Series

2nd Edition

Object-Oriented Programming using C++

- More than 300 Programs
- More than 325 Exercise Questions
- More than 500 Multiple Choice
- More than 375 Fill in the Blanks
- More than 400 True/False
- Data Structures Included

Tasleem Mustafa
Imran Saeed
Tariq Mahmood
Ahsan Raza Sattar

Freebooks.pk

IT Series

Object-Oriented Programming

using

C++

2nd Edition

Written by

Tasleem Mustafa
MS (CS), M.Sc. (CS), M.Sc. (Math), MIS (USA), OCP

Tariq Mahmood
MSc (CS), JCP, MCP

Imran Saeed
MSc (CS), JCP, MCP

Ahsan Raza Sattar
MS (CS), MSc (CS), MBA, OCP

Kitab Markaz, Faisalabad.

Freebooks.pk

Freebooks.pk

All Rights Reserved with IT Series

No part of this book may be reproduced in any form or any means without the written permission from **IT Series**.

Ch. Ashfaq Shahid
Advocate, High Court

Published by: IT Series Publications
Mob: (+92) 0321 661 56 56

Price: Rs **290/-**

Can be had from:

Kitab Markaz
Aminpur Bazar, Faisalabad,
Pakistan.
Ph: +92-041-2642707

Imtiaz Book Depot
Urdu Bazar, Lahore,
Pakistan.
Ph: +92-042-37235944

Preface

The basic objective for writing this book **"Object-Oriented Programming using C++"** is to provide complete guidance for the students to learn object-oriented programming technique for developing professional software using C++. The idea in our mind was to focus on practical aspect of programming. To fulfill this purpose, we have included hundreds of practical examples for the students.

The book provides complete programming guide for the students from beginning to advanced level. The language is very simple and easy to understand.

To cover the requirements of the students with respect to examination, there are hundreds of questions, multiple choice, fill in the blanks and true/false questions in the book so that the students may prepare for the examination from different point of views. Each chapter ends with an exercise for practice.

The book is basically meant for students but it can also be used by any person who wants to get the in-depth knowledge of programming in C++.

Readers are welcome to send suggestions for improvements of the book by sending email to us at **comments@itseries.edu.pk**. You can also visit **www.itseries.com.pk** for any interaction and latest information about the book.

Authors

Freebooks.pk

Contents

1. Introduction to Programming 1
1.1 Problem Solving 2
1.2 Program 2
1.2.1 Advantages of Computer Program 2
1.3 Algorithms & Pseudo Code 2
1.3.1 Properties of Algorithm 2
1.3.2 Advantages of Algorithms 4
1.4 Flowchart 4
1.4.1 Uses of Logic Flowchart 4
1.4.2 Flowchart Symbols 4
1.4.3 Guidelines for Drawing Flowchart 6
1.4.4 Limitations of Flowchart 7
1.4.5 Difference between Flowchart & Algorithm 8
1.5 Program Development Process 8
1.6 Programming Languages 9
1.6.1 Low Level Languages 9
1.6.2 High Level Languages 10
1.6.2.1 Procedural Languages 10
1.6.2.2 Object-Oriented Languages 11
1.6.2.3 Non-Procedural Languages 11
1.6.2.4 Difference between Procedural & Non-procedural Languages 12
1.6.3 Difference between Low-Level & High-Level Languages 12
1.6.4 Natural Programming Languages 13
1.7 Types of Codes 13
1.7.1 Source Code 13
1.7.2 Object Code 13
1.7.3 Difference between Source Code and Object Code 13
1.8 Language Processor 13
1.8.1 Compiler 13
1.8.2 Interpreter 14
1.8.3 Assembler 14
1.8.4 Difference between Compiler and Interpreter 14
1.9 Programming Techniques 14
1.9.1 Structured Programming 14
1.9.2 Object-Oriented Programming 17
1.9.3 Event-Driven Programming 17
1.9.4 Visual Programming 18

Exercise Questions 18
Multiple Choice 24
Fill in the Blanks 26
True/False 27

2. Introduction to C++ 28
2.1 History of C++ 29
2.1.1 Features of C++ 29
2.2 Basic Structure of C++ Program 30
2.2.1 Preprocessor Directive 30

Freebooks.pk

2.2.2 Header Files....31
2.2.3 main( ) Function....31
2.2.4 C++ Statements....32
2.2.5 Token....32
2.2.6 White Spaces....32
2.3 Turbo C++....33
2.3.1 Installing Turbo C++....33
2.3.2 Initializing Turbo C++ IDE....33
2.3.3 Setting up Directories....33
2.3.4 Creating and Editing a C++ Program....34
2.3.5 Saving a C++ Program....35
2.3.6 Compiling a C++ Program....36
2.3.7 Linking a C++ Program....36
2.3.8 Executing a C++ Program....36
2.4 Debugging in Turbo C++....37
2.4.1 Types of Errors....38
2.4.2 Debugging Features of Turbo C++....38
Exercise Questions....39
Multiple Choice....**40**
Fill in the Blanks....**42**
True/False....**45**
46

---

3. Programming in C++....**48**
3.1 Identifier....49
3.1.1 Types of Identifiers....49
3.2 Keywords....49
3.3 Data Types....49
3.3.1 Integer Data Types....50
3.3.2 Real Data Types....50
3.3.3 Character Data Type....51
3.4 Integer Overflow and Underflow....52
3.5 Scientific or Exponential Notation....52
3.5.1 Range and Precision....53
3.6 Variables....53
3.6.1 Variables Declaration....54
3.6.2 Rules for Declaring Variables....54
3.6.3 Variable Initialization....55
3.7 Constants....55
3.7.1 Literal Constant....56
3.7.2 Symbolic Constants....56
3.8 Expression....57
3.9 Operators....58
3.9.1 Arithmetic Operators....59
3.9.2 Assignment Statement....59
3.9.3 Compound Assignment Operators....60
3.9.4 Increment Operator....61
3.9.5 Decrement Operator....62
3.9.6 Operator Precedence....64
3.9.7 Operator Associativity....65
3.10 Type Casting....66
3.10.1 Implicit Type Casting....68

Freebooks.pk

3.10.2 Explicit Casting 69
3.11 The "sizeof" Operator 69
3.12 Comments 70

Exercise Questions 71
Multiple Choice 80
Fill in the Blanks 86
True/False 88

---

4 Input and Output 91
4.1 Input and Output 92
4.2 Standard Output 92
4.3 Escape Sequences 94
4.4 C++ Manipulators 96
4.4.1 'endl' Manipulator 96
4.4.2 'setw' Manipulator 96
4.4.3 'setprecision' Manipulator 97
4.4.4 'fixed' Manipulator 98
4.4.5 'showpoint' Manipulator 98
4.4.6 'setfill' Manipulator 99
4.5 Standard Input 100

Programming Exercise 110
Exercise Questions 111
Multiple Choice 113
Fill in the Blanks 114
True/False 115

---

5 Conditional Structures 116
5.1 Control Structures 117
5.1.1 Types of Control Structures 117
5.2 Relational Operators 119
5.2.1 Relational Expression 119
5.3 'if' Statement 119
5.3.1 Limitation of simple 'if' Statement 120
5.4 'if-else' Statement 123
5.5 Multiple 'if-else-if' Structure 126
5.6 Nested 'if' Structure 129
5.7 Compound Condition 132
5.7.1 Logical Operators 132
5.8 'switch' Structure 135
5.8.1 Difference between nested 'if-else' and 'switch' 141
5.9 Conditional Operator 141
5.10 'goto' Statement 142

Programming Exercise 143
Exercise Questions 145
Multiple Choice 153
Fill in the Blanks 157
True/False 158

Freebooks.pk

6. Looping Structures **160**
6.1 Loops 161
6.1.1 Counter-Controlled Loops 161
6.1.2 Sentinel-Controlled Loops 161
6.2 'while' Loop 161
6.3 'do-while' Loop 171
6.3.1 Difference between 'while' and 'do-while' Loop 172
6.4 Pretest and Posttest in Loops 175
6.5 'for' Loop 175
6.5.1 'continue' Statement 181
6.5.2 'break' Statement 182
6.6 Nested Loops 184

Programming Exercise **191**
Exercise Questions **196**
Multiple Choices **205**
Fill in the Blanks **209**
True/False **209**

---

7 Arrays **211**
7.1 Arrays 212
7.1.1 Advantages / Uses of Arrays 212
7.1.2 Declaring One-Dimensional Array 212
7.1.3 Array Initialization 213
7.1.4 Accessing Individual Elements of Array 213
7.1.5 Accessing Array Elements using Loops 214
7.1.6 Input and Output Values of an Array 214
7.2 Searching in Arrays 219
7.2.1 Sequential Search 220
7.2.2 Binary Search 220
7.3 Sorting Arrays 221
7.3.1 Selection Sort 222
7.3.2 Bubble Sort 225
7.4 Two-Dimensional Arrays 228
7.4.1 Accessing Individual Elements of 2-D Array 229
7.4.2 Entering Data in 2-D Arrays 229
7.4.3 Initializing 2-D Arrays 230
7.5 Multidimensional Arrays 234
7.5.1 Accessing Multidimensional Arrays 235

Programming Exercise **237**
Exercise Questions **238**
Multiple Choice **242**
Fill in the Blanks **244**
True/False **245**

---

8. Structures **246**
8.1 Structures 247
8.1.1 Declaring a Structure 247
8.1.2 Defining Structure Variable 248

Freebooks.pk

8.1.3 Accessing Members of Structure Variable 248
8.1.4 Initializing Structure Variables 251
8.1.5 Assigning One Structure Variable to Other 253
8.1.6 Array as Member of Structure 254
8.2 Array of Structures 255
8.2.1 Initializing Array of Structures 256
8.3 Nested Structure 257
8.3.1 Accessing Members of Nested Structure 257
8.3.2 Initializing Nested Structure 258
8.4 Union 261
8.4.1 Difference between Structure and Union 262
8.5 Enumerations 262

Programming Exercise 263
Exercise Questions 264
Multiple Choice 266
Fill in the Blanks 266
True/False 267

---

9 Functions 268
9.1 Functions 269
9.1.1 Importance of Functions 269
9.1.2 Advantages of Functions 269
9.2 Types of Functions in C++ 270
9.3 User Defined Functions 270
9.3.1 Function Declaration or Function Prototype 270
9.3.2 Function Definition 271
9.3.3 Function Call 272
9.3.4 Scope of Functions 273
9.4 Passing Parameters to Functions 273
9.4.1 Pass by Value 274
9.4.2 Pass by Reference 279
9.4.3 Difference between Call by Value and Call by Reference 280
9.4.4 Returning Value from Function 281
9.5 Local Variable 286
9.5.1 Scope of Local Variable 287
9.5.2 Lifetime of Local Variable 287
9.6 Global Variable 287
9.6.1 Scope of Global Variable 287
9.6.2 Lifetime of Global Variable 287
9.6.3 Difference between Local and Global Variable 288
9.7 Static Variable 288
9.7.1 Scope of Static Variable 288
9.7.2 Lifetime of Static Variable 288
9.8 Register Variables 289
9.9 Functions and Arrays 290
9.9.1 Calling a Function with Array Parameter 290
9.9.2 Passing Individual Array Element to Function 292
9.9.3 Passing Two-Dimensional Array to Function 292
9.10 Functions and Structures 293
9.10.1 Passing Structure by Value 293

Freebooks.pk

9.10.2 Passing Structure by Reference 297
9.10.3 Returning Structure from Function 299
9.11 Default Parameters 300
9.12 Inline Functions 302
9.13 Command Line Parameters 302
9.14 Function Overloading 303
9.15 Recursion 306

Programming Exercise 308
Exercise Questions 310
Multiple Choice 316
Fill in the Blanks 319
True/False 321

---

10. Built-in Functions 323
10.1 Built-in Functions 324
10.2 The 'conio.h' Header File 324
10.3 The 'stdio.h' Header File 326
10.4 Math Functions (math.h) 327
10.5 Type Functions / Character Functions (ctype.h) 331

Multiple Choice 334
Fill in the Blanks 334

---

11. Pointers 335
11.1 Memory and References 336
11.2 Pointers 337
11.2.1 Pointer Declaration 337
11.2.2 The 'void' Pointer 337
11.2.3 Dereference Operator 338
11.2.4 Pointer Initialization 339
11.3 Operations on Pointers 339
11.3.1 Pointer Addition 340
11.3.2 Pointer Subtraction 340
11.4 Pointers and Arrays 341
11.4.1 Accessing Array Elements with Pointers 341
11.5 Pointers and Strings 343
11.6 Array of Pointers 344
11.7 Pointers and Functions 345
11.8 Pointers and Structures 346
11.8.1 Passing Structure to Function using Pointers 347
11.9 Memory Management with Pointers 349
11.9.1 Dynamic Variables 349
11.9.2 The new Operator 349
11.9.3 The delete Operator 350

Programming Exercises 352
Exercise Questions 352
Multiple Choice 355
Fill in the Blanks 357
True/False 358

Freebooks.pk

12. String Handling 359
12.1 String 360
12.1.1 String Declaration 360
12.1.2 String Initialization 361
12.2 String Input 361
12.2.1 The cin Object 361
12.2.2 cin.getline() 362
12.2.3 cin.get() 363
12.3 Array of Strings 366
12.3.1 Initializing Array of Strings 366
12.3.2 Input/Output with Array of Strings 367
12.4 String Functions (string.h) 368
12.4.1 memchr() 368
12.4.2 memcmp() 369
12.4.3 memcpy() 370
12.4.4 memmove() 371
12.4.5 memset() 371
12.4.6 strcat() 372
12.4.7 strncat() 372
12.4.8 strchr() 373
12.4.9 strrchr() 374
12.4.10 strcmp() 374
12.4.11 stricmp() 375
12.4.12 strncmp() 376
12.4.13 strcoll() 377
12.4.14 strcpy() 378
12.4.15 strncpy() 378
12.4.16 strcspn() 378
12.4.17 strlen() 379
12.4.18 strstr() 381
12.4.19 strpbrk() 381
12.4.20 strspn() 382
12.4.21 strtok() 382
12.4.22 strerror() 383
12.4.23 strxfrm() 383
12.4.24 strrev() 384
12.4.25 strset() 384
12.4.26 strnset() 385
12.4.27 strlwr() 385
12.4.28 strupr() 386
12.4.29 strdup() 386
12.4.30 stpcpy() 387

Programming Exercise 387
Multiple Choice 388
Fill in the Blanks 389
True/False 389

---

13. Basics of Object-oriented programming 390
13.1 Object-Oriented Programming 391
13.1.1 Features of Object-Oriented Programming 391

Freebooks.pk

13.2 Objects....391
13.2.1 Properties of Object....392
13.2.2 Functions of Object....392
13.3 Classes....393
13.3.1 Declaring a Class....393
13.3.2 Access Specifiers....394
13.4 Creating Objects....394
13.4.1 Executing Member Functions....395
13.4.2 Defining Member Functions outside Class....400
13.5 Constructors....402
13.5.1 Passing Parameters to Constructors....403
13.5.2 Constructor Overloading....405
13.6 Default Copy Constructor....406
13.7 Destructors....407
13.8 Objects as Function Parameters....408
13.8.1 Returning Objects from Member Functions....410
13.9 Static Data Member....411
13.10 Friend Functions....413
13.11 Friend Classes....415
13.12 Static Functions....416

Programming Exercise....418
Exercise Questions....419
Multiple Choice....420
Fill in the Blanks....422
True/False....423

---

14. Operator Overloading....425
14.1 Operator Overloading....426
14.1.1 Overloading an Operator....426
14.2 Overloading Unary Operators....426
14.2.1 Overloading ++ Operator....427
14.2.2 Operator Overloading with Returned Value....427
14.2.3 Overloading Postfix Increment Operator....428
14.3 Overloading Binary Operators....429
14.3.1 Arithmetic Operators....430
14.3.2 Overloading Comparison Operator....432
14.3.3 Overloading Arithmetic Assignment Operators....433

Programming Exercise....434
Exercise Questions....435
Fill in the Blanks....435

---

15. Inheritance....436
15.1 Inheritance....437
15.1.1 Advantages of Inheritance....437
15.1.2 Categories of Inheritance....438
15.1.3 Protected Access Specifier....438
15.2 Specifying a Derived Class....438
15.3 Accessing Members of Parent Class....440

Freebooks.pk

15.3.1 Accessing Constructors of Parent Class 440
15.3.2 Accessing Member Functions of Parent Class 442
15.4 Function Overriding 444
15.5 Types of Inheritance 448
15.5.1 Public Inheritance 448
15.5.2 Protected Inheritance 450
15.5.3 Private Inheritance 452
15.6 Multilevel Inheritance 454
15.6.1 Multilevel Inheritance with Parameters 457
15.7 Multiple Inheritance 459
15.7.1 Constructors in Multiple Inheritance 461
15.7.1.2 Constructors with Parameters 462
15.7.2 Ambiguity in Multiple Inheritance 464
15.7.2.1 Removing Ambiguity 465
15.8 Containership 465

Programming Exercise 467
Exercise Questions 468
Multiple Choice 469
Fill in the Blanks 471
True/False 472

---

16. Polymorphism and Virtual Function 473
16.1 Polymorphism 474
16.2 Pointer to Objects 474
16.2.1 Array of Pointers to Objects 475
16.3 Pointers and Inheritance 476
16.4 Virtual Functions 478
16.4.1 Early Binding 479
16.4.2 Late Binding 479
16.5 Pure Virtual Functions 481
16.5.1 Abstract Classes 481
16.6 Virtual Base Classes 482

Exercise Questions 484
Multiple Choice 487
Fill in the Blanks 487
True/False 488

---

17. Template 489
17.1 Templates 490
17.2 Function Templates 490
17.2.1 Declaring Function Template 490
17.2.2 Using Function Templates 491
17.3 Class Templates 493
17.3.1 Using Class Templates 494

Programming Exercise 495
Multiple Choice 495
True/False 496

Freebooks.pk

18. File Handling 497
18.1 Files 498
18.1.1 Advantages of Files 498
18.1.2 Types of Files 498
18.2 File Access Methods 498
18.2.1 Sequential Access Method 498
18.2.2 Random Access Method 499
18.3 Stream 499
18.3.1 Types of Streams 499
18.3.2 Predefined Stream objects 500
18.3.3 Stream Class Hierarchy 500
18.4 Opening Files 501
18.4.1 Default Opening Modes 501
18.4.2 Verifying File Open 502
18.5 Closing Files 502
18.6 Formatted Files I/O 502
18.6.1 Writing Data to Formatted Files I/O 502
18.6.2 Reading Data from Formatted Files I/O 504
18.6.3 Detecting End-of-File 504
18.6.4 Reading Lines from Files 505
18.7 Character I/O 506
18.7.1 Writing Single Character 506
18.7.2 Reading Single Character 507
18.8 Binary I/O 508
18.8.1 Writing Data in Binary I/O 508
18.8.2 Reading Data in Binary I/O 509
18.9 Object I/O 510
18.9.1 Writing an Object to Disk 511
18.9.2 Reading an Object from Disk 511
18.10 Accessing Records Randomly 512
18.10.1 File Pointers 512
18.10.2 The seekg() Function 512
18.10.3 The seekp() Function 513
18.11 Printing the Files through Streams 517

Programming Exercise 517
Exercise Questions 518
Multiple Choice 518
Fill in the Blanks 519
True/False 520

---

19. Data Structures 521
19.1 Data Structures 522
19.1.1 Types of Data Structures 522
19.1.2 Data Structure Operations 522
19.2 Linked List 523
19.2.1 Types of Linked Lists 523
19.2.1.1 Singly-linked List 523
19.2.1.2 Doubly-Linked List 525
19.2.1.3 Circularly-Linked List 529
19.3 Stack 532

Freebooks.pk

19.3.1 Representation of Stack 533
19.3.1.1 Stack using Arrays 533
19.3.1.2 Stack using Linked List 533
19.4 Queue 536
19.4.1 Representation of Queue 537
19.4.1.1 Queue using Arrays 537
19.4.1.2 Queue using Linked List 537
19.5 Trees 540
19.5.1 Tree Terminology 540
19.5.2 Binary Trees 541
19.5.3 Traversing Binary Trees 542
19.5.4 Insertion in Binary Tree 542
19.5.5 Binary Tree Deletion 543
19.5.6 Advantages of Binary Tree 545
19.5.7 Complete Binary Trees 545
19.5.8 Binary Search Trees 545
19.6 Graphs 551
19.6.1 Types of Graphs 551
19.6.1.1 Directed Graph 551
19.6.1.2 Undirected Graphs 551

Exercise Questions 552
Multiple Choices 553
True/False 554

Note: The solutions of all programming exercises are available in the following book:

A Key to Programming Exercises of

OBJECT ORIENTED PROGRAMMING USING C++

Freebooks.pk

CHAPTER 1

INTRODUCTION TO PROGRAMMING

Chapter Overview

1.1 Problem Solving
1.2 Program
1.2.1 Advantages of Computer Program
1.3 Algorithms & Pseudo Code
1.3.1 Properties of Algorithm
1.3.2 Advantages of Algorithms
1.4 Flowchart
1.4.1 Uses of Logic Flowchart
1.4.2 Flowchart Symbols
1.4.3 Guidelines for Drawing Flowchart
1.4.4 Limitations of Flowchart
1.4.5 Difference between Flowchart & Algorithm
1.5 Program Development Process
1.6 Programming Languages
1.6.1 Low Level Languages
1.6.2 High Level Languages
1.6.2.1 Procedural Languages
1.6.2.2 Object-Oriented Languages
1.6.2.3 Non-Procedural Languages
1.6.2.4 Difference between Procedural & Non-procedural Languages
1.6.3 Difference between Low-Level & High-Level Languages
1.6.4 Natural Programming Languages
1.7 Types of Codes
1.7.1 Source Code
1.7.2 Object Code
1.7.3 Difference between Source Code and Object Code
1.8 Language Processor
1.8.1 Compiler
1.8.2 Interpreter
1.8.3 Assembler
1.8.4 Difference between Compiler and Interpreter
1.9 Programming Techniques
1.9.1 Structured Programming
1.9.2 Object-Oriented Programming
1.9.3 Event-Driven Programming
1.9.4 Visual Programming

Exercise Questions
Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

1.1 Problem Solving

Problem solving is a process of identifying a problem and finding the best solution for it. Problem solving is a skill that can be developed by following a well organized approach. We solve different problems every day. Every problem is different in its nature. Some problems are very difficult and require more attention to identify the solution. A problem may be solved in different ways. One solution may be faster, less expensive and more reliable than others. It is important to select the best suitable solution.

Different strategies, techniques and tools are used to solve a problem. Computers are used as a tool to solve complex problems by developing computer programs. Computer programs contain different instruction for computer. A programmer writes instructions and computer executes these instructions to solve a problem. A person can be good programmer if he has the skill of solving problems. Different problem-solving techniques are as follows:

- Program
- Algorithm
- Flowchart etc.

1.2 Program

A set of instructions that tells a computer what to do is called **program**. A computer works according to the given instructions in the program. Computer programs are written in programming languages. A person who develops a program is called **programmer**. The programmer develops programs to instruct the computer how to process data to convert into information. Programmer uses programming languages or tools to write programs.

1.2.1 Advantages of Computer Program

Different advantages of computer program are as follows:

- A computer program can solve many problems by giving instructions to computer.
- A computer program can be used to perform a task repeatedly and quickly.
- A program can process a large amount of data easily.
- It can display the results in different styles.
- The processing of a program is more efficient and less time consuming.
- Different types of programs are used in different fields to perform certain tasks.

1.3 Algorithms & Pseudo Code

An algorithm is a step-by-step procedure to solve a problem. The process of solving a problem becomes simpler and easier with help of algorithm. It is better to write algorithm before writing the actual computer program.

1.3.1 Properties of Algorithm

Following are some properties of an algorithm:

- The given problem should be broken down into simple and meaningful steps.
- The steps should be numbered sequentially.
- The steps should be descriptive and written in simple English.

Algorithms are written in a language that is similar to simple English called **pseudo code**. There is no standard to write pseudo code. It is used to specify program logic in an English like manner that is independent of any particular programming language.

Freebooks.pk

Pseudo code simplifies program development by separating it into two main parts.

1. Logic Design

In this part, the logic of the program is designed. We specify different steps required to solve the problem and the sequence of these steps.

2. Coding

In this part, the algorithm is converted into a program. The steps of algorithm are translated into instructions of any programming language.

The use of pseudo code allows the programmer to focus on the planning of the program. After the planning is final, it can be written in any programming language.

Example 1

The following algorithm inputs two numbers, calculates sum and then displays the result on screen.

1. Start
2. Input A
3. Input B
4. Total = A + B
5. Display Total
6. Exit

Example 2

The following algorithm inputs radius from the user and calculates the area of circle. (Hint: Area = 3.14 * radius * radius)

1. Start
2. Input radius in r
3. area = 3.14 * r * r
4. Print area
5. End

Example 3

The following algorithm calculates the distance covered by a car moving at an average speed of V m/s in time T. It should input average speed v and time t. (Hint: s = vt where s is the distance traveled).

1. Start
2. Input average speed in v
3. Input time in t
4. s = v * t
5. Print s
6. End

Example 4

The following algorithm finds the sum of first fifty natural numbers.

1. Start
2. sum = 0
3. N = 0
4. Repeat Step 5 and 6 While (N <= 50)
5. sum = sum + N

Freebooks.pk

6. N = N + 1
7. Print sum
8. End

1.3.2 Advantages of Algorithms

Some advantages of algorithm are as follows:

1. Reduced Complexity

Writing algorithm and program separately simplifies the overall task by dividing it into two simpler tasks. While writing the algorithm, we can focus on solving the problem instead of concentrating on a particular language.

2. Increased Flexibility

Algorithm is written so that the code may be written in any language. Using the algorithm, the program could be written in Visual Basic, Java or C++ etc.

3. Ease of Understanding

It is not necessary to understand a particular programming language to understand an algorithm. It is written in an English like manner.

1.4 Flowchart

Flowchart is combination of two words **flow** and **chart**. A chart consists of different symbols to display information about any program. Flow indicates the direction of processing that takes place in the program.

Flowchart is a graphical representation of an algorithm. It is a way of visually presenting the flow of data, operations performed on data and sequence of these operations. Flowchart is similar to the layout plan of a building. A designer draws the layout plan of the building before constructing it. Similarly, a programmer prefers to design the flowchart before writing the computer program. Flowchart is designed according to the defined rules.

1.4.1 Uses of Logic Flowchart

Flowchart is used for the following reasons:

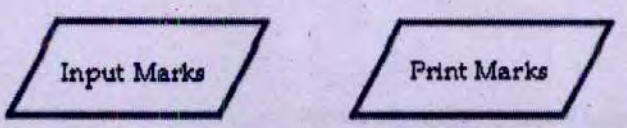
1. Flowchart is used to represent an algorithm in simple graphical manner.
2. Flowchart is used to show the steps of an algorithm in an easy way.
3. Flowchart is used to understand the flow of the program.
4. Flowchart is used to improve the logic for solving a problem.
5. Programs can be reviewed and debugged easily.

1.4.2 Flowchart Symbols

Flowchart uses simple symbols to show different types of statements:

1. Input/Output

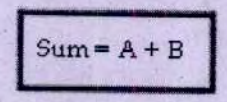
Parallelogram symbol is used to represent an input or output step. Input statement is used to get input from the user. The output statement is used to display a message to the user or to display a value.



Freebooks.pk

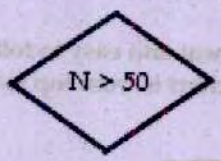
2. Process

Rectangle symbol is used to represent a process step. A process may be a complex calculation or simply an assignment statement.



3. Selection

Diamond symbol is used to represent a selection step. A condition is given in the diamond. The flow of control from diamond may go in two directions i.e. one direction if the condition is true and the second direction if the condition is false.

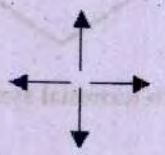


4. Start/End

Oval symbol is used to represent the start or end of the flowchart.

5. Flow Lines

Arrow symbols are used to represent the direction of flow in the flowchart. There are four types of flow lines.

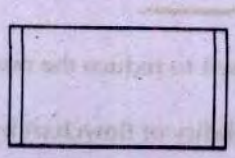


6. Connector

Circle symbol is used to combine different flow lines. It is used when two or more flow symbols come from different directions and move to one direction.

7. Function Call

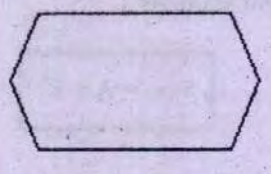
Rectangle symbol with double lines on left and right sides is used to call a function. The name of function with parameters is written inside the symbol.



Freebooks.pk

8. Preparation

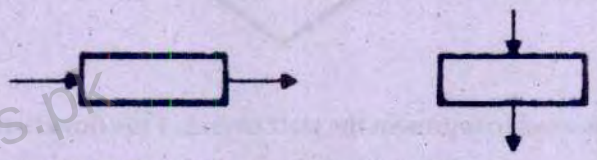
This symbol is used with for loops to specify start and stop conditions.



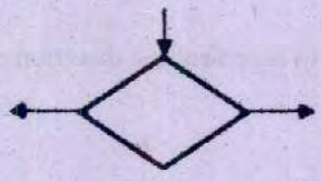
1.4.3 Guidelines for Drawing Flowchart

Different guidelines for drawing a flowchart are as follows:

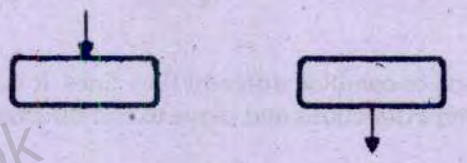
1. All necessary requirements should be listed in a logical order while drawing a flowchart.
2. Flowchart should be clear, neat and easy to follow.
3. The usual direction of flowchart is from top to bottom or left to right.



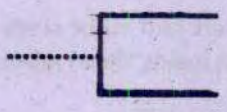
4. Only one line should come out from process symbol.
5. Only one flow line must enter a decision symbol. But two flow lines, one for each possible answer, must come out of decision symbol.



6. Only one flow line is used with terminal symbol.



7. The start and end of flowchart must be logical.
8. The comments should be written in remarks symbol.



9. The connector should be used to reduce the number of flow lines if the flowchart becomes complex.
10. It is useful to ensure the validity of flowchart by testing it with sample data.

Freebooks.pk

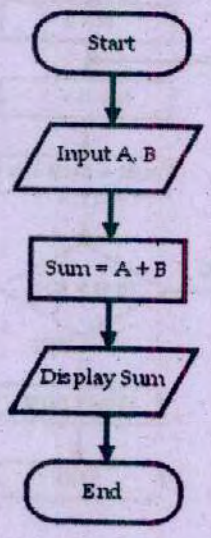
1.4.4 Limitations of Flowchart

The limitations of flowchart are as follows:

1. It is difficult to draw flowcharts for complex problems.
2. The flowchart has to be redesigned if any change is required.

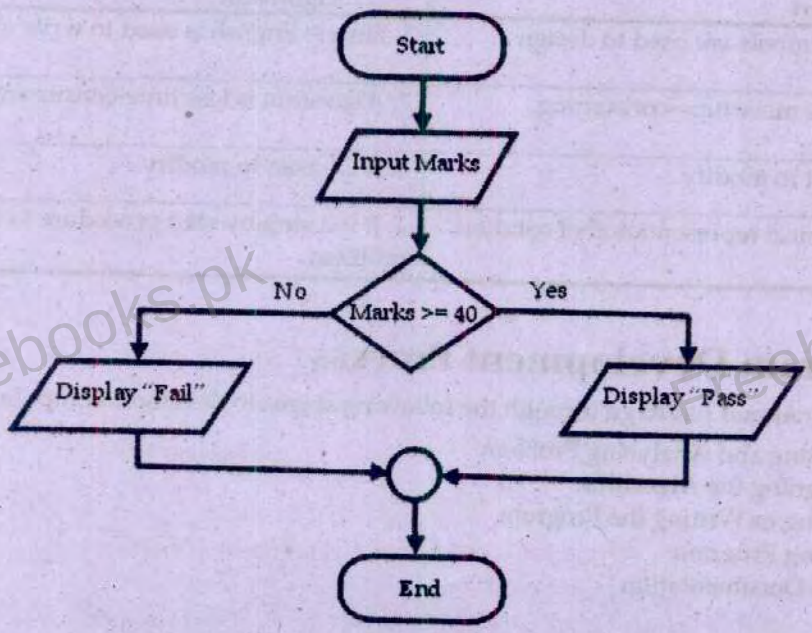
Example 1

The following flowchart represents the steps for adding two numbers.



Example 2

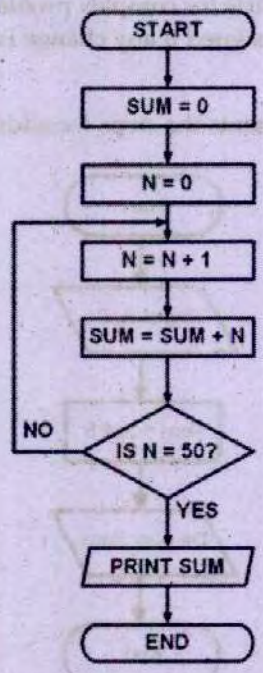
The following flowchart represents the steps of an algorithm that inputs marks of a student. If marks are 40 or more, it displays "Pass". Otherwise it displays "Fail".



Freebooks.pk

Example 3

The following flowchart finds the sum of first 50 natural numbers.



1.4.5 Difference between Flowchart & Algorithm

The difference between flowchat and algorithm is as follows:

Flowchart	Algorithm
1. Standard symbols are used to design flowchart.	1. Simple English is used to write algorithm.
2. Flowchart is more time-consuming.	2. Algorithm is less time-consuming.
3. It is difficult to modify.	3. It is easier to modify.
4. It is a graphical representation of solution.	4. It is a step-by-step procedure to solve a problem.

1.5 Program Development Process

A programmer has to go through the following stages to develop a computer program:

1. Defining and Analyzing Problem
2. Designing the Algorithm
3. Coding or Writing the Program
4. Testing Program
5. Final Documentation

Freebooks.pk

1. Defining and Analyzing the Problem

In this step, a programmer studies the problem. He decides the best way to solve this problem. Studying a problem is necessary because it helps a programmer to decide about the following things:

- The facts and figures which are necessary for developing the program.
- The way in which the program will be designed.
- The language in which the program will be most suitable.
- What is the desired output and in which form it is needed, etc.

2. Designing the Algorithm

An algorithm is a sequence of steps that must be carried out before a programmer starts preparing his program. The programmer designs an algorithm to help visualize possible alternatives in a program.

3. Coding or Writing the Program

The process of writing a program is a very important step in program development. In this step, an algorithm is converted into program. The program consists of different steps given in the algorithm. A large number of programming languages are available to write programs. A programmer selects programming language according to the nature of problem.

4. Testing Program

A program must be tested in the process of program development. This process verifies the accuracy of a program. The program is tested by executing it again and again. Different values are given as input and output is checked. The program may not give required results if it contains any error. The errors must be detected and corrected if the output is not correct. All bugs in the program are detected and removed during program testing. It ensures that the program gives desired results and the problem is solved correctly.

5. Final Documentation

When the program is finalized, its documentation is prepared. Final documentation is provided to the user. It guides the user how to use the program in the most efficient way. Another purpose of documentation is to allow some other programmer to modify the code if necessary. Documentation should be done in each step during development of a program.

1.6 Programming Languages

A set of words, symbols and codes used to write programs is called **program language**. Different programming languages are available for writing different types of programs. Some languages are specially used for writing business programs, other are used for writing scientific programs etc.

There are two types of computer programming languages:

- Low-level languages
- High-level languages

1.6.1 Low Level Languages

These languages are near to computer hardware and far from human languages. Computer can understand these languages easily. Writing a program in low-level languages requires a deep knowledge of the internal structure of computer hardware. Two low-level languages are machine language and assembly language.

Freebooks.pk

1. Machine Language

A type of language in which instructions are written in binary form is called **machine language**. It is the only language that is directly understood by the computer. It is the fundamental language of the computer.

Program written in machine language can be executed very fast by the computer. Programs written in machine language are machine-dependent. Every computer has its own machine language. Machine language is difficult to understand. Writing and modifying program in machine language takes a lot of time. Machine language is also known as **first generation language**.

2. Assembly Language

Assembly language is a low-level language. It is one step higher than machine language. In assembly language, symbols are used instead of binary code. These symbols are called **mnemonics**. For example **Sub** instruction is used to subtract two numbers.

Assembly language is also called **symbolic language.** Programs written in assembly language are easier to write and modify than machine language. Assembly language is mostly used for writing system software. Assembly language is also known as **second generation language.**

1.6.2 High Level Languages

A type of language that is close to human languages is called high level language. High-level languages are easy to understand. Instructions of these languages are written in English-like words such as **input** and **print** etc.

A program written in high-level language is easier to write and modify. High-level languages are further divided into following categories:

- Procedural Languages
- Object-Oriented Languages
- Non-Procedural Languages

1.6.2.1 Procedural Languages

Procedural languages are also known as **third-generation languages** or **3GL**. In these languages, program is a predefined set of instructions. Computer executes these instructions in the same sequence in which the instructions are written. Each instruction in this language tells the computer what to do and how to do. Some most popular procedural languages are:

1. FORTRAN

FORTRAN stands for **FORmula TRANslation**. It is mainly used for engineering application and scientific use.

2. BASIC

BASIC stands for **Beginner All Purpose Symbolic Instruction Code**. It was created in the late 1960. It was used mainly by students to use the computer for solving simple problems. It is easy to understand. It is widely used for education purpose.

3. COBOL

COBOL stands for **Common Business Oriented Language**. It is specially designed for business application. It was developed in early 1960s. The programs written in COBOL are lengthy but easy to read, write and maintain.

Freebooks.pk

4. PASCAL

This language is used for both scientific and business applications. Its name was assigned in the honor of a French mathematician Pascal.

5. C

C language is a popular high-level language. It was developed by **Dennis Ritchie** at **AT&T Bell Laboratories** in 1972. It was written as part of **UNIX** operating system. It is also known as middle-level language because it provides the facilities to write application software as well as system software.

1.6.2.2 Object-Oriented Languages

OOP is a technique in which programs are written on the basis of objects. An object is a collection of data and functions. Object may represent a person, thing or place in real world.

In OOP, data and all possible functions on data are grouped together. Object oriented programs are easier to learn and modify. C++ & Java are popular object-oriented languages.

Features of Object-Oriented Programming

Following are some features of object-oriented programming:

- **Objects** – OOP provides the facility of programming based on objects. Object is an entity that consists of data and functions.
- **Classes** – Classes are designs to create objects. OOP provides facility to design classes to create different objects. Properties and functions of objects are specified in classes.
- **Real-world Modeling** – OOP is based on real-world modeling. As in real world, things have properties and working capabilities. Similarly, objects have data and functions. Data represents properties and functions represent working of objects.
- **Reusability** – OOP provides ways to reuse data and code. **Inheritance** is a technique that allows a programmer to use code of existing program to create new programs.
- **Information Hiding** – OOP allows the programmer to hide important data from the user. It is performed by encapsulation.
- **Polymorphism** – Polymorphism is an ability of an object to behave in multiple ways.

C++

C++ is an object-oriented language. It was developed in 1980 at Bell Laboratories. It is an improved version of C language. It provides the facility of working with objects and classes. It is very powerful language and is used to develop a variety of programs.

Java

Java is a high-level language. It was designed by **Sun Microsystems**. It was primarily developed to control the microprocessors used in VCR, toasters and cable receivers etc. It provides powerful capabilities of network programming, Internet applications and **graphical user interface (GUI)**.

1.6.2.3 Non-Procedural Languages

Non-procedural languages are also known as **fourth generation languages** or **4GL**. In non-procedural languages, user only needs to tell the computer "what to do" not "how to do". An important advantage of non procedural languages is that they can be used by non-technical user to perform a specifice task. These languages accerlate program process and reduce coding errors. 4GL are normally used in database applications and report generation.

Freebooks.pk

Some important non-procedural languages are as follows:

1. SQL

SQL stands for **Structured Query Language**. It is the most popular database query language. SQL was developed by IBM. It is a national standard by the **American National Standards Institute (ANSI)**. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

2. RPG

RPG stands for **Report Program Generator**. It was developed in early 1960s by IBM. It is used to generate business reports. It is a nonprocedural language. It is mostly used with IBM mid-range computers.

1.6.2.4 Difference between Procedural & Non-procedural Languages

Procedural Languages / 3GL	Non-procedural Languages / 4GL
1. Procedural language tells the computer what to do and how to do.	1. Non-procedural language tells the computer what to do, not how to do.
2. It is difficult to learn.	2. It is easy to learn.
3. It is difficult to debug.	3. It is easy to debug.
4. It requires large number of procedural instructions.	4. It requires a few non-procedural instructions.
5. It is normally used by professional programmers.	5. It can be used professional and non-technical users.
6. It is typically file-oriented.	6. It is typically database-oriented.
7. Procedural language provides many programming capabilities.	7. Non-procedural language provides less programming capabilities.

1.6.3 Difference between Low-Level & High-Level Languages

High-level Language	Low-level Language
1. High-level languages are easy to learn.	1. Low-level languages are difficult to learn.
2. High-level languages are near to human languages.	2. Low-level languages are far from human languages.
3. Translator is required.	3. No translator is required.
4. Programs in high-level languages are slow in execution.	4. Programs in low-level languages are fast in execution.
5. Programs in high-level languages are easy to modify.	5. Programs in low-level languages are difficult to modify.
6. High-level languages do not provide much facility at hardware level.	6. Low-level languages provide facility to write programs at hardware level.
7. Deep knowledge of hardware is not required to write programs.	7. Deep knowledge of hardware is required to write programs.
8. These languages are normally used to write application programs.	8. These languages are normally used to write hardware programs.

Freebooks.pk

1.6.4 Natural Programming Languages

Natural programming languages are also known as **fifth generation languages (5GL)** or **intelligent languages**. Translator programs for these languages are very complex and require a large amount of computer resources. That is why most of these languages are still in experimental phase.

1.7 Types of Codes

There are two types of codes that are as follows:

1.7.1 Source Code

A program written in a high-level language is called **source code**. Source code is also called **source program**. Computer cannot understand the statements of high-level language. The source code cannot be executed by computer directly. It is converted into object code and then executed.

1.7.2 Object Code

A program in machine language is called **object code**. It is also called **object program** or **machine code**. Computer understands object code directly.

1.7.3 Difference between Source Code and Object Code

The main difference between source code and object code is as follows:

Source Code	Object Code
1. Source code is written in high-level or assembly language.	1. Object code is written in machine language through compilers.
2. Source code is easy to understand.	2. Object code is difficult to understand.
3. Source code is easy to modify.	3. Object code is difficult to modify.
4. Source code contains fewer statements than object code.	4. Object code contains more statements than source code.

1.8 Language Processor

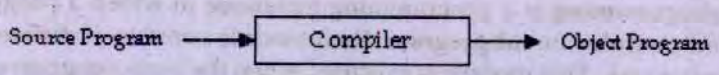
Computer understands only machine language. A program written in high-level or assembly language cannot be run on a computer directly. It must be converted into machine language before execution. **Language processor** or translator is a software that converts these programs into machine language. Every computer language has its own translators.

Different types of language processors are as follows:

1.8.1 Compiler

A **compiler** is a program that converts the instruction of a high level language into machine language as a whole. A program written in high-level language is called **source program**. Compiler converts source program into machine code known as **object program**.

The compiler checks each statement in the source program and generates machine instructions. Compiler also checks **syntax errors** in the program. A source program containing an error cannot be compiled.



A compiler can translate the programs of only that language for which it is written. For example C compiler can translate only those programs that are written in C language.

1.8.2 Interpreter

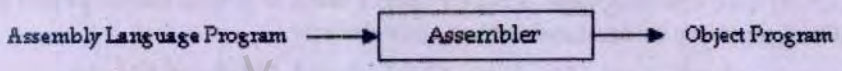
An **interpreter** is a program that converts one statement of a program at one time. It executes this statement before translating the next statement of the source program. If there is an error in the statements, the interpreter stops working and displays an errors message.

The advantage of interpreters over compilers is that an error is found immediately. So the programmer can correct errors during program development.

The disadvantage of interpreter is that it is not very efficient. The interpreter does not produce an object program. It must convert the program each time it is executed. Visual Basic uses interpreter.

1.8.3 Assembler

An **assembler** is translating program that translates the instruction of a assembly language into machine language.



1.8.4 Difference between Compiler and Interpreter

Following is a brief difference between compiler and interpreter:

Compiler	Interpreter
1. Compiler converts a program into machine code as a whole.	1. Interpreter converts a program into machine code statement by statement.
2. Compiler creates object code file.	2. Interpreter does not create object code file.
3. Compiler converts high-level program that can be executed many times.	3. Interpreter converts high-level program each time it is executed.
4. Program execution is fast.	4. Program execution is slow.
5. Compiler displays syntax errors after compiling the whole program.	5. Interpreter displays the syntax error on each statement of program.

Relationship of Object Program, Source Program and Compiler

A source program is written by a programmer in a programming language such as C. This program is not in a form that a computer can understand. A compiler translates that source program into the object program that the computer can understand and execute.

1.9 Programming Techniques

Computer programs are developed by using different programming techniques. Some important programming techniques are as follows:

1.9.1 Structured Programming

Structured programming is a programming technique in which a program is divided in small units called **modules** or **subprogram**. Each module consists of different instructions to perform a particular task. This module is executed when the main program calls it.

Freebooks.pk

When main program calls a module, the control moves to the called module temporarily. The control moves back to the main program after executing the instructions of the module.

Structured programming is an easy and simple technique for writing programs. It makes programs easier to write, check, read and modify. This technique uses only three types of instructions. So the programs are not very complex. Following types of instructions are used in this technique:

- Sequential Structure
- Conditional / Selective Structure
- Iterative / Repetitive Structure

1. Sequential Structure

In **sequential structure**, the statements are executed in the same order in which they are specified in program. The control flows from one statement to other in a logical sequence. **All statements are executed exactly once. It means that no statement is skipped and no statement is executed more than once**

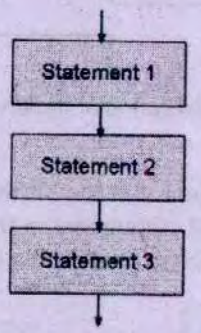


Figure 1.1: Execution of Sequential Statements

2. Conditional Structure

A **selection structure** selects a statement or set of statements to execute on the basis of a condition. In this structure, statement or set of statements is executed when a particular condition is **true** and ignored when the condition is **false** It is also known as **branching** or **conditional structure.**

Suppose a program displays **Pass** if the student gets 40 or more than 40 marks. It displays **Fail** when the marks are below 40. The program checks the marks before displaying the message.

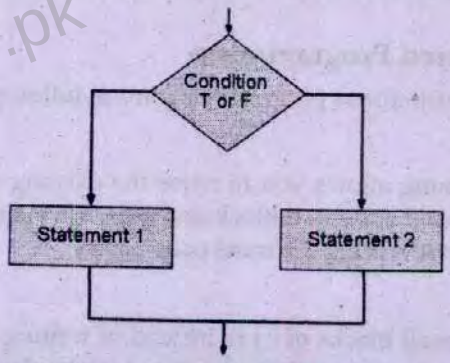


Figure 1.2: Execution of Conditional Statements

Freebooks.pk

3. Iterative Structure

A **repetition structure** executes a statement or set of statements repeatedly. It is also known as **iteration structure** or **loop**. Loops are basically used for two purposes:

- To execute a statement or number of statements for a specified number of times. For example, a user may display his name on screen for 10 times.
- To use a sequence of values. For example, a user may display a set of natural numbers from 1 to 10.

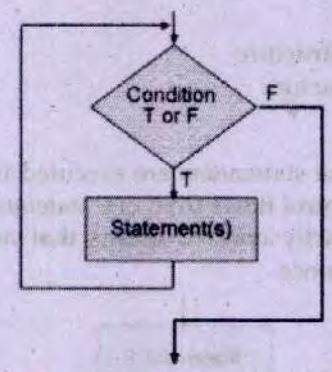


Figure 1.3: Execution of Iterative Statements

4. Function call

Function call is a type of statement that moves the control to another block of code. The control returns back after executing all statements in the block. The remaining statements are executed immediately after the function call when the control is returned.

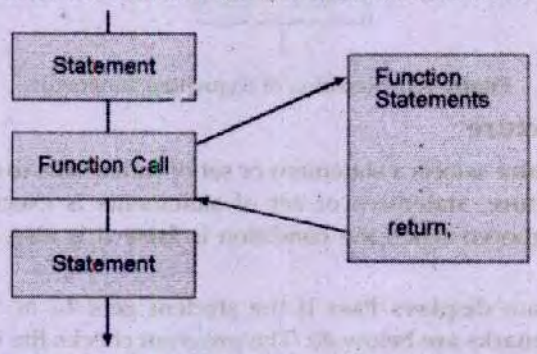


Figure 1.4: Execution of Function Call Statements

Advantages of Structured Programming

Some advantages of structured programming are as follows:

1. Reusability

Structured programming allows you to reuse the existing code as and when required. You can write frequently used code in a block and give it a specific name. This code can be used again and again without writing the same code again.

2. Easier to Code

It is easier to write small blocks of code instead of writing a long program as a whole. You can focus your attention on a specific problem in a particular block.

Freebooks.pk

3. Easier to Modify

Each block of code has a unique name and is written separately from other code. So it is easier to modify than a long program.

4. Easier to Maintain

Small blocks are comparatively easier to maintain as compared to long programs. Each block is independent so any change in it does not affect other blocks.

Disadvantage of Structured Programming

In structured programming, data cannot be considered as independent from functions that manipulate it. Procedural programming prevents the programmer from seperating data from functions.

1.9.2 Object-Oriented Programming

OOP is a technique in which programs are written on the basis of objects. An **object** is a collection of data and functions. Object may represent a person, thing or place in real world. In OOP, data and all possible functions on data are grouped together. Object oriented programs are easier to learn and modify. C++ and Java are most popular object-oriented languages.

Example

A person can be an example of an object. It has some **properties** or **characteristics** that describe what it is. Some properties of **Person** object can be as follows:

- Name
- Age
- Height

The Person object also has some **functions** that describe what it can do. Some functions of **Person** object can be as follows:

- Walk
- Talk
- Eat

1.9.3 Event-Driven Programming

An event occurs when something happens. Following are some examples of events:

1. Clicking the mouse button.
2. Typing a character from keyboard.
3. Moving the mouse.
4. Deleting or changing a value.

Event-driven programming is a programming technique in which statements are executed in response to an event. For example:

1. When the user right clicks on a window, a popup menu appears.
2. When the user presses ALT+F4, the current window is closed.
3. When the user clicks on Print button, the file is printed etc.

In event-driven programming, the programmer writes code that is executed when a particular event occurs. Normally, event-driven programming technique is used with visual programming environments.

Freebooks.pk

1.9.4 Visual Programming

Visual programming is used to provide a user-friendly interface for interacting with the program. Different objects like windows, menus, buttons and list boxes are used to develop the **graphical user interface**.

A visual programming environment provides all objects as built-in components to develop a graphical user interface. The programmer does not have to create these objects. When the programmer needs a specific object such as a button, he selects it from the toolbox. These components can be moved, resized and renamed as required.

In visual programming environment, the programmer designs user interface visually instead of writing code. The visual programming environment also provides the facility to attach code with each component. The attached code is executed when the user interacts with objects. For example, a window is closed when the user clicks on **Close** button. Some popular visual programming environments are Visual Basic, Developer 2000 and Power Builder.

Advantages / Features of Visual Programming

Some advantages of visual programming are as follows:

1. Visual programming is easy to use and learn.
2. A programmer does not have to write code to create components.
3. The visual components can be moved, resized and deleted easily.
4. The programmer can create the user interface visually by using mouse.
5. The components provided by the visual programming environment contain some built-in code.
6. The visual components can react to different events. For example, a button can display a message when the user clicks it.

Disadvantages of Visual Programming

Some disadvantages of visual programming are as follows:

1. Visual programming environments are heavy and require more memory.
2. Visual programming requires a computer with higher capacity of hard disk.
3. Visual programs need a computer with faster processor for execution.
4. Visual programming can be implemented only on graphical operating systems like Microsoft Windows.

Exercise Questions

Q.1. Define problem solving.

Problem solving is a process of identifying a problem and finding the best solution for it. It is a skill that can be developed by following a well organized approach. Every problem is different in its nature. Different solutions of a problem are identified and the best solution is selected.

Q.2. Discuss the advantages of computer program?

There are many advantages of computer program. A program can solve many problems by giving instructions to the computer. It can be used to perform a task repeatedly and quickly. A program can process a large amount of data easily. It can display the results in different styles. The processing of a program is more efficient and less time consuming. Different types of programs are used in different fields to perform certain tasks.

Freebooks.pk

Q.3. Define programming.

Programming is a process of solving a problem with the help of computer system. It prepares different instructions for computer.

Q.4. Who is a programmer?

A person who develops a program is called programmer. The programmer develops programs to instruct the computer how to process data to convert into information. Programmer uses programming languages or tools to write programs.

Q.5. Can a person be a good programmer if he is expert in problem solving?

Programming is a problem solving activity performed with the help of computer system. A programmer writes instructions in programming language and computer executes these instructions to solve a problem. A person can be a good programmer if he is expert in problem solving.

Q.6. Why is it important to identify a problem?

Problem identification is an important step to find the correct solution of a problem. A problem cannot be solved if it is not identified correctly. In this stage, the problem is observed carefully and major areas of concern are identified. The irrelevant information is removed in this step.

Q.7. Why is it important to specify requirements to solve a problem?

The requirement specification is a very important step to solve a problem. The user requirements are specified clearly in this stage so that a proper solution can be suggested. A requirements document is prepared that describes the expected features of the system. It contains the details of the system that will be designed and used.

Q.8. Write the disadvantages of not specifying the requirements to solve a problem?

It is not possible to find the correct solution of a problem if the requirements are not specified clearly. Many users cannot explain the exact requirements of the software. The confusing requirements in the mind of user may lead to a wrong solution. So it is very important to specify the exact requirement clearly to find correct solution.

Q.9. Define the process of analyzing a problem.

Analyzing problem is a process of understanding the problem. The analysis of problem is very important step to solve any problem. A problem cannot be solved without understanding it correctly.

Q.10. Describe the important of analyzing a problem.

The process of analyzing a prblem is very important. In this step, a problem is divided into smaller parts. The smaller parts are solved independently. The process of problem solving becomes simpler and easier with this method.

Q.11. Define top down design.

The technique of dividing a problem in small parts is called top down design. It is also called divided and conquer rule.

Q.12. Why is an algorithm designed to solve a problem?

The designing of algorithm requires a list of steps required to solve a problem. It is then verified whether the algorithm solves the problems according to the requirement or not. The process of problem solving becomes simpler and easier with the help of algorithm.

Q.13. Why is flowchart designed to solve a problem?

A flowchart is designed after the designing of algorithm. Flowchart is a graphical representation of an algorithm. It helps to understand the algorithm easily. It is also helpful in understanding the flow of control and data in algorithm.

Q.14. Define desk checking.

Desk checking is used to verify the accuracy of an algorithm. In this process, each step of algorithm is performed on the paper. Different values are given as input and the output is checked. The algorithm can be modified if there is any error occurs during desk checking.

Q.15. Explain the process of writing program in solving a problem.

The process of writing a program is a very important step in solving a problem. In this step, an algorithm is converted into a program. A program is a set of instructions written in a programming

Freebooks.pk

language. The program consists of different steps given in the algorithm. The problem is solved when the computer executes the program.

Q.16. Why is it important to know the syntax of programming language to solve a problem on computer?

It is very important to know the syntax of programming language to solve a problem on computer. The set of rules of a programming language is known as syntax. The programmer selects a programming language whose syntax is known to him. The programmer cannot write a program if he does not know the syntax. The problem cannot be solved on computer.

Q.17. What is algorithm? What are important properties of an algorithm?

An algorithm is a step-by-step procedure to solve a problem. Some properties of algorithm are:

1. The given problem should be broken down into simple and meaningful steps.
2. The steps should be numbered sequentially.
3. The steps should be descriptive and written in simple English.

Q.18. Is it necessary for an algorithm to solve problem in finite number of steps? If yes, why?

Yes, it is necessary for an algorithm to solve problem in finite number of steps. The reason is that the solution of any problem may not consist of infinite steps. The algorithm has a clear start and end. The solution is found when all steps of alogorithm are performed in a sequence from start to end.

Q.19. State the purpose of refining an algorithm.

The process of refining an algorithm means that the algorithm is made simpler and easier to understand.

Q.20. Which language is used to write algorithms?

Algorithms are written in a language, which is similar to simple English. This language is known as pseudo language. Pseudo code is used to specify program logic in an English like manner that is independent of any particular programming language.

Q.21. What are advantages of algorithms?

Some advantages of using pseudo code to specify an algorithm are as follows:

Reduced Complexity – Writing algorithm and program separately simplifies the overall task by dividing it into two simpler tasks. While writing the algorithm, we can focus on solving the problem instead of concentrating on a particular language.

Increased Flexibility – Algorithm is written so that the code may be written in any language. Using the algorithm, the program could be written in Visual Basic, Java or C++ etc.

Ease of Understanding – You don't have to understand a particular programming language to understand an algorithm. It is written in an English like manner.

Q.22. What is a flowchart? Why is it used?

Flowchart is a graphical representation of an algorithm. It is used to show all the steps of an algorithm in a sequence. Flowchart is used to represent an algorithm in simple graphical manner, to show the steps of an algorithm in an easy way, to understand the flow of the program and to improve the logic for solving a problem.

Q.23. List some advantages of flowchart.

Some advantages of flowchart are as follows:

- The logical of an algorithm can be described more effectively with the help of flowchart.
- The flowchart acts like a guide for program development.
- Flowchart helps in debugging process.

Q.24. Write down the limitations of flowchart.

- It is difficult to draw flowcharts for complex problems.
- The flowchart has to be redesigned if any change is required.

Q.25. Differentiate between flowchart and algorithm.

Flowchart consists of standard symbols but algorithm consists of simple English. Flowchart is more time-consuming but algorithm is less time-consuming. Flowchart is difficult to modify but algorithm is easier to modify.

Freebooks.pk

Q.26. State the purpose of testing a program?

The purpose of testing a program is to verify the accuracy of a program. The program is tested by executing it again and again. All bugs in the program are detected and removed during program testing. It ensures that the program gives desired results and the problem is solved correctly.

Q.27. Why a program should be maintained and updated?

A program should be maintained and updated continuously. It includes the upgrading of the program to meet new requirements of hardware and software. The program is modified if there is any change in user requirements. The continuous usefulness of a program depends on regular maintenance.

Q.28. Why documentation is considered vital in problem solving process?

Documentation is very vital in problem solving process. It helps user to operate and understand program easily. It also describes how input data is identified in program and how output is formatted. It provides instructions to install program and requirement of operating system and hardware.

Q.29. What is the use of programming languages?

A set of words, symbols and codes used to write programs is called **program language**. Different programming languages are available for writing different types of programs. Some languages are specially used for writing business programs, other are used for writing scientific programs etc.

Q.30. List out major types of programming languages.

There are two types of computer programming languages:

Low Level Languages – These languages are near to computer hardware and far from human languages. Computer can understand these languages easily. Writing a program in low-level languages requires a deep knowledge of the internal structure of computer hardware.

High Level Languages – A type of language that is close to human languages is called high level language. High-level languages are easy to understand. Instructions of these languages are written in English-like words such as **input** and **print** etc.

Q.31. What is machine language?

A type of language in which instructions are written in binary form is called machine language. It is the only language that is directly understood by the computer. It is the fundamental language of the computer.

Q.32. List some advantages of machine language?

Different advantages of machine language are as follows:

Fast Execution – Instruction written in machine language run faster.

No Translator Required – Machine language program is directly understood by the computer and is not required to convert.

Q.33. List some disadvantages of machine language?

Different disadvantages of machine language are as follows:

Machine Dependent – Machine language is machine-dependent. Each type of hardware requires its own machine language.

Difficult – Machine language is difficult to understand.

Time-Consuming – Writing programs in machine language takes a lot of time

Q.34. What is assembly language?

Assembly language is a low-level language. It is one step higher than machine language. In assembly language, symbols are used instead of binary code.

Q.35. Why assembly language is called symbolic language?

Assembly language is called symbolic language because it uses symbols called **mnemonics**. For example **Sub** instruction is used to subtract two numbers.

Q.36. List advantages of assembly language over machine language.

Some advantages of assembly language are as follows:

- Easier to understand and use as compared to machine language.
- Easier to locate and correct errors.
- Program written in assembly language are easier to modify.

Freebooks.pk

Q.37. What are procedural languages?

In these languages, a program is a predefined set of instructions. Computer executes these instructions in the same sequence in which these instructions are written. Each instruction in this language tells the computer what to do and how to do.

Q.38. List out some procedural language.

Some of the most popular procedural languages are FORTRAN, BASIC, COBOL, PASCAL & C.

Q.39. What are object-oriented languages?

OOP is a technique in which programs are written on the basis of objects. An object is a collection of data and functions. Object may represent a person, thing or place in real world. In OOP, data and all possible functions on data are grouped together.

Q.40. Discuss some important features of object-oriented programming.

Objects – OOP provides the facility of programming based on objects.

Classes – Classes are designs for creating objects.

Real-world Modeling – OOP is based on real-world modeling. Objects have data and functions to represent properties and functions.

Reusability – OOP provides ways of resuing the data and code through inheritence.

Information Hiding – OOP allows the programmer to hide important data from the user.

Polymorphism – Polymorphism is an ability of an object to behave in multiple ways.

Q.41. What are database query languages?

Database is a collection of data in an organized way. Database query languages are used to retrieve, insert, update, delete or search data from databases. These are non-procedural languages. In non-procedural languages, user only needs to tell the computer "what to do" not "how to do".

Q.42. What is the use of SQL?

SQL stands for Structured Query Language. It is the most popular database query language. SQL was developed by IBM. It is a national standard by the American National Standards Institute (ANSI). SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

Q.43. What is RPG?

RPG stands for Report Program Generator. It was developed in early 1960s by IBM. It is a nonprocedural language to generate business reports. It is mostly used with IBM mid-range computers.

Q.44. Why would a programmer prefer to use high-level languages?

A progrmmer would prefer to use high-level languages for the following reasons:

Easy to Learn – These languages are easier to learn than low-level languages.

Machine Independence – The programs written in high-level languages for one type of computer system can be executed on a different type of computer system.

Easy Error Detection – It is easy to find errors in programs written in these languages.

Standardized Syntax – The syntaxe of high-level language is standardized.

Easy to Understand - High-level languages are easier to understand than low-level languages.

Shorter Programs - Programs written in high-level languages are shorter than low-level languages.

Q.45. Can you imagine some disadvantages of high-level languages?

Some disadvantages of high-level languages are as follows:

Translator Required – A program written in high-level languages cannot be run directly. A language translator is required to convert it into machine language.

Time Consuming Execution – High-level programs are first translated into machine code and then executed. So the execution of high-level programs takes more time.

Less Hardware Support – High-level languages provide less support for writing hardware level programs. These languages are normally used for writing application software.

Q.46. What is the difference between high level and low level languages?

High-level languages are easy to learn but low-level languages are difficult to learn. High-level languages provide less facility at hardware level but low-level languages provide much facility at hardware level. High-level languages are used to write application software but low-level languages are used to write syste software.

Freebooks.pk

Q.47. What is source code? Why it cannot be executed directly?

Source code is a computer program written in a high-level programming language like C, C++, Visual Basic or Java. Computer cannot understand the statements of high-level language. So the source code cannot be executed by the computer directly. It is converted into machine code and then executed.

Q.48. What is object code?

An object code is the program that is translated by a language processor. It is also called machine code. Computer understands object code directly.

Q.49. What is the purpose of language processor?

A program written in high-level or assembly language cannot be run on a computer directly. It must be converted into machine language before execution. Language processor or translator is a software that converts these programs into machine language.

Q.50. List out different types of language processors.

Different types of language processors are as follows:

Compiler – Compiler converts the instruction of high level language into machine language as a whole.

Interpreter – An interpreter is a program that converts one statement of a program at one time.

Assembler – Assembler translates the instruction of a assembly language into machine language.

Q.51. What is advantage and disadvantage of interpreter?

The advantage of interpreters over compilers is that an error is found immediately. So the programmer can correct errors during program development. The disadvantage of interpreter is that it is not very efficient. The interpreter does not produce an object program. It must convert the program each time it is executed.

Q.52. Why source code is converted into object code?

Computer cannot understand the statements of high-level language. Computer cannot execute source code directly. It is converted into object code and then executed.

Q.53. What is difference between source code and object code?

Source code is easy to understand and modify. Object code is difficult to understand and modify. Source code contains fewer statements than object code.

Q.54. What is compiling?

The process of converting source code into object code is known as compiling. A compiler converts source program into object program and saves it in a separate file.

Q.55. How object program, source program and compiler are related?

A source program is written by a programmer in a programming language such as C. This program is not in a form that a computer can understand. A compiler translates that source program into the object program that the computer can understand and execute.

Q.56. What is difference between structured and object-oriented programming?

In structured programming, data cannot be considered as independent from functions that manipulate it. Procedural programming prevents the programmer from separating data from functions

OOP is a technique in which programs are written on the basis of objects. In OOP, data and all possible functions on data are grouped together. Object oriented programs are easier to learn and modify.

Q.57. What is the difference between high level and low level languages?

High-level languages are easy to learn but low-level languages are difficult to learn. High-level languages provide less facility at hardware level but low-level languages provide much facility hardware level. High-level languages are used to write application software but low-level langu are used to write syste software.

Q.58. Name any five high-level languages.

The five high-level languages are FORTRAN, Pascal, C++, Java and Visual Basic.

Freebooks.pk

Multiple Choice

1. **Computer programs are also known as:**
   a. Software b. Signal c. Procedure d. Debugger
2. **Which of the following is a problem solving technique?**
   a. Algorithm b. Programming c. Flow chart d. All
3. **How many possible solutions are there for a problem?**
   a. One b. Two c. Three d. Multiple
4. **A person who develops a program is called:**
   a. Programmer b. Database designer c. Network Administrator d. None
5. **The step-by-step procedure to solve a problem is called:**
   a. Programming language b. Flowchart c. Program d. Algorithm
6. **A graphical representation of the steps involved in an algorithm is called:**
   a. Algorithmic diagram b. Flowchart. c. Data flow diagram d. None
7. **The diamond symbol in a flowchart indicates:**
   a. Progress. b. Condition. c. Input. d. Output.
8. **The rectangle symbol in a flow chart indicates:**
   a. Process. b. Condition. c. Input. d. Output
9. **The parallelogram symbol in a flow chart indicates:**
   a. Input. b. Output c. Both a and b d. None
10. **The process of carefully observing the working of an algorithm is called:**
    a. Desk Checking b. Compiling c. Debugging d. Coding
11. **The process of writing a program in programming language is called:**
    a. Flow chart b. Coding c. Desk Checking d. None
12. **The set of rules for writing a program in any programming language is called:**
    a. Syntax b. Bug c. Debug d. None
13. **Program up gradation refers to:**
    a. Program enhancement b. Program Identification
    c. Program development d. Program implementation
14. **Which of the following documents describe various features of the software and the way it is used?**
    a. Software requirement specification b. Problem description
    b. User manual d. Algorithm
15. **The most difficult part of problem solving is:**
    a. Design Algorithm b. Testing program
    c. Documentation d. Program execution
16. **Compiled programs typically execute faster because:**
    a. Compiled programs are read and executed a line at a time.
    b. Compiled programs are already in a machine-readable form.
    c. Compiled programs do not require any data.
    d. None of the above

The output of the compiler is called

a. The program b. Source code c. Linked code d. Object code

of the following are part of the documentation process EXCEPT:

est procedures b. Program algorithm
chine language code d. User Manual

Freebooks.pk

19. **Machine language:**
    a. is the language in which programs were first written.
    b. is the only language understood by the computer.
    c. Differs from one type of computer to another.
    d. All of above
20. **A type of language in which instructions are written in binary form is called:**
    a. Machine language b. Assembly language c. High level language d. None
21. **Which of the following is the native language of the computer?**
    a. C language b. Pascal c. Machine language d. DOS
22. **Writing programs in machine language is:**
    a. Complex b. Simple c. Time-consuming d. Both a and c
23. **Writing programs in high-level languages is _____ than machine language.**
    a. Less time consuming b. More time consuming
    c. Less simple d. Both a & c
24. **Which of the following is not a high level language?**
    a. Assembly language b. Pascal c. BASIC d. FORTRAN
25. **The lowest level of programming language is:**
    a. Java b. Assembly Language c. Pascal d. C++
26. **An assembly language uses:**
    a. English words b. Mnemonic codes c. Os and 1s d. Binary digit
27. **What is the name of program that translates assembly source code to machine code?**
    a. Complier b. Interpreter c. Assembler d. Debugger
28. **Which one of the following translates source code to object code as a whole?**
    a. Interpreter b. Compiler c. Assembler d. Debugger
29. **Choose the program that translates and executes one line of the source code at a time.**
    a. Compiler b. Assembler c. Interpreter d. None
30. **Which of the following language is ideal to write business application?**
    a. C++. b. FORTRAN. c. Assembly. d. COBOL.
31. **Which language was originally created for teaching the students how to program?**
    a. C++ b. COBOL c. Assembly d. BASIC
32. **Which of the following languages is an object-oriented language?**
    a. COBOL b. Assembly c. C++ d. BASIC
33. **Java is a product of :**
    a. Microsoft b. IBM c. Sun System d. Hewlett
34. **A programming language used for mathematical and engineering applications was:**
    a. BASIC b. FORTRAN c. Assembly d. COBOL

Answers

1. a	2. d	3. d	4. a	5. d	6. b
7. b	8. a	9. c	10. a	11. b	12. a
13. a	14. b	15. a	16. c	17. d	18. c
19. d	20. a	21. c	22. d	23. d	24. a
25. b	26. b	27. c	28. b	29. c	30. d
31. d	32. c	33. c	34. b		

Freebooks.pk

Fill in the Blanks

1. The set of instructions given to the computer to solve a problem is called _____.
2. The set of rules for writing programs in programming language is known as_____ of language.
3. Flow chart is a_____ representation of algorithm.
4. An algorithm solves a problem in _____ number of steps.
5. Two main categories of computer programming languages are _____ and _____.
6. Low-level languages are divided into _____ broad categories.
7. Machine language and assembly language are examples of _____ language.
8. _____ is the native language of the computer.
9. Languages that are quite close to computer hardware are called _____.
10. A code written in binary form indicates _____language.
11. _____ language is hardware dependent.
12. Assembly language uses_____ to represent each instruction type.
13. _____ is used to translate an assembly language programs to machine language.
14. The only fundamental language that does not use any translator is called _____.
15. The programming languages whose instructions resemble English language are called _____.
16. Every high level language defines a set of rules for writing programs called_____ of the language.
17. The code written in high-Level language is converted into_____ language before execution.
18. C++, java, Pascal and FORTRAN are examples of _____languages.
19. _____ language is ideal for writing business applications.
20. _____ language has very powerful mathematical capabilities.
21. A(n) _____ is a step-by-step logical sequence for a problem's solution.
22. The programming language FORTRAN stands for _____.
23. The programming language BASIC stands for _____.
24. OOP stands for _____.

Answers

1. Program	2. Syntax	3. Graphical
4. Finite	5. low level, high level	6. Two
7. Low-level	8. Machine language	9. Machine language
10. Machine	11. Machine	12. Mnemonic codes
13. Assembler	14. Machine language	15. High-level languages
16. Syntax	17. Machine	18. High-level
19. COBOL	20. FORTRAN	21. Algorithm
22. FORmula TRANslation	23. Beginners' All-Purpose Symbolic Instruction Code	
24. Object-oriented programming		

Freebooks.pk

True / False

1. Machine code is a complex code written in octal form.
2. Hexadecimal code is the basic language of computer.
3. Programs written in high-level languages are machine independent.
4. Assembly language is a middle level language.
5. Mnemonics are symbols used by Assembly language instead of complex binary code.
6. FORTRAN is also called symbolic language.
7. LISP & PERL are two examples of symbolic language.
8. A computer program written in high-level language is called Source Code.
9. Computer does not understand object code directly.
10. A program translated only by interpreter is called object code.
11. Software that converts high-level programs into machine language is called language processor.
12. Compiler is a program that converts programs line by line in machine code.
13. Compiler converts object code into machine code, which is called source code.
14. Interpreter does not produce an object program.
15. Assembler translates instructions of C language into machine language.
16. Compiler displays syntax errors after compiling each line.
17. C language can run without the presence of a translator.
18. High-level languages provide less support for writing hardware level programs.
19. Programs in high-level languages are much faster in execution than low-level languages.
20. High-level language is less cryptic than machine language
21. All programs must be brought into main memory before they can be executed.
22. A bit is a binary digit, o or 1.
23. A byte is a sequence of eight bits.
24. An object consists of data and operations on those data.
25. Software is a programs run by the computer.

Answers

1. F	2. F	3. T	4. F	5. T
6. F	7. F	8. T	9. F	10. F
11. T	12. F	13. F	14. T	15. F
16. F	17. F	18. T	19. F	20. T
21. T	22. T	23. T	24. T	25. T

Freebooks.pk

CHAPTER 2

INTRODUCTION TO C++

Chapter Overview

2.1 History of C++
- 2.1.1 Features of C++

2.2 Basic Structure of C++ Program
- 2.2.1 Preprocessor Directive
- 2.2.2 Header Files
- 2.2.3 main( ) Function
- 2.2.4 C++ Statements
- 2.2.5 Token
- 2.2.6 White Spaces

2.3 Turbo C++
- 2.3.1 Installing Turbo C++
- 2.3.2 Initializing Turbo C++ IDE
- 2.3.3 Setting up Directories
- 2.3.4 Creating and Editing a C++ Program
- 2.3.5 Saving a C++ Program
- 2.3.6 Compiling a C++ Program
- 2.3.7 Linking a C++ Program
- 2.3.8 Executing a C++ Program

2.4 Debugging in Turbo C++
- 2.4.1 Types of Errors
- 2.4.2 Debugging Features of Turbo C++

Exercise Questions
Multiple Choice
Fill in the Blanks
True/False

Freebooks.pk

2.1 History of C++

Many new programming languages appeared during the 1960s. The computers at that time were still in early stage of development. The language **ALGOL 60** was developed as an alternative to **FORTRAN**. The language **CPL (Combined Programming language)** was developed in 1963. It was more specific for concrete programming tasks of that time than ALGOL or FORTRAN. But it was very difficult to learn. **Martin Richards** developed **BCPL (Basic Combined Programming Language)** in 1967 that was a simplification of CPL.

Ken Thompson created the **B** language in 1970. **C** language was derived from B language. The **B** language provided the basis for the development of **C**. C language was originally designed to write system program under **UNIX** operating system. The power and flexibility of **C** language made it popular in industry for a wide range of applications.

The earlier version of C was known as **K&R** (Kernighan and Ritchie) C. The **American National Standard Institute (ANSI)** developed a standard version of the language. The standard version is known as **ANSI C**. This new version provided many features that were not available in the older version.

Bjarne Stroustrup from **Bell labs** started the development of C++ language in 1980. The first commercial release of the language appeared in October 1985. It was originally named "C with Classes". It was an enhancement to C language. It allows the use of a programming technique known as **object oriented programming (OOP)**.

C++ language was refined during 1980s and it became a unique language. It was very much compatibile with the code of C and provided the most important characteristics of C. **ANSI** committee **X3J16** began the development of a specific standard for C++ from 1990. In mid 1998, **ANSI/ISO C++** language standards were approved. This standard is used by most of the compilers used today to compile programs. The language became very popular and now it is the preferred language to develop professional applications on all platforms.

2.1.1 Features of C++

Some important features of C++ are as follows:

1. Convenient Language

C++ is very convenient language. It provides many facilities in easier way that are difficult to use in low-level languages. Programmers can write complex programs more easily as compared to low-level languages.

2. Well-Structured Language

C++ is a well-structured language. Its syntax is very easy to understand. The programs written in C++ language are easy to maintain and modify.

3. Case Sensitivity

C++ is case sensitive language. It means that it can differentiate uppercase and lowercase words. All keywords are written in lowercase. This feature makes it easier to maintain the source code.

4. Machine Independence

C++ language provides machine independence. It means that the programs written in C++ language can be executed on different types of computers. For example, a program written in C++ can be executed on Intel processors and Motorola processors with a little modification. It is preferable to write program in C++ rather than machine language.

Freebooks.pk

5. Object Oriented

C++ is an object-oriented language. In object-oriented technique, the programs are written on the basis of objects. An object is a collection of data and functions. Object may represent a person, thing or place in real world. In OOP, data and all possible functions on data are grouped together. Object oriented programs are easier to learn and modify.

6. Modular Programming

C++ language provides the facility of modular programming. It means that a program can be divided into small modules. These modules can be developed and compiled independently and then linked together.

7. Standard Libraries

The standard C++ library is a set of functions, constants, classes and objects to extend C++ language. It provides the basic functionality to interact with the operating system. These libraries can be reused by any programmer for writing programs more easily in less time.

8. Hardware Control

C++ language provides close control on hardware. The programmer can write efficient programs to control hardware components of computer system.

9. Brevity

C++ is a small language. It has a small number of keywords and programming controls. But still it is every powerful for developing different types of programs. The code written in C++ is very short as compared with other languages.

10. Speed

C++ compilers generate very fast code. The resulting code from a C++ compilation is very efficient. This code executes very efficiently. So the programs take less time to execute.

11. Popular Language

C++ is a very widely used programming language. There are many tools available for C++ programming and a broad base of programmers contributing to C++ community.

12. C Compatibility

C++ is backward compatible with C language. Any code written in C can easily be included in a C++ program without hardly making any change.

2.2 Basic Structure of C++ Program

The format of writing program in C++ is called its structure. The basic structure of C++ program is very flexible. It increases the power of language. It consists of the following parts:

- Preprocessor directive
- Main() function
- Program body (C++ statements)

2.2.1 Preprocessor Directive

Preprocessor directive is an instruction given to the compiler before the execution of actual program. Preprocessor directive is also known as **compiler directive**. The preprocessor directives are processed by a program known as **preprocessor**. It is part of C++ compiler. It modifies C++ source program before compilation. The semicolon is not used at the end of preprocessor directives.

Freebooks.pk

The preprocessor directives start with hash symbol #. These directives are written at the start of program. The following preprocessor directive is used in C++ to include header files in the program:

Include Preprocessor

Include preprocessor directive is used to include header files in the program. The syntax of using this directive is as follows:

```
#include <iostream.h>
```

The above statement tells the compiler to include the file **iostream.h** in source program before compiling it.

2.2.2 Header Files

Header files are collection of standard library functions to perform different tasks. There are many header files for different purposes. Each header files contains different types of predefined functions. Many header files can be included in one program. The header file must be included in the program before calling any of its functions in the program.

The extension of a header file is **.h**. The **include** preprocessor directive is used to include header files in programs. These files are provided by C++ compiler system.

The header files are normally stored in **INCLUDE** subdirectory. The name of header file is written in angle brackets.

Syntax

The syntax of using header files is as follows:

```
#include <header_file_name>
```

The name of header file can also be used in double quotes as follows:

```
#include "header_file_name"
```

Example

```
#include "iostream.h"
```

The word "iostream" stands for **input/output stream**. This header file contains the definitions of built-in input and output functions and objects.

A header file **math.h** is used in programs to use predefined mathematical functions. The following statement is used to include this file in program:

```
#include "math.h"
```

2.2.3 main( ) Function

The **main()** function is the starting point of a C++ program. When the program is run, the control enters **main()** function and starts executing its statements.

Each program must contain **main()** function. If a program does not contain main function, it can be compiled but cannot be executed. Any number of statements can be written in the body of the **main()** function. The syntax of **main()** function is as follows:

```
void main( )
{
        body of main function
}
```

Freebooks.pk

2.2.4 C++ Statements

The statements of the program are written in curly braces. The curly brace { is called **opening brace** and } is called **closing brace**. The braces are also known as **delimiters**. These statements are collectively known as the **body** of a program. Each statement in C++ language is terminated by semicolon.

A statement in C++ language is an instruction for the computer to perform a task. Computer performs these instructions one by one in the same sequence in which these instructions are written.

Example

The following example explains the basic structure of C++ program:

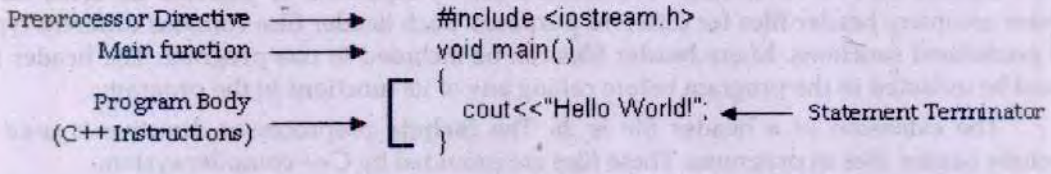


Figure 2.1: Basic Structure of C++ program

In the above example,

- The first line is preprocessor directive to include a header file **iostream.h**.
- The second line is **main** function where the execution of program starts.

Using 'cout'

The **cout** object is used to print a message on the screen. The message may consist of strings and values. A **string** is a set of characters. The **cout** object is used with the insertion operator <<. Anything written after the insertion operator is displayed on the screen.

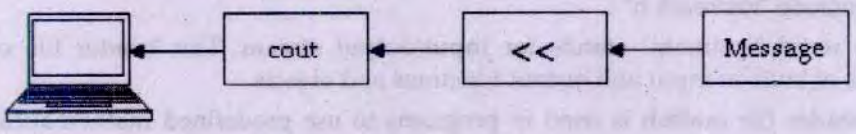


Figure 2.2: Working of 'cout' object

2.2.5 Token

A **token** is a language element that is used to form a statement. C++ statement may consist of different tokens. Different types of tokens are as follows:

- **Keywords:** Keyword is a word in C language that has a predefined meaning and purpose. The meaning and purpose of a keyword is defined by the developer of the language.
  Example: for double if const
- **Identifiers:** Identifier is the name of a variable or function etc. It is also called **user-defined word**.
  Example: Student_age item20 counter number_of_character
- **Constants:** Constant is a quantity that cannot be changed during execution of a program.
  Example: 213 520 420.7 -220.7 'j' '%'

Freebooks.pk

- **String literals:** A collection of characters written in double quotations is called string or string literal.
  Example: "This is a string constant" "120" "99-Mall Road, Lahore"
- **Operators:** Operator is a symbol that performs some operation. It acts on different operands.
  Example: + - * / %
- **Punctuators:** Punctuator is a symbol that is used to separate two tokens.
  Example: [ ] ( ) { } , ; . :

2.2.6 White Spaces

The white spaces are used in programs to increase readability. Different types of white spaces include space, tab and carriage return etc. C++ compiler ignores the white spaces. All statements of a program can be written on one line but it will be difficult to read. The white spaces makes the source program more readable. A single statement can be written on two or more lines or order to increase readability but it will remain same for the compiler.

2.3 Turbo C++

Turbo C++ is an **integrated development environment (IDE)** to write C++ programs. It is the implementation of **Borland International**. It is used to create, edit and save C++ programs. It also provides a powerful debugger. The debugger helps the users in detecting and removing errors in programs.

2.3.1 Installing Turbo C++

There are different types of installers available for Turbo C++ such as Turbo C++ 3.0. The installation of Turbo C++ is very simple. The user can follow a simple wizard for installing Turbo C++. The wizard prompts the user to specify the root folder. The default folder is **TC**. Some CDs also provide TC folder that requires no installation process. The user can simply copy the folder and paste it on hard disk. The installation of Turbo C++ creates TC folder on hard disk. This folder contains different subfolders as follows:

- **INCLUDE:** It contains the header files.
- **LIB:** It contains the library files.
- **BIN:** It contains the source files and executable files.

2.3.2 Initializing Turbo C++ IDE

The following procedure is used to initialize Turbo C++ IDE:

1. Go to BIN folder in TC folder.
2. Double-click on **TC.EXE** file. The Turbo C++ start and its main window will appear.

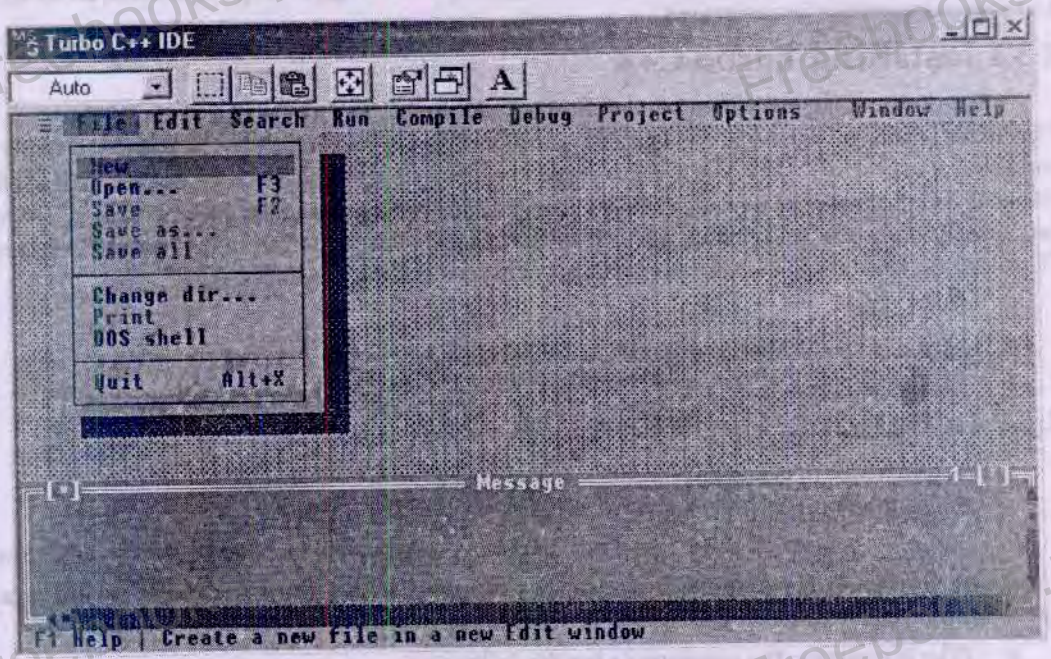
The words "Turbo C++ IDE" appear at the top of the window. It stands for **integrated development environment**. It displays the menu bar that includes the following menus:

- **File:** It contains the commands for the following:
  - Creating new files
  - Opening existing files
  - Printing source code
  - Exiting Turbo C++ IDE etc.
- **Edit:** It contains the command for the following:
  - Copying the code
  - Pasting the code

Freebooks.pk

- Undo and Redo
- **Search:** It contains the commands for the following:
  - Find
  - Replace
  - Go to a specific line number etc.
- **Run:** It contains the commands for running C++ programs.
- **Compile:** It contains the commands for compiling C++ programs.
- **Debug:** It contains the commands for debugging C++ programs.
- **Project:** It contains the commands for working with C++ projects.
- **Options:** It contains the commands for setting directories and configuring IDE environment.
- **Windows:** It contains the commands for managing different windows in Turbo C++ IDE.
- **Help:** It contains the commands for getting help about Turbo C++ IDE. The user can press **F1** to display help.

The menu bar can be activated by pressing **F10**. The user can then select any menu by pressing the arrow keys to right or left side. A menu can also be activated with a combination of **Alt** key and first letter of any menu. For example, the user can press **Alt+F** to activate File menu. A menu will appear as follows:



The **File** menu is activated in the above figure. The user can select different options in the menu by moving to any option by arrow keys.

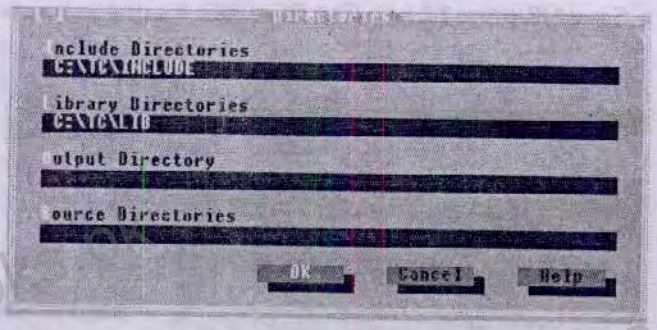
2.3.3 Setting up Directories

By default, **Turbo C++** stores object files and executable files in **BIN** subdirectory of **TC** directory. It is not the right place for storing these files. These files should be stored in the same directory in which source files are stored. The user can change the setting for output and source directories.

The following procedure is used to set the output and source directories:

1. Select **Options > Directories**. A window will appear with four fields.

- **Include Directories:** It indicates the location of standard header files included in C++ programs. It should already be set to ***drive*:\TC\INCLUDE**. The *drive* can be any drive on computer.
- **Library Directories**: It indicates the location of library files. It should already be set to ***drive*:\TC\LIB**. The *drive* can be any drive on the computer.
- **Output Directories:** It indicates the location where the compiler stores object files and linker stores executable files.
- **Source Directories:** It indicates the location of source code of the libraries that do not belong to the open project.

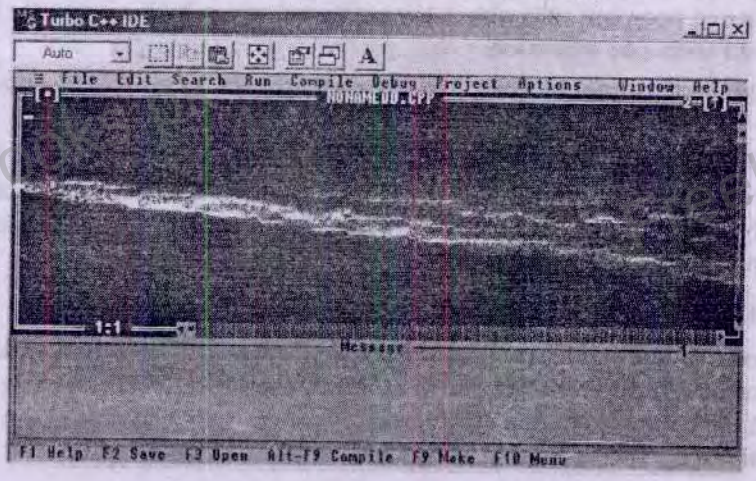


2. Enter the path for output directory like "C:\MyPrograms" in **Output Directory** field.
3. Enter the path for Source directory like "C:\MyPrograms" in **Output Directory** field.
4. Click **OK**. The new setting will be applied.

The above setting indicates that Turbo C++ in installed on C drive and the name of the folder is TC. These setting will change if the user has copied or installed C++ on different drive or with different name.

2.3.4 Creating and Editing a C++ Program

The process of writing C++ program is known as **editing**. This process includes writing, modifying and deleting program statements. The part of Turbo C++ IDE that is used to write C++ programs is called **edit window**. A new edit window can be opened by selecting **File > New** option from menu bar. The edit window appears as follows:



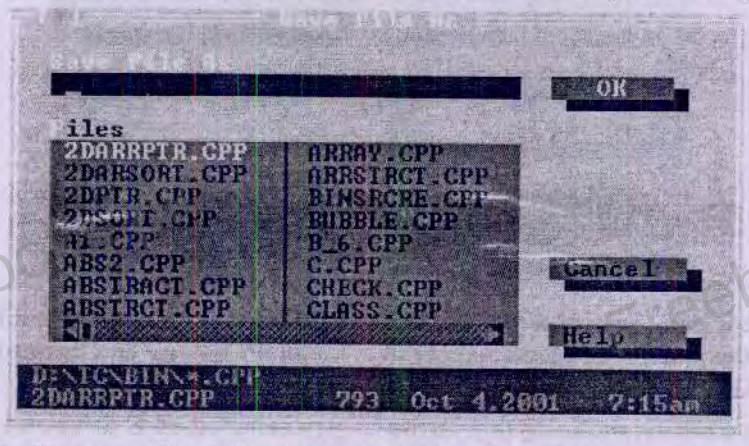
The edit window has a double-lined border. The cursor blinks in the window. The position of the cursor indicates the starting point to write a program. The user can expand the window by clicking the arrow in the upper right corner. The user can also select **Window > Zoom** to expand the window. The vertical and horizontal scrollbars are used to navigate through the program.

2.3.5 Saving a C++ Program

The process of storing the program on disk is known as **saving**. A program should be saved on disk to be used repeatedly. C++ programs are saved with **.cpp** extension. For example, a program can be saved like **program.cpp**.

The following procedure is used to save a C++ program:

1. Select **File > Save** OR press **F2** key. The **Save File As** dialog box will appear.



2. Enter the file name. The default name **NONAME00.CPP** appears in the dialog box.
3. Enter the path to save file. The default location to save programs is **BIN** directory.
4. Click **OK**. The program will be saved at specified location with specified name.

2.3.6 Compiling a C++ Program

The process of converting source program into object program is known as **compiling**. The program saved with **.cpp** extension contains the statement of C++ language. It is known as **source program**. The source program cannot be executed by computer directly. A compiler converts the source program into object program and saves it in a separate file. The object program is saved with **.obj** extension.

The source code cannot be compiled if it contains syntax error. The compiler generates error message to describe the cause of error. All errors must be removed to successfully compile a source program. The following procedure is used to compile a C++ program:

- Select **Compile > Compile** OR press **ALT+F9** key. The program will be translated into object program if it contains no error. The compiler will generate error message if the program contains any error.

2.3.7 Linking a C++ Program

The process of linking library files with object program is known as **linking**. These files are used to accomplish different tasks such as input/output etc. A **library file** must be linked with the object file before execution of program. A program that combines the object program with additional library files is known as **linker**. It is part of C++ compiler.

Freebooks.pk

The linker generates error message if the library file does not exist. A new file is created with **.exe** extension if the process of linking is successful. This file is known as **executable file** and can be executed by the computer directly. The linker can be invoked in Turbo C++ by selecting **Compiler > Link** from the menu bar.

2.3.8 Executing a C++ Program

The process of running an executable file is known as **executing**. The C++ program can be executed after compiling and linking. The program must be loaded into the memory to execute it. A program that places an executable file in the memory is known as **loader**. The program can be loaded in the memory by selecting **Run > Run** from menu bar or pressing **CTRL+F9**.

The screen flickers for some time when a program is executed. The output screen displays the output of the program and disappears. The user can display the output screen by selecting **Window > User Screen** or pressing **ALT+F5**.

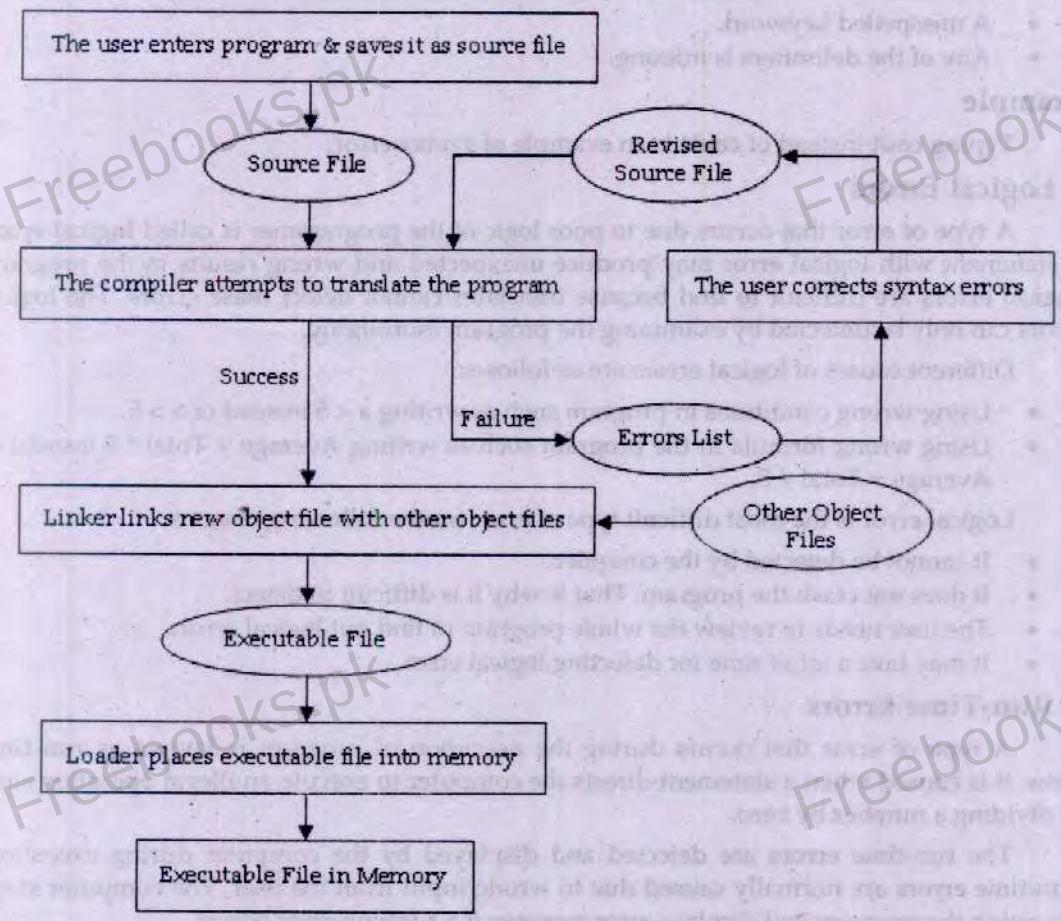


Figure 2.3: Different steps from writing to executing C++ programs

Freebooks.pk

2.4 Debugging in Turbo C++

An error in a computer program is known as **bug**. The process of finding and removing bugs is known as **debugging**.

2.4.1 Types of Errors

Different types of errors can occur in C++ programs. These errors are syntax errors, logical errors and run-time errors.

1. Syntax Errors

A collection of rules for writing programs in a programming language is known as **syntax**. All program statements are written according to these rules. Syntax error is a type of error that occurs when an invalid statement is written in program. The compiler detects syntax errors and display error message to describe the cause of error. A program containing syntax errors cannot be compiled successfully. There can be many causes of syntax errors. Some important causes are as follows:

- The statement terminator is missing at the end of statement.
- A misspelled keyword.
- Any of the delimiters is missing.

Example

Typing **cout** instead of **coutt** is an example of syntax error.

2. Logical Errors

A type of error that occurs due to poor logic of the programmer is called **logical error**. A statement with logical error may produce unexpected and wrong results in the program. Logical errors are difficult to find because translator cannot detect these errors. The logical errors can only be detected by examining the program thoroughly.

Different causes of logical errors are as follows:

- Using wrong conditions in program such as writing a < 5 instead of a > 5.
- Using wrong formula in the program such as writing Average = Total * 5 instead of Average = Total / 5.

Logical error is the most difficult type of error for the following reasons:

- It cannot be detected by the compiler.
- It does not crash the program. That is why it is difficult to detect.
- The user needs to review the whole program to find out logical error.
- It may take a lot of time for detecting logical error.

3. Run-Time Errors

A type of error that occurs during the execution of program is known as **run-time error**. It is caused when a statement directs the computer to execute an illegal operation such as dividing a number by zero.

The run-time errors are detected and displayed by the computer during execution. Run-time errors are normally caused due to wrong input from the user. The computer stops executing the program and displays error message if a runtime error occurs.

Example

The user may ask the program to open a file that does not exist. Similarly, the user may enter wrong type of data etc.

Freebooks.pk

2.4.2 Debugging Features of Turbo C++

Turbo C++ provides many useful debugging features. The debugging features of Turbo C++ are available in **Debug** menu. Some important debugging features of Turbo C++ are as follows:

1. Single Stepping

A program may not execute even if it has been compiled successfully. The debugger provides the facility to find errors by executing one line of a program at one time. It enables the programmer to detect the exact place of error.

The following procedure is used to execute one step at a time with single stepping:

- Select **Run > Trace Into** OR press **F7**.

2. Watches

The **watch** or **watch expression** is used to check the value of a variable as the program executes. It indicates how the values of variables change during program execution. It is normally used in combination with single stepping.

The following procedure is sued to use watch or watch expression:

- Place the cursor on the variable whose value is to be checked.
- Select **Debug > Watches**. A submenu will appear.
- Select **Add Watch** from the submenu. OR press **CTRL+F7**. A dialog box will appear. The selected variable will appear in **Watch expression** field.
- Click **OK** OR press **Enter**. A new window will appear indicating that the selected symbol is undefined.
- Select **Run > Trace Into** OR press **F7** to execute single stepping. The value of the selected variable will appear in **Watch** window.

3. Breakpoints

A breakpoint is used to mark a part of program where program execution will stop. The program executes all statements up to the breakpoint and then stops. The user can check the values of a variable at this point by using **Watch** window by single stepping remaining part of the program.

The following procedure is used to insert a breakpoint in the program:

- Place the cursor on the line on which the breakpoint is to be inserted.
- Select **Debug > Toggle** breakpoint OR press **CTRL+F8**. The breakpoint will be inserted and the line will be highlighted.

4. Evaluate/Modify Window

The **evaluate/modify** window is used to change the value of variable during program execution. It can be useful if the user is single stepping the program and wants to change the value of a certain variable.

The following procedure is used to use evaluate/modify window:

- Select **Debug > Evaluate/Modify**. A new window will appear with three fields.
- Enter the name of the variable whose value is to be modified in **Expression** field.
- Enter the new value for the variable in **New Value** field. The value of the third field **Result** will also change automatically.

Freebooks.pk

Exercise Questions

Freebooks.pk

Q.1. What is the use of Turbo C++?

Turbo C++ is an integrated development environment to write C++ programs. It is the implementation of Borland International. It is used to create, edit and save C++ programs. It also provides a powerful debugger. The debugger helps users to detect and remove errors in programs.

Q.2. What do you know about White Spaces?

The white spaces are used in programs to increase readability. Different types of white spaces include space, tab and carriage return etc. C++ compiler ignores the white spaces. All statements of a program can be written on one line but it will be difficult to read. The white spaces make the source program more readable.

Q.3. What is the process of linking?

The process of linking library files with object program is known as **linking**. These files are used to accomplish different tasks such as input/output etc. A **library file** must be linked with the object file before execution of program.

Q.4. What do you know about linker?

A program that combines the object program with additional library files is known as linker. It is a part of compiler. It combines object program and library files and saves the final machine language program as executable file. The extension of executable files is **.exe**.

Q.5. What is preprocessor directive?

The preprocessor directives are commands that give instructions to C++ preprocessor. The preprocessor is a program that modifies C++ source program before compilation. It directives start with hash symbol #. These directives are written at the start of program.

Q.6. Write an example of preprocessor directive.

#include <iostream.h>

Q.7. What is the purpose of the statement #include <iostream.h> in C++ program?

The # sign indicates that this is an instruction for compiler. iostream.h stands for input/output stream. The word 'include' informs C++ compiler to include the file iostream.h in the user's program.

Q.8. What is the use of "include" preprocessor?

The **"include"** preprocessor directive enables a program to access a library. It allows the preprocessor to insert definitions from a standard header file into C++ program before compilation. It is used to include header files in the program.

Q.9. What is the purpose of header files?

Header files are collection of standard library functions to perform different tasks. Each header files contains different types of predefined functions. Many header files can be included in one program. The header file must be included in the program before calling any of its functions in the program. The extension of a header file is **.h**. The **include** preprocessor directive is used to include header files in programs. These files are provided by C++ compiler system.

Q.10. What is main function used in C++ programs?

The **main()** function is the place where the execution of a C++ program starts. When the program is executed, the control enters **main()** function and starts executing its statements. Each program must contain **main()** function.

Q.11. What do you know about C++ statements?

A statement in C++ language is an instruction for the computer to perform a task. The statements are written in curly brackets. Each statement in C++ is terminated with semicolon. The compiler generates an error if any statement is not terminated by semi colon.

Q.12. What do you mean by delimiters?

The statements of the program are written in curly braces. The curly brace { is called opening brace and } is called closing brace. The braces are also known as **delimiters**. These statements are collectively known as the **body** of a program.

Q.13. Why are comments used in C++ program?

Comments are used to increase the readability of the program. Comments are notes about different lines of code that explain the purpose of the code. The user can insert information notes in the code. It helps in debugging and modifying the program later.

Q.14. Where comments can be added in programs?

Comments can be added anywhere in the code. Comments can be added in programs in two ways. The user can write comments on single line or on multiple lines.

Q.15. How are comments added on single line?

Comments on single line are added by using double slash "//". Any thing written on the right side of double slash is considered as comments and is ignored during execution.

Q.16. How are comments added on multiple lines?

Comments on multiple lines are added by using "/*" and "*/" symbols. Any thing written between two symbols is considered as comments and is ignored during execution.

Q.17. Why comments do not affect the size of program?

The compiler ignores comments and does not include them in the executable program. That is why the comments do not affect the size of executable program.

Q.18. Define a bug.

An error in a computer program is known as **bug**. The programmer can make different errors while writing programs. A program that contains syntax errors can be compiled. The compiler detects errors and displays errors message to describe the cause of error.

Q.19. What is a syntax error?

A collection of rules for writing programs in a programming language is known as **syntax**. All program statements are written according to these rules. Syntax error is a type of error that occurs when an invalid statement is written. The compiler detects syntax errors and display error message to describe the cause of error. A program containing syntax errors cannot be compiled successfully.

Q.20. What are the different causes of Syntax error

1. The statement terminator is missing at the end of statement.
2. A misspelled keyword
3. Any of the delimiters is missing.

Q.21. What are logical errors? When and why does it occur?

A type of error that occurs due to poor logic of programmer is known as **logical error**. A logical error occurs when the program follows a faulty algorithm. A statement with logical error may produce unexpected and wrong results. For example, the programmer may have written a wrong formula to computer an answer. The formula will be executed without any error but will produce wrong results.

Q.22. What are run-time errors?

A type of error that occurs during the execution of program is known as **run-time error**. It is caused when a statement directs the computer to execute an illegal operation such as dividing a number by zero.

Q.23. Why is logical error the most difficult error?

1. It cannot be detected by the compiler.
2. It does not crash the program. That is why it is difficult to detect.
3. The user needs to review the whole program to find out logical error.

Freebooks.pk

Q.24. Give two reasons why good style is important when writing programs.

1. Good style improves readability, understandability, and consistency.
2. Good style makes debugging and modification easier.

Q.25. What kind of file is produced when a CPP file with no syntax error is compiled?

An object file with extension obj is produced when a CPP file with no syntax error is compiled.

Q.26. Write the shortcut key to compile C++ program.

The shortcut key to compile C++ program is ALT+F9.

Q.27. Write the shortcut key to run C++ program.

The shortcut key to run C++ program is CTRL+F9.

Q.28. Write the shortcut key to view output screen in Turbo C++ IDE.

The shortcut key to view output screen in Turbo C++ IDE is ALT+F5.

Q.29. C++ is a case sensitive language. What does that mean?

Case sensitive means that it can differentiate between uppercase and lowercase words. All keywords are written in lowercase.

Q.30. Which header file is necessary to use the objects cout and cin?

The header file iostream.h is necessary to use the objects cout and cin.

Q.31. Categorize each of the following situations as a compile-time error, run-time error, or logical error.

a) Multiplying two numbers when the user wanted to add them.

Answer: a logical error

b) Dividing by zero

Answer: a run-time error

c) Forgetting a semicolon at the end of a programming statement

Answer: a compile-time error

d) Spelling a word wrong in the output

Answer: a logical error

e) Producing inaccurate results

Answer: a logical error

f) Typing a { instead f (

Answer: a compile-time error

Q.32. Identify different syntax errors in the following program:

```
include <iostream.h>
void main()
  cout>>'Hello'
}
```

Answer: 1. # sign missing before include. 2. The starting braces { is missing after main function. 3. Replace >> with << with cout object. 4. The word Hello should be enclosed in double quotation. 5. A semicolon must be used at the end of statement.

Multiple Choice

1. **Who developed C++?**

a. Von-Neumann b. Al-Khuwarizmi c. Charles Babbage d. Bjarne Stroustrup

Freebooks.pk

2. **C++ was designed to write program for:**
   a. Windows operating system b. Solaris operating system
   c. Unix operating system d. OS/2 operating system
3. **C++ is a:**
   a. High level language b. Low-level language
   c. Assembly language d. Machine language
4. **Turbo C++ can compile:**
   a. C programs only b. C and C++ programs
   c. Turbo C Programs only d. Turbo C++ programs only
5. **Debug is the process of:**
   a. Creating bugs in program b. Identifying and removing errors
   c. Identifying Errors d. Removing Errors
6. **Processor directives are command for:**
   a. Microprocessor b. Language processor c. C++ preprocessor d. Loader
7. **The expression in defined directive:**
   a. Can only be changed at the end of the program b. Can not e changed
   c. Can not be changed but can be redefined d. Cannot be assigned a value
8. **Which of the following language requires no translator to execute the program?**
   a. C b. C++ c. Machine language d. Assembly language
9. **The ".exe" file is produced by:**
   a. Linker b. Loader c. Compiler d. Interpreter
10. **void occupy how many bytes in memory?**
    a. Zero b. One c. Two d. four
11. **Which of the following key is used to save a file?**
    a. F2 b. F3 c. F5 d. F9
12. **The basic structure of C++ program consists of:**
    a. Preprocessor Directive b. main () function c. Program body d. All
13. **Which of the following is a compiler directive?**
    a. #include <iostream..h> b. Using namespace std;
    c. int main ( ) d. All of these
14. **iostream.h is part of:**
    a. Comment section b. C++ standard library c. Compiler d. main function
15. **Which of the following symbol is used to denote a pre-processor statement?**
    a. % b. $ c. # d. @
16. **Which of the following syntax is used to include header file?**
    a. #include<name of header file> b. #include "name of the header file"
    c. Both a or b d. None of these
17. **main is a:**
    a. Compiler directive b. Comment c. Function d. Expression
18. **C++ statements end with a:**
    a. Period b. Comma c. Semicolon d. Nothing
19. **For every opening brace in a C++ program, there must be a:**
    a. String literal b. Variable c. Closing brace d. None
20. **Which of the following causes the contents of another file to be inserted into a program?**
    a. Backslash b. Semicolon c. #include directive d. None
21. **Which of the following is first step for creating and executing C++ program?**
    a. Editing b. Compiling c. Linking d. Executing
22. **Which is the correct sequence of steps for creating and executing C++ program?**
    a. Editing → saving → compiling → linking → Executing
    b. Compiling → Editing → saving → executing → linking

Freebooks.pk

c. Editing → Executing → Compiling→ Linking
d. Linking → Executing → saving → Editing→ Compiling

23. The process of converting source code into object code is knows as:
a. Compiling b. Executing c. Linking d. Saving

24. The process of linking library files with object code is known as:
a. Compiler b. Executing c. Linking d. Saving

25. The processing of running an executable file is known as:
a. Debugging b. Compiling c. Executing d. Saving

26. Which file contains the program after translation into machine language?
a. Object file b. exe file c. Source file d. None

27. Which term is commonly used to refer to software or program errors:
a. Crash b. Short circuit c. Down d. Bug

28. Which of the following is NOT an example of a program bug?
a. Run-time error b. Operator error c. Syntax error d. Logic error

29. If a program does something different than thinking of programmer, it is called:
a. Syntax error. b. Logic error. c. Program error. d. None

30. The output of the compiler is called:
a. The program b. Source code c. Linked code d. Object code

31. What does the following code print on screen?
Cout<<"hello";
a. Hello b. hello c. HELLO d. Nothing

32. Syntax refers to the:
a. Saving computer program on hard disk. b. Rules used to create a computer program.
c. Both a and b. d. None of the above.

33. A syntax error occurs when:
a. The program instructions violate grammatical rules.
b. The logic of program is incorrect.
c. Both a and b
d. None of the above

34. A program's syntax errors is detected by:
a. Compiler b. Linker c. Loader d. Debugger

35. Which of the following errors are NOT detected by compiler?
a. Syntax error b. Logical error c. Both a and b d. None

36. Which of the following does not contain machine language code?
a. Source module b. Object module c. Library module d. None

37. A spelling error (e.g. typing intt instead of int) is an example of:
a. Syntax error b. Run-time error c. Logic error d. None

38. Division by zero (e.g. 4 / 0) is an example of:
a. Compile error b. Run-time error c. Logic error d. None

39. In a C++ program, two slash marks (//) indicate:
a. The beginning of a comment b. The end of the program
c. The beginning of a block of code d. None

40. Text enclosed in /* */ in a C++ program:
a. Gives instructions to the processor b. Declares memory requirements
c. Causes a syntax error d. is ignored by compiler

41. Comments are used to:
a. To help others read and understand the program b. Make the program run faster.
c. Make the program compile easier.
d. Increase the size of the executable program.

Freebooks.pk

42. **The programmer usually enters source code into a computer using:**
    a. A compiler b. Pseudocode c. A text editor d. None
43. **Which of the following is TRUE about void function?**
    a. Any value is returned b. 0 is returned
    c. 1 is returned d. No value is returned

Answers

1. d	2. d	3. b	4. b	5. c	6. a
7. c	8. a	9. b	10. a	11. b	12. d
13. a	14. b	15. c	16. c	17. c	18. c
19. c	20. c	21. a	22. a	23. a	24. c
25. c	26. a	27. d	28. b	29. b	30. d
31. b	32. b	33. a	34. a	35. b	36. a
37. a	38. b	39. a	40. d	41. a	42. c
43. d					

Fill in the Blanks

1. C was derived from an earlier programming language named_____.
2. B language was developed by _____.
3. The default name for saving C++ program in _____ folder.
4. Turbo C++ assigns a default name _____ to the file.
5. The C++ program is saved with_____ extension.
6. C++ _____ translates the source program into an object program.
7. The extension of object program is_____.
8. The shortcut key to invoke Turbo C++ compiler is _____.
9. _____ is a process in which object file produced by the compiler is linked to many other library files by the linker.
10. The_____ combines different library files to the object file and produces an executable file with .exe extension.
11. _____ is a Borland International's implementation of compiler for C++ language.
12. The process of linking generates an executable file with an extension _____.
13. A program to be executed must be in the form of _____ language.
14. _____ is a program that places executable file in memory.
15. Press _____ to see the program's output.
16. Turbo C++ places object and executable files in _____ subdirectory of _____ directory.
17. _____ are commands that give instructions to C++ preprocessor.
18. The _____is a program that modifies C++ program before compilation.
19. The _____directive gives program access to a library.
20. The _____ directive tells the compiler where to find the meanings of standard identifiers used in the program.
21. A preprocessor directive always begins with the symbol_____.
22. _____ is a preprocessor directive.
23. The extension of header file is _____.
24. Preprocessor directive is also called _____ directive.
25. Header files are provided by C++ _____.
26. The header file ____ contains information about standard input and output functions.
27. _____is the function where the execution of C++ program begins.
28. Every program must have a _____ function.
29. The definition of main function starts with a reserved word _____.

Freebooks.pk

30. The body of the main function enclosed in _____.
31. _____ indicates the beginning and end of the function body.
32. The _____ indicates the beginning of the block of code.
33. The _____ represent the end of a block of code.
34. The braces that indicate the beginning and end of function body are called _____.
35. Every statement in a C++ program must be terminated with a _____.
36. _____ function is used to display the output of the program on screen.
37. There are _____ types of programming errors.
38. The errors in the program are also known as _____.
39. The process of finding and removing these bugs is called_____.
40. _____, _____, and _____ are types of programming error.
41. A _____error occurs when the program violates any rules of C++ language.
42. The compiler detects the _____ errors.
43. A _____error occurs when the program directs computer to perform an illegal operation.
44. Dividing a number by zero is an example of _____ error.
45. _____ errors are detected and displayed by computer during program execution.
46. _____ errors occur when program follows a faulty algorithm.
47. _____ errors are very difficult to detect.
48. Program consisting of _____ errors cannot be executed.

Answers

1.	B	2.	Ken Thompson	3.	BIN
4.	NONAME00.cpp	5.	CPP	6.	Compiler
7.	.obj	8.	Alt+F9	9.	Linking
10.	Linker	11.	Turbo C++	12.	.exe
13.	Machine	14.	loader	15.	Alt+F5
16.	BIN,TC	17.	Preprocessor directives	18.	Preprocessor
19.	include	20.	include	21.	#
22.	include	23.	.h	24.	Compiler
25.	Compiler	26.	iostream.h	27.	main
28.	main	29.	void	30.	braces
31.	braces	32.	opening brace {	33.	closing }
34.	Delimiters	35.	Semicolon (;)	36.	printf
37.	Three	38.	Bugs	39.	Debugging
40.	syntax, Error, Runtime, Logic	41.	Syntax	42.	syntax
43.	runtime	44.	Runtime	45.	Runtime
46.	Logical	47.	Logical	48.	syntax

True/ False

1. The C++ programming language was developed by Dennis Ritchie in 1972.
2. C was derived from earlier programming language named B.
3. The B language was developed by Ken Thomson in 1980.
4. The shortcut key for compiling a program is Alt+F9.
5. The compiler produces the source file from an object file.
6. The linker is a program that combines the object program with additional files.
7. The shortcut key for executing C++ programs is Alt+F5.
8. In structured programming, the entire program is divided into smaller modules.
9. The value of a constant can be changed during program execution.

Freebooks.pk

10. Modularity is one of the main features of C++ language.
11. All keywords in C++ language are written in lower case most usually.
12. Basic structure of C++ program consists of preprocessor directive, main function and program body.
13. Each statement in C++ language is terminated by colon or semi-colon.
14. Preprocessor directive is also called compiler directive as it is part of C++ compiler.
15. The extension of a header file is .hf.
16. The header files in Turbo C++ are commonly stored in BIN subdirectory.
17. The maximum statements that can be written in main() function are 30.
18. A process of converting source code into object code is called compiling.
19. Linking is the process of linking library files with source code.
20. A set of rules for writing programs in a programming language is called syntax.
21. Logical errors can be detected easily by compiler.
22. The errors that occur during the execution of program are called syntax errors.
23. Computer does not understand object code directly.
24. Software that converts high-level programs into machine language is called language processor.
25. A preprocessor directive always begins with a double slash (//).
26. ANSI is the abbreviation of American Nation Standard Institute.
27. C++ is very useful for writing system software.
28. The keyword void in main function represents data type of the value returned by the function.
29. F3 is a shortcut key to save a C++ program.
30. The include directive tells the compiler where to find the meanings of standard identifier used in the program.
31. The preprocessor directive start with % sign and keyword include or process.
32. The math.h header file contains the definition of mathematical functions.
33. main is the function where the execution of C++ program begins.
34. The include directive gives a program access to a library.
35. Every C++ program must have a main function.
36. A string can be stored in an array of characters.
37. A preprocessor directive does not require a semicolon at the end.
38. The linker can be invoked by selecting Compile > Link from menu bar.
39. The loader contains different library files.
40. The preprocessor executes after the compiler.
41. The C++ language is derived from the C+ language which preceded it:
42. In C++, each statement must be on a separate line.
43. A program containing a logic error will compile and run, but not produce correct results
44. An attempt to divide by zero during program execution will result in a run-time error
45. Every C++ program has a function called main.

Answers

1. F	2. T	3. F	4. T	5. T
6. T	7. F	8. T	9. F	10. T
11. T	12. T	13. F	14. T	15. F
16. T	17. F	18. T	19. F	20. T
21. F	22. F	23. F	24. T	25. F
26. T	27. T	28. T	29. F	30. T
31. F	32. T	33. T	34. T	35. T
36. T	37. T	38. T	39. F	40. F
41. F	42. F	43. T	44. T	45. T

Freebooks.pk

CHAPTER 3

PROGRAMMING IN C++

Chapter Overview

3.1 Identifier
- 3.1.1 Types of Identifiers

3.2 Keywords

3.3 Data Types
- 3.3.1 Integer Data Types
- 3.3.2 Real Data Types
- 3.3.3 Character Data Type

3.4 Integer Overflow and Underflow

3.5 Scientific or Exponential Notation
- 3.5.1 Range and Precision

3.6 Variables
- 3.6.1 Variables Declaration
- 3.6.2 Rules for Declaring Variables
- 3.6.3 Variable Initialization

3.7 Constants
- 3.7.1 Literal Constant
- 3.7.2 Symbolic Constants

3.8 Expression

3.9 Operators
- 3.9.1 Arithmetic Operators
- 3.9.2 Assignment Statement
- 3.9.3 Compound Assignment Operators
- 3.9.4 Increment Operator
- 3.9.5 Decrement Operator
- 3.9.6 Operator Precedence
- 3.9.7 Operator Associativity

3.10 Type Casting
- 3.10.1 Implicit Type Casting
- 3.10.2 Explicit Casting

3.11 The "sizeof" Operator

3.12 Comments

Exercise Questions

Multiple Choice

Fill in the Blanks

True/False

Freebooks.pk

3.1 Identifier

The **identifiers** are the names used to represent variable, constants, types, functions and labels in the program. Identifier is an important feature of all computer languages. A good identifier name should be descriptive but short.

An identifier in C++ may consist of 31 characters. If the name of an identifier is longer than 31 characters, the first 31 character will be used. The remaining characters will be ignored by C++ compiler. Some important rules for identifier name are as follows:

- The first character must be an alphabetic or underscore (_).
- The identifier name must consist of only alphabetic characters, digits or underscores.
- The reserved word cannot be used as identifier name.

3.1.1 Types of Identifiers

C++ provides the following types of identifiers:

1. Standard Identifiers

A type of identifier that has special meaning in C++ is known as **standard identifier**. C++ cannot use a standard identifier for its original purpose if it is redefined.

Example

cout and **cin** are examples of standard identifiers. These are the names of input/output objects defined in standard input/output library **iostream.h**.

2. User-defined Identifiers

The type of identifier that is defined by the programmer to access memory location is known as **user-defined identifier**. The user-defined identifiers are used to store data and program results.

Example

Some examples of user-defined identifiers are **a**, **marks** and **age** etc.

3.2 Keywords

Keyword is a word in C++ language that has a predefined meaning and purpose. The meaning and purpose of a keyword is defined by the developer of the language. It cannot be changed or redefined by the user. Keyword can be used for the same purpose for which it is defined. Keywords are also known as **reserved words**. There are different types of keywords in C++ language. The total number of keywords is 63.

List of Keywords

asm	auto	bool	break	case	catch
char	class	const	const_cast	continue	default
delete	do	double	dynamic_cast	else	enum
explicit	export	extern	false	float	for
friend	goto	if	inline	int	long
mutable	namespace	new	operator	private	protected
public	register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch	template
this	throw	true	try	typedef	typeid
typename	union	unsigned	using	virtual	void
volatile	wchar_t	while			

Freebooks.pk

3.3 Data Types

The **data type** defines a set of values and a set of operations on those values. The computer manipulates various types of data. The data is given to the program as input. The data is processed according to the program instructions and output is returned. The data and its type are defined before designing the actual program used to process the data. The type of each data value is identified at the beginning of program design.

A C++ program may need to process different types of data. Each data type require different amount of memory. C++ language provides the following data types:

Data type	Purpose
int	To store numeric values
float	To store real values
double	To store large real values
char	To store character values

Table 3.1: Data types in C++ language

3.3.1 Integer Data Types

Integer data is numeric value with no decimal point or fraction. It includes both positive and negative values. The minus sign – is used to indicate negative value. If no sign is used, the value is positive by default.

Examples

Some examples of integer values are 10, 520 and -20 etc.

Types of Integers

C++ provides different types of **integer** data. These are as follows:

Data Type	Size in Bytes	Description
int	2	Ranges from –32,768 to 32,767.
short	2	Ranges from –32,768 to 32,767.
unsigned int	2	Ranges from 0 to 65,535.
long	4	Ranges from –2,147,483,648 to 2,147,483,647.
unsigned long	4	Ranges from 0 to 4,294,967,295.

Table 3.2: Data types for integers

1. int Data Type

int data type is used to store integer values. It takes two or four bytes in memory depending on the computer and compiler being used. In **MS-DOS**, it takes two bytes and its range is from -32768 to 32767.

INT_MIN represents the smallest value that can be stored in **int** data type. **INT_MAX** represents the largest value that can be stored in **int** data type. Both **INT_MIN** and **INT_MAX** are found in **limits.h** header file.

2. short int Data Type

short int data type is used to store integer values. It takes two bytes in memory. Its range is from -32768 to 32767.

SHRT_MIN represents the smallest value that can be stored in **short int** data type. **SHRT_MAX** represents the largest value that can be stored in **short int** data type. Both **SHRT_MIN** and **SHRT_MAX** are found in **limits.h** header file.

Freebooks.pk

3. unsigned int Data Type

unsigned int data type is used to store only positive integer values. It takes two bytes in memory. Its range is from 0 to 65,535.

UINT_MAX represents the largest value that can be stored in **unsigned int** data type. It is found in **limits.h** header file.

4. long int Data Type

long int data type is used to store large integer values. It takes four bytes in memory. Its range is from –2,147,483,648 to 2,147,483,647.

LONG_MIN represents the smallest value that can be stored in **long int** data type. **LONG_MAX** represents the largest value that can be stored in **long int** data type. Both **LONG_MIN** and **LONG_MAX** are found in **limits.h** header file.

5. unsigned long int Data Type

unsigned long int data type is used to store large positive integer values. It takes four bytes in memory. Its range is from 0 to 4,294,967,295.

ULONG_MAX represents the largest value that can be stored in **unsigned long int** data type. It is found in **limits.h** header file.

3.3.2 Real Data Types

Real data is numeric value with decimal point or fraction. It is also called floating point number. It includes both positive and negative values. The minus sign – is used to indicate negative value. If no sign is used, the value is positive by default.

Examples

Some examples of real values are 10.5, 5.3 and -10.91 etc.

Types of Real

C++ provides different types of real data. These are as follows:

Data Type	Size in Bytes	Description
float	4	3.4×10^{-38} to $3.4 \times 10^{+38}$
double	8	1.7×10^{-308} to $1.7 \times 10^{+308}$
long double	10	1.7×10^{-4932} to $1.7 \times 10^{+4932}$

Table 3.3: Data types for real

1. float Data Type

float data type is used to store real values. It takes four bytes in memory. Its range is from 3.4×10^{-38} to $3.4 \times 10^{+38}$. It provides accuracy of 6 decimal places.

FLT_MIN represents the smallest value that can be stored in **float** data type. **FLT_MAX** represents the largest value that can be stored in **float** data type. Both **FLT_MIN** and **FLT_MAX** are found in **float.h** header file.

2. double Data Type

double data type is used to store large real values. It takes eight bytes in memory. Its range is from 1.7×10^{-308} to $1.7 \times 10^{+308}$. It provides accuracy of 15 decimal places.

DBL_MIN represents the smallest value that can be stored in **double** data type. **DBL_MAX** represents the largest value that can be stored in **double** data type. Both **DBL_MIN** and **DBL_MAX** are found in **float.h** header file.

Freebooks.pk

3. long double Data Type

long double data type is used to store very large real values. It takes ten bytes in memory. Its range is from $1.7x10^{-4932}$ to $1.7x10^{+4932}$. It provides accuracy of 19 decimal places.

LDBL_MIN represents the smallest value that can be stored in **long double** data type. **LDBL_MAX** represents the largest value that can be stored in **long double** data type. Both **LDBL_MIN** and **LDBL_MAX** are found in **float.h** header file.

3.3.3 Character Data Type

char data type is used to store character value. It takes 1 byte in memory. It is used to represent a letter, number or punctuation mark and a few other symbols.

Character values are normally given in single quotes. It can represent individual characters such as 'a', 'x', '5', and '#'. The character '5' is manipulated differently than integer 5. It is possible to perform mathematical operation on character values. The characters can be added, subtracted and compared like numbers.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char ch1, ch2, sum;
  ch1 = 'A';
  ch2 = 'B';
  cout<<"Characters are: "<<ch1<<ch2;
  getch();
}
```

Output:
Characters are: AB

CHAR_MIN represents the smallest value that can be stored in **char** data type. **CHAR_MAX** represents the largest value that can be stored in **char** data type. Both **CHAR_MIN** and **CHAR_MAX** are found in **limits.h** header file.

3.4 Integer Overflow and Underflow

Integer overflow occurs when the value assigned to an integer variable is more than maximum possible value. Integer underflow occurs when the value assigned to an integer variable is less than possible minimum value.

An integer variable can stored values from -32768 to 32767. If the assigned value is more than 32767, it is known as **integer overflow**. If the assigned value is less than -32768, it is called **integer underflow**.

Integer overflow and underflow is not detected by the compiler. But the results may become wrong. It is the responsibility of the user to ensure that the values assigned to integer variables are within the allowed range.

Program 3.1

Write a program that explains the concept of overflow and underflow.

```
#include <conio.h>
#include <iostream.h>
void main()
{
  clrscr();
```

Freebooks.pk

```
    short testVar = 32767;
    cout << testVar << endl;
    testVar = testVar + 1;
    cout << testVar << endl;
    testVar = testVar - 1;
    cout << testVar << endl;
    getch();
}
```

Output:
32767
-32768
32767

3.5 Scientific or Exponential Notation

The real numbers can also be written in exponential or scientific notation. This notation represents large real numbers in a short way.

The exponential notation consists of two parts:

- Mantissa
- Exponent

The general form of writing real values in exponential notation is as follows:

±m **e** ±n OR ±m **E** ±n

The value before **e** is known as **mantissa** and the value after **e** is known as **exponent**. Some important points about exponential notation are as follows:

- Mantissa and exponent may be positive or negative value.
- Exponent is always an integer value. It cannot be a real number.
- The absolute value of mantissa must be greater or equal to 1 and less than 10.

Suppose we have a real number 1000.0. It can be represented in scientific notation as **1.0×10^3**. It can also be written in exponential form as **1.0e3.** In this example, 1.0 is mantissa and 3 is the value of exponent. Similarly, $0.004694 = 4.694 \times 10^{-3}$. It can also be written as **4.694e-3.**

Some examples of real numbers with scientific and exponential notations are as follows:

Real Number	Scientific Notation	Exponential Notation
1.98	1.98×10^0	1.98e0
4,679,000.0	4.679×10^6	4.679e6
0.00053	5.3×10^{-4}	5.3e-4

3.5.1 Range and Precision

The possible values that a floating type variable can store are described in terms of precision and range:

- **Precision:** Precision is the number of digits after the decimal point.
- **Range:** Range is the exponential power of 10.

A **float** variable has a precision of 6 digits and a range of 10^{-38} to 10^{+38}. Suppose we have a value 0.1234567×10^3. The precision of this value is 7 and range is 3. Similarly, a value 0.123456 x 10-3 has a precision of 6 digits and a range of -3.

A **double** variable has a precision of 15 digits and a range of 10^{-308} to 10^{+308}. A large precision provides more accuracy. If the value to be stored requires more precision, **double** data type is more suitable than **float**.

3.6 Variables

A **variable** is a named memory location or memory cell. It is used to store program's input data and its computational results during execution. The value of a variable may change during the execution of program. However, the name of variable cannot be changed.

The variables are created in RAM. RAM is a temporary memory. That is why the data stored in variables is also temporary. It can only be used and processed during the execution of program. The data stored in the variable is automatically removed when program ends.

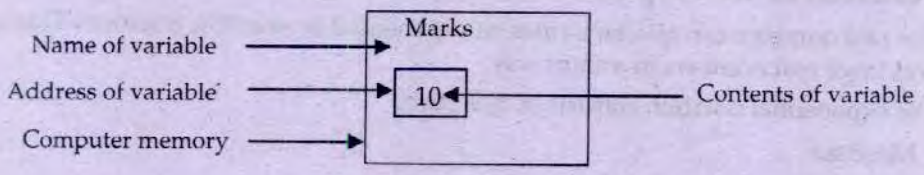


Figure 3.1: Variable in memory

In the above figure:

Name of variable It refers to an identifier that represents a memory location.
Address of variable It refers to the memory location of the variable.
Contents of variable It refers to the value stored in memory location referred by variable.

3.6.1 Variables Declaration

The process of specifying the variable name and its type is called **variable declaration**. A program can have as many variables as needed. All variables must be declared before they are used in the program. A variable can be declared anywhere in the program before its use.

The variable declaration provides information to the compiler about the variable. The compiler uses the declaration to determine how much memory is needed for each variable. Different types of data require different amount of memory. The required number of bytes are allocated when a variable is declared. For example, an integer variable requires 2 bytes whereas a character variable requires 1 byte. The memory bytes are used to store the value of the variable. Once a variable is declared, its data type cannot be changed during program execution. However, the value of variable can be changed during execution.

Syntax

The syntax of declaring variables is as follows:

```
data_type variable_name;
```

data_type It indicates type of data that can be stored in variable.
variable_name It refers to the memory location of the variable.

Examples

Different types of variables are declared as follows:

```
int marks;
float average;
char grade;
double salary;
```

In the above examples, **int**, **float**, **char** and **double** are keywords that indicate data types. The words **marks**, **average**, **grade** and **salary** are variable names.

Freebooks.pk

Many variables of same data type can also be declared in single line. In this case, each variable is separated by comma as follows:

```
int a, b, c;
```

3.6.2 Rules for Declaring Variables

Following are some rules for naming variables in C++ language:

- Variable may include letters, numbers and underscore (_).
- The first character of variable must be a letter or underscore _. The use of underscore is not recommended. The variables 9minute, #home and 2kg are invalid.
- Blank spaces are not allowed in variable names. The variables **my var** and **your car** are invalid.
- Both upper and lower cases are allowed. A user-defined variable is conventionally written in lower case. The constants are conventionally written in upper case.
- Special symbols cannot be used as variable name.
- Reserved word cannot be used as variable name. The names **int**, **void** and **while** are invalid variables.
- A variable can be up to 31 characters long for many compilers. If a variable consists of more than 31 characters, only first 31 characters will be used. The remaining characters will be ignored.
- A variable can be declared only for one data type.

The variable name must be meaningful and readable. It helps the user to understand the purpose of the variable. For example, a variable to store the marks of student should be **Marks**. If a variable consists of multiple words, first letter of each word should be capitalized to increases readability. A variable MyPhoneNo is more readable than myphoneno.

Examples

Some examples of valid variables are as follows:

```
Marks        sub1        _test        f_name        Integer
```

3.6.3 Variable Initialization

The process of assigning a value to a variable at the time of declaration is known as **variable initialization**. The equal sign = is used to initialize a variable. Variable name is given on left side and value is given on the right side of equal sign.

The C++ compiler automatically sets aside some memory for the variable when it is declared. The memory location may contain meaningless data known as **garbage value**. It may produce unexpected results in some computations. All variables should be initialized to avoid this problem.

Syntax

The syntax of initializing a variable is as follows:

```
type_name variable = value;
```

type_name	It indicates the data type of the variable to be initialized.
variable	It is name of the variable to be initialized.
=	It is the assignment operator used to initialize a variable.
value	It is the value to initialize a variable.

Freebooks.pk

Example

Examples of variable initialization are as follows:

```
int n = 100;
int x = 50, y = 75;
float average = 50.73;
char grade = 'A';
```

3.7 Constants

Constant is a quantity that cannot be changed during program execution. Following are two types of constants in C++:

- Literal constant
- Symbolic constant

3.7.1 Literal Constant

A literal constant is a value that is typed directly in a program. It appears in the program wherever it is needed.

Example

The following statement contains string literal constant **Hello World**:

```
cout<<"Hello World";
```

Similarly, in the following statement:

```
int age = 19;
```

age is integer variable and **19** is a literal constant.

Types of Literal Constants

Different types of literal constants are as follows:

1. Integer constant
2. Floating point constant
3. Character constant
4. String constant

1. Integer Constants

Integer constants are numeric values without fraction or decimal point. Both positive and negative integer constant are used. The minus sign – is used for negative integer constants. If no sign is used, the value is positive by default.

Examples

Some examples of integer constants are as follows:

```
87      45      -10      -5
```

The use of **L** at the end of **integer** indicates that it is a **long integer**.

```
65000L          -55000L
```

2. Floating Point Constants

Floating point constants are numeric values with fraction or decimal point. Both positive and negative floating point constants are used. The minus sign – is used for negative floating point constants. If no sign is used, the value is positive by default. The use of **F** or **f** at the end of real value indicates that the value is **float**.

Freebooks.pk

Examples

Some examples of floating point constants are as follows:

50.75F 10.22F

If **F** or **f** is not used, the value is considered as **double**.

15.32 42.79

The use of **L** at the end of real value indicates that it is a **long double**.

52000.33L

3. Character Constants

Any character written in single quotes is known as character constant. All alphabetic characters, digits and special symbols can be used as character constants.

Examples

Some examples of character constants are as follows:

'A' 'n' '9' '=' '$'

4. String Constants

A set of characters written in double quotations is called **string** or **string constant**. It may consist of any alphabetic characters, digits and special symbols.

Examples

Some examples of sting constants are as follows:

"Pakistan" "123" "99-Mall Road, Lahore"

3.7.2 Symbolic Constants

A symbolic constant is a name given to values that cannot be changed. A constant must be initialized. After a constant is initialized, its value cannot be changed. Symbolic constants are used to represent a value that is frequently used in a program.

A symbolic constant **PI** can be used to indicate the value of Pi which is 3.141593. PI can be written in program wherever its value is required.

Symbolic constants can be declared in two ways:

1. **const** Qualifier
2. **define** Directive

1. const Qualifier

const qualifier is used to define a constant. The constant is declared by specifying its name and data type. The constant must be initialized with some value. The initialized value cannot be changed during program execution.

Syntax

The syntax of declaring constants with **const** qualifier is as follows:

```
const data_type identifier = value;
```

const	It is a keyword used to define a constant.
data_type	It indicates the data type of the constant.
identifier	It represents the name of the constant.
value	It represents the value with which the constant is initialized.

Freebooks.pk

Example

An example of constant declaration is as follows:

```
const int N = 100;
```

The above example defines a constant **N** that is initialized with 100. The value 100 cannot be changed during the execution of program.

2. define Directive

define directive is also used to define a constant. The difference between **const** qualifier and **define** directive is that **define** directive does not specify the data type of the constant. **define** directive starts with the symbol **#**. It is not terminated with semicolon.

Syntax

The syntax of **define** directive is as follows:

```
#define  identifier  value
```

#	It indicates the start of preprocessor directive.
define	It is used to define a constant.
identifier	It is the name of the constant.
value	It represents the value associated with the identifier.

The preprocessor directive replaces all occurrences of the **identifier** with the **value**. The identifier is conventionally written in uppercase.

Example

```
#define PI  3.141593
```

Program 3.2

Write a program that inputs the radius of a circle and displays the circumference by using formula $2\pi R$. Store the value of π in a constant by using DEFINE directive.

```
#include <iostream.h>
#include <conio.h>
#define PI 3.141
void main()
{
  float r, area;
  clrscr();
  cout<<"Enter radius: ";
  cin>>r;
  area = 2.0 * PI * r;
  cout<<"Area = "<<area;
  getch();
}
```

3.8 Expression

A statement that evaluates to a value is called an **expression**. An expression gives a single value. An expression consists of **operators** and **operands**. An operator is a symbol that performs some operation. Operand is the value on which the operator performs some operation. Operand can be a constant, variable or function. An expression may consist of any number of operators and operands. Some examples of expressions are as follows:

```
A + B;
m / n;
```

Freebooks.pk

x + 100;

In the expression **A + B**, + is the operator. **A** and **B** are variables used as operands in the expression.

3.9 Operators

Operators are the symbols that are used to perform certain operations on data. C++ provides a variety of operators. These include arithmetic operators, relational operators, logical operators, bitwise operators etc. The operators can be categorized as follows:

1. Unary Operators

A type of operator that works with one operand is known as **unary operator**. Following operators are unary operators:

–, ++, – –

The above operators are used with one operand as follows:

– a;
N++;
– –x;

2. Binary Operators

A type of operator that works with two operands is known as **binary operator**. Following operators are binary operators:

+, –, *, /, %

The above operators are used with two operands as follows:

a + b;
x / y;

3.9.1 Arithmetic Operators

Arithmetic operator is a symbol that performs mathematical operation on data. C++ provides many arithmetic operators. Following is a list of all arithmetic operators in C++:

Operation	Symbol	Description
Addition	+	Adds two values
Subtraction	–	Subtracts one value from another value
Multiplication	*	Multiplies two values
Division	/	Divides one value by another value
Modulus	%	Gives the remainder of division of two integers

Table 3.4: Mathematical operators in C++ language

All arithmetic operators work with numeric values. Suppose we have two variables A and B where A = 10 and B = 5. Arithmetic operators can be used on A and B as follows:

Operation	Result
A + B	15
A – B	5
A * B	50
A / B	2
A % B	0

Table 3.5: Use of mathematical operators

Freebooks.pk

Some important points about modulus operator are as follows:

- Modulus operator is also called **remainder operator**.
- The modulus operator works only with integer values.
- If modulus operator is used with the division of 0, the result will always be 0. For example the expression 0 % 5 will give 0 as a result.
- In expression like 3 % 5, 3 is not divisible by 5. Its result is 3.

Working of Division Operator

The manipulation of division operator is different from other operators. The result of division operation is always an integer if divisor and dividend are integers. The fractional part of the quotient is truncated. For example, the result of 7.0/2.0 is 3.5 but the result of 7/2 is 3. The floating point number must be used to get the accurate result of a division operation.

3.9.2 Assignment Statement

A statement that assigns a value to a variable is known as **assignment statement**. The assignment operator = is used in assignment statement to assign a value or computational result to a variable. The name of variable is written on the left side of the assignment operator and the value is written on the right side of assignment operator. The value can be a constant, variable, expression or function.

Syntax

The syntax of writing an assignment statement is as follows:

```
variable = expression;
```

variable	It is the name of variable to which the value is assigned.
=	It is the assignment operator used to assign the value to variable.
expression	It is the expression whose returned value is assigned to variable. It can be a constant, variable or combination of operands and arithmetical operators. The expression on right side is evaluated first. The result is then assigned to the variable on left side.

Examples

Some examples of assignment statements are as follows:

```
A = 100;
C = A + B;
X = C – D + 10;
```

Lvalue and Rvalue

An **lvalue** is an operand that can be written on the left side of assignment operator =. It must always be a single value. An **rvalue** is an operand that can be written on the right side of assignment operator =. All lvalues can be used as rvalues. But all rvalues cannot be used as lvalues. For example, a constant can be used as rvalue but it cannot be used as lvalue. The expression x = 5 is valid but the expression 5 = x is not valid.

Program 3.3

Write a program that performs all mathematical operations on two variables.

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
void main()
{
  clrscr();
  int a, b;
  a = 10;
  b = 5;
  cout<<"a + b = "<<a + b<<endl;
  cout<<"a – b = "<<a – b<<endl;
  cout<<"a * b = "<<a * b<<endl;
  cout<<"a / b = "<<a / b<<endl;
  cout<<"a % b = "<<a % b<<endl;
  getch();
}
```

Output:
a + b = 15
a – b = 5
a * b = 50
a / b = 2
a % b = 0

Compound Assignment Statement

An assignment statement that assigns a value to many variables is known as **compound assignment** statement. The assignment operator = is used in this statement.

Example

The following statement

```
A = B = 10;
```

assigns the value 10 to both A and B. Some examples of compound assignment statement are as follows:

```
x = y = z = 100;
m = n = 50;
```

3.9.3 Compound Assignment Operators

C++ language provides **compound assignment operators** that combine assignment operator with arithmetic operators. Compound assignment operators are used to perform mathematical operations more easily.

Syntax

The general syntax of compound assignment operator is as follows:

```
variable op = expression;
```

variable	The variable to assign a value.
op	It can be any arithmetic operator.
expression	It can be a constant, variable or arithmetic expression

For example, the statement:

```
N += 10;
```

is equivalent to

```
N = N + 10;
```

Both statements are equivalent and perform the same operation. Both statements add 10 in the current value of N. All other mathematical operators can also be combined with assignment operator.

Freebooks.pk

Examples

Some examples of compound assignment are as follows:

A += 10	is equivalent to	A = A + 10
A -= 10	is equivalent to	A = A - 10
A *= 10	is equivalent to	A = A * 10
A /= 10	is equivalent to	A = A / 10
A %= 10	is equivalent to	A = A % 10

Program 3.4

Write a program that performs all compound assignment operations on an integer.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a;
  clrscr();
  a = 10;
  cout<<"Value of a: "<<a<<endl;
  a += 5;
  cout<<"Value of a after a+=5: "<<a<<endl;
  a -= 5;
  cout<<"Value of a after a-=5: "<<a<<endl;
  a *= 2;
  cout<<"Value of a after a*=2: "<<a<<endl;
  a /= 2;
  cout<<"Value of a after a/=2: "<<a<<endl;
  a %= 2;
  cout<<"Value of a after a%=2: "<<a<<endl;
  getch();
}
```

Output:
Value of a : 10
Value of a after a+=5 : 15
Value of a after a-=5 : 10
Value of a after a*=2 : 20
Value of a after a/=2 : 10
Value of a after a%=2 : 0

3.9.4 Increment Operator

The increment operator is used to increase the value of a variable by 1. It is denoted by the symbol ++. It is a unary operator and works with single variable.

The increment operator cannot increment the value of constants and expressions. For example, **A++** and **X++** are valid statements but **10++** is an invalid statement. Similarly, **(a+b)++** or **++(a+b)** are also invalid. Increment operator can be used in two forms:

- Prefix Form
- Postfix Form

Prefix Form

In prefix form, the increment operator is written **before** the variable as follows:

```
++y;
```

The above line increments the value of variable **y** by 1.

Postfix Form

In postfix form, the increment operator is written **after** the variable as follows:

```
y++;
```

The above line also increments the value of variable **y** by 1.

Freebooks.pk

Difference between Prefix & Postfix Increment

When increment operator is used independently, prefix and postfix form work similarly. For example, the result of A++ and ++A is same. But when increment operator is used in a larger expression with other operators, prefix and postfix forms work differently. For example, the results of two statements **A = ++B** and **A = B++** are different.

The statement **A = ++ B** contains two operators ++ and =. It works in following order:

1. It increments the value of **B** by 1.
2. It assigns the value of **B** to **A**.

The above statement is equivalent to the following two statements:

```
++B;
A = B;
```

In postfix form, the statement **A = B++** works in the following order:

1. It assigns the value of **B** to **A**.
2. It increments the value of **B** by 1.

The above statement is equivalent to the following two statements:

```
A = B;
B++;
```

Program 3.5

Write a program that explains the difference of postfix increment operator and prefix increment operator used as independent expression.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int a, b, x, y;
  a = b = x = y = 0;
  a++;
  b = a;
  ++x;
  y = x;
  cout<<"a = "<<a<<endl<<" b = "<<b<<endl;
  cout<<"x = "<<x<<endl<<" y = "<<y<<endl;
  getch();
}
```

Output:
a = 1
b = 1
x = 1
y = 1

Program 3.6

Write a program that explains the difference of postfix increment operator and prefix increment operator used as part of a larger expression.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int a, b, x, y;
```

Freebooks.pk

```
    a = b = x = y = 0;
    b = a++;
    y = ++x;
    cout<<"a = "<<a<<endl<<" b = "<<b<<endl;
    cout<<"x = "<<x<<endl<<" y = "<<y<<endl;
    getch();
}
```

Output:
a = 1
b = 0
x = 1
y = 1

3.9.5 Decrement Operator

The decrement operator is used to decrement the value of a variable by 1. It is denoted by the symbol - -. It is a unary operator and works with single variable.

The decrement operator cannot decrement the value of constants and expressions. For example, **A- -** and **X- -** are valid statements but **10- -** is an invalid statement. Similarly, **(a+b)- -** or **- -(a+b)** are also invalid. Decrement operator can be used in two forms:

- Prefix Form
- Postfix Form

Prefix Form

In prefix form, decrement operator is written **before** the variable as follows:

```
--y;
```

The above line decrements the value of variable **y** by 1.

Postfix Form

In postfix form, the decrement operator is written **after** the variable as follows:

```
y--;
```

The above line also decrements the value of variable **y** by 1.

Difference between Prefix & Postfix Decrement

When decrement operator is used independently, prefix and postfix form work similarly. For example, the result of **A- -** and **- -A** is same. But when increment operator is used in a larger expression with other operators, prefix and postfix forms work differently. For example, the results of two statements **A = - -B** and **A = B- -** are different.

The statement **A = - - B** contains two operators i.e. - - and =. It works in the following order:

3. It decrements the value of **B** by 1.
4. It assigns the value of **B** to **A**.

The above statement is equivalent to the following two statements:

```
--B;
A = B;
```

In postfix form, the statement **A = B- -** works in the following order:

5. It assigns the value of **B** to **A**.
6. It decrements the value of **B** by 1.

The above statement is equivalent to the following two statements:

```
A = B;
B--;
```

Freebooks.pk

Program 3.7

Write a program that explains the difference of postfix decrement operator and prefix decrement operator used as independent expression.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int a, b, x, y;
  a = b = x = y = 0;
  a--;
  b = a;
  --x;
  y = x;
  cout<<"a = "<<a<<endl<<" b = "<<b<<endl;
  cout<<"x = "<<x<<endl<<" y = "<<y<<endl;
  getch();
}
```

Output:
a = - 1
b = -1
x = -1
y = -1

Program 3.8

Write a program that explains the difference of postfix decrement operator and prefix decrement operator used as part of a larger expression.

```
#include <iostream.h>
#include <conio.h>
{
  clrscr();
  int a, b, x, y;
  a = b = x = y = 0;
  b = a--;
  y = --x;
  cout<<"a = "<<a<<endl<<" b = "<<b<<endl;
  cout<<"x = "<<x<<endl<<" y = "<<y<<endl;
  getch();
}
```

Output:
a = -1
b = 0
x = -1
y = -1

3.9.6 Operator Precedence

The order in which different types of operators in an expression are evaluated is known as **operator precedence**. It is also known as **hierarchy of operators**.

Each operator has its own precedence level. If an expression contains different types of operators, the operators with higher precedence are evaluated before the operators with lower precedence. The order of precedence in C++ language is as follows:

- Any expression given in parentheses is evaluated first.
- Then multiplication * and division / operators are evaluated.
- Then plus + and minus - operators are evaluated.
- In case of parentheses within parentheses, the expression of the inner parentheses will be evaluated first.

Freebooks.pk

Example

The expression **10 * (24 / (5 – 2)) + 13** is evaluated in the following order:

1. First of all, the expression **5–2** will be evaluated. It gives a value **3**.
2. Secondly, **24** will be divided by the result of last line i.e. **24 / 3** giving value **8**.
3. Thirdly, **10** will be multiplied by **8** i.e. giving a result **80**.
4. Finally, **80** will be added in **13** and the last result will be **93**.

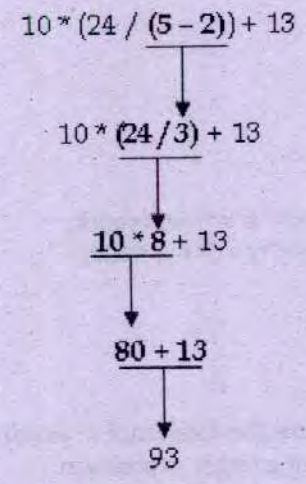


Figure 3.2: Operator precedence

3.9.7 Operator Associativity

The order in which operators of same precedence are evaluated is known as **operator associativity**. If an expression contains some operators that have same precedence level, the expression is evaluated either from **left-to-right** or **right-to-left**.

Operator associativity in C++ language is as follows:

Operators	Associativity
() ++ (postfix) - - (postfix)	Left-to-right
+ (unary) – (unary) ++ (prefix) – – (prefix)	Left-to-right
* / %	Left-to-right
+ –	Left-to-right
= += –= *= /=	Right-to-left

Table 3.6: Associativity of operators

Example

The expression **10 * 24 / 5 – 2 + 13** contains four operators. Two operators * and / have equal precedence. These expressions will be evaluated from left to right. Similarly, two operators + and - also have equal precedence. The expression will be evaluated as follows:

- First of all, the expression **10 * 24** is evaluated. It gives a value **240**.
- Secondly, **240** will be divided by **5** i.e. **240 / 5** giving a value **48**.
- Thirdly, **2** will be subtracted from **48** i.e. **48 – 2** giving a result **46**.
- Finally, **46** will be added in **13** and the last result will be **59**.

Freebooks.pk

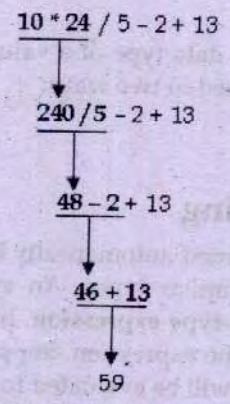


Figure 3.3: Operator Associativity

Program 3.9

Write a program that solves the following expression:

a * b / (- c * 31 % 13) * d

Assuming the values of variables are as follows:

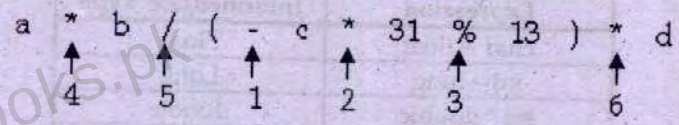
a = 10, b = 20, c = 15, d = 8, e = 40

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c, d, r;
  clrscr();
  a = 10;
  b = 20;
  c = 15;
  d = 8;
  r = a * b / ( - c * 31 % 13 ) * d;
  cout<<"Result of expression is : "<<r<<endl;
  getch();
}
```

Output:
Result of expression is : -160

How above Program Works?

The above expression will be evaluated in the following order:



The working of above expression in different steps will be as follows:

Step	Operator	Reduced expression
1	Unary -	a * b / (- 15 * 31 % 13) * d
2	*	a * b / (- 465 % 13) * d
3	%	a * b / (- 10) * d
4	*	200 / (- 10) * d
5	/	-20 * d
6	*	-160

3.10 Type Casting

The process of converting the data type of a value during execution is known as **type casting**. Type casting can be performed in two ways:

- Implicit type casting
- Explicit type casting

3.10.1 Implicit Type Casting

Implicit type casting is performed automatically by the C++ compiler. The operands in arithmetic operation must be of similar types. An expression in which operands are of different data types is called **mixed-type expression**. In this case, the result of an expression is evaluated to larger data type in the expression. Suppose an expression contains an integer and double as operands. The result will be evaluated to a double data type.

Different types of variables are as follows:

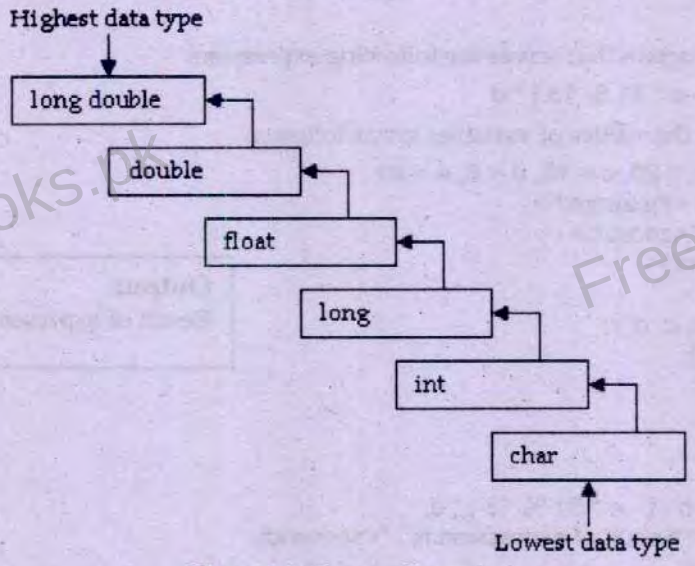


Figure 3.4: Type casting order

Some examples of implicit conversions are as follows:

Expression	Intermediate Type
char + float	Float
int – long	Long
int * double	double
float / long double	long double

Example

Suppose **x** is an **integer** and **y** is a **long** variable and following expression is evaluated:

x + y

In the above expression, data type of **x** is lower than data type of **y**. So the value of **x** will be converted into **long** during the evaluation of expression. The data type of **x** is not changed. Only the data type of the value of **x** is changed during the evaluation of expression.

Freebooks.pk

3.10.2 Explicit Casting

Explicit casting is performed by programmer. It is performed by using **cast operator.** The cast operator tells the computer to convert the data type of a value.

Syntax

The syntax of using cast operator is as follows:

```
(type) expression;
```

type It indicates the data type to which operand is to be converted.

expression It indicates the constant, variable or expression whose data type is to be converted.

Example

Suppose **x** and **y** are two **float** variables. **x** contains 10.3 and **y** contains 5.2 and the following expression is evaluated:

```
x % y
```

The above expression will generate an error because data type of **x** and **y** is **float**. The modulus operator cannot work with float variables. It only works with integers. So the expression can be written as follows:

```
(int) x % (int) y
```

The above statement converts the values of **x** and **y** into integers and evaluates the expression. The value of **x** will be converted to 10 and value of **y** will be converted to 5. So the result of the above expression will be 0.

Program 3.10

Write a program that divides two float variables and finds the remainder by using explicit casting.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  float a, b;
  int c;
  a = 10.3;
  b = 5.2;
  c = (int)a % (int)b;
  cout<<"Result is "<<c;
  getch();
}
```

Output:
Result is 0

How above Program Works?

The above program finds the remainder of two **float** variables using modulus operator. The modulus operator cannot work with float variables. So type casting is used to convert these variables into integers in the expression. The result of the expression will be 0.

3.11 The "sizeof" Operator

The **sizeof** operator is used to find the size of any data value. It gives the number of bytes occupied by that value.

Freebooks.pk

Syntax

The syntax of using this operator is:

```
sizeof(operand);
```

The **operand** can be a variable or a constant.

Example

Following are some examples of using **sizeof** operator:

```
sizeof(n);
sizeof(10);
sizeof("Pakistan");
```

Program 3.11

Write a program that displays the sizes of different data types.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  cout<<"size of char = "<<sizeof(char)<<endl;
  cout<<"size of int = "<<sizeof(int)<<endl;
  cout<<"size of float = "<<sizeof(float)<<endl;
  cout<<"size of long = "<<sizeof(long)<<endl;
  cout<<"size of double = "<<sizeof(double)<<endl;
  cout<<"size of long double = "<<sizeof(long double)<<endl;
  getch();
}
```

Output:
The size of char : 1
The size of int: 2
The size of float : 4
The size of long : 4
The size of double : 8
The size of long double : 10

3.12 Comments

Comments are the lines of program that are not executed. The compiler ignores comments and does not include them in the executable program. That is why the comments do not affect the size of executable program. Comments are used to increase the readability of the program. Comments are notes about different lines of code that explain the purpose of the code. The user can insert information notes in the code. It helps in debugging and modifying the program later. Comments can be added anywhere in programs in two ways:

- **Sing-line Comments:** Comments on single line are added by using double slash "//". Any thing written on the right side of double slash is considered as comments and is ignored during execution.
- **Multi-line Comments:** Multi-line comments are inserted to the code by placing /* at the beginning of the comments. The character */ is used to end multi-line comments.

Examples

Following line is an example of comments:

```
// Practice makes a man programmer.
```

Another way to add comments on single line is as follows:

```
/* These are also comments... */
```

In order to add comments on many lines in program, following way is used:

```
/* These lines are for the purpose of
writing comments */
```

Freebooks.pk

Exercise Questions

Freebooks.pk

Q.1. What is an identifier? What are the legal characters for an identifier?

An identifier is a word that represents certain program entity. Some examples of identifiers are Student_Age, Item20, Sum etc. Identifiers are used for variables, constants, functions etc. The legal characters to construct identifier include upper alphabets from A to Z, lower alphabets from a to z, digits from 0 to 9 and underscore character.

Q.2. What is the difference between standard identifiers and user-defined identifiers?

A type of identifier that has special meaning in C++ is known as **standard identifier**. C++ cannot use a standard identifier for its original purpose if it is redefined. cout and cin are examples of standard identifiers

The type of identifier that is defined by the programmer to access memory location is known as **user-defined identifier**. The user-defined identifiers are used to store data and program results. Some examples of user-defined identifiers area, marks and age etc.

Q.3. What is keyword? Also, give an example.

Keyword is a word in C++ language that has a predefined meaning and purpose. The meaning and purpose of a keyword is defined by the developer of the language. It cannot be changed or redefined by the user. Keyword can be used for the same purpose for which it is defined. Examples include if, while, int, and const

Q.4. What is meant by data type?

The data type defines a set of values and a set of operations on those values. The data and its type are defined before designing the actual program used to process the data. The type of each data value is identified at the beginning of program design.

Q.5. Name and give the purpose of different data type available in C++.

int are used to store whole numbers (integers). 10,20 and –520 are examples of integer data type. float is used to store a real or floating point value (value with a decimal point). it is also called real data. 10.5,5.2 and –10.91 are examples of real data. A char stores a single character including letters, punctuation marks and digits. 'a','5' and '#' are examples of character data type. However, storing the numeric digit '5' is not the same as storing the number 5.

Q.6. List at least three data types in C++. How much memory does each require?

char	1 byte
int	2 bytes
double	8 bytes

Q.7. What is a variable?

A quantity whose value can be changed during execution of program is known as variable. The value of a variable is referred by variable name. Each variable represents a memory location. The data stored in the variables is automatically removed when program ends. The value of a variable can change during program execution but the name of variable cannot be changed.

Q.8. What does declaring a variable mean?

The process of specifying the variable name and its type is called **variable declaration**. A variable must always be declared before it can be used in a program. The compiler gives an error if an undeclared variable is used in a program.

Q.9. List out different rules for declaring variables in C++.

The first character of variable must be a letter or underscore. Blank spaces are not allowed in variable names. Variable may include letters, numbers and underscore (_). Reserved word cannot be used as variable name.

Q.10. Why is it important to assign a data type to a variable?

Variables are assigned data types to specify the amount of memory occupied by that variable. Different data types occupy different amount of memory. Using proper data type for a variable can save the memory.

Q.11. Why is C++ known as strongly typed language?

C++ is a strongly typed language. It means that a variable must always be declared before it can be used in a program. The compiler gives an error if an undeclared variable is used in a program.

Q.12. Which things are specified in variable declaration?

Variable Name: It refers to the memory location of the variable.
Variable Data Type: It indicates the type of data that can be stored in variable.

Q.13. Explain the difference between a constant and a variable.

Variables and constants are temporary memory locations with a name and data type. The value stored in a variable can be changed during the execution of the program. The values stored in constants cannot change.

Q.14. What is a constant? Also, give an example of one.

A constant is an unchanging value. For instance: 12, 1.2E6, or 'a'. A symbolic constant is a variable defined with the const keyword. For instance:

```
const double PI = 3.14159;
```

Q.15. List out different types of constant?

There are two types of constants in C++ language. These are literal constant and symbolic constant.

Q.16. What is literal constant?

A literal constant is a value typed directly in a program. It appears in the program wherever it is needed. Different types of literal constants are Integer constant, Floating point constant, Character constant and String constant.

Q.17. What are Integer constants?

Integer constants are numeric values without fraction or decimal point. Integer constants represent values that are counted. Both positive and negative integer constant are used. The minus sign - is used for negative integer constants. If no sign is used, the value is positive by default. Some examples of integer constants are 87, 45, -10 and -5.

Q.18. Why "L" is used with integer constants?

The use of L at the end of integer indicates that it is a long integer like 65000L.

Q.19. What are floating point constants?

Floating point constants are numeric values with fraction or decimal point. Floating point constants represent values that are measured. Both positive and negative floating point constants are used. The minus sign – is used for negative floating point constants. If no sign is used, the value is positive by default.

Q.20. Why "F" or "f" is used with floating point constants?

The use of F or f at the end of a real value indicates that the value is float. If F or f is not used, the value is considered as double.

Q.21. Why "L" is used with floating point constants?

The use of L at the end of real value indicates that it is a long double like 52000.33L.

Q.22. What are character constants? Give some examples.

Any character written within apostrophe is known as character constant. All alphabetic characters, digits and special symbols can be used as character constants. The maximum length of a character constant is 1 character. Some examples of character constants are 'A','9','=' and '$' etc.

Freebooks.pk

Q.23. What are string constants? Give some examples.

A collection of characters is called string. String constant is written in double quotes. It may consist of any alphabetic characters, digits and special symbols. Some examples of sting constants are "Pakistan", "123" and "99-Mall Road, Lahore".

Q.24. What are symbolic constants? Give some examples.

A symbolic constant is a type of constant that represents a memory location. A constant m .t be initialized. Its value cannot be changed after initialization. Symbolic constants can be declared in two ways. These are using const qualifier or using define directive.

Q.25. What is const qualifier?

The const qualifier is used to define a constant. The constant is declared by specifying its name and data type. The constant must be initialized with some value. An example of declaring constant using const qualifier is const int N = 100;

Q.26. What is difference between const qualifier and define directive?

The difference between const qualifier and define directive is that define directive does not specify the data type of the constant.

Q.27. How is characters are stored in memory?

Characters are stored in memory using ASCII code. An ASCII code is associated with each character. For example, ASCII code represents the character A.

Q.28. What are operators?

Operators are the symbols that are used to perform certain operations on data. C++ provides a variety of operators. These include arithmetic operators, relational operators, logical operators, bitwise operators etc.

Q.29. What is arithmetic operator?

Arithmetic operator is a symbol that performs mathematical operation on data. C++ language provides many arithmetic operators. C++ language.provides many arithmetic operators. These are +, -, *, / and %.

Q.30. What does the symbol = do in C++?

The symbol = is called assignment operator. It is used to assign values to variables in C++.

Q.31. What is the difference between z='a' and z=a?

The expression z='a' will assign the character 'a' to variable z. The expression z=a will assign the value of variable a to variable z.

Q.32. What is the difference between s="word" and s=word?

The expression s="word" will assign the string "word" to variable s. The expression s=word will assign the value of variable word to variable s.

Q.33. What is the difference between preincrement and postincrement operators ++x and x++, respectively?

Preincrement operator increments the value of x then uses the new value of x in the statement. Postincrement operator uses the current value of x in the statement and then performs the increment.

Q.34. What is difference between unary and binary operators.

The unary operators work with one operand such as ++x. The binary operators work with two operands such as x + y.

Q.35. List out and explain two forms of increment operator?

Increment operator can be used in two forms of prefix form and postfix form. In prefix form, the increment operator is written before variable like ++y; It increments value of variable by 1. In postfix form, increment operator is written after the variable like y++; It increments the value of variable y by 1

Freebooks.pk

Q.36. List out and explain two forms of decrement operator?

Decrement operator can be used in two forms of prefix form and postfix form. In prefix form, the decrement operator is written before variable like --y; It decrements the value of variable by 1. In postfix form, the decrement operator is written after variable like y--; It decrements the value of variable **y** by 1.

Q.37. What is assignment operator?

An operator that is used in assignment statement to assign a value to a variable is called assignment operator. The symbol = is used as assignment operator in C++.

Q.38. Define operator precedence.

The order in which different types of operators in an expression are evaluated is known as operator precedence. It is also known as hierarchy of operators.

Q.39. Define associativity of operators.

The order in which operators of same precedence are evaluated is known as operator associativity. These operators are evaluated from left-to-right or right-to-left in an expression.

Q.40. What is type casting?

Converting the data type of a value during execution is known as type casting. Type casting can be performed in two ways. These are implicit type casting and explicit type casting.

Q.41. What is implicit type casting? Why is implicit type casting used?

A type of type casting that is performed automatically by the C++ compiler is called implicit type casting. The operands in arithmetic operation must be of similar types. If the data types of operands are different, the operand with lower data type is converted into higher data type.

Q.42. Can you represent each of the following values using a data type? If you can which primitive data type would you choose and why? If you cannot why not?

4	int
23.5	double or float
'0'	char
7.8E-03	double or float
' 876'	invalid, character variable contains only 1 character
2147483650	long
'+'	char
1.986E56	double (not float)
'8'	char
-32.4789877689	double (not float)
'hello'	invalid, character variable contains only 1 character

Q.43. What data type would you use to represent the following items?

a. Number of children at your school
b. A letter grade on an exam
c. The average marks of your class

The data types used to represent the above items are as follows:

a. Number of children at your school	int
b. A letter grade on an exam	char
c. The average marks of your class	float

Q.44. Which of the following are valid variable names in C++?

income	Valid.
total marks	Invalid. A variable cannot have spaces in it.
double	Invalid. A keyword cannot be used as variable name.
averge-score	Invalid. A variable cannot have hyphen in it.
room#	Invalid. A variable cannot have hash sign in it.
_area	Valid.
no_of_students	Valid.

Freebooks.pk

long	Invalid. A keyword cannot be used as variable name.
Item	Valid.
MAX_SPEED	Valid
My.school	Invalid. A variable cannot have period "." in it.
2ndTry	Invalide. A variable cannot start with a digit.
$cost	Invalid. The special symbol $ cannot be used in variable name.

Q.45. Answer the following:

a. Write a C++ constant declaration to give PI a value 3.142. Ans: const float PI = 3.142;

b. What is the difference between the literal string "area" and the identifier area?

The literal string "area" is a string constant value. The identifier area is a variable to store a value.

c. Write a declaration and an assignment statement to assign 30 to a variable named length.

int length = 30;

d. Declare two integer variables in one declaration statement. The two integer variables should be named testNumber and testScore.

int testNumber, testScore;

e. Declare int integer i initialized to 10.

int i = 10;

f. Add variable a to variable sum and assign the result to variable sum.

sum = sum + a;

Q.46. Find errors in the following statements and correct them.

a.	const int n;	b.	define char ch 'B'
c.	char c = "ABC";	d.	float a b c;
e.	#define msg "Hello World";	f.	const double = 100;
g.	char n[] = Hello World;	h.	int = 35;
i.	long, population = 15000;	j.	float average = 90.15.

Answer:

a. Invalid. Constant must be initialized.
b. Invalid. The sign # is used with define. It does not use data types.
c. Invalid. Character variable can store single character.
d. Invalid. Variables are separated by commas in declaration.
e. Invalid. Define directive is not terminated by semicolon.
f. Invalid. Data types cannot be used as variables.
g. Invalid. String values are written in double quotes.
h. Invalid. Value cannot be assigned to data types.
i. Invalid. Comma cannot be used between data type and variable name.
j. Invalid. Statement must be terminated by semicolon instead of dot.

Q.47. Convert the following numbers exponential form.

a. 126.	b. 556.68	c. 1234.24	d. .421	e. .0321	f. .007869
Answer: a. 1.26e2	b. 5.5668e3	c. 1.23424e3	d.4.21e-1	e.3.21e-2	f. 7.869e-3

Q.48. Which of the following is valid or invalid assignment expression?

a. x=30;	b. a=9=3;	c. 10=x;	d. x=y=10;	e. x%=5;

Answer:

a. Valid	b. Invalid	c. Invalid	d. Valid	e. Valid

Freebooks.pk

Q.49. Suppose w, x, y and z are four float variables and a, b and c are three int variables. Each of the following statements contains one or more violations of rules for forming arithmetic expressions. Rewrite these statements so that it is consistent with rules.

Invalid expression	Valid Expression
z = 4.0 w * y;	z = 4.0 * w * y;
y = yz	y = y * z;
a = 6b4;	a = 6 * b * 4;
c = 3(a+b);	c = 3 * (a + b);
z = 7w + xy;	z = 7 * w + x * y;

Q.50. Convert the following expressions into C++ expressions.

a. x2+3x-4
b. (x+y)z
c. x+3y / 2x-y
d. 1 / x2+x+3
e. x+y / 7
f. 2bc3
g. $x = \frac{3y}{5-z}$
h. $z = \text{area}\sqrt{area}$
i. $\frac{x + 32}{y - 32} - (x - 2y)$
j. $res = \frac{3ijk + k^9}{7ik - 5\sqrt{j} + k}$

Answer:

a. x*x + 3*x -4
b. (x + y) * x
c. (x + 3*y) / (2*x - y)
d. 1/(x*x + x + 3)
e. (x + y) / 7
f. 2 * b * c * c * c
g. x = (3 * y)/(5 – z)
h. z = area *sqrt(area);
i. (x + 32) / (y – 32) – (x – 2 * y)
j. res = (3 * I * j * k + pow (k, 9)) / (7 * I * k – 5 * sqrt(j + k))

Q.51. For the following questions, assume the following declarations:

```
int i = 1;
int j = 2;
int k = 3;
double x = 1.0;
double y = 2.0;
double z = 3.0;
```

a. Parenthesize this expression and write its value: i + j == k;

((i + j) == k); ==> 1

b. Parenthesize this expression and write its value: k - i < y > x;

(((k - i) < y) > x); ==> 0

c. Parenthesize this expression and write its value: z - x + 1 != j + k - 2 * i;

(((z - x) + 1) != ((j + k) - (2 * i))); ==> 0

Q.52. What is the output of the following expressions?

a. cout<<(1+8/2+((1*4)+(5*4))/4);
b. cout<<((1+1+1+1)/2+(1+1+1)/3);
c. cout<<(5*5+5/5+6);
d. cout<<(((3+4)+(4*7))/5);
e. cout<<((3*6*7*2)+12/2);
f. cout<<5-3*4%(6-1)
g. cout << (8 * 4 * 2 + 6) / 2 + 4

Answer:

a. 11 b. 3 c. 32 d. 12 e. 258
f. 3 g. 39

Freebooks.pk

Q.53. What is the output of the following code?

a.
```
int x=20,y=35;
x=y++ + x++;
y= ++y + ++x;
cout<<x<<y;
```
Answer: 5794

b.
```
int x=10, y=15;
x = x++;
y = ++y;
cout<<x<<","<<y;
```
Answer. 11,16

c.
```
int a = 10;
a++;
cout << a << endl;
```
Answer: 11

d.
```
int a = 10;
cout << a++ << endl;
```
Answer: 10

e.
```
int n = 10;
int x = 0;
x = n--;
cout << x << endl;
```
Answer: 10

f.
```
int i = 5, j = 6, k = 7, n = 3;
cout << i + j * k - k % n << endl;
cout << i / n << endl;
```
Answer:
46
1

g.
```
int n;
float x = 3.8;
n = int(x);
cout << "n = " << n << endl;
```
Answer: n=3

h.
```
int a = 6;
cout << a;
a = a + 3;
cout << a;
a -= 5;
cout << a;
a++;
cout << a;
```
Answer: 6945

Q.54. What is the output of this snippet of code:

```
int i = 5;
cout << i << endl;
cout << (i++) << endl;
cout << (++i) << endl;
```

Answer:
5
5
7

Q.55. Consider the following code and answer the questions.

```
int a=3,b=4;
++a* = b++;
cout<<a<<endl;
```

a. What is the output the above code?

Answer: 16

b. What is the output if we replace "++a*=b++" by "++a*=++b"?

Answer: 20

c. The elementary operations executed by "a = b++" can be written as "a=b; b=b+1;". Write down the elementary operations executed by "++a*=++b" in correct order using only three elementary operations: +, * and =.

Freebooks.pk

Answer:

```
a=a+1;
b=b+1;
a=a*b;
```

or

```
b=b+1;
a=a+1;
a=a*b;
```

Q.56. What value does a have after the following code?

```
int a, b;
a = 5;
a++;
a = a + 1;
b = a;
b++;
a = b;
```

Answer: 8

Q.57. Write four different program statements that increment the value of an integer variable sum.

Answer:

```
sum = sum + 1;
sum += 1;
sum++;
++sum;
```

Q.58. For each of the following, state whether or not the statement is valid. If it is valid, state the values for a, b, and c. Assume each is preceded by the following code:

```
int a, b, c;
a= 2;
b = 3;
c = 5;
```

a.	++a;	b.	abc;	c.	a++;
d.	b+=a;	e.	b+=a*c;	f.	a++b;
g.	a+=2; b=2/2; c=a*b;	h.	++b++;	i.	b = a++ + b++;

Answer:

a.	Valid, 3, 3, 5	b.	Invalid	c.	Valid, 3,3,5
d.	Valid, 2,5,5	e.	Valid, 2,13,5	f.	Invalid
g.	4, 1, 4	h.	Invalid	i.	Valid, 3, 6, 5

Q.59. Calculate the value of n after each of the following C++ statements is executed.

a. int n = 22/8;
b. int n = 99/8 + 21/11*7;
c. double a = 4, b = 3, p = 8, q = 2;
double n = q/a - p/b;
d. int a = 6, b = 3, p = 8, q = 2;
double n = p/a + q/b;
e. int a = 3.3, b = 2.7, p = 8.7 , q = 5.4;
double n = p/b + q/a;
f. int n = 11 + 7/4 + 98%6*3;
g. int n = 11/31%8*5 - 12;

h. double p = 8;
int q = 5;
int n = q*p + 3.0*p*p - (q%3)*p*p*p;

i. double p = 5;
int q = 9;
int n = (q%2)*p + 6.0*p*(q%3) + (q%4)*p*p*p;

j. int n = 3, a = 6, b = 2;
n *= a + b;
a--;
n %= a;

k. int a, n = 5;
++n;
a = --n;
n = n-a;

Answer:

a. 2	b. 19	c. -2.16667	d. 1	e. 5	f. 18
g. -12	h. -792	i. 130	j. 4	k. 0	

Q.60. You want to divide 21 by 4.

a. How do you write the expression in C++ if you want the result to be the floating- point value 5.25?

21 / 4.0 or 21.0 / 4.0, or 21.0 / 4

b. How do you write the expression in C++ if you want only the integer quotient?

21 / 4

c. How do you write the expression in C++ if you want only the remainder?

21 % 4

Q.61. Suppose we have the following variables in program:

int a, b, c, d, e;
a = 10
b = 20
c = 15;
d = 8;
e = 40;

Compute the following arithmetic expressions:

1. (a + b / (c – 5)) / ((d + 7) / (e – 37) % 3
2. a + b / c – 5 / d + 7 / e – 37 % 3
3. a * (b * b) – (c * b) + d

Answer:

1. The evaluation of the expression will occur as follows:

```
(a + b / (c – 5)) / ((d + 7) / (e – 37) % 3)
  5  4  1   8    2   6   3   7
```

← Order of evaluation

The answer of the expression is 6.

2. The evaluation of the expression will occur as follows:

```
a + b / c – 5 / d + 7 / e – 37 % 3
  5  1  6  2  7  3  8   4
```

← Order of evaluation

The result of the expression is 10.

3. The evaluation of the expression will occur as follows:

```
a * (b * b) – (c * b) + d
  3  1   4  2   5
```

← Order of evaluation

The result of the expression is 3708.

Freebooks.pk

Q.62. Answer the following questions:

a. What is the value of y after the following code executes:

```
float y = 3.4 + sqrt (25.0);
```

b. Rewrite the following expression without using the compound assignment operator +=.

```
something += (thing + 2);
```

c. Rewrite the following expression without using the operator ++.

```
another++;
```

d. Rewrite the following expression using a compound assignment operator:

```
result = result / 2;
```

Answer:

a. 8.4
b. something = something + thing + 2;
c. another = another + 1;
d. result /= 2;

Q.63. Which of the following assignment statements are invalid? If invalid, explain why?

a. 3.14 * r = area;
b. c = m / n – (z +n);
c. z = n * y + z (4.2n + y);
d. float = m / n;
e. y = 5 = z;

Answer:

a. Invalid. Arithmetic expression cannot be written on the left side of an assignment statement.
b. Valid.
c. Invalid. Arithmetic operator is missing in 4.2n.
d. Invalid. No value can be assigned to a reserved word.
e. Invalid. Constant cannot be written on the left side of an assignment statement.

Multiple Choice

1. **A memory location with some data that can be changed is called:**
   a. Constant. b. Variable. c. Named constant. d. Symbolic constant.
2. **Variables are created in:**
   a. RAM b. ROM. c. Hard disk. d. Cache.
3. **A memory location with some data that cannot be changed is called:**
   a. Constant. b. Variable. c. Named constant. d. Symbolic constant.
4. **Which of the following is a valid character constant?**
   a. a. b. Variable. c. '6' d.=
5. **Which of the following operators has lowest precedence?**
   a. ! b. + c. = d. = =
6. **Which is /or are not a valid identifier(s) in C++?**
   a. Hi_There b.top 40 c. &UpAnDdOwN d.both b and c
7. **C++ is strongly typed language, it means that:**
   a. Every program must be compiled before execution
   b. Every variable must be declared before it is being used
   c. The variable declaration also defines the variable
   d. Sufficient data types are available to manipulate each type of data
8. **Which is /or are not valid identifier(s) in C++?**
   a. my-Name b. 9littles c. X123Y d. both a and b

Freebooks.pk

9. **a + =b is equivalent to:**
   a. b+=a b. a=+b c. a=a+b d. b=b+a
10. **Which is true about a variable?**
   a. The name and data value can both change.
   b. The name can change, but the data value cannot.
   c. The name cannot change, but the data value can.
   d. The name and the data value both cannot change.
11. **Which is NOT a rule for naming variables?**
   a. Use a descriptive name for the variable.
   b. Start the name of a variable with a letter.
   c. Use nothing but letters, digits, or the underscore character.
   d. All of the above
12. **Variable and constant names can not contain a(n):**
   a. Number. b. Underscore. c. Letter d. Period.
13. **Variable names can not begin with a(n):**
   a. Number. b. Underscore. c. Upper-case letter. d. Lower-case letter.
14. **Which of the following is NOT a valid identifier?**
   a. return b. myInt c. myInteger d. total3
15. **Which of the following are valid variable names**
   a. long b. Integer c. notlongenough d. SuperLong
16. **Which term describes the kind of values that a variable can store?**
   a. Variable Name b. Datatype c. Variabletype d. Variablesize
17. **Which statement is true about data types?**
   a. Each data type has no memory requirements.
   b. Each data type has different memory requirements.
   c. Each data type has the same memory requirements.
   d. None of the above
18. **Which data type is used to store numeric value with no decimal point?**
   a. int b. char c. float d. All
19. **Which is a numeric data type?**
   a. Floating point b. Integer c. Both a and b. d. None
20. **The number of bytes used by int data type in C++ is?**
   a. 2 b. 8 c. 12 d. 16
21. **The number of bytes used by long int data type in C++ is?**
   a. 2 b. 4 c. 12 d. 16
22. **An integer variable can store the value:**
   a. -1.1 b. "123" c. 32898 d. None
23. **The integer, long and short data types are known as:**
   a. Integer data types b. Non-integral data types
   c. float data types d. Non-numeric data types
24. **The data type that can handle decimal places is:**
   a. Long b. float c. char d. Integer
25. **The float, long float and double data types are known as:**
   a. Integer data b. Character data c. Integral data d. Real data
26. **What kinds of numbers are stored in float and double data types?**
   a. Floating points numbers b. single and double numbers
   c. Short integer numbers d. long integer numbers
27. **The number of bytes used by float data type in C++ is:**
   a. 2 b. 4 c. 12 d. 16

Freebooks.pk

28. The number of bytes used by double data type in C++ is:
a. 2 b. 8 c. 12 d. 16

29. What happens when the result of a calculation exceeds the capacity of data type?
a. System error b. Logic error c. Syntax error d. Overflow

30. The exponential notation consists of:
a. Mantissa b. Exponent c. Range d. Both a and b

31. The number of digits after a decimal point is called:
a. Significance b. Precision c. Range d. Scope

32. Which of the following data type is used to store string value?
a. char b. float c. string d. None

33. The number of bytes used by char data type in C++ is:
a. 2 b. 1 c. 12 d. 16

34. Which variable should be used to store the value "I want an A in this exam"
a. char b. int c. float d. character

35. Another way to write the value 3452211903 is:
a. 3.452211903e09 b. 3.452211903e-09 c. 3.452211903x09 d. 3452211903e09

36. The constant 0.15e+6 represents the same value as:
a.150000.0 b. 6.15 c. 0.75 d. 0.21

37. Which of the following statements is NOT legal?
a. char ch='b'; b. char ch='0'; c. char ch=65; d. char ch="cc";

38. Which of the following are required to declare a variable?
a. Continue keyword b. Variable name c. Data type d. Both b & c

39. Which symbol is used to separate each variable while declaring many variables one line?
a. Commas b. Colons c. Pipes d. Semicolons

40. Which is a valid statement for declaring a variable?
a. int marks; b. int a,b,c; c. double salary; d. All

41. Which of the following data types is most appropriate for storing a name?
a. float b. Integer c. char d. None

42. The process of giving a variable its starting value is called:
a. Declaring b. Initializing c. Naming d. None of these

43. Which is a valid statement for initialization of a variable?
a. int n=100; b. int x=50, y=75; c. char grade='a'; d. All

44. Which is NOT a valid statement to initialize a variable?
a. int =100; b. long population=15000;
c. char n[]="Hello word"; d. const int N=100;

45. Which of the following statements is correct?
a. float num1; num2; b. int day, night;
c. int continue = 5.0; d. string black = 'white';

46. Which of the following constant definitions is correct?
a. const float PRICE = 3,500; b. const int LENGTH = 6.5;
c. const float COST = $700.0; d. const int MY_HEIGHT = 60 / 12;

47. Which of the following can be used as character constant:
a. Characters b. Digits c. Special characters d. All

48. Which of the following are valid examples of character constant?
a. 'A' b. '9' c. '$' d. All

49. A character constant must be enclosed in:
a. Quotation marks (").
b. Single quotes (').
c. Exclamation points (!).
d. Pound signs (#).

50. What is the valid range of numbers for int type of data?
a. 0 to 256 b. -32768 to +32767 c. 65536 to +65536 d. No specific range

Freebooks.pk

51. How many bytes are occupied by the statement: float a, b;
a. 1 byte b. 4 c. 8 d. 16

52. The size of a character variable is:
a. 1 byte b. 8 byte c. 16 byte d. None

53. An expression consists of:
a. Operators b. Operand c. Both a and b d. None

54. An expression can be a:
a. Constant. b. Variable.
c. Combination of constants, variables, and arithmetic operators d. All

55. All of the following are valid expressions EXCEPT:
a. Sales – Revenues. b. Mpg, Gallons. c. Pi * Radius d. M/n.

56. An assignment statement will:
a. Perform a calculation. b. Store the results of a calculation.
c. Display the results of a calculation. d. Both a and b.

57. Which of the following is NOT arithmetic operators except
a. + b. - c. % d. >

58. The modulus operator is used for:
a. Exponentiation b. Multiplication. c. Division. d.Integer remainder

59. Which of the following operators works only with integer value?
a. % b. + c. - d. /

60. The expression 10%3 has a value equal to:
a. 1 b. 3 c. 8 d. None

61. The expression 3%5 has a value equal to:
a. 3 b. 5 c. 0 d. None

62. The expression 0%4 has a value equal to:
a. 0 b. 4 c. Both a and b d. None

63. The value of the C++ expression 9 / 5 * 2 is:
a. 3.6 b. 0.9 c. 0 d. 2

64. The value of the C++ expression 5 / 9 * 2 is:
a. 0.27 b. 1.11 c. 0 d. None

65. Given that x is a int variable and num is an float variable containing the value 10.5, what will x contain after execution of the following statement:
x = num + 2;
a. 12.5 b. 10.5 c. 12 d. nothing; a compile-time error occurs

66. Given that x is a float variable containing the value 80.9 and num is an int variable, what will num contain after execution of the statement: num = x + 2.9;
a. 83.8 b. 84.0 c. 83 d. 83.0

67. The value of the C++ expression 13 + 21 % 4 - 2 is:
a. 0 b. 12 c. 16 d. 14

68. Given that x is a float variable and num is an int variable containing the value 38, what will x contain after execution of the following statement:
x = num / 4 + 3.0;
a. 12.5 b. 13 c. 12 d. 12.0

69. Assume that we have the following variable definitions in a program:
int a = 11, b = 4, c = 10, d = 2;
What will be the value computed for the expression (a % b + c - d * 3)?
a. 6 b. 30 c. 33 d. 7

70. If originally x = 5, y = 4, and z = 2, what is the value of the following expression?
y * (x + y) % 3 / z + 10
a. 10 b. 13 c. 9 d. 8

Freebooks.pk

71. What is the output of the following code fragments?

```
double x;
x = 3.0 / 4.0 + 3 + 2 / 5;
cout << x;
```

a. 3.75 b. 4.75 c. 2.75 d. 3.57

72. What is the value of the following expression? (20.0 * (9/5)) + 32.0

a. 52.0 b. 54.0 c. 51.0 d. 50

73. What is the value of the following expression? (20.0 * (9/5.0)) + 32.0

a. 68.0 b. 67.0 c. 78.0 d. 88.0

74. Which assignment statement can be used to store letter A in the char variable someChar?

a. someChar = "A"; b. someChar = 'A'; c. someChar = A; d. a and c

75. Which of the following operators is used to assign a value to a variable?

a. > b. + c. = d. *

76. Which of the following is correct syntax for writing an assignment statement?

a. Variable = right_side b. Constant = right_side c. Constant = variable d. None

77. The left side of an assignment statement holds:

a. Variable. b. Constant. c. Expression. d. Both a and b.

78. The right side of an assignment statement holds:

a. A variable b. An expression c. Both a and b d. None of these

79. Which of the following is valid assignment statement?

a. a = 100; b. c = a + b; c. x = c – d + 10; d. All of these

80. Which of the following operators works with one operand?

a. Unary b. Binary c. Ternary d. None

81. Which of the following operators works with two operands?

a. Unary b. Binary c. Ternary d. None

82. Which of the following statements assigns a value to many variables?

a. Compound assignment b. Lvalue c. Rvalue d. Input statement

83. The statement I += 3 has the same effect as:

a. I = I+3. b. I = 3. c. I-3 = I. d. I3.

84. Which of the following operator is used to increase the value of a variable?

a. ++ b. +- c. >> d. None

85. Which of the following operator is used to decrease the value of a variable?

a. -- b. -+- c. << d. None

86. Which choice has the highest order of precedence when computing an expression?

a. () b. * c. / d. +

87. If x is int and y is float, what promotion will occur in the expression: y = 5.0 + x;

a. float to int b. int to float c. float to double d. No promotion

88. What is the value of x after the following statements?

```
int x, y, z;
y = 10;
z = 3;
x = y * z + 3;
```

a. 12 b. 60 c. 30 d. 33

89. What is the value of x after the following statements?

```
int x = 0;
x = x + 30;
```

a. 0 b. 30 c. 33 d. None

90. What is the value of x after the following statements?

```
float x;
x = 13/4;
```

a. 3.75 b. 4.0 c. 3.0 d. 60

Freebooks.pk

91. What is the value of x after the following statements?

```
int x;
x = 15/4;
```

a. 15 b. 3 c. 4 d. 3.75

92. What is the value of x after the following statements?

```
int x;
x = 17%4;
```

a. 15 b. 4 c. 1 d. 3.75

93. What is the value of the expression? ((1+4+4)/(3*3)-(3+6))

a. –8 b. 9 c. 12 d. 16

94. What is the value of the following expression? ((5+1)/((2+2+2)/2))

a. 2 b. 6 c. 12 d. 9

95. What is the value of the following expression? (1*2*3*4+5)

a. 29 b. 30 c. 33 d. 40

96. What is the value variable Result after the following code has been executed?

```
int result;    Result=14-4*6+12/6;
```

a. 12 b. 2 c. -8 d. 62

97. What is the name for a word that has a specific meaning in C++?

a. Keywords b. Comments c. Token d. Operators

98. Another name for keywords is:

a. Reserved words b. Special words c. Comments d. None

99. What is the output of the following code fragments?

```
int a = 5, b = 3, c = 2, d = 2;
        cout << ++a/3-c*d << endl;
```

a. –2 b. –1 c. 4 d. –4

100. What is the results value and its type of the expression? 11 * 3 - 35 / (11 % 4)

a. 22 b. 32 c. 11 d. 0

101. What is the results value and its type of the expression? 9 % 11 + 13 % 11 + 0 % 11

a. 22 b. 11 c. 12 d. 10

102. What is the results value and its type of the expression?

```
float f = 12.0; int z = 3;
(f / 2 * ( 3 - ++z )) – 5
```

a. 22 b. 11 c. -11.0 d. 12

103. Which of the following expression will yield 0.5?

a. 1 / 2.0 b. (double) 1 / 2 c. 1.0/ 2 d. All

104. -16 % 4 is:

a. 0 b. 4 c. 16 d. -4

105. To add a value 1 to variable y, you write:

a. y += 1; b. y = y + 1; c. y = 1 + y; d. All

106. To add number to sum, you write:

a. sum += number; b. sum = sum + number;
c. sum = Number + sum; d. both a and b

107. What is the output of the following code:

```
double x = 3.3;
int y = (int)x;
cout << "x is " << x << " and y is " << y;
```

a. x is 3.3 and y is 3.0
b. x is 3.0 and y is 3.0
c. x is 3.3 and y is 3
d. x is 3 and y is 3

Freebooks.pk

108. We want to translate the following algebraic expression into a C++ arithmetic expression. Which of the following is the correct form?

$$\frac{1}{c} + \frac{1-a}{1+b}$$

a. 1 / c + 1 - a / 1 + b
b. 1 / c + (1 - a / 1 + b)
c. (1 / c) + (1 - a / 1 + b)
d. 1 / c + (1 - a) / (1 + b)

109. Let the values of the integer variables a and b be 10 and 12 respectively. After the following statement is executed, what will their values be? a += (b -= 2);

a. 20 and 10 b. 10 and 20 c. 10 and 12 d. 12 and 10

110. These are operators that add and subtract one from their operands.

a. ++ and - - b. Plus and minus c. Binary and unary d. None

111. Which of the following is an illegal variable name?

a._customer_num b. jan2009 c. dayOfWeek d. 2dGraph

112. The = operator in C++ language indicates:

a. Assignment b. Subtraction c. equality d. Negation

113. Associativity is either right to left or:

a. Left to right b. Top to bottom c. Back to front d. None

114. Which of the following is used to declare a variable to store real number?

a. Integer data type b. Floating data type c. Character data type d. All

Answers

1. b	2. a	3. a	4. c	5. c	6. d
7. b	8. d	9. c	10. c	11. d	12. d
13. a	14. a	15. c	16. b	17. b	18. a
19. c	20. a	21. b	22. d	23. a	24. b
25. d	26. a	27. b	28. b	29. d	30. d
31. b	32. a	33. b	34. a	35. a	36. a
37. d	38. d	39. a	40. d	41. c	42. b
43. d	44. a	45. b	46. d	47. d	48. d
49. b	50. b	51. c	52. d	53. c	54. d
55. b	56. c	57. d	58. d	59. a	60. a
61. a	62. a	63. d	64. c	65. c	66. c
67. b	68. d	69. d	70. a	71. a	72. a
73. a	74. b	75. c	76. a	77. a	78. b
79. d	80. a	81. b	82. a	83. a	84. a
85. a	86. a	87. b	88. d	89. b	90. c
91. b	92. c	93. a	94. a	95. a	96. c
97. a	98. a	99. a	100. a	101. b	102. c
103. d	104. a	105. d	106. d	107. c	108. d
109. a	110. a	111. d	112. a	113. a	114. b

Fill in the Blanks

1. _____ are the names used to represent variables, constants, types, functions and label s in C++ programs.
2. The header file_____ contains functions for standard function input/output operations in C++ language.
3. C++ is a _____ language because it can differentiate uppercase and lowercase letters.

Freebooks.pk

4. Keywords are always written in _____.
5. _____ are named memory location that are used to store programs input data and its computational results during program execution.
6. The variables are created in _____.
7. C++ is a _____ language because all variables must be declared before being used.
8. Specifying the name and type of variable in C++ language is called variable _____.
9. A process of assigning a value to a variable at the time of declaration is called variable _____.
10. When many variables of same data type are declared on a single line in C++, _____ is used to separate each variable.
11. The_____ set aside some memory space when a variable is declared.
12. First character of a variable must be a _____in C++ language.
13. C++ variable name can be up to _____ characters long.
14. A _____ is a quantity whose value cannot be changed during program execution.
15. _____ can be used to define constant macro.
16. C++ has _____ types of constant.
17. _____ constant is a single alphabet, a single digit or a single symbol enclosed within apostrophes.
18. The maximum length of a character constant is _____.
19. The _____ defines a set of values and set of operations on those values.
20. _____ data type is used to represent integers.
21. Numeric value with no decimal point or fraction is called _____ data.
22. 0,1,2 and -23 are examples of _____data type.
23. Integer variable can be singed or _____.
24. The float type data takes_____ bytes in memory.
25. double type data takes _____ bytes in memory .
26. long double type data takes_____ bytes in memory.
27. int stands for _____.
28. _____ are the numbers that have a fractional part.
29. The number 5.0, 6.58 are examples of _____numbers.
30. Exponential notation consists of two parts: mantissa & _____.
31. The number 245634 in exponential notation would be represented as_____.
32. The exponent is _____ if the number is smaller than 1.
33. The data type_____ is used to represent a letter, number or punctuation mark.
34. A char type variable occupies _____ byte in memory.
35. _____ are symbols that are used to perform certain operations on data.
36. _____ operators are the symbols used to perform mathematical operations.
37. Modulus operator is also called_____ operator.
38. The _____ operator returns the quotient.
39. % is an arithmetic operator called _____.
40. The result of the expression 10 % 13 is equal to _____.
41. The result of the expression 8 % 3 is equal to _____.
42. A statement that assigns a value to a variable is called _____.
43. The equal sign (=) is treated as an operator in C++ language called _____ operator
44. The name of variable is written on the _____ side of assignment operator
45. The _____ can be a variable, a constant or arithmetic, relation or logical expression.
46. In the expression a+b, + is the operator while a and b are_____.
47. The _____increased the value of its operand by one.
48. Increment operator is denoted by the symbol _____.
49. When ++ comes before its operand, it is called _____ increment.
50. When the ++ after its operand, it is called_____ increment.
51. The _____ decrease the value of its operand by one
52. _____ denoted by the symbol - -.

Freebooks.pk

53. The operators that combine assignment operator with arithmetic operators are called _____
54. The two statements - - Q; P = Q; can be written in one statement as _____.
55. The statement P=P+1; can be written as _____ using increment operator
56. In _____ form, the statement p=q++ first assigns the value of q to p and then increments the value of q by 1.
57. The order in which different types of operators in an expression are evaluated is called _____ or hierarchy of operators.
58. The precedence of addition operator is _____ than multiplication operator
59. The result of the expression 10 + (1 % 3 * 13) /13 is equal to _____.
60. The binary operators * and / have _____ precedence
61. Modulus operator in C++ only works with _____ data type
62. The operator's ++ and - - are called _____operators
63. A _____ expression is one in which operands are of different data type.
64. The answer of the expression 7.0/2.0 is_____.
65. The answer of the expression 7/2 is_____.
66. The answer of the expression 198.0/100.0 is_____.
67. The answers of the expression 198/100 is_____.
68. In exponential notation, the number 245634 is represented as_____.
69. The number 0.00524 will be resented in computer as_____.

Answers

1. Identifier	2. iostream.h	3. Case sensitive
4. Lower case	5. Variables	6. Memory (RAM)
7. Strongly type	8. Declaration	9. Initialization
10. commas	11. compiler	12. letter
13. 31	14. constant	15. define directive
16. Two	17. Character	18. 1 character
19. data type	20. int	21. int
22. int	23. unsigned	24. four
25. eight	26. ten	27. integer
28. float point numbers	29. floating point	30. exponent
31. $2.45634*10^5$	32. negative	33. char
34. 1	35. operators	36. arithmetic
37. remainder	38. division	39. modulus operator
40. 10	41. 2	42. Assignment statement
43. assignment	44. left	45. expression
46. operand	47. increment operator	48. ++
49. prefix	50. postfix	51. decrement
52. decrement operator	53. compound assignment operator	54. P = - - Q;
55. P++;	56. postfix	57. operator's precedence
58. low	59. 11	60. same
61. int	62. unary	63. mixed-type
64. 3.5	65. 3	66. 1.98
67. 1	68. $2.45634X10^5$	69. 5.24E-3

True/ False

1. In C++, Identifier are names of thing.
2. Reserve words cannot be used as identifiers in a program.
3. _b_c is a valid identifier.
4. 7th_inning is a valid identifier.

Freebooks.pk

5. Variable names may begin with a number.
6. A good name for a variable representing the cost of an item is x.
7. There are 10 bits in the unit of storage known as a byte:
8. All items of data must be stored in a single byte of memory.
9. If 3 variables are read using 1 input statement, the values must be entered on the same line.
10. In C++, the modulus operator (%) can be used with either integers or doubles:
11. A variable of type bool can never be assigned an integer value other than 0 or 1:
12. The modulus or remainder operator (%) works equally well with integers and doubles.
13. Every variable has a name, data value and data type.
14. The float data type is used in C++ to store large real values.
15. Integer values range from -32767 to 32768.
16. Char data type takes 1 byte in the memory.
17. The double data type takes 8 bytes in memory.
18. Both long int and unsigned long int take 4 bytes in memory.
19. The name and value of variable can change while the program is running.
20. String data and numeric data are same and can be input in a program the same way.
21. Numeric data cannot include symbols such as percent sign or commas.
22. The value 0. 103301 has a precision of 6 digits and range of -3.
23. float data type is more suitable than double if the value requires more precision.
24. char data type is used to store character values.
25. Type of a variable cannot be changed during execution but its value can be changed.
26. A real value variable can be declared by the keyword Real.
27. Keywords cannot be used as variable names.
28. A quantity that cannot be changed during program execution is called constant.
29. The constants must start with a letter but not variable may not start with a letter. .
30. Constants are used to make the program easier to read and maintain.
31. Literal constant and symbolic constant are two types of constants used in C++.
32. Special symbols can also be used as literal constants.
33. Symbolic constants represent a value that is frequently used in program.
34. The define directive is never terminated with a semicolon.
35. Variables, constants and operators are combined to form expressions.
36. Operator is the value on which some operation is performed.
37. A symbol that performs some operation on operand is called operator.
38. Arithmetic operator is the symbol used to perform some logical operations on data.
39. Modulus (!%) is the operator that gives remainder of division.
40. Modulus operator cannot work with float values.
41. Result of the expression 3%10 is 10.
42. If x is a type double variable and n is of type int, the following assignment statements are equivalent: x = n; x = (double)n;
43. The name of the variable is written on the left side of assignment operator.
44. The equal sign (=) is used both as comparison operator and assignment operator.
45. An operand written on the right side of = operator is called lvalue.
46. A type of operator that works with one statement is called unary operator.
47. '+' & '++' are two binary operators.
48. The operator that works with three operands is called ternary operator.
49. Conditional operator is a binary operator.
50. An arithmetic operator always gives result in a logical output.
51. The statement i = i+13 shows a compound assignment statement.
52. The operator that increases the value of a variable by 1 is called increment operator.
53. The decrement operator cannot be used with constants.
54. The operator used to decrease the value of a variable by 1 is called Decrease variable.
55. A variable is written after the decrement operator in postfix form.
56. In postfix form, the statement i = k- -; is equivalent to k- -; i = k; .

Freebooks.pk

57. The order in which different operators in an expression are evaluated is called associativity.
58. The value of the expression x + y * z * z is always same as the value of x + ((y * z) * z).
59. The programmer can change the operator precedence by using parentheses.
60. The exponentiation operation has the highest precedence.
61. The operators with higher precedence are evaluated before the operators with lower precedence in a single expression.
62. The division operator is evaluated before the addition operator in an expression.
63. An expression that evaluates to zero is considered true.
64. An expression that evaluates to a non-zero value is considered false.
65. If a=4; and b=3;,then after the statement a=b; the value of b is still 3.
66. In a mixed expression, all operands are converted to floating-point numbers.
67. Suppose a=5, after the statement b=a++; execute b is 5 and a is 6.
68. All variables must be declared before they can be used.
69. C++ does not automatically initialize variables.
70. Every variable has a name, a value, a data type and a size.
71. When a new value is assigned to a variable, the old value is destroyed.
72. You can use the cast operator to explicitly convert values from one data type to another.
73. The arithmetic compound assignment operator has the form op=, where op is any arithmetic operator.
74. Outputting or accessing the value of a variable in an expression does not destroy or modify the contents of the variable.
75. Prompt lines are executable statements that tell the user what to do.
76. Identifiers Num and num refer to the same variable.
77. Identifiers are case-sensitive.
78. In C++, every expression ends with a semi-colon.
79. A variable is an expression.
80. A variable is a condition.
81. The expressions (a % b) and (a - (a/b) * b) are equivalent.
82. Suppose a mixed-mode arithmetic expression consists of variables of types double and int. The type of the expression becomes a double.
83. In C++, it is illegal to mix character data with numeric data in arithmetic expressions.
84. There can be multiple declarations of variables with the same name, but only one memory location is allocated

Answers

1. T	2. T	3. T	4. F	5. F	6. F
7. F	8. F	9. F	10. F	11. F	12. F
13. T	14. F	15. F	16. T	17. T	18. T
19. F	20. F	21. T	22. F	23. F	24. T
25. T	26. F	27. T	28. T	29. F	30. T
31. T	32. T	33. T	34. T	35. T	36. F
37. T	38. F	39. F	40. T	41. F	42. T
43. T	44. F	45. F	46. F	47. F	48. T
49. F	50. F	51. F	52. T	53. T	54. F
55. F	56. F	57. F	58. T	59. T	60. T
61. T	62. T	63. F	64. F	65. T	66. F
67. T	68. T	69. T	70. T	71. T	72. T
73. T	74. T	75. T	76. F	77. T	78. F
79. T	80. T	81. F	82. T	83. F	84. F

Freebooks.pk

CHAPTER 4

INPUT AND OUTPUT

Chapter Overview

4.1 Input and Output
4.2 Standard Output
4.3 Escape Sequences
4.4 C++ Manipulators
- 4.4.1 'endl' Manipulator
- 4.4.2 'setw' Manipulator
- 4.4.3 'setprecision' Manipulator
- 4.4.4 'fixed' Manipulator
- 4.4.5 'showpoint' Manipulator
- 4.4.6 'setfill' Manipulator

4.5 Standard Input

Programming Exercise
Exercise Questions
Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

4.1 Input and Output

The process of giving something to computer is known as **input**. The input is mostly given by keyboard. A program may need certain inputs from the user for working properly. The term **standard input** refers to the input via keyboard.

The process of getting something from computer is known as **output**. The output is mostly displayed on monitor. The term **standard output** refers to the output displayed on monitor. The result of a program is the output of that program.

C++ uses different streams to perform input and output operations. A **stream** can be defined as flow of data. It is an object that is used by a program to insert or extract characters. The standard C++ library includes the header file **iostream.h**. It contains the declarations of all standard input and output stream objects. The header file **iostream.h** must be included in a program that uses any input or output statement.

4.2 Standard Output

By default, the term **standard output** refers to the output displayed on monitor. C++ uses the **cout** stream object to display standard output. The word 'cout' stands for **console output**. The **cout** object is used with insertion operator (<<). The message or the value of variable followed by the insertion operator is displayed on the screen.

Syntax

The syntax of using **cout** object is as follows:

```
cout<< Variable/Constant/Expression;
```

cout It is the name of the object used to display standard output.

<< It is known as insertion operator or put to operator. It sends the output to cout object. The cout object then sends the output to the screen. The left side of << operator must be an output stream like cout. The right side of << operator must be an expression or manipulator. One statement can use multiple insertion operators to display the values of multiple variables or constants etc.

Variable/Constant/Expression It is the variable, constant or expression whose value is displayed on the screen.

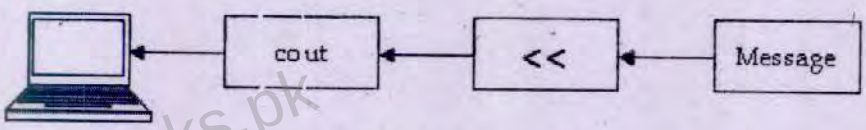


Figure 4.1: Working of 'cout' object

Examples

```
cout<<"Pakistan Zindabad";
```

The above line will display "Pakistan Zindabad" on the screen. The string constant is always enclosed in double quotes.

```
cout<<720;
```

The above line will display 720 on the screen. The numeric constant is not enclosed in quotation marks.

```
cout<<a;
```

The above line will display the value of variable **a** on the screen. The variables are not enclosed in quotation marks.

```
cout<<"The value of N is "<<N;
```

The above line will display "The value of of N is " along with the value of variable N on the screen. The insertion operator is used for twice to concatenate the string value with the value of N.

Program 4.1

Write a program that displays a message and values of integer and character variables.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int n = 10;
  char ch = '*';
  cout<<"Testing output...";
  cout<<n;
  cout<<ch;
  getch();
}
```

Output:
Testing output...10*

How above Program Works

The above program declares and initializes two variables **n** and **ch**. It displays message "Testing output..." on the screen. It then displays the values of variables **n** and **ch**. The output appears on the same line. The output of second statement starts where the output of first statement ends.

Program 4.2

Write a program that adds two floating point numbers and shows the sum on screen.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  float var1, var2, res;
  var1 = 24.27;
  var2 = 41.50;
  res = var1 + var2;
  cout<<var1<<" +  "<<var2<<" = "<<res;
  getch();
}
```

Output:
24.27 + 41.50 = 65.770004

Program 4.3

Write a program to calculate and print the area of square with given height and width.

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
void main()
{
  clrscr();
  int height, width, area;
  height = 5;
  width = 4;
  area = height * width;
  cout<<"Area of Square = "<<area;
  getch();
}
```

Output:
Area of Square = 20

Working of above Program

The first line declares three integer variables. The second and third lines assign values to the variables **height** and **width**. The next line uses an arithmetic expression to calculate the area of square and stores the result in variable **area**. The last line displays the result on screen.

4.3 Escape Sequences

Escape sequences are special characters used in control string to modify the format of output. These characters are not displayed in the output. These characters are used in combination with backslash "\". The backslash is called **escape character**. Different escape sequences used in C language are as follows:

Escape Sequence	Purpose
\a	Alarm
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\'	Single quote
\"	Double quote

Table 4.1: Escape sequences

\a

This escape sequence is used to play beep during execution. For example:

```
cout<<"Hello \aWorld"; will display
Hello World
```

After displaying "Hello", the computer will play beep and then print "World".

\b

This escape sequence is used to insert backspace in the output. For example:

```
cout<<"Hello\bWorld"; will display
HellWorld
```

First of all, "Hello" is printed but **\b** deletes 'o'. Then "World" is printed. So "HellWorld" is displayed on the screen.

\f

This escape sequence is used to insert a blank paper in the printed output. It is called **form feed**. For example:

```
cout<<"Hello\fWorld"; will display
```

Freebooks.pk

```
HelloWorld
```

After printing "Hello", the computer will include a blank paper and then print the remaining output. It is used during printing.

\n

This escape sequence is used to insert new line in output. For example:

```
cout<<"Hello\nWorld"; will display
Hello
World
```

First of all, "Hello" is printed and "\n" shifts the cursor to next line. Then "World" is printed. The output is displayed on two lines.

\r

This escape sequence is used to move the cursor at the beginning of current line. For example:

```
cout<<"Hello\rWorld"; will display
World
```

First of all, "Hello" is printed but \r moves the cursor at the beginning of current line. After this, "World" is printed that overwrites "Hello". So only "World" is displayed on screen.

\t

This escape sequence is used to insert a **TAB** in the output. For example:

```
cout<<"Hello\tWorld"); will display
Hello	World
```

First of all, "Hello" is printed and \t inserts a TAB. Then "World" is printed. So it displays "Hello World" on the screen.

\'

This escape sequence is used to display single quotes in the output. For example:

```
cout<<" \'Hello World\' "; will display
'Hello World'
```

\"

This escape sequence is used to display double quotes in the output. For example:

```
cout<<" \"Hello World\" "; will display
"Hello World"
```

****

This escape sequence is used to display backslash in the output. For example:

```
cout<<"C:\\ "; will display
C:\
```

Program 4.4

Write a program to display the following output using single **cout** statement.

```
*
* *
* * *
* * * *
```

Freebooks.pk

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  cout<<"*\n**\n***\n****";
  getch();
}
```

4.4 C++ Manipulators

C++ manipulators are used to format the output in different styles. The manipulators are the most common way to control output formating.

Some important manipulators in C++ are as follows:

- endl
- setprecision
- showpoint
- setw
- setfill
- fixed

The **endl** manipulator can be used by including **iostream.h** file in the program and setw, setprecision and setfill manipulators can be used by including **iomanip.h** file.

4.4.1 'endl' Manipulator

The word 'endl' stands for **end of line**. The **endl** manipulator is used to move cursor to the beginning of next line. It works similar to '\n' escape sequence. It requires no parameter.

Example

```
cout<<"Hello"<<endl<<"World";
```

The above line will display the following output:

```
Hello
World
```

The **endl** manipulator is not used as part of string. It is written with separate insertioin operator (<<) without quotation marks. The above example first displays 'Hello'. The 'endl' manipulator displays a new line and then 'World' is displayed on the next line.

4.4.2 'setw' Manipulator

The word 'setw' stands for **set width**. The **setw** manipulator is used to display the value of an expression in specified columns. The value of expression can be string or number. If the value of expression is less than specified columns, the additional columns are left blank from left side. The output automatically uses the required columns if output is larger than the specified columns. The use of **setw** has no effect on the output in this case.

The **setw** manipulator is applied only to the value that is inserted after it. The output is right justified by default. The **setw** manipulator is part of **iomanip.h**.

Syntax

The syntax of 'setw' manipulator is as follows:

```
setw(n)
```

The **n** indicates the number of columns in which the value is to be displayed. It can be an unsigned positive integer constant, variable or expression.

Freebooks.pk

Example

Suppose that the value of **a** is 33 and the value of **b** is 7132. represents the space.

Statement	Output
cout << setw(4) << a<< setw(5) <<b<< setw(4) << "Hi";	33 7132 Hi
cout << setw(2) << a << setw(4) << b << setw(2) << "Hi";	337132Hi
cout << setw(6) << a << setw(5) << b << setw(3) << "Hi";	33 7134 Hi
cout << setw(1) << a << setw(5) << b << setw(1) << "Hi";	33 7134Hi

Program 4.5

Write a program that explains the use of **setw** manipulator.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  int n = 3928;
  double d = 91.5;
  char str[] = "OOP using C++";
  cout<<"("<<setw(5)<<n<<")"<<endl;
  cout<<"("<<setw(8)<<d<<")"<<endl;
  cout<<"("<<setw(16)<<str<<")"<<endl;
  getch();
}
```

Column Numbers

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	3	9	2	8											
				9	1	.	5								
			O	O	P		u	s	i	n	g		C	+	+

4.4.3 'setprecision' Manipulator

The 'setprecision' manipulator is used to set the number of digits to be displayed after decimal point. It is applied to all subsequent floating point numbers written to that output stream. The value is rounded with the use of this manipulator.

Syntax

The syntax of 'setprecision' manipulator is as follows:

setprecision(n)

The **n** indicates the number of digits displayed after the decimal point. It can be an unsigned positive integer constant, variable or expression.

Example

Value of x	Statement	Output
310.0	cout << setw(10) << setprecision(2) << x;	310.00
310.0	cout << setw(10) << setprecision(5) << x;	310.00000
310.0	cout << setw(7) << setprecision(5) << x;	310.00000
4.827	cout << setw(6) << setprecision(2) << x;	4.83

Freebooks.pk

Program 4.6

Write a program that dispalys the values of different variables using **setprecision** manipulator.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  clrscr();
  double r, n1 = 132.364, n2 = 26.91;
  r = n1 / n2;
  cout<<r<<endl;
  cout<<setprecision(5)<<r<<endl;
  cout<<setprecision(4)<<r<<endl;
  cout<<setprecision(3)<<r<<endl;
  cout<<setprecision(2)<<r<<endl;
  cout<<setprecision(1)<<r<<endl;
  getch();
}
```

Output:
4.918766
4.91877
4.9188
4.919
4.92
4.9

4.4.4 'fixed' Manipulator

The 'fixed' manipulator is used to further control the output of floating-point numbers. It displays the floating-point numbers in a fixed decimal format. The following statement sets the output of floating-point numbers in a fixed decimal format:

```
cout<<fixed;
```

All floating-point number after the above statement executes are displayed in the fixed decimal format until the manipulator fixed is disabled. It can be disabled by using the stream member function **unsetf** as follows:

```
cout.unsetf(ios:fixed);
```

4.4.5 'showpoint' Manipulator

The decimal part of a floating-point number is not displayed if the decimal part of the number is zero and the user displays the number using fixed decimal format. The showpoint manipulator is used to displays the decimal part even if the decimal part is zero.

The following statement setes the output numbers with a decimal point and trailing zeros on the standard input device.

```
cout<<showpoint;
```

Program 4.7

Write a program that displays the values of different variables using **showpoint** manipulator.

```
#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
void main()
{
  clrscr();
```

double ..

```
x = 15.674;
y = 235.73;
z = 9525.9864;
cout.setf(ios::fixed, ios::floatfield);
cout.setf(ios::showpoint);
cout<<setprecision(2)<<"setprecision(2)"<<endl;
cout<<"x = "<<x<<endl;
cout<<"y = "<<y<<endl;
cout<<"z = "<<z<<endl;
cout<<setprecision(3)<<"setprecision(3)"<<endl;
cout<<"x = "<<x<<endl;
cout<<"y = "<<y<<endl;
cout<<"z = "<<z<<endl;
cout<<setprecision(4)<<"setprecision(4)"<<endl;
cout<<"x = "<<x<<endl;
cout<<"y = "<<y<<endl;
cout<<"z = "<<z<<endl;
cout<<setprecision(3)<<x<<" ";
cout<<setprecision(2)<<y<<" "<<setprecision(4)<<z<<endl;
getch();
}
```

Output:

```
setprecision(2)
x = 15.67
y = 235.73
z = 9525.99
setprecision(3)
x = 15.674
y = 235.730
z = 9525.986
setprecision(4)
x = 15.6740
y = 235.7300
z = 9525.9864
15.674  235.730  9525.986
```

4.4.6 'setfill' Manipulator

The setfill manipulator is used to replace the leading or trailing blanks in the output by the specified character. It requires one parameter to specify the fill character. The parameter can be a character constant, character variable or an integer that represents the fill character in ASCII system. The manipulator must be used with fixed width output.

Example

The manipulator setfill ('*') or setfill(42) will replace the leading or trailing blanks in the output by *. The value 42 is the decimal equivalent of the character '*' in ASCII.

Example

```
cout<<endl<<setfill ( ' * ')<<"Result:";
```

The above statement will simply display the following:

R	E	S	U	L	T	S	:

The above statement has not specified a field of fixed width to display the string. It will appear in the field of minimum width. It takes exactly eight fields to display the value. That is why, the setfill ('*') has no effect.

Example

```
cout<<endl<<setw(15)<<setfill ( ' * ')<<"Result:";
```

The above statement defines a field width of 15. It is more than the length of the value to be displayed. The statement will display the value as follows:

*	*	*	*	*	*	*	R	E	S	U	L	T	S	:

Program 4.8

Write a program to display the values of different variables using **setfill** manipulator.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  clrscr();
  char str[] = "OOP using C++";
  cout<<setw(20)<<setfill('*')<<str<<endl;
  cout<<setw(20)<<setfill('@')<<str<<endl;
  cout<<setw(20)<<setfill('=')<<str<<endl;
  getch();
}
```

Output:

```
*******OOP using C++
@@@@@@@OOP using C++
=======OOP using C++
```

4.5 Standard Input

By default, the term **standard input** refers to the input given via keyboard. C++ uses **cin** stream object to get standard input. The word 'cin' stands for **console input**. C++ handles standard input by applying the **extraction operator (>>)** with **cin** stream. The operator must be followed by a variable. The variable is used to store the input data.

Syntax

The syntax of using **cin** object is as follows:

```
cin>>var;
```

cin	It is the name of the object used to get standard input.
>>	It is known as extraction operator or get from operator. It gets the input from cin object. The cin object then stores the input in the variable. The left side of >> operator must be an input stream like cin. The right side of >> operator must be a variable of simple data type. One statement can use multiple extraction operators to get multiple values.
var	It is the variable in which the input value is stored.

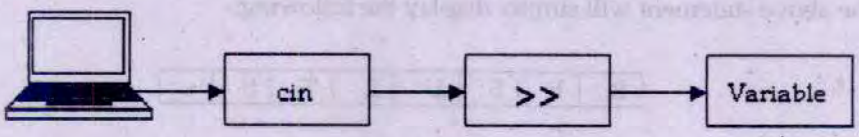


Figure 4.2: Working of 'cin' object

Examples

```
cin>>x;
```

The above line will get a value from keyboard and store it in variable **x**.

```
cin>>a>>b>>c;
```

The above line will get three values from keyboard and store it in variables **a**, **b** and **c**.

Program 4.9

Write a program that inputs name, age and address from the user and then displays these values on the screen.

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
void main()
{
    char name[25],city[30];
    int age;
    clrscr();
    cout<<"Enter your age:";
    cin>>age;
    cout<<"Enter your first name:";
    cin>>name;
    cout<<"Enter your city:";
    cin>>city;
    cout<<"\nYour first name is "<<name<<endl;
    cout<<"Your city is "<<city<<endl;
    cout<<"Your age is "<<age<<endl;
    getch();
}
```

Output:
Enter your age: 16
Enter your first name: Usman
Enter your city: Faisalabad

Your first name is Usman
Your city is Faisalabad
Your age is 16

Program 4.10

Write a program to calculate the simple interest. It inputs principal amount, rate of interest and the number of years and displays the simple interest.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
    double p, r, t, i;
    clrscr();
    cout<< "Enter principal amount, rate , time:";
    cin>> p >> r >> t;
    i = (p * r* t) / 100;
    cout<< "\n Principal Amt = Rs. " << p;
    cout<< "\n Rate = " << r << "%" ;
    cout << "\n Time = " << t << "yrs.";
    cout<< "\n Simple Interest Amt = Rs. " << i ;
    getch();
}
```

Output:
Enter amount, rate, time: 1000 5 3
Principal Amt = Rs. 1000
Rate = 5%
Time = 3yrs
Simple Interest Amt = Rs. 150

Program 4.11

Write a program that inputs a character and displays its ASCII code.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    char charac;
    cout << "Enter the character : ";
    cin>>charac;
    int num1 = charac;
    cout << "The ASCII code for " << charac << " is " << num1 << "." << endl;
    getch();
}
```

Output:
Enter a character: a
The ASCII code for a is 97

Freebooks.pk

Program 4.12

Write a program that inputs dividend and divisor. It then calculates and displays the quotient and remainder.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int div, dis, q , r;
  clrscr();
  cout<< "Enter dividend & divisor :";
  cin >> div >> dis;
  q = div / dis;
  r  = div % dis;
  cout << " Quotient = " << q<<endl;
  cout << "Remainder = " << r;
  getch();
}
```

Output:
Enter dividend & divisor: 20 3
Quotient = 6
Remainder = 2

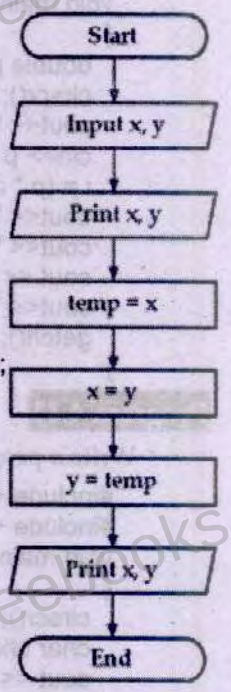
Program 4.13

Write a program that inputs two numbers, swaps the values and then displays them.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, temp;
  clrscr();
  cout<<"Enter the first number: ";
  cin>>a;
  cout<<"Enter the second number: ";
  cin>>b;
  cout<<"You input the numbers as "<<a<<" and "<<b<<endl;
  temp = a;
  a = b;
  b = temp;
  cout<<"The values after swapping are "<< a <<" and "<<b<<endl;
  getch();
}
```

Output:
Enter first number: 10
Enter second number: 20
You input the numbers as 10 and 20
The values after swapping are 20 and 10

Flowchart:



Program 4.14

Write a program that inputs two numbers, swaps these values without using third variable and displays them.

```
#include<iostream.h>
#include<conio.h>
```

```
void main()
{
  int x, y;
  clrscr();
  cout<<"\n Enter 2 integers respectively:";
  cin>>x>>y;
  cout<<"\n The original value in x = " << x << " and  y = " << y;
  x = x+y;
  y = x - y;
  x = x - y;
  cout<<"\n The swapped value in x = " << x << " and y = " << y;
  getch();
}
```

Output:
Enter 2 integers respectively: 2 3
The original value in x = 2 and y = 3
The swapped value in x = 3 and y = 2

Program 4.15

Write a program that inputs the distance traveled and the speed of vehicle. It calculates the time required to reach the destination and displays it.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
  clrscr();
  double distance, time, speed;
  cout<<"Enter the distance traveled in miles: ";
  cin>>distance;
  cout<<"Enter the speed of vehicle (mph): ";
  cin>>speed;
  time = distance / speed;
  cout<<"Time required to reach destination: "<<time<<" hours."<<endl;
  getch();
}
```

Output:
Enter the distance traveled in miles: 100
Enter the speed of vehicle (mph): 10
Time required to reach destination: 10 hours

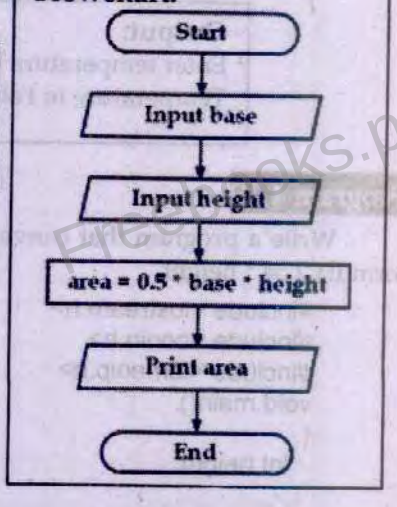
Program 4.16

Write a program that inputs base and height from the user. It calculates and displays the area of a triangle by using the formula Area = ½ * Base * Height.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  float base, height;
  double area;
  clrscr();
  cout<<"Enter base: ";
  cin>>base;
  cout<<"Enter height: ";
  cin>>height;
  area = 0.5 * base * height;
  cout<<"Area = "<<setprecision(2)<<area;
  getch();
}
```

Output:
Enter base: 10.5
Enter height: 5.4
Area = 28.35

Flowchart:



Freebooks.pk

Program 4.17

Write a program that inputs time in seconds and converts it into hh-mm-ss format.

```
#include <iostream.h>
#include<conio.h>
void main()
{
  int sec, s, m, h;
  clrscr();
  cout<< "\n Enter time in seconds :";
  cin >> sec;
  h = sec / 3600;
  sec = sec % 3600;
  m = sec / 60;
  s = sec % 60;
  cout << "\n HH-MM-SS= " << h << ":" << m << ":" << s;
  getch();
}
```

Output:
Enter time in seconds: 5300

HH–MM–SS= 1:28:20

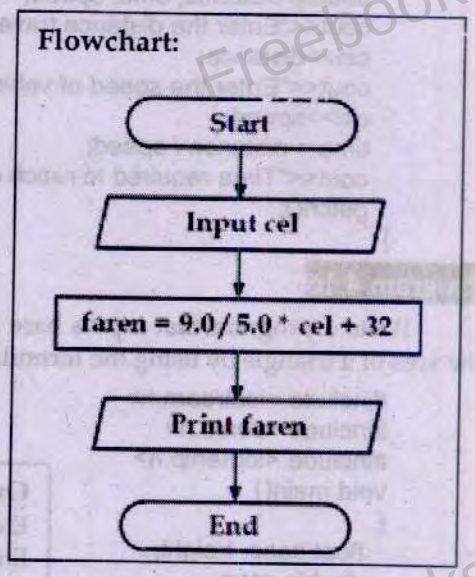
Program 4.18

Write a program that gets temperature from the user in Celsius and converts it into Fahrenheit using the formula F = 9/5 * C + 32.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  float cel, faren;
  clrscr();
  cout<<"Enter temperature in Celcius:";
  cin>>cel;
  faren = 9.0 / 5.0 * cel + 32;
  cout<<"Temperature in Fahrenheit is ";
  cout<<setprecision(2)<<faren;
  getch();
}
```

Output:
Enter temperature in Celsius: 15.50
Temperature in Fahrenheit is 59.9

Flowchart:



Program 4.19

Write a program that converts a person's height from inches to centimeters using the formula 2.54 * height.

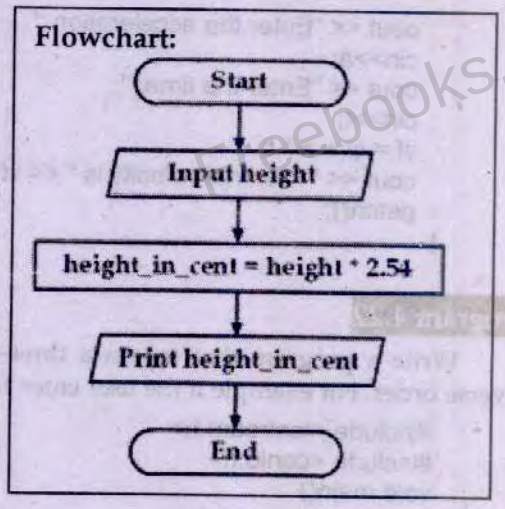
```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  int height;
```

Freebooks.pk

```
    float height_in_cent;
    clrscr();
    cout<<"Enter height in inches:";
    cin>>height;
    height_in_cent = height * 2.54;
    cout<<"Your height in centimeters is: ";
    cout<<setprecision(2)<<height_in_cent;
    getch();
}
```

Output:
Enter height in inches: 20
Your height in centimeters is: 50.8

Flowchart:



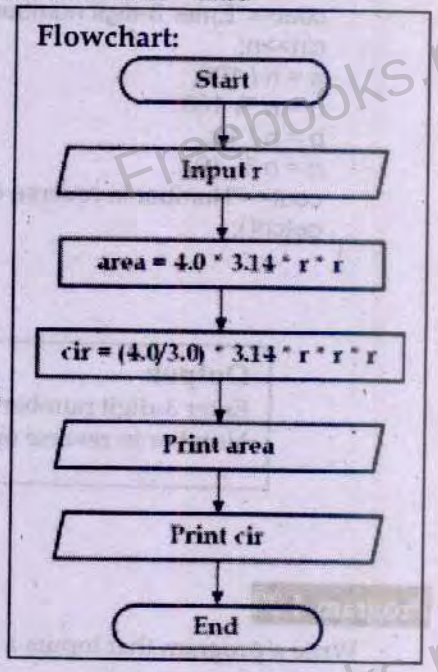
Program 4.20

Write a program that inputs radius from the user and calculates area and circumference of circle using the formula Area = πR^2 and circumference = $2\pi R$.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  float area, radius, cir;
  clrscr();
  cout<<"Enter radius: ";
  cin>>radius;
  area = radius * radius * 3.141;
  cir = 2.0 * 3.141 * radius;
  cout<<"Area: "<<setprecision(2)<<area<<endl;
  cout<<"Circumference: "<<setprecision(2)<<cir;
  getch();
}
```

Output:
Enter radius: 5
Area: 78.53
Circumference: 31.41

Flowchart:



Program 4.21

Write a program that calculates the final velocity of an object by taking following inputs from the user: v*i* = Initial velocity, a = acceleration, t = time span. Formula v*f* = vi + at

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int vi, vf, a, t;
  cout << "Enter the initial velocity: ";
  cin>>vi;
```

Output:
Enter the initial velocity: 20
Enter the acceleration: 10
Enter the time: 5
The final velocity is 70

Freebooks.pk

```
    cout << "Enter the acceleration: ";
    cin>>a;
    cout << "Enter the time: ";
    cin>>t;
    vf = vi + a * t;
    cout << "The final velocity is " << vf << endl;
    getch();
}
```

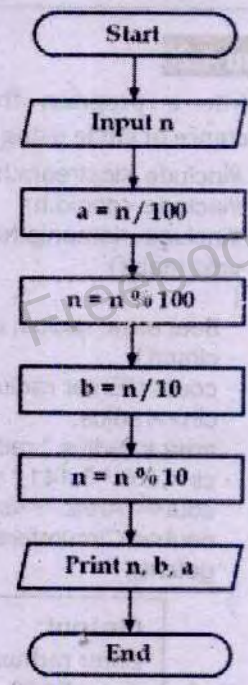
Program 4.22

Write a program that inputs a three-digit number from the user and displays it in reverse order. For example if the user enter 123, it displays 321.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int n, a, b;
    clrscr();
    cout<<"Enter 3-digit number:";
    cin>>n;
    a = n / 100;
    n = n % 100;
    b = n / 10;
    n = n % 10;
    cout<<"Number in reverse order is "<<n<<b<<a;
    getch();
}
```

Output:
Enter 3-digit number: 512
Number in reverse order is 215

Flowchart:



Program 4.23

Write a program that inputs a five-digit number as input and reverse the number.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    long int n, a, b, c, d;
    clrscr();
    cout<<"Enter 5-digit number: ";
    cin>>n;
    a = n / 10000;
    n = n % 10000;
    b = n / 1000;
```

Output:
Enter 5-digit number: 92174
Number in reverse order is 47129

Freebooks.pk

```
    n = n % 1000;
    c = n / 100;
    n = n % 100;
    d = n / 10;
    n = n % 10;
    cout<<"Number in reverse order is "<<n<<d<<c<<b<<a;
    getch();
}
```

Program 4.24

Write a program that inputs an even and odd number through keyboard, multiplies even with 5 and odd with 3 and adds both results. It subtracts the result from 1000 and finally prints the difference.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int ev, od, r;
  clrscr();
  cout<<"Enter an even number: ";
  cin>>ev;
  cout<<"Enter an odd number: ";
  cin>>od;
  r = 1000 - ((ev*5) + (od*3));
  cout<<"Difference = "<<r;
  getch();
}
```

Output:
Enter an even number: 10
Enter an odd number: 5
Difference = 935

Program 4.25

Write a program that generates the following output:

```
10
20
19
```

Use integer constant for 10, arithmetic assignment operator to generate the 20 and decrement operator to generate 19.

```
#include <iostream.h>
#include <conio.h>
void main()

{
  int n = 10;
  clrscr();
  cout<<n<<endl;
  n *= 2;
  cout<<n<<endl;
  cout<<--n;
  getch();
}
```

Freebooks.pk

Program 4.26

Write a program that will prompt the user to enter number of hours. It computes and displays the number of weeks, days and hours within the input number of hours.

(e.g. 4000 hrs = 23, weeks, 5 days, 16 hrs)

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int hrs, w, d;
  cout << "Enter number of hours: ";
  cin>>hrs;
  w = hrs / 168;
  hrs = hrs % 168;
  d = hrs / 24;
  hrs = hrs % 24;
  cout<<"Weeks: "<<w<<endl;
  cout<<"Days: "<<d<<endl;
  cout<<"Hours: "<<hrs;
  getch();
}
```

Output:
Enter number of hours: 200
Weeks: 1
Days: 1
Hours: 8

Program 4.27

Write a program that will prompt the user to enter the current rates of electricity, gas and petrol per unit. Give each item's rate an increment of 10%. Compute and display the new prices per unit of electricity, gas and petrol.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  clrscr();
  float e1, e2, g1, g2, p1, p2;
  cout << "Enter current electricity rate: ";
  cin>>e1;
  cout << "Enter current petrol rate: ";
  cin>>p1;
  cout << "Enter current gas rate: ";
  cin>>g1;
  e2 = e1 * 1.1;
  p2 = p1 * 1.1;
  g2 = g1 * 1.1;
  cout<<"New electricity rate: "<<setprecision(2)<<e2<<endl;
  cout<<"New petrol rate: "<<setprecision(2)<<p2<<endl;
  cout<<"New gas rate: "<<setprecision(2)<<g2<<endl;
  getch();
}
```

Output:
Enter current electricity rate: 11
Enter current petrol rate: 69
Enter current gas rate: 8
New electricity rate: 12.1
New petrol rate: 75.9
New gas rate: 8.8

Freebooks.pk

Program 4.28

Write a program to print the following message:

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  cout<<"C"<<endl;
  cout<<"O M"<<endl;
  cout<<"P U T"<<endl;
  cout<<"E R I S"<<endl;
  cout<<"A W O R L"<<endl;
  cout<<"D O F S"<<endl;
  cout<<"C I E"<<endl;
  cout<<"N C"<<endl;
  cout<<"E"<<endl;
  getch();
}
```

```
C
O M
P U T
E R I S
A W O R L
D O F S
C I E
N C
E
```

Program 4.29

Write a program that generates the following checker board by using eight output statements.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  cout<<" . . . . . . . ."<<endl;
  getch();
}
```

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Freebooks.pk

Programming Exercises

1. Write a program that prints a text of 4 lines consisting of characters, integer values and floating-point values using cout statement.
2. Write a program that inputs radius of sphere from the user. Calculates its volume and surface area using the formula Area = $4\pi R^2$ and circumference = $4/3\pi R^3$ where π=3.14.
3. Write a program to find out the area of triangle when three sides a, b and c of the triangle are given. Use appropriate statements to input the values of a, b and c from the keyboard. Formula for the area of triangle is area = $\sqrt{s(s-a)(s-b)(s-c)}$ where s = (a + b + c) / 2.
4. Write a program that inputs miles from the user and convert miles into kilometers. One mile is equal to 1.609 kilometer.
5. Write a program that inputs 4 numbers and calculates the sum, average, and product of all the numbers.
6. Write a program that inputs age in years and displays age in days and months.
7. Write a program that inputs a number from user and displays its square and cube.
8. Write a program that inputs total pages of a book, number of pages a person reads in one day and number of days a person has read the book. It displays number of pages that have been read and number of pages remaining.
9. A car can travel 5.3 miles in 1 liter. Write a program that inputs petrol in liters and displays how much distance the car can cover using the available petrol.
10. Write a program that inputs total number of student in a class and fee per student. It displays total fee collected from the class.
11. Write a program that inputs temperature from the user in Fahrenheit and converts it into Celsius degree using formula C = 5/9 (F – 32).
12. Write a program that inputs a 3-digit number and displays its digits in separate three lines. For example if the user enters 123, the program displays the output as follows:

    1
    2
    3

13. Write a program to show following output using one cout statement:

1	2	3	4	5
6	7	8	9	10

14. Write a program to calculate the volume (V) of a cube by taking measures from the user. (Formula: V = length * width * height)
15. Write a program that inputs the x, y coordinates for two points and computes the distance between two points using the formula: Distance = $\sqrt{(x2-x1)^2+(y2-y1)^2}$
16. Write a program to swap the values of three variables with using fourth variable.
17. Write a program that calculates the arc of length of a convex lens by taking radius of arc and angle made by arc. (Formula: length = radius * angle)
18. Write a program that inputs pounds from the user and converts it into kilograms.
19. Write a program that computes the area of a sector of a circle when theta is the angle in radians between the radii.

Freebooks.pk

20. Write a program that reads a positive number and then computes the logarithm of that value to the base 2.
21. Write a program to enter a letter and display the next two letters.
22. Write a program that inputs five-digit number through the keyboard and calculates the sum of its digits.
23. Write a program that inputs Basic Salary and calculates 35% dearness allowance, 25% house rent and then displays the gross salary.
24. Write a program that inputs two times in hh:mm:ss format, adds both times and displays the total time.
25. Write a program that inputs principal amount, rate of interest and total time. It calculates the compound interest and displays it.
26. Write a program that inputs a number and displays its corresponding ASCII code.
27. Write a program that displays the following output:

Number	Square	Cube
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

28. Write a program that inputs marks obtained by a student in five subjects. It then calculates and displays the total marks and percentage.

Exercise Questions

Q.1. What is stream? What are two types of streams?

A stream can be defined as flow of data. It is an object that is used by a program to insert or extract characters. There are two types of stream known as input stream and output stream. A sequence of characters from an input device to computer is called input stream. A sequence of characters from computer to output device is called output stream.

Q.2. What is standard output and input?

By default, the term **standard output** refers to the output displayed on monitor. C++ uses the **cout** stream object to display standard output. By default, the term **standard output** refers to the input given via keyboard. C++ uses **cin** stream object to get standard input.

Q.3. Which header must be included to use cin and cout objects?

The cin and cout objects can be used by including iostream.h header file.

Q.4. What is the use of insertion and extraction operators?

The insertion operator << is used with **cout** stream object to display standard output. The message or the value of variable followed by the insertion operator is displayed on the screen. The extraction operator >> is used with **cin** stream object to get standard input. The operator must be followed by a variable. The variable is used to store the input data.

Q.5. What is the output of the following?

```
char C1, C2, C3;
C1 = 'A';
C2 = 'B';
C3 = 'C';
cout<<C1<<C2<<"C3";
```

Answer: ABC3

Q.6. What is the output of the following?

```
cout << "One" << "Two" << endl;
cout << "Three" << "Four" << endl;
cout << "Five" << "Six" << endl;
```

Answer:

OneTwo
ThreeFour
FiveSix

Q.7. Write the output of the following segments of code:

```
int x = 2;
double y = 3.0;
cout << setprecision(2);
cout << x * y << " and " << x / y;
```

Answer: 6 and 0.67

Q.8. Write the output of the following segments of code:

```
int x = 13;
int y = 3;
cout << x/y << y/x << x%y;
```

Answer: 401

Q.9. What is the output of following code if user enters 3, 2 and 4?

```
int x, y, z;
cout<<"Enter three numbers: ";
cin >> x >> y >> z;
y = x + z;
x = z + 1;
cout << x << " " << y<< " " << z;
```

Answer: 5 7 4

Q.10. Write a single cout statement to print the following output.

```
cout << "This\nisa\nC++\nprogram\n" ;
```

Answer:

This
is a
C++
program

Q.11. Write the output of the following segments of code.

```
cout << "\top ba\na\\na";
```

Answer:

op ba
a\na

Q.12. What is the output of the following?

```
float a, b, temp;
a = 2.467;
b= 5.254;
temp = a;
a = b;
b = temp;
cout << fixed << setprecision(2);
cout << setw(5) << a << setw(7) << b << endl;
```

Answer:

| | 5 | . | 2 | 5 | | | | 2 | . | 4 | 7 | | | | | | |

Freebooks.pk

Q.13. What is the output of the following?

```
cout << "One\n" << "Two\n";
cout << "Three" << "\tFour";
cout << "\nFive" << "\tSix" << endl;
```

Answer:

```
One
Two
Three	Four
Five	Six
```

Q.14. If i and j are int variables with i = 2 and j = 6, give output of each of the following:

Statement	What is printed
cout << i;	2
cout << "i = " << i;	i = 2
cout << "Sum = " << i + j;	Sum = 8
cout << "i = " << i << endl << "j = " << j << endl << "sum = " << i + j << endl;	i = 2 j = 6 sum = 8
cout << j % i << endl;	0

Multiple Choice

1. **cin is:**
   a. an object b. variable c. Command d. None
2. **The _____ object causes data to be input from the keyboard:**
   a. cin b. cout c. standard input d. None
3. **The cin object is followed by _____ operator:**
   a. iostream b. the stream insertion (<<) c. stream extraction (>>) d. None
4. **The functions used for input and output is stored in:**
   a. iostream.h b. conio.h c. math.h d. None
5. **Which of the following is used to display output on screen is called:**
   a. cout object b. cin object c. Opening and closing braces d. None
6. **Which of the following is displayed by cout object?**
   a. Text b. constant or values of variable c. A and b d. None
7. **How many variables can be used in one cout object?**
   a. One b. Two c. Many d. None
8. **Which escape sequence can be used to begin a new line in C++?**
   a. \a b. \b c. \m d. \n
9. **Which escape sequence can be used to beep from speaker in C++?**
   a. \a b. \b c. \m d. \n
10. **Which escape sequence can be used to insert a Tab in C++?**
    a. \a b. \b c. \t d. \n
11. **Which escape sequence can be used to move cursor at the beginning of current line in C++?**
    a. \a b. \b c. \m d. \r
12. **What library contains the definitions of setprecision and fixed?**
    a. iostream b. iomanip C. both a and b d. string
13. **What is the output of the following statement? (# Indicates a blank space:**
    cout << setw (6) << left << "world" << setw(6) << right << 1.97;
    a. #world1.97## b. #world1.97### c. world###1.97 d. world####1.97
14. **Which manipulator is used to set the number of digits to be displayed after decimal point:**
    a. Setw b. setprecision c. Setfill d. None

Freebooks.pk

15. Which of the following manipulator is used to replace the leading or trailing blanks in the output by the specified character:

a. setw b. setprecision c. setfill d. None

16. This manipulator is used to establish a field width for the value immediately following it:

a. setw b. setprecision c. iomanip d. None

17. What will the following code display?

```
int a = 2, b = 4, c = 6;
cout << a << b << c << endl;
```

a. 246 b. 426 c. 642 d. abc

18. What will the following code display?

```
int n= 2;
cout << "The number is " << "n" << endl;
```

a. The number is 2 b. THE number is 2
c. The number is n d. The number is 0

19. What is printed by the following statement?

```
cout << "My  name is" << "Abdullah" << endl;
```

a. My name is Abdullah b. my Name is Abdullah
c. My name is ABDULLAH d. MY name is Abdullah

20. Which of the following are correct?

```
a.  cout<<"Hello World"<<endl;
b.  cout<<"Hello" ;
        <<"World"<<endl;
    cout<<" Hello"
        <<"Wordl"<<endl;
```

d. both a and c

21. Which of the following statement prints Abdullah\exam1\test.txt?

a. cout << "Abdullah\exercise\test.txt";
b. cout << "Abdullah\\exercise\\test.txt";
c. cout << "Abdullah\"exercise\"test.txt";
d. cout << "Abdullah"\exercise"\test.txt";

Answers

1. a	2. a	3. c	4. a	5. a	6. c	7. c
8. d	9. a	10. c	11. d	12. b	13. c	14. b
15. c	16. a	17. a	18. c	19. a	20. d	21. b

Fill in the Blanks

1. The process of giving something to computer is known as_____.
2. The process of getting something from computer is known as _____.
3. _____stream object to display standard output
4. cout stands for _____.
5. The cout object is used with _____
6. The endl stands for _____.
7. _____ manipulator is used to set number of digits to be displayed after decimal point.
8. _____manipulator replaces leading or trailing blanks in output by specified character.

Freebooks.pk

9. C++ uses _____ to get standard input
10. C++ handles standard input by applying the _____.
11. The word 'cin' stands for _____.
12. Escape sequences always begins with a_____.

Answers

1. Input	2. Output	3. cout
4. console output	5. insertion operator (<<).	6. end of line
7. setprecision	8. setfill	9. cin stream object
10. extraction operator (>>)	11. console input	12. blackslash

True/ False

1. The process of giving something to the computer is called output.
2. The cout object is used with extraction operator (>>).
3. one statement can use multiple insertion pointer to display output.
4. Escape characters are not displayed in the output.
5. Escape sequence are special characters used in combination with "/".
6. \t is used to insert new line in output.
7. endl requies no parameter.
8. setw manipulator is part of iomanip.h.
9. C++ handles standard input by applying the extraction operator(>>) with cin object.
10. The cin object can be used to input more than one value in a single statement.
11. The extraction operator must be followed by variable.
12. It is impossible to display 34.789 in a field of 9 spaces with 2 decimal places of precision.
13. An input stream is a stream from a source to a computer.
14. An output stream is a stream from a computer to destination
15. When inputting data into a variable, the operator >> skips all leading whitespace characters.
16. The program must include the header file iostream to use cin and cout.
17. The manipulator setprecision formats the output of floating-point numbers to a specified number of decimal places.
18. The manipulator fixed output floating-point numbers in the fixed decimal format.
19. The manipulator setw formats the output of an expression in a specific number of columns.
20. The default out put in setw format is right justified.
21. The extraction operator is a binary operator

Answers

1. F	2. F	3. T	4. T	5. F	6. F	7. T
8. T	9. T	10. T	11. T	12. F	13. T	14. T
15. T	16. T	17. T	18. T	19. T	20. T	21. T

Freebooks.pk

CHAPTER 5

CONDITIONAL STRUCTURES

Chapter Overview

5.1 Control Structure
- 5.1.1 Types of Control Structures

5.2 Relational Operators
- 5.2.1 Relational Expression

5.3 'if' Structure
- 5.3.1 Limitation of simple 'if' Statement

5.4 'if-else' Structure

5.5 Multiple 'if-else-if' Structure

5.6 Nested 'if' Structure

5.7 Compound Condition
- 5.7.1 Logical Operators

5.8 'switch' Structure
- 5.8.1 Difference between nested 'if-else' and 'switch'

5.9 Conditional Operator

5.10 'goto' Statement

Programming Exercise

Exercise Questions

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

5.1 Control Structure

A statement used to control the flow of execution in a program is called **control structure**. The instructions in a program can be organized in three kinds of control structures to control execution flow. The control structures are used to implement the program logic.

5.1.1 Types of Control Structures

Different types of control structures are as follows:

1. Sequence

In **sequential structure**, the statements are executed in the same order in which they are specified in program. The control flows from one statement to other in a logical sequence. All statements are executed exactly once. It means that no statement is skipped and no statement is executed more than once

Example

Suppose a program inputs two numbers and displays average on screen. The program uses two statements to input numbers, one statement to calculate average and one statement to display average number. These statements are executed in a sequence to find the average number. All statements are executed once when the program is executed. It is an example of sequence control structure.

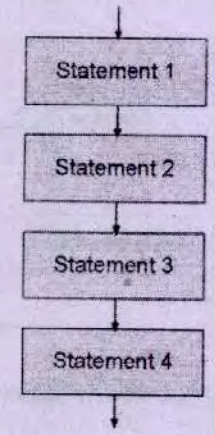


Figure 5.1: Execution of Sequential Statements

2. Selection

A **selection structure** selects a statement or set of statements to execute on the basis of a condition. In this structure, statement or set of statements is executed when a particular condition is **true** and ignored when the condition is **false.** There are different types of selection structures in C++. These are **if, if...else,** and **switch.**

Example

Suppose a program inputs the marks of a student and displays a message on screen whether the student is pass or fail. It displays **Pass** if the student gets 40 or more than 40 marks. It displays **Fail** when the marks are below 40. The program checks the marks before displaying the message. It is an example of selection structure.

Freebooks.pk

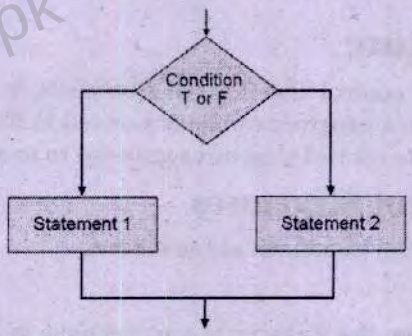


Figure 5.2: Execution of Conditional Statements

3. Repetition

A **repetition structure** executes a statement or set of statements repeatedly. It is also known as **iteration structure** or **loop**. There are different types of repetition structures in C++. These are **while, do while** and **for.**

Example

Suppose we want to display a message "I love Pakistan" on screen for 100 times. It is time consuming to perform this task using control sequence structure. However, it is very easy to perform this task using repetition structure. A single loop statement can display the message for 100 times.

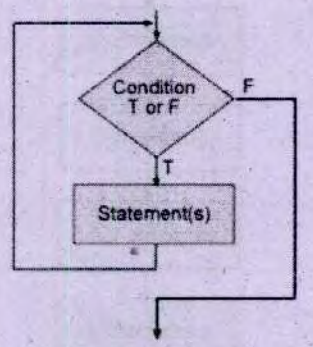


Figure 5.3: Execution of Iterative Statements

4. Function call

Function call is a type of statement that moves the control to another block of code. The control returns back after executing all statements in the block. The remaining statements are executed immediately after the function call when the control is returned.

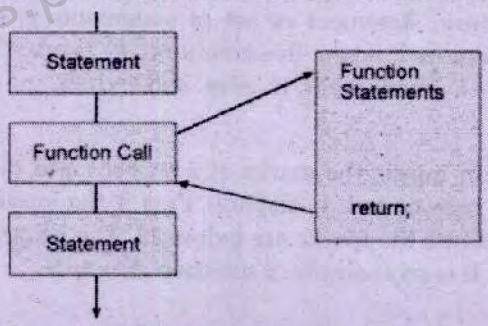


Figure 5.4: Execution of Function Call Statements

5.2 Relational Operators

The relational operators are used to specify conditions in programs. A relational operator compares two values. It produces result as **true** or **false**. The relational operators are sometimes called the **conditional operators** or **comparison operators** as they test conditions that are either **true** or **false**. C++ provides the following relational operators:

Operator	Description
>	**Greater than operator** returns **true** if the value on left side of > is greater than the value on the right side. Otherwise returns **false**.
<	**Less than operator** returns **true** if the value on left side of < is less than the value on right side. Otherwise returns **false**.
= =	**Equal to operator** returns **true** if the values on both sides of == are equal. Otherwise returns **false**.
>=	**Greater than or equal to operator** returns **true** if value on left side of >= is greater than or equal to the value on right side. Otherwise returns **false**.
<=	**Less than or equal to operator** returns **true** if the value on left side of <= is less than or equal to the value on right side. Otherwise returns **false**.
!=	The not equal to operator. Returns **true** if the value on the left side of <> is not equal to the value on the right. Otherwise returns **false**.

Table 5.1: Relational Operators

5.2.1 Relational Expression

A **relational expression** is a statement that uses relational operators to compare two values. The result of a relational expression can be **true** or **false**. Both sides of a relational expression can be a **constant, variable** or **arithmetic expression**.

Examples

Some examples of different relational expressions are as follows:

Relation Expression	Result
100 > 15	True
25 < 5	False
30 <= 12	False
40 >= 20	True
!('A' > 'B')	True
0 >= 0	True
0 <= 0	True
1 != 2	True
5 != 5	False

Table 5.2: Relational Expressions

5.3 'if' Statement

if is a keyword in C++ language. **if** statement is a decision-making statement. It is the simplest form of selection constructs. It is used to execute or skip a statement or set of statements by checking a **condition**.

Freebooks.pk

The condition is given as a relational expression. If the condition is **true**, the statement or set of statements after **if statement** is executed. If the condition is **false**, the statement or set of statements after **if statement** is not executed.

Syntax

The syntax of **if** statement is as follows:

```
if (condition)
   statement;
```

The above syntax is used for single statement. A set of statements can also be made conditional. In this case, these statements are written in curly brackets { }. The set of statements is also called **compound statements**.

The syntax for compound statements in **if statement** is as follows:

```
if (condition)
{
   statement 1;
   statement 2;
   :
   statement N;
}
```

Flowchart

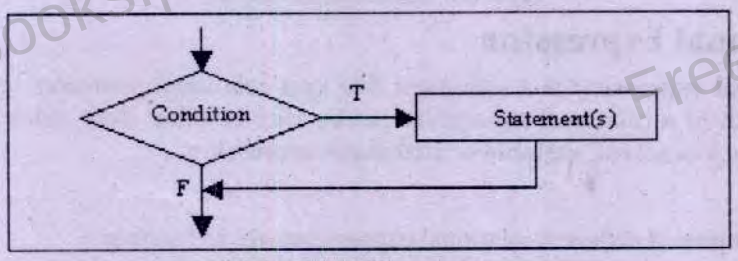


Figure 5.5: If…Else structure

5.3.1 Limitation of simple 'if' Statement

if statement is the simplest selection structure but it is very limited in its use. The statement or set of statements is executed if the condition is **true**. But if the condition is **false** then nothing happens. A user may want to:

- Execute one statement or set of statements if the condition is **true**.
- Execute other statement or set of statements if the condition is **false**.

In this situation, simple **if** statement cannot be used effectively.

Example

For example, a program should display **'Pass'** if the student gets 40 or more marks. It should display **'Fail'** if the student gets less then 40 marks. Simple if statement cannot be used to handle this situation.

Program 5.1

Write a program that inputs marks and displays "Congratulations! You have passed." if the marks are 40 or more.

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
void main()
{
  int marks;
  clrscr();
  cout<<"Enter your marks: ";
  cin>> marks;
  if(marks >= 40)
    cout<<"Congratulations! You have passed.";
  getch();
}
```

Output:
Enter your marks: 50
Congratulation! You have passed.

Program 5.2

Write a program that inputs two numbers and finds whether both are equal.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b;
  clrscr();
  cout<<"Enter a number: ";
  cin>>a;
  cout<<"Enter a number: ";
  cin>>b;
  if(a == b)
    cout<<"Both numbers are equal.";
  getch();
}
```

Output:
Enter a number: 15
Enter a number: 15
Both numbers are equal.

Program 5.3

Write a program that inputs two numbers and finds if second number is square of first.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b;
  clrscr();
  cout<<"Enter a number: ";
  cin>>a;
  cout<<"Enter a number: ";
  cin>>b;
  if(a * a == b)
    cout<<"2nd number is square of 1st number.";
  getch();
}
```

Output:
Enter a number: 5
Enter a number: 25
2nd number is square of 1st number.

Program 5.4

Write a program that inputs marks of three subjects. If the average of marks is more than 80, it displays two messages "You are above standard!" and "Admission granted!"

```
#include <iostream.h>
#include <conio.h>
void main()
```

Freebooks.pk

```
{
  int sub1, sub2, sub3;
  float avg;
  clrscr();
  cout<<"Enter marks of first subject: ";
  cin>>sub1;
  cout<<"Enter marks of second subject: ";
  cin>>sub2;
  cout<<"Enter marks of third subject: ";
  cin>>sub3;
  avg = (sub1+sub2+sub3)/3.0;
  if(avg>80)
  {
    cout<<"You are above standard! \n";
    cout<<"Admission granted!";
  }
  getch();
}
```

Output:
Enter marks of first subject: 90
Enter marks of second subject: 80
Enter marks of third subject: 80
You are above standard!
Admission granted!

Program 5.5

Write a program that inputs three numbers and displays the maximum number.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c, max;
  cout<<"Enter first  number: ";
  cin>>a;
  cout<<"Enter second number: ";
  cin>>b;
  cout<<"Enter third number: ";
  cin>>c;
  max = a;
  if(b > max)
    max = b;
  if(c > max)
    max = c;
  cout<<"The maximum number is "<<max;
  getch();
}
```

Output:
Enter first number: 20
Enter second number: 30
Enter third number: 12
The maximum number is 30

Program 5.6

Write a program to input a number and determine whether it is positive, negative or 0.

```
#include <iostream.h>
#include<conio.h>
void main( )
{
  int n;
  cout<<"Enter a number";
  cin>>n;
  if(n>0)
    cout<<"The number is positive";
```

Output:
Enter a number: 20
The number is positive.

Freebooks.pk

```
    if (n<0)
      cout<<"The number is negative";
    if (n==0)
      cout<<"The number is zero";
    getch();
}
```

Program 5.7

Write a program that inputs five integers. It finds and prints the largest and smallest integer.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c, d, e, max, min;
  clrscr();
  cout<<"Enter five integers: ";
  cin>>a>>b>>c>>d>>e;
  min = max = a;
  if(b < min) min = b;
  if(c < min) min = c;
  if(d < min) min = d;
  if(e < min) min = e;
  if(b > max) max = b;
  if(c > max) max = c;
  if(d > max) max = d;
  if(e > max) max = e;
  cout<<"Largest number is "<<max<<endl;
  cout<<"Smallest number is "<<min<<endl;
  getch();
}
```

Output:
Enter five integers: 10 32 76 61 40
Largest number is 76
Smallest number is 10

5.4 'if-else' Statement

if else statement is another type of **if** statement. It executes one block of statement(s) when the **condition** is **true** and the other when it is **false**. In any situation, one block is executed and the other is skipped. In **if else** statement:

- Both blocks of statement can never be executed.
- Both blocks of statements can never be skipped.

Syntax

Its syntax is as follows:

```
if (condition)
  statement;
else
  statement;
```

Two or more statements are written in curly brackets { }. The syntax for compound statements in **if else** statement is as follows:

```
if (condition)
{
  statement 1;
```

Freebooks.pk

```
    statement 2;
        :
    statement N;
  }
  else
  {
    statement 1;
    statement 2;
        :
    statement N;
  }
```

Flowchart

The flowchart of **if else** statement is as follows:

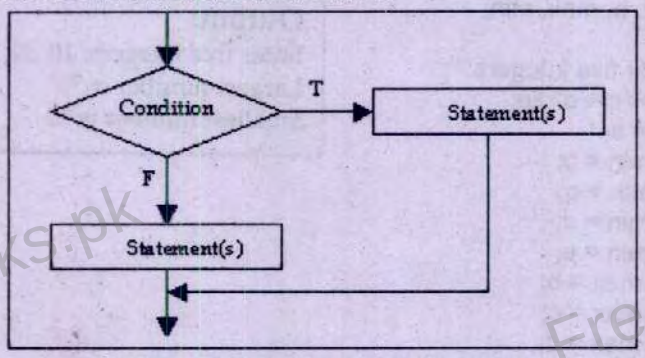


Figure 5.6: Flowchart of if-else structure

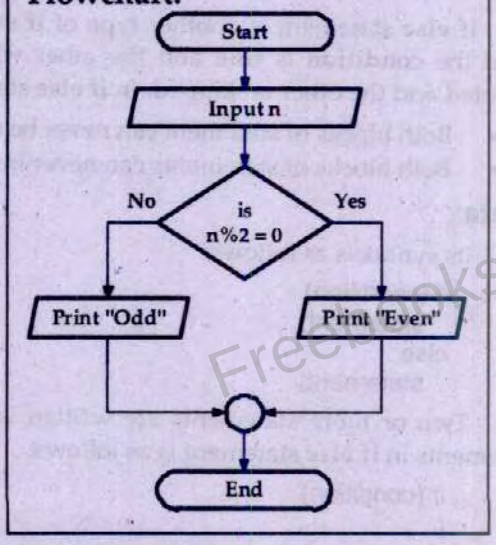
Program 5.8

Write a program that inputs a number and finds whether it is even or odd using if-else structure.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  clrscr();
  cout<<"Enter a number: ";
  cin>>n;
  if(n%2 == 0)
    cout<<n<<" is even.";
  else
    cout<<n<<" is odd.";
  getch();
}
```

Output:
Enter a number: 10
10 is even.

Flowchart:



Freebooks.pk

Program 5.9

Write a program that inputs a year and finds whether it is a leap year or not using if-else structure.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int y;
  clrscr();
  cout<<"Enter a year: ";
  cin>>y;
  if(y % 4 == 0)
    cout<<y<<" is a leap year.";
  else
    cout<<y<<" is not a leap year.";
  getch();
}
```

Output:
Enter a year: 2003
2003 is not a leap year.

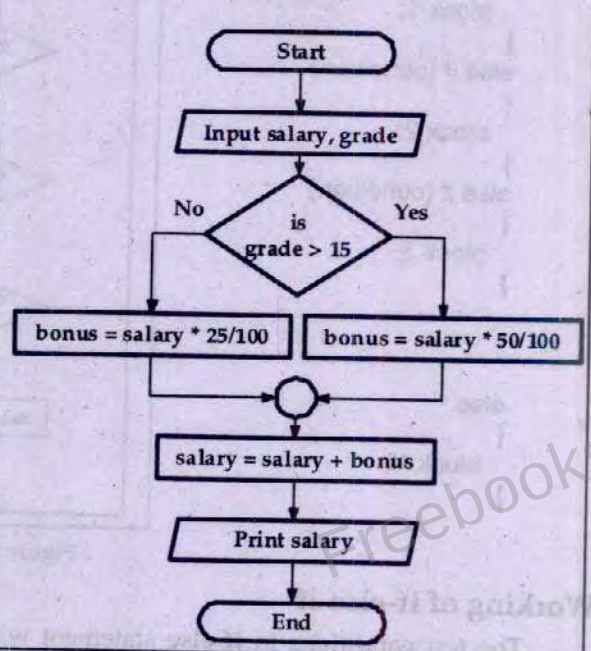
Program 5.10

Write a program that inputs salary and grade. It adds 50% bonus if the grade is greater than 15. It adds 25% bonus if the grade is 15 or less and then displays the total salary.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  float salary, bonus;
  int grade;
  clrscr();
  cout<<"Enter your salary: ";
  cin>>salary;
  cout<<"Enter your grade: ";
  cin>>grade;
  if(grade>15)
    bonus = salary * 50.0/100.0;
  else
    bonus = salary * 25.0/100.0;
  salary = salary + bonus;
  cout<<"Your total salary is Rs. ";
  cout<<salary;
  getch();
}
```

Output:
Enter your salary: 16000
Enter your grade: 17
Your total salary is 24000

Flowchart:



Program 5.11

Write a program that inputs two integers. It determines and prints if the first integer is a multiple of second integer.

```
#include < iostream.h>
#include <conio.h>
void main()
{
  int a, b;
  clrscr();
  cout<<"Enter first integer: ";
  cin>>a;
  cout<<"Enter second integer: ";
  cin>>b;
 if(a % b == 0)
    cout<<"The first number is a multiple of second.";
 else
    cout<<"The first number is not a multiple of second.";
  getch();
}
```

Output:
Enter first integer: 8
Enter second integer: 2
The first number is a multiple of second.

5.5 Multiple 'if-else-if' Structure

if-else-if statement can be used to choose one block of statements from many blocks of statements. It is used when there are many options and only one block of statements should be executed on the basis of a condition.

Syntax

The syntax of this structure is:

```
if (condition)
{
  block 1;
}
else if (condition)
{
  block 2;
}
else if (condition)
{
  block 3;
}
.
.
else
{
  block N;
}
```

Flowchart

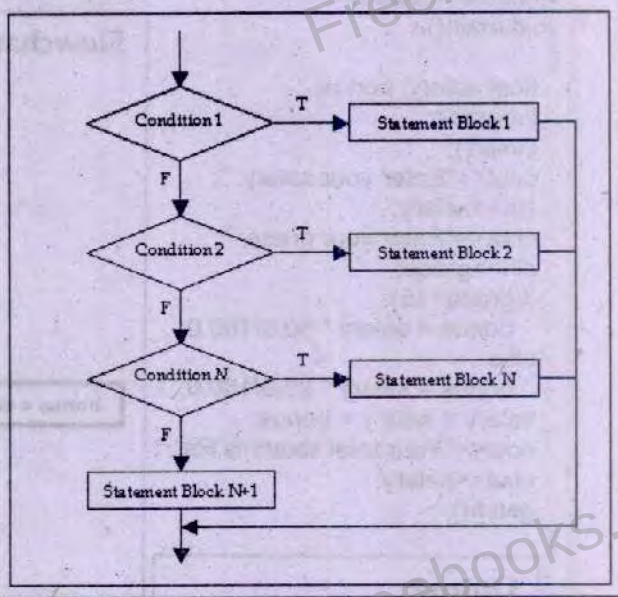


Figure 5.7: Flowchart of multiple if-else-if structure

Working of if-else-if

The test conditions in **if-else** statement with multiple alternatives are executed in a sequence until a **true** condition is reached. If a condition is **true**, the block of statements following the condition is executed. The remaining blocks are skipped. If a condition is **false**, the block of statements following the condition is skipped. The statement after the last **else** are executed if all conditions are **false**.

Freebooks.pk

Program 5.12

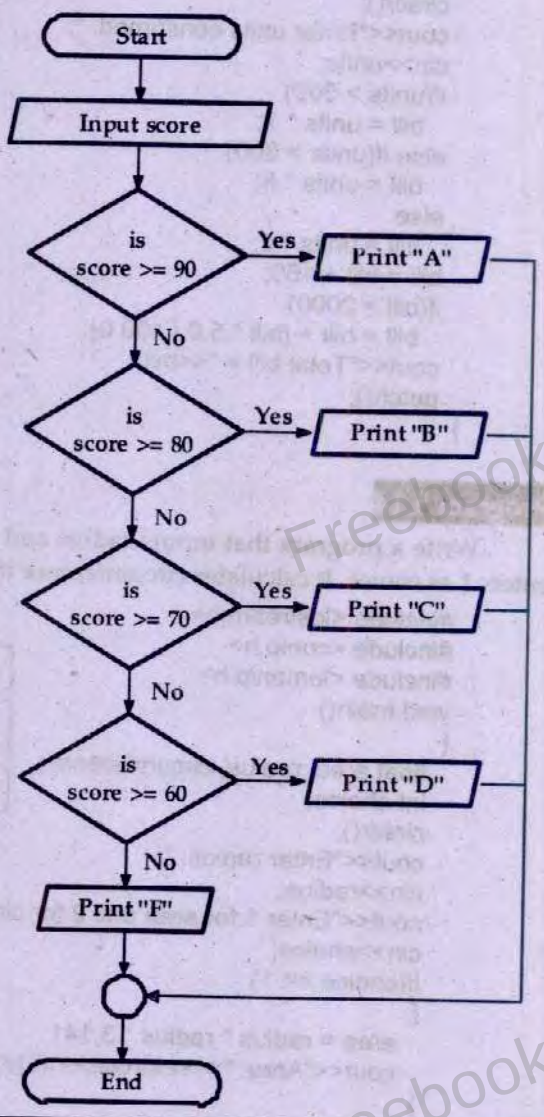
Write a program that inputs test score of a student and displays his grade according to the following criteria:

Test Score	Grade
>= 90	A
80 – 89	B
70 – 79	C
60 – 69	D
Below 60	F

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int score;
  clrscr();
  cout<<"Enter your test score: ";
  cin>>score;
  if(score>=90)
    cout<<"Your grade is A.";
  else if(score>=80)
    cout<<"Your grade is B.";
  else if(score>=70)
    cout<<"Your grade is C.";
  else if(score>=60)
    cout<<"Your grade is D.";
  else
    cout<<"Your grade is F.";
  getch();
}
```

Output:
Enter your test score: 74
Your grade is C.

Flowchart:



Program 5.13

Write a program that calculates the electricity bill. The rates of electricity per unit are as follows:

- If the units consumed are <=300, then the cost is Rs. 2 per unit
- If the units consumed are >300 and <=500, then the cost is Rs. 5 per unit.
- If the units consumed exceed 500 then the cost per unit is Rs. 7

A line rent Rs. 150 is also added to the total bill and a surcharge of 5% extra if the bill exceeds Rs.2000. Calculate the total bill with all the conditions given above.

Freebooks.pk

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int units;
  float bill;
  clrscr();
  cout<<"Enter units consumed: ";
  cin>>units;
  if(units > 500)
    bill = units * 7;
  else if(units > 300)
    bill = units * 5;
  else
    bill = units * 2;
  bill = bill + 150;
  if(bill > 2000)
    bill = bill + (bill * 5.0 / 100.0);
  cout<<"Total bill = "<<bill;
  getch();
}
```

Output:
Enter units consumed: 300
Total bill = 750

Program 5.14

Write a program that inputs radius and user's choice. It calculates area of circle if user enters 1 as choice. It calculates circumference if the user enters 2 as choice.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
  float area, radius, circumference;
  int choice;
  clrscr();
  cout<<"Enter radius: ";
  cin>>radius;
  cout<<"Enter 1 for area and 2 for circumference: ";
  cin>>choice;
  if(choice == 1)
  {
    area = radius * radius * 3.141;
    cout<<"Area: "<<setprecision(2)<<area;
  }
  else if(choice == 2)
  {
    circumference = 2.0 * 3.141 * radius;
    cout<<"Circumference: "<<circumference;
  }
  else
    cout<<"Invalid choice.";
  getch();
}
```

Output:
Enter radius: 5
Enter 1 for area and 2 for circumference: 1
Area: 78.53

Freebooks.pk

Program 5.15

Write a program that inputs salary. If the salary is 20000 or more, it deducts 7% of salary. If the salary is 10000 or more but less than 20000, it deducts 1000 from the salary. If salary is less than 10000, it deducts nothing. It finally displays the net salary.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int salary;
  float net;
  clrscr();
  cout<<"Enter salary: ";
  cin>>salary;
  if(salary >= 20000)
    net = salary - (salary * 7.0/100);
  else if(salary >= 10000)
    net = salary - 1000;
  else
    net = salary;
  cout<<"Your net salary is "<<net;
  getch();
}
```

Output:
Enter your salary: 15000
Your net salary is 14000

5.6 Nested 'if' Structure

An **if** statement within an **if** statement is called **nested if** statement. In nested structure, the control enters into the **inner if** only when the outer condition is **true**. Only one block of statements are executed and the remaining blocks are skipped automatically.

The user can use as many **if** statements inside another **if** statement as required. The increase in the level of nesting increases the complexity of **nested if** statement.

Syntax

The syntax of nested If is as follows:

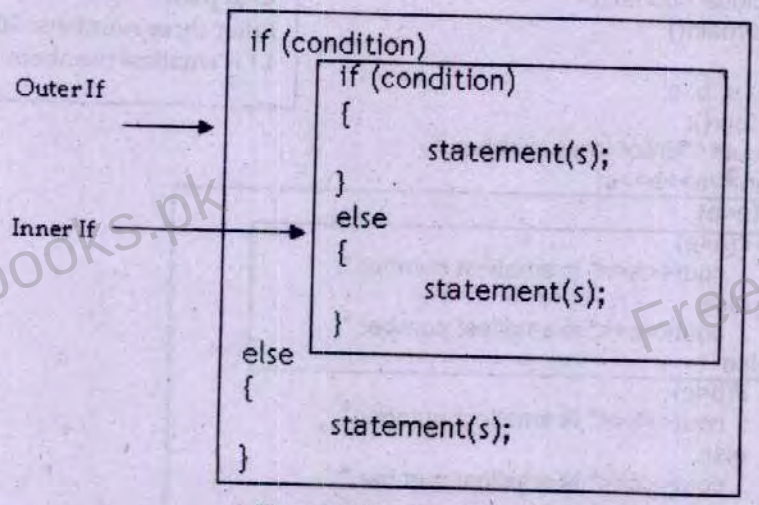


Figure 5.8: Syntax of nested if structure

Freebooks.pk

Working of Nested IF

In **nested if** statement, the condition of **outer if** is evaluated first. If it is **true**, the control enters in the **inner if** block. If the condition is **false**, the **inner if** is skipped and control directly moves to the **else** part of **outer if**. If **outer if** is **true** then control enters in the **inner if** statement. The **inner if** evaluated according to **simple if** statement.

Flowchart

The flowchart of **nested if** statement is as follows:

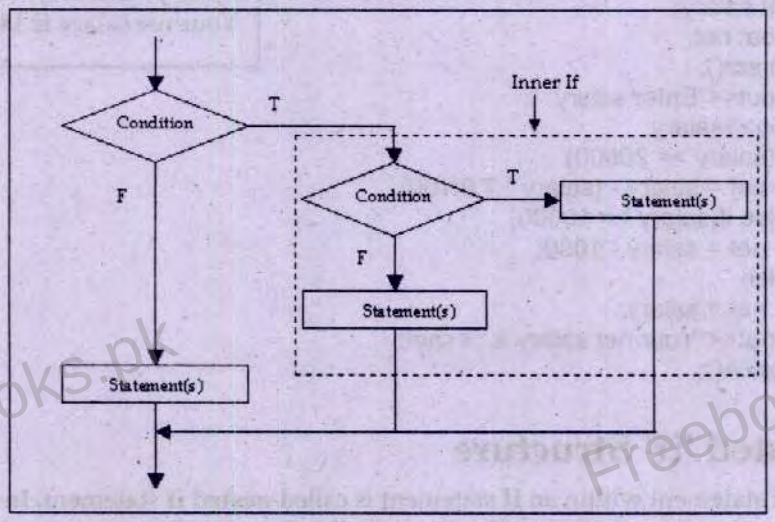


Figure 5.9: Flowchart of nested if structure

Program 5.16

Write a program that inputs three numbers and displays the smallest number by using nested if condition.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c;
  clrscr();
  cout<<"Enter three number:";
  cin>>a>>b>>c;
  if(a<b)
    if(a<c)
      cout<<a<<" is smallest number.";
    else
      cout<<c<<" is smallest number.";
  else
    if(b<c)
      cout<<b<<" is smallest number.";
    else
      cout<<c<<" is smallest number.";
  getch();
}
```

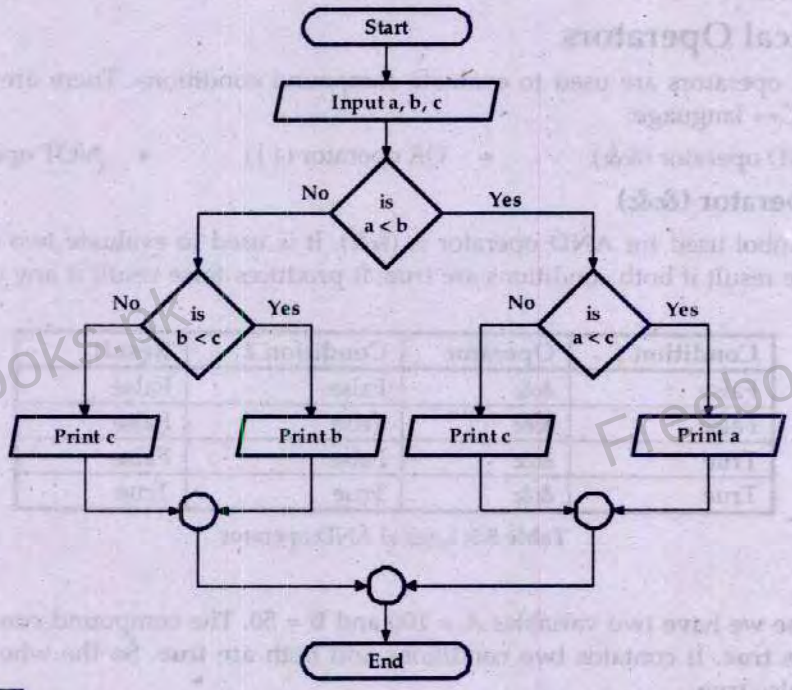
Output:
Enter three numbers: 20 35 13
13 is smallest numbers.

Freebooks.pk

How it Works?

The above program inputs three numbers and finds the smallest one. When the control enters the outer box, first condition **if(a<b)** is evaluated. If it is true, the controls enters in the smaller box and the condition **if(a<c)** is evaluated. If it is also **true**, the value of **a** is displayed, otherwise the value of **c** is displayed. If the first condition **if(a<b)** is **false**, the control shifts to **else** part of **if** statement. Now the condition **if(b<c)** is evaluated. If it is **true** the value of **b** is displayed, otherwise the value of **c** is displayed.

Flowchart



Program 5.17

Write a program that inputs three numbers and displays whether all numbers are equal or not by using nested if condition.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c;
  cout<<"Enter three number:";
  cin>>a>>b>>c;
  if(a==b)
    if(a==c)
      cout<<"All numbers are equal.";
    else
      cout<<"Numbers are different.";
  else
    cout<<"Numbers are different.";
  getch();
}
```

Output:
Enter three numbers: 15 25 30
Numbers are different.

Freebooks.pk

5.7 Compound Condition

A type of comparison in which more than one conditions are evaluated is called **compound condition**. It is used to execute a statement or set of statements by testing many conditions.

Example

For example, a program inputs two numbers. It displays **OK** if one numbers is greater than 100 and second number is less than 100. Compound condition is executed by using logical operators.

5.7.1 Logical Operators

Logical operators are used to evaluate compound conditions. There are three logical operators in C++ language:

- AND operator (&&)
- OR operator (| |)
- NOT operator (!)

1. AND Operator (&&)

The symbol used for **AND** operator is (&&). It is used to evaluate two conditions. It produces **true** result if both conditions are **true**. It produces **false** result if any one condition is **false**.

Condition 1	Operator	Condition 2	Result
False	&&	False	False
False	&&	True	False
True	&&	False	False
True	&&	True	True

Table 5.3: Logical AND operator

Example

Suppose we have two variables A = 100 and B = 50. The compound condition **(A>10) && (B>10)** is **true**. It contains two conditions and both are **true**. So the whole compound condition is also **true**.

The compound condition **(A>50) && (B>50)** is **false.** It contains two conditions. One condition **(A>50)** is **true** and second condition **(B>50)** is **false**. So the whole compound condition is **false**.

Program 5.18

Write a program that inputs three numbers and displays the maximum number by using logical operators.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c;
  cout<<"Enter three numbers: ";
  cin>>a>>b>>c;
  if(a>b && a>c)
    cout<<"Maximum number is "<<a;
  else if(b>a && b>c)
    cout<<"Maximum number is "<<b;
```

Output:
Enter three numbers: 10 20 30
Maximum number is 30

Freebooks.pk

```
    else
      cout<<"Maximum number is "<<c;
    getch();
  }
```

2. OR Operator (| |)

The symbol used for **OR** operator is (| |). It is used to evaluate two conditions. It produces **true** result if either condition is **true**. It produces **false** result if both conditions are **false**.

Condition 1	Operator	Condition 2	Result
False	||	False	False
False	||	True	True
True	||	False	True
True	||	True	True

Table 5.4: Logical OR operator

Example

Suppose we have two variables A = 100 and B = 50. The compound condition **(A>50) | | (B>50)** is **true**. It contains two conditions and one condition **(A>50)** is **true**. So the whole compound condition is also **true**. The compound condition **(A>500) | | (B>500)** is **false** because both conditions are **false**.

Program 5.19

Write a program that inputs a character and displays whether it is a vowel or not.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char ch;
  clrscr();
  cout<<"Enter any character: ";
  cin>>ch;
  if(ch=='A' || ch=='a' || ch=='E' || ch=='e' || ch=='I' || ch=='i' || ch=='O' || ch=='o' ||
  ch=='U' || ch=='u')
    cout<<"You entered a vowel: "<<ch;
  else
    cout<<"You did not enter a vowel: "<<ch;
  getch();
}
```

Output:
Enter any character: A
Your entered a vowel: A

Program 5.20

Write a program that allows the user to enter any character through the keyboard and determines whether it is a capital letter, small case letter, a digit number or a special symbol.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char ch;
```

Output:
Enter any character: #
The character # is a symbol.

Freebooks.pk

```
    cout << "Enter any character: "<<endl;
    cin>>ch;
    if ((ch >= 'A') && (ch <= 'Z'))
      cout<< "The character "<<ch<<" is a capital letter."<<endl;
    else if ((ch >= 'a') && (ch <= 'z'))
      cout<< "The character "<<ch<<" is a small case letter."<<endl;
    else if ((ch >= '0') && (ch <= '9'))
      cout<< "The character "<<ch<<" is a digit."<<endl;
    else
      cout<<"The character "<<ch<<" is a symbol."<<endl;
    getch();
  }
```

3. NOT Operator (!)

The symbol used for **NOT** operator is (!). It is used to reverse the result of a condition. It produces **true** result if the condition is **false**. It produces **false** result if the condition is **true**.

Operator	Condition	Result
!	True	False
!	False	True

Table 5.5: Logical NOT operator

Example

Suppose we have two variables A = 100 and B = 50. The condition !(A==B) is true. The result of (A==B) is false but NOT operator converts it into true. The condition !(A>B) is false. The condition (A>B) is true but NOT operator converts it into false.

Program 5.21

Write a program that inputs a number and displays whether it is even or odd by using logical operator "!".

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  clrscr();
  cout<<"Enter any number: ";
  cin>>n;
  if(!(n%2==0))
    cout<<"You entered odd number.";
  else
    cout<<"You enter even number.";
  getch();
}
```

Output:
Enter any number: 20
You entered even number.

Program 5.22

Write a program that inputs three digits and displays all possible combinations of these digits.

```
#include <iostream.h>
#include<conio.h>
void main()
{
```

Freebooks.pk

```
int a,b,c;
clrscr();
cout<<"Enter three digits:";
cin>>a>>b>>c;
if((a!=b) && (b!=c) && (c!=a))
{
  cout<<a<<b<<c<<"\t";
  cout<<a<<c<<b<<"\t";
  cout<<b<<a<<c<<"\t";
  cout<<c<<a<<b<<"\t";
  cout<<c<<b<<a;
}
else
{
  if((a==b) && (a==c))
 cout<<a<<b<<c;
  else
  {
   if(a==b)
   {
     cout<<a<<b<<c<<"\t";
     cout<<a<<c<<b<<"\t";
     cout<<c<<b<<a;
   }
   else
   {
    if(a==c)
    {
       cout<<a<<c<<b<<"\t";
       cout<<a<<b<<c<<"\t";
       cout<<b<<a<<c;
    }
    else
    {
       cout<<b<<c<<a<<"\t";
       cout<<b<<a<<c<<"\t";
       cout<<a<<b<<c;
    }
   }
  }
 }
getch();
}
```

Output:

```
Enter three digits: 1 2 3
123     132     213     312     321
```

5.8 'switch' Structure

The **switch** statement is another conditional structure. It is a good alternative of **nested if-else**. It can be used easily when there are many choices available and only one should be executed. Nested if becomes very difficult in such situation.

Freebooks.pk

Syntax

The syntax for writing this structure is as follows:

```
                    → Integer or character variable
switch (expression)
{                   → Integer or character constant
  case constant 1:
    statement(s);
    break;          First case body → Causes exit from case body
  case constant 2:
    statement(s);   Second case body
    break;
    :
  case constant N:
    statement(s);   N case body
    break;
  default:
    statement(s);   Default body
}
```

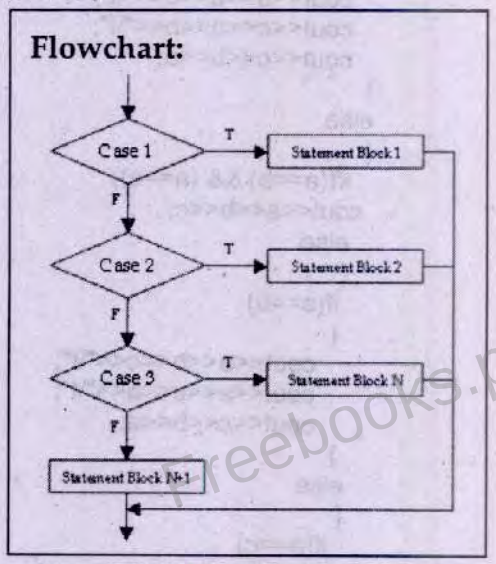


Figure 5.10: Syntax of switch statement

Working of 'switch' Structure

switch statement compares the result of a single expression with multiple cases. Expression can be any valid expression that results in integer or character value. The expression is evaluated at the top of switch statement and its result is compared with different cases. Each case represents one choice. If the result matches with any case, the corresponding block of statements is executed. Any number of cases can be used in one switch statement.

The **default** label appears at the end of all **case** blocks. It is executed only when the result of **expression** does not match with any case label. Its use is optional. The position of default label is not fixed. It may be placed before the first case statement or after the last one.

The **break** statement in each case is used to exit from **switch** body. It is used at the end of each **case** block. When the result of the expression matches with a **case** block, the corresponding statements are executed. The **break** statement comes after these statements and the control exits from **switch** body. If **break** is not used, all case blocks that come after the matching case, will also be executed.

Program 5.23

Write a program that inputs number of week's day and displays the name of the day. For example if user enters 1, it displays "Friday" and so on.

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
void main()
{
  int n;
  clrscr();
  cout<<"Enter number of a weekday: ";
  cin>>n;
  switch(n)
  {
    case 1:
      cout<<"Friday";
      break;
    case 2:
      cout<<"Saturday";
      break;
    case 3:
      cout<<"Sunday";
      break;
    case 4:
      cout<<"Monday";
      break;
    case 5:
      cout<<"Tuesday";
      break;
    case 6:
      cout<<"Wednesday";
      break;
    case 7:
      cout<<"Thursday";
      break;
    default:
      cout<<"Invalid number";
  }
  getch();
}
```

Output:
Enter number of a weekday: 3
Sunday

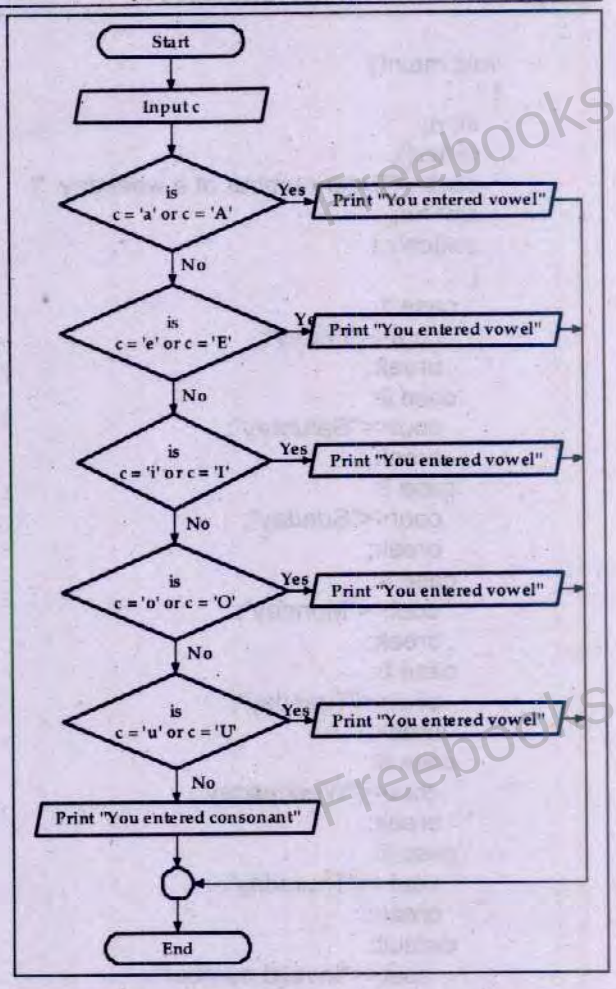
Program 5.24

Write a program that inputs a character from the user and checks whether it is a vowel or consonant.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char c;
  clrscr();
  cout<<"Enter an alphabet: ";
  cin>>c;
  switch(c)
  {
    case 'a':
    case 'A':
      cout<<"You entered vowel.";
      break;
```

Freebooks.pk

```
    case 'e':
    case 'E':
      cout<<"You entered vowel.";
      break;
    case 'i':
    case 'I':
      cout<<"You entered vowel.";
      break;
    case 'o':
    case 'O':
      cout<<"You entered vowel.";
      break;
    case 'u':
    case 'U':
      cout<<"You entered vowel.";
      break;
    default:
      cout<<"Not vowel";
  }
  getch();
}
```



Output:
Enter an alphabet: u
You entered a vowel.

Program 5.25

Write a program that inputs a floating point number, an operator and another floating point number. It displays the result by performing the operation on the given numbers. If the operator is a division, it should check to make sure that the divisor is not equal to zero. If the operator is not a +, -, *, or / then the program should print an error message.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  float a, b;
  char op;
  clrscr();
  cout<<"Enter a floating point number: ";
  cin>>a;
  cout<<"Enter an operator: ";
  cin>>op;
  cout<<"Enter a floating point number: ";
  cin>>b;
  switch(op)
  {
    case '+':
```

Output:
Enter a floating point number: 5.2
Enter an operator: *
Enter a floating point number: 1.3
6.759999

Freebooks.pk

```
            cout<<a+b<<endl;
            break;
        case '-':
            cout<<a-b<<endl;
            break;
        case '*':
            cout<<a*b<<endl;
            break;
        case '/':
            if (b == 0)
              cout<<"Division by zero!"<<endl;
            else
              cout<<a/b<<endl;
            break;
        default:
            cout<<"Invalid operator!"<<endl;
    }
    getch();
}
```

Program 5.26

Write a program that displays the following menu of a health club:

1. Standard Adult Membership
2. Child Membership
3. Senior Citizen Membership
4. Quit the Program

It inputs choice and number of months and calculates membership charges as follows:

Choice	Charges per month
1. Standard Adult Membership	Rs. 50
2. Child Membership	Rs. 20
3. Senior Citizen Membership	Rs. 30

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
  int choice, months;
  double charges;
  cout << "   Health Club Membership Menu\n\n";
  cout << "1. Standard Adult Membership\n";
  cout << "2. Child Membership\n";
  cout << "3. Senior Citizen Membership\n";
  cout << "4. Quit the Program\n\n";
  cout << "Enter your choice: ";
  cin  >> choice;
  if (choice >= 1 && choice <= 3)
  {
    cout << "For how many months? ";
    cin  >> months;
    switch (choice)
    {
```

Output:

```
Health Club Membership Menu
1. Standard Adult Membership
2. Child Membership
3. Senior Citizen Membership
4. Quit the Program

Enter your choice: 2
For how many months? 6
Total charges are Rs. 120
```

Freebooks.pk

```
            case 1:   charges = months * 50.0;
            break;
            case 2:   charges = months * 20.0;
            break;
            case 3:   charges = months * 30.0;
        }
        cout<<"The total charges are Rs. "<<charges<< endl;
    }
    else if (choice != 4)
    {
        cout << "The valid choices are 1 to 4.\n";
        cout << "Run the program again and select one of these.\n";
    }
}
```

Program 5.27

Write a program that converts ASCII number to character and vice versa. The program should display the following menu:

1. Convert ASCII value to Character
2. Convert Character to ASCII value

Output:

1. Convert ASCII value to Character
2. Convert Character to ASCII value
Enter your choice: 1
Enter a number: 97
The corresponding character is a

```
#include<iostream.h>
#include <stdlib.h>
#include<conio.h>
void main()
{
    int number, option;
    char letter;
    cout<<"1. Convert ASCII value to Character"<<endl;
    cout<<"2. Convert Character to ASCII value"<<endl;
    cout<<"Enter your choice: ";
    cin>>option;
    switch (option)
    {
      case 1:
        cout<<"Enter a number : ";
        cin >> number;
        cout<<"The corresponding character is: "<<char(number)<<endl;
        break;
      case 2:
        cout<<"Enter a letter : ";
        cin >> letter;
        cout<<"ASCII value of the letter is: "<<int(letter)<<endl;
        break;
      default:
        cout<<"Invalid Option!";
        break;
    }
    getch();
}
```

5.8.1 Difference between nested 'if-else' and 'switch'

The switch statement and nested if-else statements are used when there are multiple choices and only one is to be executed. Difference between if-else and switch is as follows:

	Switch	Nested-If
1.	It is easy to use when there are multiple choices.	It is difficult to use when there are multiple choices.
2.	It uses single expression for multiple choices.	It uses multiple expressions for multiple choices.
3.	It cannot check a range of values.	It can check a range of values.
4.	It checks only constant values. You cannot use variables with case statement.	It can check variables also. You can use a constant or variable in relational expressions.

Table 5.6: Difference between switch and nested if

5.9 Conditional Operator

Conditional operator is a decision-making structure. It can be used in place of simple **if-else** structure. It is also called **ternary operator** as it uses three operands.

Syntax

The syntax of conditional operator is as follows:

```
(condition) ? true-case statement: false-case statement;
```

condition The condition is specified as relational or logical expression. The condition is evaluated to **true** or **false**.

true-case It is executed if expression evaluates to **true**.

false-case It is executed if expression evaluates to **false**.

Example

Suppose we have a variable A. The following statement:

```
X = (A>50) ? 1 : 0;
```

will assign 1 to X if the condition **A>50** is **true**. It will assign 0 to X the condition is **false**. The above statement can be written using **if-else** statement as follows:

```
if (A>50)
  X = 1;
else
  X = 0;
```

Program 5.28

Write a program that inputs marks of a student and displays "Pass" if marks are more than 40 and "Fail" otherwise by using conditional operator.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int marks;
  clrscr();
  cout<<"Enter your marks:";
  cin>>marks;
```

Output:
Enter your marks: 92
Result is Pass

Freebooks.pk

```
    cout<<"Result is "<<(marks>40 ? "Pass" : "Fail");
    getch();
}
```

Program 5.29

Write a program that inputs a number and displays whether it is divisible by 3 or not by using conditional operator.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  clrscr();
  cout<<"Enter number:";
  cin>>n;
  (n%3==0 ? cout<<"Divisible by 3" : cout<<"Not divisible by 3");
  getch();
}
```

Output:
Enter number: 15
Divisible by 3

5.10 'goto' Statement

The 'goto' statement is used to move the control directly to a particular location of the program by using label. A label is a name given to a particular line of the program. A label is created with a valid identifier followed by a colon (:).

Syntax

```
goto Label;
```

The 'Label' indicates the lable to which the control is transferred.

Program 5.30

Write a program that displays "C++" five times using goto statement.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
  clrscr();
  int n=1;
  loop:
  cout<<n<<": C++"<<endl;
  n++;
  if (n<=5) goto loop;
  cout<<"End of Program";
  getch();
}
```

Output:
1: C++
2: C++
3: C++
4: C++
5: C++
End of Program

How above Program Works?

The above program uses **goto** statement to repeat a statement. The **if** statement checks the value of n. If the value is 5 or less, the control moves back to the **loop** label. In this way, the required message appears five times.

Freebooks.pk

Programming Exercises

1. Writing a program that accepts a character and determines whether the character is a lowercase letter. A lowercase letter is any character that is greater than equal to 'a' and less than or equal to 'z'. If the entered character is a lowercase letter, display the message "Entered character is a lowercase letter", otherwise display the message "Entered character is not a lowercase letter".
2. Senior salesperson is paid Rs. 400 a week, and a junior salesperson is paid Rs. 275 a week. Write a program that accepts as input a salesperson's status in the character variable *status*. If *status* is 's' or 'S', the senior person's salary should be displayed; if status is 'j' or 'J', the junior person's salary should be displayed, otherwise display error message.
3. Write a program to get three numbers from user for integer variables a, b and c. If a is not zero, find out whether it is the common divisor of b and c.
4. Write a program that contains an if statement that may be used to compute the area of a square (area = side * side) or a triangle (area = ½ * base * height) after prompting the user to type the first character of the figure name (S or T).
5. Write a program that gets the number and a letter. If the letter is 'f', the program should treat the number entered as temperature in degrees Fahrenheit and convert it to the temperature in degree Celsius and print a suitable message. If the letter is 'c', the program should consider the number as Celsius temperature and convert it to Fahrenheit temperature and print a suitable message. The program should display error message and then exit if the user enters any other letter.
6. Write a program that accepts the code number as an input and display the correct disk drive manufacture as follows:

Code	disk drive manufacture
1	Western Digital
2	3M Corporation
3	Maxell Corporation
4	Sony Corporation
5	Verbatim Corporation

7. Write a program that uses the following categories of movies:

   A for Adventure movies
   C for Comedy movies
   F for Family movies
   H for Horror movies
   S for Science Fiction movies

The program inputs code for movie type and displays its category. For example if the user enters H, it displays "Horror Movies". The program should also display a menu of movie categories.

8. Write a program that inputs a value and type of conversion. The program should then output the value after conversion. The program should include the following conversions:

   1 inch = 2.54 centimeters
   1 gallon = 3.785 liters

Freebooks.pk

1 mile = 1.609 kilometers
1 pound = .4536 kilograms

Make sure that program accepts only valid choices for type of conversion to perform.

9. A year is a leap year if it is divisible by four, except that any year divisible by 100 is a leap year only if it is divisible by 400. Write a program that inputs a year such as 1996, 1800 and 2010 and display "Leap year" if it is a leap year otherwise displays "Not a leap year".

10. Write a program that inputs temperature and displays a message as follows:

Temperature	Message
Greater than 35	Hot day
Between 25 and 35 (inclusive)	Pleasant day
Less than 25	Cool day

11. Write a program that inputs obtained marks of a student, calculates the percentage (assuming total marks are 1100) and displays grade according to the following rules:

Percentage	Grade
More than or equal to 80	A+
Between 70 (inclusive) and 80	A
Between 60 (inclusive) and 70	B
Between 50 (inclusive) and 60	C
Between 40 (inclusive) and 50	D
Between 33 (inclusive) and 40	E
Less than 33	F

12. Write a program that converts MILITARY time to STANDARD time.

13. Write a program that will take three values a, b and c and print the roots, if real, of the quadratic equation $ax^2 + bx + c = 0$. Sampel input 1: (a=1, b=1, c=-6 the output should be "Roots of the equation are 2 and -3"). Sample input 2 (if the input is a=1, b=0, c=9, the output should be "Sorry the roots are not real.")

14. Write a program that inputs the salary of an employee from the user. It deducts the income tax from the salary on the following basis :

- 20% income tax if the salary is above Rs. 30000.
- 15% income tax if the salary is between Rs. 20000 and Rs. 30000.
- 10% income tax if the salary is below Rs. 20000.

The program finally displays salary, income tax and the net salary.

15. Write a program that inputs year and month. It displays the number of days in the month of the year entered by the user. For example, if the user enters 2010 in year and 3 in month, the program should display "March 2010 has 31 days".

16. Write a program that displays the following menu for a parking area:

- M = Motorcycle
- C = Car
- B = Bus

The program inputs the type of vehicle and number of days to park the vehicle. It finally displays the total charges for the parking according to the following:

- Motorcycle Rs. 10 per day
- Car Rs. 20 per day
- Bus Rs. 30 per day

Freebooks.pk

17. Write a program that inputs a value and type of conversion. The program should then display the output after conversion. The program should include the following conversions:

- 1 cm = .394 inches
- 1 liter = .264 gallons
- 1 kilometer = .622 miles
- 1 kilogram = 2.2 pounds

Make sure that the program accepts only valid choices for the type of conversion.

Exercise Questions

Q.1. What is a control structure?

A statement used to control the flow of execution in a program or function is called control structure. The control structures in C are used to combine individual instruction into a single logical unit. The logical unit has one entry point and one exit point.

Q.2. What are the basic control structures for writing programs?

The basic control structures for writing programs are sequence, selection and repetition.

Q.3. Name and describe three types of branching mechanisms.

Three types of branching mechanisms include if...else statement, multi-way if...else statement and switch statement. An if...else statement selects between two alternative statements based on a condition. A multi-way if...else statement evaluates several conditions and selects one block of statements from many blocks. The switch statement evaluates an expression and executes statements in the matching case label.

Q.4. Write three selection statements and three repetition statements.

Selection statements are if, if...else, switch. The repetition statements are while, do...while, for.

Q.5. What is Relational Operators? List out different relational operators in C++ language?

The relational operators are used to specify conditions in programs. A relational operator compares two values. It produces result as true or false. The relational operators are sometimes called the conditional operators or comparison operators as they test conditions that are either true or false. C++ provides different relational operators. These are >, <,==, >=, <=, and !=.

Q.6. What is relational expression? Give some examples of relational expressions

Relational expression is a statement that uses relational operators to compare two values. Examples of relational expressions are A>B, A<B, A<=B, A>=B, A==B and A!=B.

Q.7. When must you use curly braces { } with a selection or repetition statement?

The curly braces are required if the selection or repetition statement has more than one statement in its body.

Q.8. What is Compound Condition? Give its example.

A type of comparison in which more than one conditions are evaluated is called **compound condition**. It is used to execute a statement or set of statements by testing many conditions. For example, a program inputs two numbers. It displays **OK** if one numbers is greater than 100 and second number is less than 100. Compound condition is executed by using logical operators.

Q.9. What is dangling else?

When an if-else statement is used as the statement part of another if statement, it is not clear that else is associated with which if statement. It is known as dangling else.

Q.10. What happens if break is missed in case block?

If break is not used, all case blocks coming after matching case will also be executed.

Q.11. Which data types can be used in the expression in the switch statement?

The data types that can be used in the expression in switch statement are int and char.

Q.12. What is the alternative of if-else statement in C++?

Conditional operator is an alternative of if-else statement in C. It is a decision-making structure. It is also called ternary operator as it uses three operands.

Freebooks.pk

Q.13. Describe the importance of break & default statement in switch statement.

break statement in each case label is used to exit from switch body. If break is not used, all case blocks coming after matching case will also be executed. The default block is used to execute a statement or set of statements when the result of expression in switch statement does not match with any case.

Q.14. Why break statement is used in a switch ( ) structure?

The break statement in each case label is used to exit from switch body. If break is not used, all case blocks coming after matching case will also be executed.

Q.15. Why should you use a default label in a switch statement?

The default block is used to execute a statement or set of statements when the result of expression in switch statement does not match with any case. It makes sure that switch executes properly if the selector is not in the range of case labels.

Q.16. What is conditional operator? Write the syntax of conditional operator.

Conditional operator is a decision-making structure. It can be used in place of simple "if-else" structure. It is also called ternary operator because it uses three operands. The syntax of conditional operator is as follows:

(condition) ? true-case statement : false-case statement;

Q.17. Write one similarity and one difference between if...else and conditional operator.

The similarity between if...else and conditional operator is that both select a statement depending on a condition. The difference is that conditional operator can choose between two values but if...else can be extended to check multiple conditions to select one statement from many available statements.

Q.18. Convert the following code to an equivalent if-else statement:

```
Answer= (x >y)? x*y : x+y;
```

```
if (x>y)
  Answer = x*y;
else
  Answer = x+y;
```

Q.19. Write C++ expressions that represent the following English expressions. Assume that all variables have been properly defined and initialized.

1. x is even and y is odd (x and y are integers)
2. x is not between –47.5 and +132.0 (x is double)
3. x is either greater than 10, or less than or equal to -142
4. w is more than 200, and one or more of the following three conditions is fulfilled: x is less than 98.6; y is less than 60.0; z is more than 160.0
5. age is from 18 to 25
6. Temperature is less than 40.0 and greater than 25.0
7. Year is divisible by 4
8. Y is greater than x and less than z.
9. W is either equal to 6 or not greater than 3
10. p and q are less than zero
11. score is between 50 and 60 inclusive
12. y is equal to –5 or 5
13. age is outside the range p to q
14. digit is either equal to 50 or not less than 100
15. at least one of x or y is odd
16. The hit is no more than 6.7 units away from target
17. ch is an upper case alphabetic character (between 'A' and 'Z')

Answer:

1. (x%2 == 0) && (y%2 == 1)
2. !((-47.5 < x) && (x < 132.0)) or: !(-47.5 < x) || !(x < 132.0)
3. (x > 10) || (x <= -142)
4. (w > 200.0) && ((x < 98.6) || (y < 60.0) || (y > 160.0))
5. (age>18 && age<25)

Freebooks.pk

6. (temperature<40.0 && temperature>25.0)
7. (year%4==0)
8. (y>x && y<z)
9. ((w==6 || (!(w>3))
10. (p < 0) && (q < 0)
11. (score >= 50) && (score <= 60)
12. (y = = -5) || (y = = 5)
13. (age < p) || (age > q)
14. digit == 50 || !(digit < 100)
15. x%2 || y%2
16. ((hit - target) <= 6.7) && ((hit - target) <= -6.7)
17. ('A' <= ch) && (ch <= 'Z')

Q.20. Write C++ statements for the following:

a. **Assign the value of 1 to variable test if k is in the range -m through +m inclusive. Otherwise, assign a value of zero.**

```
if (k >= -m && k <= +m)
        test = 1;
else
        test = 0;
```

b. **Assign a value of 1 to variable lowercase if ch is a lowercase letter; otherwise, assign a value of zero.**

```
if (ch >= 97 && ch <= 122)
        lowercase = 1;
else
        lowercase = 0;
```

c. **Assigns a value of 1 to variable divisor if m is a divisor of n; otherwise assign a value of 0.**

```
if (m%n == 0)
        divisor = 1;
else
        divisor = 0;
```

Q.21. Do the following expressions evaluate to true or false?

a. (3<4) || (3>4)
b. (3!=3) && (4 == 4)
c. (15>=15) && (16==16) || (14<2)
d. 6 <= 6) && (5 < 3)
e. (6 <= 6) || (5 < 3)
f. (5 != 6)
g. (5 < 3) && (6 <= 6) || (5 != 6)
h. (5 < 3) && ((6 <= 6) || (5 != 6))
i. !((5 < 3) && ((6 <= 6) || (5 != 6)))
j. !(12 > 25) && !(18 < 17)

Answer:

a. True	b. False	c. True	d. False	e. True
f. True	g. True	h. False	i. True	j. True

Q.22. What is the output by the following code segment?

a)
```
int p = 1, q = 9;
if ((p >= 3) || (q < 11 && p + q < 15))
```

Freebooks.pk

```
        cout<<("hello\n");
      else
        cout<<("goodbye\n");
```

Answer: hello

b)
```
      int n, k = 5;
    n = (100 % k ? k + 1 : k - 1);
    cout << "n = " << n << " k = " << k << endl;
```

Answer: n = 4 k = 5

c)
```
      int found = 0, count = 5;
      if (!found || --count == 0)
        cout << "danger" << endl;
    cout << "count = " << count << endl;
```

Answer: danger
count = 5

Q.23. What is the output by the following code segment when score has the value 62?

```
      if (score < 50)
              cout << "Failing" ;
      if (score < 60)
              cout << "Below average" ;
      if (score < 70)
              cout << "Average" ;
      if (score < 80)
              cout << "Above average" ;
      if (score < 90)
              cout << "Very good" ;
      if (score < 100)
              cout << "Excellent" ;
```

Answer: **AverageAbove averageVery goodExcellent**

Q.24. What is the output of the following statement, if grade = 'B'?

```
      if (grade == 'A' || grade == 'B' && grade == 'C')
        cout << "Fail";
      else
        cout << "Pass";
```

Answer: Pass

Q.25. What is the output of the following program?

```
      int b=6, c=5;
      if(b++==7 && ++c==5)
      {
              b*=c;
              cout<<++b<<endl;
      }
      else
              cout<<b--<<endl;
```

Answer: 7

Q.26. What is the output of the following program?

```
    int n1, n2;
    n1 = 25;
    n2 = 50;
    if (n1 < n2 && !(n2 != 2 * n1))
    {
      n1 = n1 * 2;
      n2 = n2 * 2;
```

```
    cout<<n1<<n2;
  }
  if (n2 < 2 * n1 || n1 == n2)
    n1 = n1 / 2;
    n2 = n2 / 2;
  cout<<n1<<n2;
```

Answer: 501005050

Q.27. What is the output of the following?

```
int n = 0;
  if('A' == 'a')
    n = 1;
  else if('A' > 'a')
    n = 2;
  else
    n = 3;
  cout << n << endl;
```

Answer: 3

Q.28. What is the output of the following?

```
int n = 5, a = -1, b = 3;
switch((--n)%4) {
  case 0: n++;
  break;
  case 1: n *= a++;
  case 2: n +=--b;
  case 3: --n;
  default:n++;
}
cout<<n;
```

Answer: 5

Q.29. What is the output of the following?

```
int n = 5, a = 4, b = 0, c = 8;
if (b<=6 && c>4 || a++==6)
n ++;
if (a!=4 || b>5 || c<=9)
n += a++;
if (c<=2 || a>=5 && b--==0)
n *= a++;
```

Answer: 50

Q.30. What is the output of the following?

```
int a = 4, b = 2, c = 5;
if(a > b)
{
        a = 5;
}
if(c == a)
{
        a = 6;
}
else
{
        a = 7;
}
cout << a;
```

Freebooks.pk

Answer: 6

Q.31. What is the output of the following?

```
char choose = 'b';
switch (choose)
{
        case 'a':
        case 'A':
                cout << "you win!\n";
        case 'b':
        case 'B':
                cout << "you lose!\n";
        case 'c':
        case 'C':
                cout << "you draw!\n";
        default:
                cout << "I don't know you!\n";
}
```

Answer:
you lose!
you draw!
I don't know you!

Q.32. What is the output of the following?

```
int x = 5 + 7/2;
switch (x)
{
        case 5: cout << 'p'; break;
        case 7: cout << 'q'; break;
        case 8: cout << 'r'; break;
        default: cout << "No output";
}
```

Answer: r

Q.33. Write C++ code to accomplish each of the following (just the code fragment required to achieve the output - no headers, declarations, etc):

a. Display "child" if x is less than 10, "youth" if x is between 10 and 20 (inclusive) and "adult" if x is greater than 20. The statement should output "error" if x is negative or zero.

Answer:

```
if( x <= 0 )
        cout << "error" << << "\n";
else if ( x < 10 )
        cout << "child" << "\n";
else if ( x <= 20 )
        cout << "youth" << "\n";
else
        cout << "adult" << "\n";
```

b. Display "x is even" if the integer variable x is even and non-zero, if it is odd, display "x is odd" and if it is zero output "x is zero".

Answer:

```
if( x == 0 )
        cout << "x is zero" << "\n";
else if( x % 2 == 0 )
        cout << "x is even" << "\n";
else
        cout << "x is odd"  << "\n";
```

c. A passing score is 60 or higher, below 60 is considered failing.

Answer:

```
if (score >= 60)
```

Freebooks.pk

```
            cout << "Passing score.";
else
            cout << "Failing score.";
```

d. Display a message telling whether or not letter follows 'M' in the alphabet

```
if (letter > 'M' && letter <= 'Z')
            cout <<"Yes, it follows the letter 'M' in the alphabet" <<endl;
else
            cout <<"No, it does not" <<endl;
```

e. Use an if - else structure to display a student's course result. The inputs are lab_grade and theory_grade. Both are character variables and can take the values 'p' or 'f'. A student passes the course only if both lab_ grade and theory_grade are equal to 'p'.

Answer:

```
if( lab_grade == 'p' && theory_grade == 'p' )
            cout << "pass" << endl;
else
            cout << "fail" << endl;
```

f. Write a nested if statement to display a message indicating the educational level of a student based on the number of years of schooling (0 is none, 1-6 is elementary, 7-8 is middle school, 9-12 is high school, greater than 12 is college). Display a message to indicate if the data is bad as well.

```
if (level < 0)
            cout <<"Invalid data has beeen supplied" <<endl;
else if (level == 0)
            cout <<"No education level" <<endl;
else if ( level <= 6)
            cout <<"An elementary education level" <<endl;
else if (level <= 8)
            cout <<"A middle school education level" <<endl;
else if (level <= 12)
            cout <<"A high school education level" <<endl;
else
            cout <<"A college education level" <<endl;
```

Q.34. Rewrite the following if, else if, else statement as a switch statement:

a)

```
if ((x=='I') || (x=='i'))
 cout<<"Roman Number 1"<<endl;
else if ((x=='V') || (x=='v'))
 cout<<"Roman Number 2"<<endl;
else if ((x=='X') || (x=='x'))
 cout<<"Roman Number 3"<<endl;
else
 cout<<"Invalid Number"<<endl;
```

Answer:

```
switch (x)
{
 case 'I':
 case 'i':
  cout<<"Roman Number 1"<<endl;
  break;
 case 'V':
 case 'v':
  cout<<"Roman Number 2"<<endl;
  break;
 case 'X':
 case 'x':
  cout<<"Roman Number 3"<<endl;
  break;
 default:
   cout<<"Invalid Number"<<endl;
}
```

b)

```
if (option==1)
            cout<<"The option was 1";
else if (option ==2)
            cout<<The option was 2";
else if (option ==3)
            cout<<The option was 3";
else if (option ==4)
            cout<<The option was 4";
else if (option ==5)
            cout<<The option was 5";
```

Freebooks.pk

Answer:

```
switch (option)
{
 case 1:
        cout<<"The option was 1";
  break;
 case 2:
        cout<<"The option was 2";
        break;
 case 3:
        cout<<"The option was 3";
        break;
 case 4:
        cout<<"The option was 4";
        break;
 case 5:
        cout<<"The option was 5";
        break;
}
```

Q.35. Rewrite the following switch statement as if, else if, else statement:

```
switch (grade)
{
        case 'A' :
        gradePt = 4.0;
        break;
        case 'B' :
        gradePt = 3.0;
        break;
        case 'C' :
        gradePt = 2.0;
        break;
        case 'D' :
        gradePt = 1.0;
        break;
        case 'F' :
        gradePt = 0.0;
        break;
        default :
        cout << "Invalid grade";
```

Answer:

```
if (grade = 'A')
  gradePt = 4.0;
else if (grade = 'B')
  gradePt = 3.0;
else if (grade = 'C')
  gradePt = 2.0;
else if (grade = 'D')
  gradePt = 1.0;
else if (grade = 'F')
  gradePt = 0.0;
else
  cout << "Invalid grade";
```

Q.36. Find the errors in the following codes:

a.

```
if ( a > b );
  a = b;
else
  b = a;
```

b.

```
if ( a > b )
  a = b;
else
  b = a;
```

c.

```
if (a > b)
  a = b
else
```

Freebooks.pk

d.
```
    b = a;
if ( x != 0 )
  a = a / x;
switch ( number % 2 )
{
  case 0 :
    cout << " The number " << number << " is divisible by 3 " << endl ;
  default
    cout << " The number " << number << " is not divisible by 3 " << endl ;
}
```

Answer:

a. Compilation Error. Semicolon is not allowed after condition.

b. Correct

c. Semicolon missing after the statement a = b

d. Semicolon missing after the statement a = a / x

e. default statement needs a colon (:) – syntax error. case 0 needs a break statement and number % 2 must be number % 3 – logic errors

Multiple Choice

1. **The three programming structures are:**
   a. Sequence, decision, and repetition. b. Process, decision, and alternation.
   c. Sequence, definition, and process. d. Relation, comparison, and process.
2. **Which programming structure executes program statements in order?**
   a. Relation b. Decision c. Sequence d. Repetition
3. **Another term for a computer making a decision is:**
   a. Sequential. b. Selection. c. Repetition. d. Iteration.
4. **Which programming structure makes a comparison?**
   a. Relation b. Decision c. Sequence d. Repetition
5. **An expression that uses a relational operator is known as:**
   a. Condition. b. Decision. c. Series. d. Relation.
6. **Relational operators allow you to _____ numbers.**
   a. Compare b. Add c. Multiply d. Divide
7. **When a relational expression is false, it has the value _____.**
   a. Zero b. one c. Less than 0 d. None
8. **You can use a decision statement to:**
   a. Run a series of statements if a test fails. b. Test a series of conditions.
   c. Test whether a condition is true or false. d. All
9. **Operators that are used to compare operands and decide whether the relation is true or false are called:**
   a. Arithmetic operators. b. Logical operators.
   c. Syntax operators. d. Relational operators
10. **Which of the following statements is the simplest form of a decision structure?**
    a. Select...Case b. If statement c. Try...Catch...Finally d. Nested if
11. **In if statement, false is represented by:**
    a. 0 b. 1 c. 2 d. 3

Freebooks.pk

12. **In if statement, true is represented by:**
    a. 0 b. 1 c. 2 d. 3
13. **Which of the following is used for making two way decision**
    a. if-else b. if c. nested if d. switch
14. **Conditional operator is an alternative of:**
    a. if b. if-else c. nested if d. None
15. **The conditional portion of the If statement can contain:**
    a. Any valid constant.
    b. Any expression that can be evaluated to a Boolean value.
    c. Any valid variable.
    d. Any valid constant or variable.
16. **An if statement within an if statement is called:**
    a. A nested if statement. b. A branched if statement.
    c. A infinite if statement d. None
17. **What does a compound condition use to join two conditions?**
    a. Relational Operator b. Logical Operator
    c. Relational Results d. Logical Results
18. **Which is an example of multiple branches from single expression?**
    a. If statement b. switch statement c. while loop d. All
19. **Which of the following is valid case statement in a switch?**
    a. case 1: b. case x<4: c. case 'ab': d. case 1.5:
20. **A case can be defined as:**
    a. An individual condition to test. b. A set of conditions to test.
    c. A statement that includes an If statement. d. A block of selection code.
21. **The first line in switch block contains:**
    a. The value of the first criterion.
    B. The statement to be executed if the first criterion is true.
    c. The expression to be evaluated.
    d. The statement to be executed if none of the criteria are true.
22. **The Case block ends with:**
    a. End Select. b. End Case. c. break d. Case Else.
23. **Which of the following is equivalent to (p >= q)?**
    a. p < q b. !(p < q) c. p > q d. !p < q
24. **Another term for a conditional operator is:**
    a. Ternary b. Binary c. Byte d. Iteration.
25. **Conditional operator takes:**
    a. Three operands b. Two operands c. One operand d. None
26. **Which operator in C++ is called a ternary operator?**
    a. if b. ++ c. ? d. ( )
27. **All of the following are relational operators EXCEPT:**
    a. != b. ||. c. = d. >=.
28. **Which of the following returns True if A = 25 and B = 35:**
    a. A+B b. A>=B c. A!=B A=B
29. **For A = 4 and B = 4 which evaluates as true?**
    a. A <> B b. A < B c. A > B d. A >= B
30. **What operator is used in a compound condition to join two conditions?**
    a. Relational Operator b. Logical Operator c. Relational Results d. Logical Results
31. **Operators that perform operations on Boolean data are:**
    a. Arithmetic operators. b. Logical operators.
    c. Relational operators. d. Math operators.

Freebooks.pk

32. Which of the following symbol is used for OR operator?

a. || b. && c. ! d. None

33. Which of the following symbol is used for AND operator?

a. || b. && c. ! d. None

34. Which of the following symbol is used for NOT operator?

a. || b. && c. ! d. None

35. All of the following are logical operators EXCEPT:

a. && b. || c. ! d. =

36. Which of the following is not a C++ relational operator?

a. == b. < c. != d.&&

37. Which C++ logical expression correctly determines whether the value of beta lies between 0 and 100?

a. 0 < beta < 100 b. 0 < beta && beta < 100

c. (0 < beta) && (beta < 100) d. b and c above

38. If the int variables i, j, and k contain the values 3, 20, and 100, respectively, what is the value of the following logical expression:

j < 4 || j == 5 && i <= k3 < 4 || 20 == 5 && 3 < 100

a. 3 b. false c. 20 d. True

39. If p is a Boolean variable, which of the following logical expressions always has the value false ?

a. p && p b. p || p c. p && !p d. p || !p

40. What is output of the following code?

```
if ((1==1) || (2==2) || (3==3))
  cout<<"Good";
else
   cout<<"Bad";
```

a. Good b. Bad c. Goodbad d. No output

41. What is output of the following code?

```
int x=1;
int y=2;
int z=3;
if ((x==y) || (y==z) || (z==2))
  printf("YES");
else
  printf("NO");
```

a. No b. YES c. yes d. No output

42. What is output of the following code?

```
if (1==2)
  cout<<"Hello";
else if (2==1)
  cout<<"World";
```

a. No output b. Hello c. World d. Helloworld

43. After execution of the following code, what will be the value of angle if the input value is 0?

```
cin >> angle;
if (angle > 5)
   angle = angle + 5;
else if (angle > 2)
   angle = angle + 10;
else
```

Freebooks.pk

```
angle = angle + 15;
```

a. 15 b. 25 c 35 d. 40

44. What is output of the following code?

```
int p, q, r;
p = 10;
q = 3;
r = 2;
if ((p + q) < 14 && (r < q – 3))
  cout << r;
else
  cout << p;
```

a. –2 b. 4 c. 9 d. -4

45. What is output of the following code?

```
if(('A'=='A') && ('B'=='B') && ('C'=='c'))
 cout<<"Equal";
else
 cout<<"Not Equal";
```

a. Not Equal b. Equal c. No output d. Both a and b

46. What is output of the following

```
if(9<5)
cout<<"x";
else
   if(5<6)
   cout<<"#";
else
   cout<<"@";
```

a. x b. # c. @ d. None

47. What is the output of the following C++ Code for the input data 25 35 20

```
cin >> a >> b >> c;
if (a > b && a > c)
   m = a;
else if (b > c)
   m = b;
else
   m = c;
cout << m << endl;
```

a. 35 b. 25 c. 34 d. 31

48. What is the output of the following C++ Code for the input data 35 30 36 24

```
cin >> a >> b >> c >> d;
if (a > b || a > c && c != d )
   m = a;
else if (b > d)
   m = b;
else
   m = c;
cout << m << endl;
```

a. 35 b. 45 c. 31 d. 40

49. What is the opposite of the boolean expression: (1 < x && x < 100)

a. x <= 1 || 100 <= x) b. !(1 < x && x < 100)

c. Both a or b d. None

Freebooks.pk

50. **What would be the value of x if y is 10 and z is 4?**

```
switch (y - z)
{
 case 8:  x = y + z; break;
 case 9:  x = y; break;
 case 10: x = z; break;
 default: x = y * 2;
}
```

a. 20 b. 42 c. 35 d. 45

51. **Which of the following has the highest precedence?**

a. ++ b. && c. || d. -

Answers

1. a	2. c	3. b	4. b	5. a
6. a	7. a	8. d	9. d	10. c
11. a	12. b	13. a	14. b	15. a
16. a	17. a	18. b	19. a	20. d
21. b	22. c	23. b	24. a	25. b
26. c	27. b	28. c	29. d	30. d
31. b	32. a	33. b	34. c	35. d
36. d	37. d	38. d	39. c	40. a
41. a	42. a	43. a	44. a	45. a
46. c	47. a	48. a	49. c	50. a
51. a				

Fill in the Blanks

1. A statement that is executed when a particular condition is true and ignored otherwise is called _____ statement.
2. The statements that are executed in the same order in which they are written are called _____.
3. A _____ is a group of statements enclosed in braces.
4. A _____ is a pictorial representation of a program.
5. _____ are statement used to control the flow of execution in a program.
6. _____ refers to a structure with one if statement inside another if statement.
7. _____ statements can be used as an alternative to if-else if statement.
8. In switch statement the value of expression must be_____ or _____.
9. The _____statement switches the control outside the block in which it is used.
10. The purpose of_____ label in switch case statement is the same as that of else in if-else statement.
11. A condition is an expression that is either _____ or _____.
12. _____ statement executes one block of statements if the condition is true and the other if condition is false.
13. _____ statement is the simplest form of decision constructs.
14. The statements in the block of _____ statement are executed if the condition is true otherwise skipped.
15. The _____ label in switch statement is used to provide code go be executed if none of the case label is matched.
16. In nested-if structure, control enters in the inner if only when ____ condition is true.
17. _____ condition is a type of comparison in which more than one conditions are evaluated.

Freebooks.pk

18. _____ statement is best to use when there are many choices available and only one should be executed.
19. Each ____ in the switch statement represents one choice.
20. The _____ statement is used at end of each case label to exit form switch body.
21. If ____ statement is not used in switch structure, all case labels will be executed that come after the matching case block.
22. _____ operator also called ternary operator can be used in the place of simple if-else structure.
23. Ternary operator requires _____ operands.
24. Conditional operator is also called_____ operator.
25. _____ operators are used to compare two values.
26. The relational operator always evaluate to _____ or _____.
27. The relational expression is assigned a value of _____ if it is true.
28. The relational expression is assigned a value of _____ if it is false.
29. If a=1, b=2, the value of the relational expression (a>b) is_____.
30. If a=1, b=2, the value of the relational expression (a<b) is_____.
31. Operators used to combine two or more relational expression to construct compound expression are called _____ operators.
32. && is a logical operator while != is a ____.
33. C++ has three logical operators !, && and _____.
34. _____operator produces the true result only if both the conditions are true.
35. _____ operator produces the true result only if anyone of the conditions are true.
36. Logical operators have _____ precedence than relational operators.

Answers

1. Conditional	2. Sequential statements	3. Compound statement
4. Flow chart	5. Control structures	6. Nested if statement
7. switch	8. integer or character	9. break
10. default	11. true(1), false (0)	12. if-else
13. if	14. if	15. default
16. Outer	17. Logical	18. More
19. case	20. break	21. break
22. Conditional	23. three	24. Ternary
25. Relational	26. true, false	27. 1
28. 0	29. 0	30. 1
31. Logical	32. Logical NOT	33. || (Logical OR)
34. && (Logical AND)	35. || (Logical OR)	36. High

True / False

1. No statement is skipped in execution of sequential statements.
2. A logical operator is used to compare two values.
3. There are six relational operators.
4. ! is a relational operator used in C++ language.
5. >= returns false if value on left side of operator is greater than or equal to value on right side.
6. A statement that uses relational operators to compare two values is called relational expression.
7. All conditions of an If statement can evaluate to a Boolean expression.
8. The only decision structure available to a computer program is IF statement.
9. 'if statement' is used to execute a statement by checking a condition.
10. Every 'else' statement has a condition after it.

Freebooks.pk

11. 'if else' statement is a type of 'if' statement that executes one block of statement when the condition is true and executes no thing if condition is false.
12. The 'else' is attached with the far most 'if' when there are many 'if-else' used.
13. An 'if' statement within another 'else' statement is called nested if statement.
14. In nested if statement, control enters the inner if block if outer if block returns true.
15. Every If statement has two or more conditions.
16. A compound condition has at least three or more conditions joined by a logical operator.
17. Conditional operator is also called ternary operator.
18. Relational operators have less precedence than logical operators.
19. AND operator returns true if both conditions are true.
20. OR operator returns true if any one of the conditions is true.
21. != is used to reverse the results of a condition.
22. Switch is a good choice when many choices are available and at most two should be executed.
23. Missing break statements are a cause of compilation error in switch statement.
24. Default block is executed after each case of switch statement.
25. Switch statement does not work without the presence of 'default' block.
26. An arithmetic expression cannot be used as condition in if statement.
27. The conditional operator is a ternary operator.
28. Logical operators are used to compare values.
29. A switch statement cannot be used within the block of an if statement.
30. The break statement stops the execution of a program for a moment and then resumes.
31. In sequence structure, statements are executed in same order in which they appear in program.
32. A false condition always evaluates to zero.
33. Use of the statement terminator at the end of an if statement causes a syntax error.
34. The switch expression cannot be of float or double type.
35. The integer zero has a boolean value of false. The boolean value true is represented by the integer(s) non-zero integers.
36. Floating point numbers cannot be used as case values for switch statements.
37. The switch structure is used to handle multiway selection.
38. Control structure alters the normal flow of control.
39. Including space between the relational operators create syntax error.
40. Selection structure incorporates decisions in a program.
41. The result of logical expression cannot assigned to an int variable.
42. Every if statement must have a corresponding else.
43. The expression ! (a>0) is true only if a is a negative number.
44. All switch structures can be converted to if/else structures.
45. The controlling value in a switch structure can be an expression.
46. Every compound condition in an it/else structure can be translated into an if/else structure without compound conditions.
47. The controlling value in a switch structure can only be of type integer.
48. If x has the value 3, the expression (x > 3 && x < 10) returns TRUE.

Answers

1. T	2. F	3. T	4. F	5. F	6. T
7. T	8. F	9. T	10. F	11. F	12. F
13. F	14. T	15. F	16. F	17. T	18. T
19. T	20. T	21. F	22. F	23. F	24. F
25. F	26. T	27. T	28. F	29. F	30. T
31. T	32. F	33. F	34. T	35. T	36. T
37. T	38. T	39. F	40. F	41. F	42. F
43. F	44. T	45. T	46. T	47. F	48. F

Freebooks.pk

CHAPTER 6

LOOPING STRUCTURES

Chapter Overview

6.1 Loops
- 6.1.1 Counter-Controlled Loops
- 6.1.2 Sentinel-Controlled Loops

6.2 'while' Loop

6.3 'do-while' Loop
- 6.3.1 Difference between 'while' and 'do-while' Loop

6.4 Pretest and Posttest in Loops

6.5 'for' Loop
- 6.5.1 'continue' Statement
- 6.5.2 'break' Statement

6.6 Nested Loops

Programming Exercise

Exercise Questions

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

6.1 Loops

A type of control structure that repeats a statement or set of statements is known as **looping** structure. It is also known as **iterative** or **repetitive** structure. In sequential structure, all statements are executed once. On the other hand, conditional structure may execute or skip a statement on the basis of some given condition. In some situations, it is required to repeat a statement or number of statements for a specified number of times. Looping structures are used for this purpose..There are different types of loops available in C++.

Loops are basically used for two purposes:

- To execute a statement or number of statements for a specified number of times. For example, a user may display his name on screen for 10 times.
- To use a sequence of values. For example, a user may display a set of natural numbers from 1 to 10.

There are three types of loops available in C++. These are as follows:

- while loop
- do-while loop
- for loop

6.1.1 Counter-Controlled Loops

This type of loop depends on the value of a variable known as **counter variable**. The value of counter variable is incremented or decremented each time the body of the loop is executed. This loop terminates when the value of counter variable reaches a particular value. The number of iterations of counter-controlled loop is known in advance. The iterations of this loop depends on the following:

- Initial value of the counter variable
- Terminating condition
- Increment or decrement value

6.1.2 Sentinel-Controlled Loops

This type of loop depends on special value known as **sentinel value**. The loop terminates when the sentinel value is encountered. These types of loops are also known as **conditional loops**. For example, a loop may execute while the value of a variable is not –1. Here –1 is the sentinel value that is used to terminate the loop.

The number of iterations of sentinel-controlled loop is unknown. It depends on the input from the user. Suppose the sentinel value to terminate a loop is –1. If the user enters –1 in first iteration, the loop will execute only once. But if the user enters –1 after entering many other values, the number of iterations will vary accordingly. A sentinel value is commonly used with **while** and **do-while** loops.

6.2 'while' Loop

while loop is the simplest loop of C++ language. This loop executes one or more statements while the given condition remains **true**. It is useful when the number of iterations is not known in advance.

Syntax

The syntax of **while** loop is as follows:

```
while (condition)
        statement;
```

Freebooks.pk

condition The condition is given as a relational expression. The statement is executed only if the given condition is **true**. If the condition is **false**, the statement is never executed.

statement Statement is the instruction that is executed when the condition is **true**. If two or more statements are used, these are given in braces { }. It is called the **body** of the loop.

The syntax for compound statements is as follows:

```
while (condition)
{
        statement 1;
        statement 2;
        :
        :
        statement N;
}
```

Working of 'while' Loop

First of all, the condition is evaluated. If it is **true**, the control enters the body of the loop and executes all statements in the body. After executing the statements, it again moves to the start of the loop and evaluates the condition again. This process continues as long as the condition remains **true**. When the condition becomes **false**, the loop is terminated. **while** loop terminates only when the condition becomes **false**. If the condition remains **true**, the loop never ends. A loop that has no end point is known as an **infinite loop**.

Flowchart

The flowchart of **while** loop is as follows:

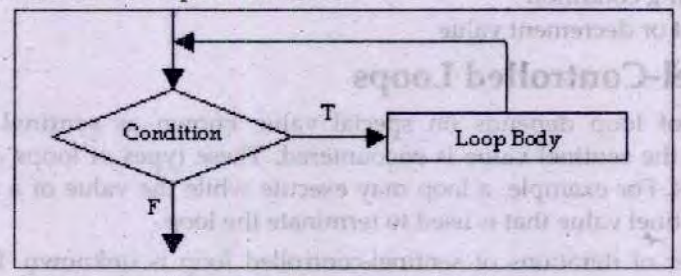


Figure 6.1: Flowchart of while loop

Program 6.1

Write a program that displays "Pakistan" for five times using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
        int n;
        clrscr();
        n = 1;
        clrscr();
        while(n<=5)
        {
                cout<<"Pakistan"<<endl;
                n++;
```

Output:
Pakistan
Pakistan
Pakistan
Pakistan
Pakistan

Freebooks.pk

```
        }
        getch();
    }
```

How above Program Works?

The above program uses a variable **n** to control the iterations of the loop. It is known as **counter variable**. The variable **n** is initialized to 1. When the condition is evaluated for the first time, the value of **n** is less than 5 so the condition is **true**. The control enters the body of the loop. The body of loop contains two statements. The first statement prints **Pakistan** on the screen. The second statement increments the value of **n** by 1 and makes it 2. The control moves back to the condition after executing both statements. This process continues for five times. When the value of **n** becomes 6, the condition becomes **false** and the loop terminates.

Program 6.2

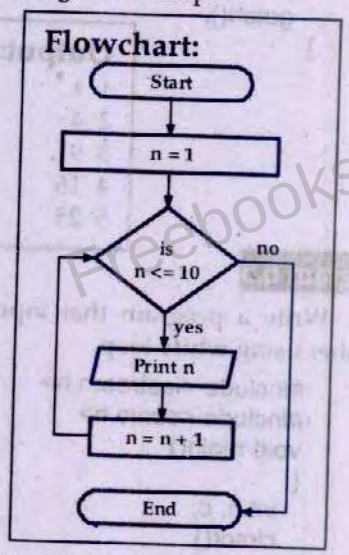
Write a program that displays counting from 1 to 10 using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  n = 1;
  clrscr();
  while(n<=10)
  {
    cout<<n<<endl;
    n++;
  }
  getch();
}
```

Output:

1
2
3
4
5
6
7
8
9
10

Flowchart:



Program 6.3

Write a program that displays first five numbers and their sum using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int c, sum;
  clrscr();
  c = 1;
  sum = 0;
  while(c <= 5)
  {
    cout<<c<<endl;
    sum = sum + c;
    c = c + 1;
  }
  cout<<"Sum is "<<sum;
  getch();
}
```

Output:

1
2
3
4
5
Sum is: 15

Freebooks.pk

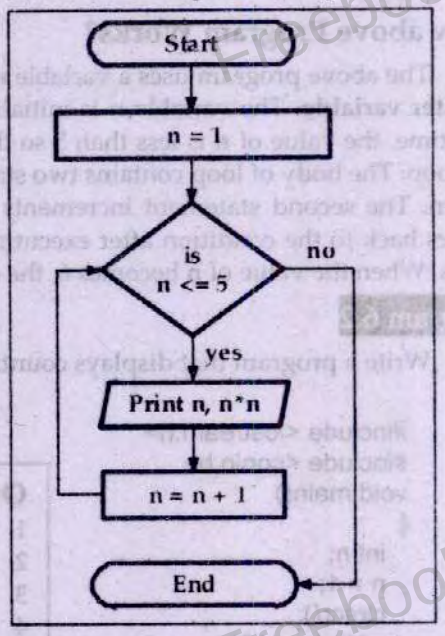
Program 6.4

Write a program that displays first five numbers with their squares using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  n = 1;
  clrscr();
  while(n<=5)
  {
    cout<<n<<" "<<n*n<<endl;
    n++;
  }
  getch();
}
```

Output:

```
1 1
2 4
3 9
4 16
5 25
```



Program 6.5

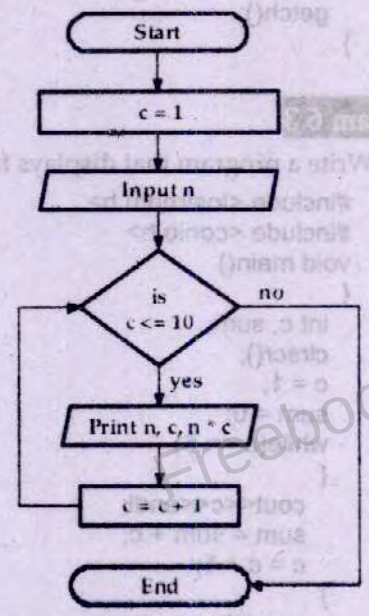
Write a program that inputs a number from the user and displays a table of that number using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n, c;
  clrscr();
  c = 1;
  cout<<"Enter a number: ";
  cin>>n;
  while(c <= 10)
  {
    cout<<n<<" * "<<c<<" = "<<n*c<<endl;
    c = c + 1;
  }
  getch();
}
```

Output:

```
Enter a number: 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
.
.
.
3 * 10 = 30
```

Flowchart:



Freebooks.pk

Program 6.6

Write a program that inputs an integer and displays the sum of its digits. For example the program should display 9 if the user enters 135.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  int x, a, r, sum = 0;
  cout<<"Enter an integer: ";
  cin>>x;
  a = x;
  while (x != 0)
  {
    r = x % 10;
    if (r == 0)
      sum = sum+x;
    else
      sum = sum+r;
    x = x / 10;
  }
  cout<<"The sum of digits of "<<a<<"= "<<sum;
  getch();
}
```

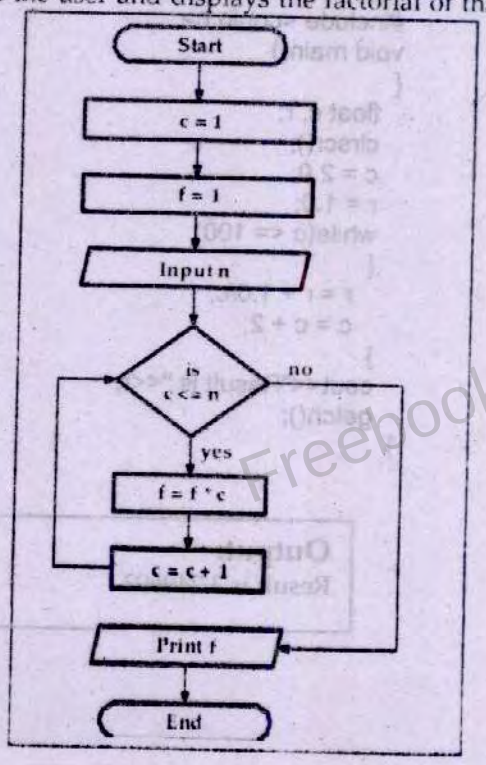
Output:
Enter an integer: 512
The sum of digits of 512 = 8

Program 6.7

Write a program that inputs a number from the user and displays the factorial of that number using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  long int n, c, f;
  clrscr();
  c = 1;
  f = 1;
  cout<<"Enter a number: ";
  cin>>n;
  while(c <= n)
  {
    f = f * c;
    c = c + 1;
  }
  cout<<"Factorial of "<<n<<" is "<<f;
  getch();
}
```

Output:
Enter a number: 8
Factorial of 8 is 40320



Freebooks.pk

Program 6.8

Write a program that displays the degree-to-radian tabel using **while** loop.

```
#include <iostream.h>
#include<conio.h>
#include <iomanip.h>
const double PI = 3.141593;
void main()
{
  clrscr();
  int degrees=0;
  double radians;
  cout.setf(ios::fixed);
  cout.precision(6);
  cout << "Degrees to Radians \n";
  while (degrees <= 360)
  {
    radians = degrees*PI/180;
    cout << setw(6) << degrees << setw(10) << radians << endl;
    degrees += 10;
  }
  getch();
}
```

Output:

Degree to Radians

0	0
10	0.174533
20	0.349066
.	
.	
.	
360	6.283186

Program 6.9

Write a program that displays the sum of the following series using **while** loop.

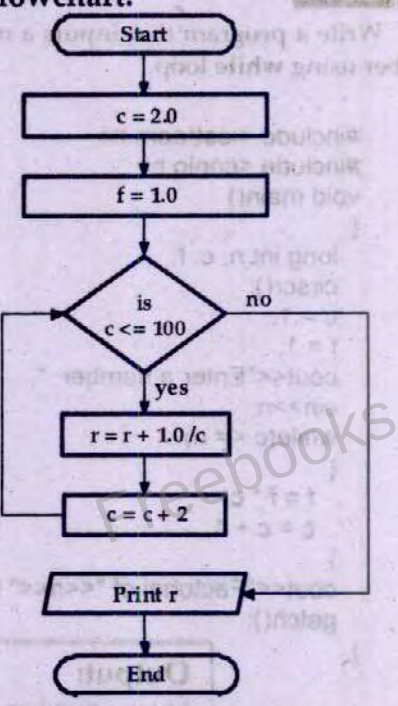
1 + 1/2 + 1/4 + 1/6 + ... + 1/100

```
#include <iostream.h>
#include <conio.h>
void main()
{
  float c, r;
  clrscr();
  c = 2.0;
  r = 1.0;
  while(c <= 100)
  {
    r = r + 1.0/c;
    c = c + 2;
  }
  cout<<"Result is "<<r;
  getch();
}
```

Output:

Result is 3.249602

Flowchart:



Freebooks.pk

Program 6.10

Write a program that inputs a positive number. It then displays the sum of all odd numbers and the sum of all even numbers from 1 to the number entered by the user.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int n,oddsum=0,evensum=0;
  clrscr();
  cout<<"Enter a positive number:";
  cin>>n;
  while (n >= 0)
  {
    if(n%2==0)
      evensum=evensum+n;
    else
      oddsum=oddsum+n;
    n--;
  }
  cout<<"The sum of even digits is:" <<evensum<<endl;
  cout<<"The sum of odd digits is:" <<oddsum;
  getch();
}
```

Output:
Enter a positive number: 5
The sum of even digits is: 6
The sum of odd digits is: 9

Program 6.11

Write a program that inputs a number and checks whether it is an Armstrong number or not. A number is an Arstrong number if the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since $3^3 + 7^3 + 1^3 = 371$

```
#include <iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  int num, n, r, sum;
  cout<<"Enter a number:";
  cin>>num;
  n = num;
  sum = 0;
  while(n != 0)
  {
    r = n % 10;
    sum = sum + (r * r * r);
    n /= 10;
  }
  if (sum == num)
    cout<<num<<" is an Armstrong number.";
  else
    cout<<num<<" is not an Armstrong number.";
  getch();
}
```

Output:
Enter a number: 371
371 is an Armstrong number.

Freebooks.pk

Program 6.12

Write a program that inputs numbers until the user enters a negative number. The program calculates the average, maximum and minimum of all positive numbers.

```
#include <iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  float num, sum;
  float avg, min, max;
  int count;
  sum = 0.0;
  count = 0;
  cout<<"Enter positive number: ";
  cin >> num;
  min = num;
  max = num;
  while (num >= 0.0)
  {
     sum += num;
     count++;
     if (num > max)
       max = num;
     else if (num < min)
       min = num;
     cout << "Enter positive number: ";
     cin >> num;
  }
  if (count == 0)
     cout << "No positive number entered: "<< endl;
  else
  {
     avg = sum / count;
     cout << "You entered "<<count<<" numbers "<<endl;
     cout << "Average = "<<avg <<endl;
     cout << "Minimum = "<<min <<endl;
     cout << "Maximum = "<<max<<endl;
  }
  getch();
}
```

Output:

```
Enter a positive number: 3
Enter a positive number: 5
Enter a positive number: 7
Enter a positive number: 9
Enter a positive number: 10
Enter a positive number: 9
Enter a positive number: -1
You entered 6 numbers.
Average = 7.166667
Minimum = 3
Maximum = 10
```

Program 6.13

Write a program that inputs a sentence from the user and counts the number of words and characters in the sentence.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int countch=0;
  int countwd=1;
```

```
    cout << "Enter a sentence: " << endl;
    char ch='a';
    while(ch!='\r')
    {
      ch=getche();
      if(ch==' ')
        countwd++;
      else
        countch++;
    }
    cout<< "\nWords = "<<countwd<<endl;
    cout<<"Characters = "<<countch-1<<endl;
    getch();
  }
```

Output:
Enter a sentence:
I love my Pakistan
Words = 4
Characters = 15

Program 6.14

Write a program that inputs starting and ending number from the user and displays all even numbers in the given range using **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n, s, e;
  clrscr();
  cout<<"Enter starting number: ";
  cin>>s;
  cout<<"Enter ending number: ";
  cin>>e;
  n = s;
  while(n<=e)
  {
    if(n%2==0)
          cout<<n<<endl;
    n++;
  }
  getch();
}
```

Output:
Enter starting number: 1
Enter ending number: 10
2
4
6
8
10

How above Program Works?

The above program inputs two numbers from the user in **s** and **e**. It uses a counter variable **n** to control the iterations of the loop. The value of **s** is assigned to variable **n**. The number of iterations depends on the values entered in **s** and **e**. For example, if the user enters 11 in **s** and 20 in **e**, loop will execute for ten times and program displays 12, 14, 16, 18 and 20.

Program 6.15

Write a program that uses a **while** loop to enter number from the user and then display it. The loop is terminated when the user enters -1.

```
#include <iostream.h>
#include <conio.h>
void main()
{
```

Freebooks.pk

```
    int n;
    n = 1;
    while(n != -1)
    {
      cout<<"Enter a number: ";
      cin>>n;
      cout<<"You entered "<<n<<endl;
    }
    cout<<"End of Program";
    getch();
  }
```

Output:
Enter a number: 5
You entered 5
Enter a number: 10
You entered 10
Enter a number: -1
You entered -1
End of Program…

How above Program Works?

The above example uses –1 to terminate the loop. The variable **n** is initialized to 1. First of all, the condition is evaluated and the control enters the body of the loop because the condition is **true**. The first statement in the body gets the input from the user and prints that value on the screen. Then the control moves the beginning of the loop and the condition is evaluated again. This process continues and the value entered by the user is displayed. The loop is terminated when the user enters -1.

Program 6.16

Write a program that inputs a number from user and displays **n** fibonacci terms. In Fibonacci sequence, sum of two successive terms gives the third term.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int a,b,next,n,count;
  clrscr();
  cout<<"How many Fibonacci terms required:";
  cin>>n;
  a=0;
  b=1;
  cout<<"Fibonacci terms are"<<endl;
  cout<<a<<"\t"<<b;
  count=2;
  while(count<n)
  {
    next=a+b;
    cout<<"\t"<<next;
    count++;
    a=b;
    b=next;
  }
  getch();
}
```

Output:
How many Fibonacci terms required: 6
Fibonacci terms are:
0 1 1 2 3 5

Program 6.17

Write a program that inputs a number and checks if it is a Fibonacci number or not.

```
#include <iostream.h>
#include<conio.h>
void main()
```

Freebooks.pk

```
{
  clrscr();
  int a,b,next,n;
  cout<<"Enter the number :";
  cin>>n;
  if ((n==0) || (n==1))
    cout<<n<<" is a Fibonacci number.";
  else
  {
     a=0;
     b=1;
     next=a+b;
  while(next<n)
  {
    a=b;
    b=next;
    next=a+b;
  }
  if (next==n)
    cout<<n<<" is Fibonacci number.";
  else
    cout<<n<<" is not a Fibonacci number.";
  }
  getch();
}
```

Output:
Enter a number: 8
8 is a Fibonacci number.

6.3 'do-while' Loop

do-while is an **iterative** control in C++ language. This loop executes one or more statements while the given condition is **true**. In this loop, the condition comes after the body of the loop. It is an important loop in a situation when the loop body must be executed at least once.

Syntax

The syntax of while loop is as follows:

```
do
{
        statement 1;
        statement 2;
        :
        :
        statement N;
}
while (condition);
```

do — It is the keyword that indicate the beginning of the loop.

Condition — The condition is given as a relational expression. The statement is executed only if the given condition is **true**. If the condition is **false**, the statement is never executed.

Statement — Statement is the instruction that is executed when the condition is **true**. Two or more statements are written in braces {}. It is called the **body** of the loop.

Freebooks.pk

Working of 'do-while' Loop

First of all, the body of loop is executed. After executing the statements in loop body, the condition is evaluated. If it is **true**, the control again enters the body of the loop and executes all statements in the body again. This process continues as long as the condition remains **true**. The loop terminates when the condition becomes **false**. This loop is executed at least once even if the condition is **false** in the beginning.

Flowchart

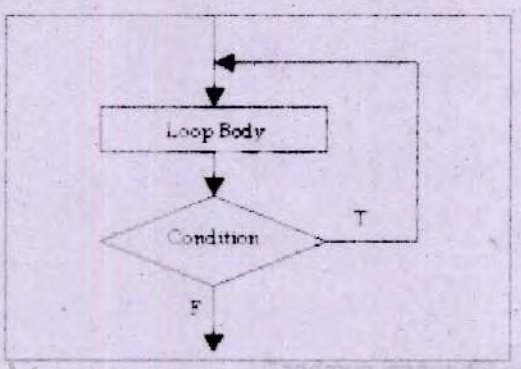


Figure 6.2: Flowchart of do-while loop

6.3.1 Difference between 'while' and 'do-while' Loop

A difference between 'while' and 'do-while' loop is as follows:

while loop	do-while loop
In while loop, condition comes before the body of the loop.	In do-while loop, condition comes after the body of the loop.
If condition is false in the beginning, while loop is never executed.	do-while is executed at least once even if condition is false in the beginning.

Program 6.18

Write a program that displays back-counting from 10 to 1 using **do-while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int c;
  clrscr();
  c = 10;
  do
  {
    cout<<c<<endl;
    c = c - 1;
  }
  while(c>=1);
  getch();
}
```

Output:

```
10
9
8
7
6
5
4
3
2
1
```

Freebooks.pk

Program 6.19

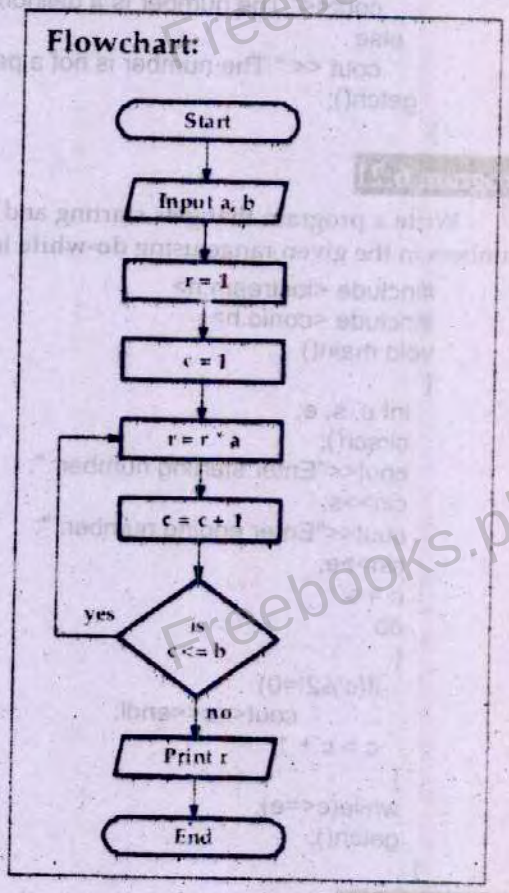
Write a program that gets two numbers from the user and displays the result of first number raise to the power of second number using **do-while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, c, r;
  clrscr();
  cout<<"Enter first number: ";
  cin>>a;
  cout<<"Enter second number: ";
  cin>>b;
  c = 1;
  r = 1;
  do
  {
    r = r * a;
    c = c + 1;
  }
  while(c<=b);
  cout<<"Result is "<<r;
  getch();
}
```

Output:

Enter first number: 2
Enter second number: 3
Result is 8

Flowchart:



Program 6.20

Write a program that inputs a number and checks whether it is a palindrome or not. A palindrome is a number that reads the same backwards as forwards such as 62526 and 4994.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  long int n, num, digit, rev = 0;
  cout << "Enter a positive number: ";
  cin >> num;
  n = num;
  do
  {
    digit = num%10;
    rev = (rev*10) + digit;
    num = num/10;
  }
```

Output:

Enter a positive number: 62526
The reverse of the number is 62526
The number is a palindrome

Freebooks.pk

```
    while (num!=0);
      cout<<" The reverse of the number is: " << rev << endl;
    if (n == rev)
      cout<<" The number is a palindrome";
    else
      cout << " The number is not a palindrome";
  getch();
}
```

Program 6.21

Write a program that gets starting and ending point from the user and displays all odd numbers in the given range using **do-while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int c, s, e;
  clrscr();
  cout<<"Enter starting number: ";
  cin>>s;
  cout<<"Enter ending number: ";
  cin>>e;
  c = s;
  do
  {
    if(c%2!=0)
          cout<<c<<endl;
    c = c + 1;
  }
  while(c<=e);
  getch();
}
```

Output:
Enter starting number: 5
Enter ending number: 15
5
7
9
11
13
15

Program 6.22

Write a program that reads the current state of a telephone line. The user should enter **w** for working state and **d** for dead state. Any other input should be invalid. Use do-while loop to force the user to enter a valid input value.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char s;
  clrscr();
  do
  {
    cout<<"Enter the state of phone (\'w\' for working \'d\' for dead): ";
    cin>>s;
  }
  while(s != 'w' && s != 'd');
  getch();
}
```

Output:
Enter the state of phone ('w' for working 'd' for dead): a
Enter the state of phone ('w' for working 'd' for dead): m
Enter the state of phone ('w' for working 'd' for dead): w
Telephone is working.

Freebooks.pk

How above Program Works?

The above program does not proceed until the user enters a valid input of **d** or **w**. The program repeatedly shows a message to the user to enter a valid input. The condition indicates that the loop continues if the user enters a value other than **w** and **d**.

6.4 Pretest and Posttest in Loops

A **pretest** is a condition that is used to test for the completion of loop at the top of loop. In this type of loop, the statements inside the loop body can never execute if the terminating condition is **false** the first time it is tested.

Example

Following is an example of pretest condition. The statements in the body of loop cannot be executed if the condition is **false**.

```
n = 0;
while(n == 0)
{
  loop body
}
```

A **posttest** is a condition that is used to test for the completion of loop at the bottom of loop. It means that the statements in the loop will always be executed at least once.

Example

Following is an example of posttest condition. The statements in the body of loop will be executed at least once even if the condition is **false**.

```
do
{
  loop body
}
while(n == 0)
```

6.5 'for' Loop

for loop executes one or more statements for a specified number of times. This loop is also called **counter-controlled loop**. It is the most flexible loop. That is why, it is the most frequently-used loop by the programmers.

Syntax

The syntax of **for** loop is as follows:

```
for (initialization; condition; increment/decrement)
{
        statement 1;
        statement 2;
        :
        statement N;
}
```

Initialization It specifies the starting value of counter variable. One or many variables can be initialized in this part. To initialize many variables, each variable is separated by comma.

Freebooks.pk

Condition The condition is given as a relational expression. The statement is executed only if the given condition is **true**. If the condition is **false**, the statement is never executed.

Increment/decrement This part of loop specifies the change in counter variable after each execution of the loop. To change many variables, each variable must be separated by comma.

Statement Statement is the instruction that is executed when the condition is **true**. If two or more statements are used, these are given in braces { }. It is called the **body** of the loop.

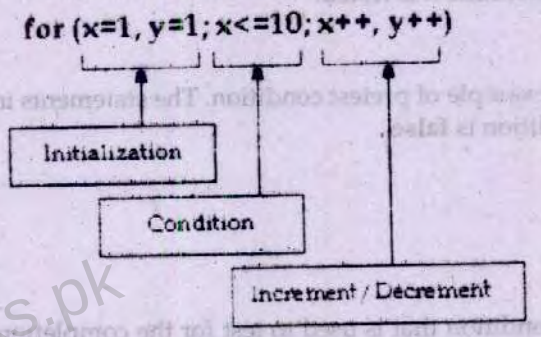


Figure 6.3: Parts of for loop

Working of 'for' Loop

The number of iterations depends on the initialization, condition and increment/decrement parts. The initialization part is executed only once when control enters the loop. After initialization, the given condition is evaluated. If it is **true**, the control enters the body of loop and executes all statements in it. Then the increment/decrement part is executed that changes the value of counter variable. The control again moves to condition part. This process continues while the condition remains **true**. The loop is terminated when the condition becomes **false**.

Flowchart

The flowchart of **for** loop is as follows:

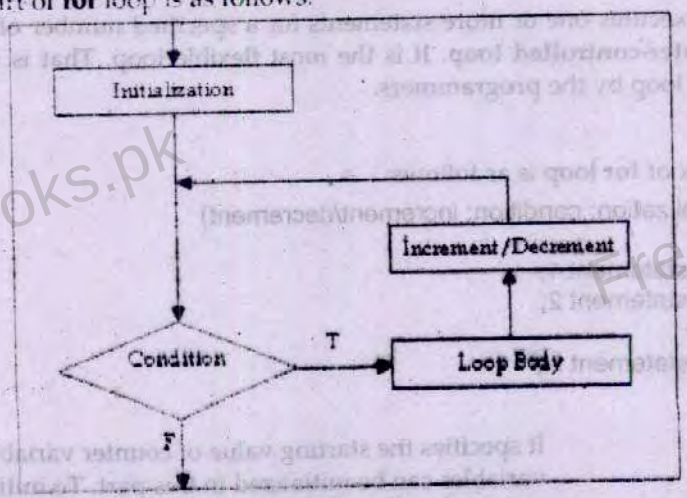


Figure 6.4: For ... Next loop

Freebooks.pk

Program 6.23

Write a program that displays counting from 1 to 5 using **for** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  clrscr();
  for(n=1; n<=5; n++)
      cout<<n<<endl;
  getch();
}
```

Output:

```
1
2
3
4
5
```

How above Program Works?

In the above example, the variable **n** is initialized 1. The termination condition is **n <= 10**. It means that the loop will continue while the value of counter is less than or equal to 10. The loop will terminate when the value of counter is greater than 10. The third part contains **n++**. It indicates that the value of **n** will be incremented by 1 after each execution of the loop.

Program 6.24

Write a program that displays product of all odd numbers from 1 to 10 using **for** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  long int product = 1;
  int c;
  clrscr();
  for(c=1; c<=10; c=c+2)
        product *= c;
  cout<<"Result is "<<product;
  getch();
}
```

Output:

```
Result is: 945
```

Program 6.25

Write a program that inputs table number and length of table and then displays the table using **for** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int tab, len, c;
  cout<<"Enter number for table: ";
  cin>>tab;
  cout<<"Enter length of table: ";
  cin>>len;
  for(c=1; c<=len; c++)
    cout<<tab<<" * "<<c<<" = "<<tab*c<<endl;
  getch();
}
```

Output:

```
Enter number for table: 2
Enter number for table: 8
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
```

Freebooks.pk

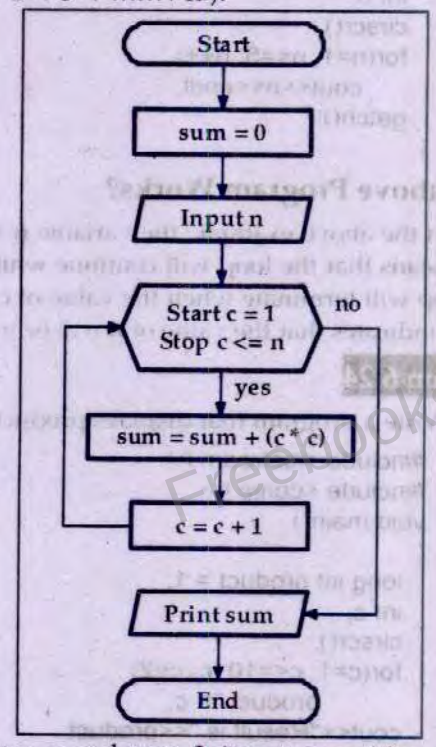
Explanation

The above program uses a variable **tab** for table instead of a constant number. It gets a number from the user as a table. It also inputs a number in variable **len** up to which the table is displayed. It execute the loop for **len** times by using the condition **c <= len**.

Program 6.26

Write a program that finds the sum of the squares of integers from 1 to n. Where **n** is a positive value entered by the user (i.e. Sum = $1^2 + 2^2 + 3^2 + \ldots\ldots + n^2$).

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n, c;
  long int sum;
  clrscr();
  sum = 0;
  cout<<"Enter a number: ";
  cin>>n;
  for(c=1; c<=n; c++)
    sum = sum + (c * c);
  cout<<"Sum is "<<sum;
  getch();
}
```



Output:
Enter a number: 5
Sum of squares: 55

Explanation

The above program declares three variables **c**, **n** and **sum**. It inputs a positive number from the user and stores this number in the variable **n**. In the initialization part of loop, **c** is initialized to 1. The condition statement **c<=n** uses **n** that is entered by the user. The loop will execute **n** times. The increment part increments the counter variable by 1.

In each iteration, the statement **sum = sum + (c * c)** is executed. It takes the square of the counter variable **c** and adds it to the variable **sum**. The loop runs **n** times and the squares of numbers from 1 to **n** are summed up. After completing **for** loop the **cout** statement is executed that displays the sum of the squares of number from 1 to **n**.

Program 6.27

Write a program that inputs a number from the user and prints the lowest and highest digit in the number.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int n;
  clrscr();
```

Freebooks.pk

```
    cout<< "\n Enter a number ";
    cin>> n;
    int num = n;
    int high = n%10, low = n%10, rem;
    n = n/10;
    for(int i = n; i>=1; i=i/10)
    {
      rem =  i%10;
      if (rem>high)
        high = rem;
      if (rem < low)
        low = rem;
    }
    cout<< "\n The highest digit of "<< num <<" is " << high;
    cout<< "\n The lowest digit of "<< num <<" is " << low;
    getch();
  }
```

Output:
Enter a number: 4591
The highest digit of 4591 is 1
The lowest digit of 4591 is 9

Program 6.28

Write a program that inputs the value of x and range. It then calculates and prints the sum of the following series:

1 + 1/x + 1/x^2 + 1/x^3 + ...

```
#include <iostream.h>
#include <math.h>
#include<conio.h>
void main ()
{
  clrscr();
  int i,n,x;
  float sum=0,den;
  cout<<"\nEnter the value of x : ";
  cin>>x;
  cout<<"\nEnter the range ; ";
  cin>>n;
  for (i=0;i<n;i++)
  {
    den=pow (x,i);
    sum = sum + ( 1/den);
  }
  cout<<"Sum of series:"<<sum;
  getch();
}
```

Output:
Enter the value of x: 2
Enter the range: 5
Sum of series: 1.9375

Program 6.29

Write a program to print the following series:

1 4 7 10 ... 40

```
# include<iostream.h>
#include<conio.h>
void main()
{
  int a = 1,i , incre = 3;
```

Freebooks.pk

```
    clrscr();
    cout<<"\n The Series is as follows:";
    for( i = 1 ; a<=40; i++)
    {
      cout<< a << " " ;
      a = a+incre;
    }
    getch();
  }
```

Program 6.30

Write a program to print the following series:

1 -4 7 -10 ..... -40

```
# include<iostream.h>
#include<conio.h>
void main()
{
  int a = 1, i,p , incre = 3;
  cout<<"\n The Series:";
  for( i = 1 ; a<=40; i++)
  {
    if(i%2 == 0)
    {
      p=-a;
      cout<< p << " " ;
    }
    else
      cout<< a << " ";
    a = a+incre;
  }
  getch();
}
```

Program 6.31

Write a program that inputs a number and checks whether it is a perfect number or not. A perfect number is a number that is numerically equal to the sum of its divisors. For example, 6 is a perfect number because the divisors of 6 are 1, 2, 3 and 1+2+3 = 6.

```
#include <iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  int i, num, mid, sum=0;
  cout<<"Enter the number:";
  cin>>num;
  mid = num / 2;
  for(i=1;i<=mid;i++)
  {
    if ((num%i)==0)
      sum=sum+i;
  }
```

Output:
Enter a number: 6
6 is a perfect number

Freebooks.pk

```
    if (sum==num)
      cout<<num<<"is a perfect number";
    else
      cout<<num<<"is not a perfect number";
    getch();
  }
```

Program 6.32

Write a program that inputs an integer and prints if it is prime or composite. A number is prime if it has factors 1 and itself, otherwise it is a composite number.

```
#include <iostream.h>
#include<conio.h>
void main()
{
  int c, num, p = 1;
  clrscr();
  cout<<"Enter an integer: ";
  cin>>num;
  for(c=2; c<=num/2; c++)
    if(num%c==0)
    {
        p = 0;
        break;
    }
  if(p==1)
    cout<<num<<" is a prime number.";
  else
    cout<<num<<" is a composite number.";
  getch();
}
```

Output:
Enter an integer: 97
97 is a prime number.

6.5.1 'continue' Statement

The **continue** statement is used in the body of the loop. It is used to move the control to the start of the loop body. When this statement is executed in the loop body, the remaining statements of current iteration are not executed. The control directly moves to the next iteration.

Example

```
int x;
for(x=1; x<=5; x++)
{
  cout<<"Hello World! \n";
  continue;
  cout<<"Knowledge is power.";
}
```

Explanation

The above example has two **cout** statements. One statement is before the **continue** statement and one is after **continue** statement. The second statement is never executed. This is because each time **continue** statement is executed, the control moves back to the start of the body. So "Knowledge is power" is never displayed.

Freebooks.pk

Program 6.33

Write a program that inputs a number from the user using **for** loop. It displays the number if it is greater than 0 otherwise it inputs next number using **continue** statement.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int x, num;
  for(x=1; x<=5; x++)
  {
    cout<<"Enter a number: ";
    cin>>num;
    if(num<=0)
          continue;
    cout<<"You entered "<<num<<endl;
  }
  getch();
}
```

Output:
Enter a number: 5
You entered 5
Enter a number: -1
Enter a number: 0
Enter a number: -5
Enter a number: 100
You entered 100

Program 6.34

Write a program that displays the sum of the following series:

1+3+5+7+..100

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int sum = 0;
  for(int i=1;i<100;i++)
  {
    if(i % 2 == 0)
      continue;
    sum = sum + i;
  }
  cout<<"The sum is "<<sum;
  getch();
}
```

Output:
The sum is 2500

6.5.2 'break' Statement

The **break** statement is used in the body of the loop to exit from the loop. When this statement is executed in the loop body, the remaining iterations of the loop are skipped. The control directly moves outside the body and the statement that comes after the body is executed.

Example

```
for(int x=1; x<=5; x++)
{
  cout<<"Questioning\n";
  break;
  cout<<"Gateway to knowledge";
}
cout<<"Bye";
```

Freebooks.pk

Explanation

The above program uses **break** statement in **for** loop. The counter variable x indicates that the loop should execute for five times. But it is executed only once. In first iteration, the **cout** statement in the loop displays "Questioning" and control moves to **break** statement. This statement moves the control out of the loop. So the message appears only once.

Program 6.35

Write a program that inputs number from the user using **for** loop. If the number is greater than 0, it is displayed and the next number is input. The program exits the loop if the number is 0 or negative using **break** statement.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int x, num;
  for(x=1; x<=5; x++)
  {
    cout<<"Enter a number: ";
    cin>>num;
    if(num<=0)
        break;
    cout<<"You entered "<<num<<endl;
  }
  getch();
}
```

Output:
Enter a number: 5
You entered 5
Enter a number: 10
You entered: 10
Enter a number: -5

Explanation

The above program uses **for** loop to get five numbers from the user. If the number is greater than 0, it is displayed on the screen. Otherwise, the loop is terminated and the control moves out of the body.

Program 6.36

Write a program that inputs two numbers and displays the greatest common divisor of both numbers.

```
#include <iostream.h>
#include<conio.h>
void main()
{
  int n1, n2, d;
  cout << "Enter the first number: ";
  cin >> n1;
  cout << "Enter the second number: ";
  cin >> n2;
  d = (n1 < n2) ? n1 : n2;
  for ( ; d >= 1; d--)
    if ((n1 % d == 0) && (n2 % d == 0))
      break;
  cout << "GCD of " << n1 << " and " << n2 << " is " << d;
  getch();
}
```

Output:
Enter the first number: 18
Enter the second number: 27
GCD of 18 and 27 is 9

Freebooks.pk

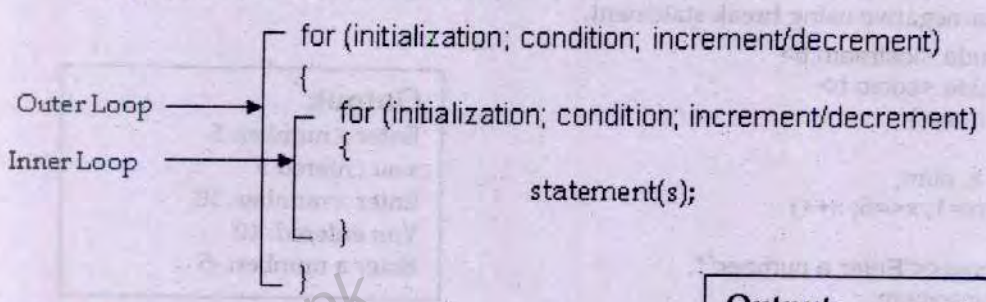
6.6 Nested Loops

A loop within a loop is called **nested loop**. In nested loops, the inner loop is executed completely with each change in the value of counter variable of outer loop.

The nesting can be done up to any level. The increase in level of nesting increases the complexity of **nested loop**. Any loop can be used as inner loop of another loop. For example, **while** loop can be used as outer loop and **for** loop can be used as inner loop in nested loop.

Syntax

The syntax of nested loop is as follows:



Example

```
int i, j;
for(i=1; i<=2; i++)
    for(j=1; j<=3; j++)
        cout<<"Outer: "<<i<<" Inner: "<<j<<endl;
```

Output:
Outer: 1 Inner: 1
Outer: 1 Inner: 2
Outer: 1 Inner: 3
Outer: 2 Inner: 1
Outer: 2 Inner: 2
Outer: 2 Inner: 3

Explanation

The above example uses nested **for** loop. The outer loop executes two times and the inner loop executes three times with each iteration of outer loop. It means that the inner loop executes six times in total. When the value of **i** is 1 in first iteration of outer loop, the value of **j** changes from 1 to 3 in three iterations of inner loop. Similarly, when the value of **i** is 2 in second iteration of outer loop, the value of **j** again changes from 1 to 3 in three iterations of inner loop.

Example

The following program explains the use of nested **while** loops.

```
int i, j;
i = 1;
while(i<=2)
{
    j = 1;
    while(j<=3)
    {
        cout<<"Outer: "<<i<<" Inner: "<<j<<endl;
        j++;
    }
    i++;
}
```

Output:
Outer: 1 Inner: 1
Outer: 1 Inner: 2
Outer: 1 Inner: 3
Outer: 2 Inner: 1
Outer: 2 Inner: 2
Outer: 2 Inner: 3

Freebooks.pk

Program 6.37

Write a program that displays the product components of a number wihtout repeating them. For example, if the user enters 24, it displays 24*1, 12*2, 8*3 and 4*6.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int n,i,j, s = 1;
  cout<<"\n Enter a number:";
  cin>>n;
  clrscr();
  cout<<"\n Product Components of "<< n << " are\n";
  for(i = n; i>= s;i--)
    for(j = 1; j<=n;j++)
    {
      if(i*j == n)
      {
        cout<< i << "*" << j << endl;
        s = j+1;
      }
    }
  getch();
}
```

Output:
Enter a number: 24
Product Components of 24 are:
24*1
12*2
8*3
6*4

Program 6.38

Write a program that displays the following output:

```
#include <iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  int i, j, sum;
  for(i=1; i<=5 ;i++)
  {
    cout<<'1';
    sum = 1;
    for(j=2; j<=i ;j++)
    {
      cout<<'+'<<j;
      sum = sum + j;
    }
    cout<<"="<<sum<<endl;
  }
  getch();
}
```

```
1 = 1
1 + 2 = 3
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
```

Program 6.39

Write a program that inputs the hieght of triangle and displays a triangle of characters. For example, if the user enters 5, it displays the following:

Freebooks.pk

```
#include<iostream.h>
#include<conio.h>
void main()
{
  char ch='A';
  int n,i,j;
  clrscr();
  cout<< "\n Enter the height of triangle :";
  cin>> n;
  for(i=1; i<= n ; i++)
  {
    for(j= 1;j<=i;j++)
    {
      cout<< ch<< " ";
      ch++;
    }
    cout<< "\n";
  }
  getch();
}
```

```
Enter the height of triangle: 5
A
B C
D E F
G H I J
K L M N O
```

Program 6.40

Write a program that displays the following shape using nested loops. The outer loop should be for loop and inner loop should be **while** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int i, j;
  clrscr();
  for(i=1; i<=7; i++)
  {
    j = i;
    while( j<=7 )
    {
        cout<<"*";
        j++;
    }
    cout<<"\n";
  }
  getch();
}
```

```
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

Program 6.41

Write a program that displays the following block using nested **for** loop.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int m, n;
  clrscr();
```

```
for(m=1; m<=5; m++)
{
  for(n=1; n<=5; n++)
        cout<<"*";
  cout<<endl;
}
getch();
}
```

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

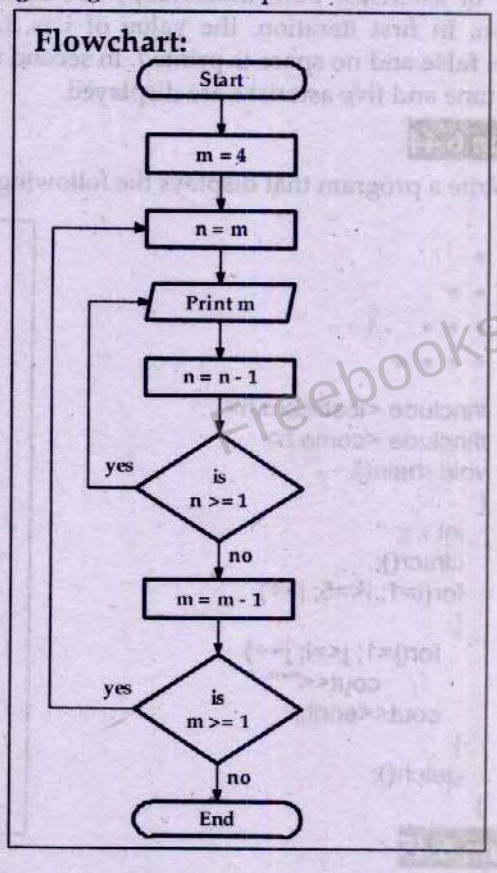
Program 6.42

Write a program that displays the following using **do-while** loop.

```
4    4    4    4
3    3    3
2    2
1
```

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int m, n;
  clrscr();
  m = 4;
  do
  {
    n = m;
    do
    {
        cout<<m<<"\t";
        n--;
    }
    while(n>=1);
    cout<<endl;
    m--;
  }
  while(m>=1);
  getch();
}
```

Flowchart:



Program 6.43

Write a program that displays the following shape using nested **for** loops.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int i, j, s;
  clrscr();
  for(i=5; i>=1; i--)
  {
    for(s=1; s<=5-i; s++)
        cout<<" ";
```

```
* * * * *
 * * * *
  * * *
   * *
    *
```

Freebooks.pk

```
        for(j=1; j<=i; j++)
            cout<<"*";
        cout<<endl;
    }
    getch();
}
```

How above Program Works?

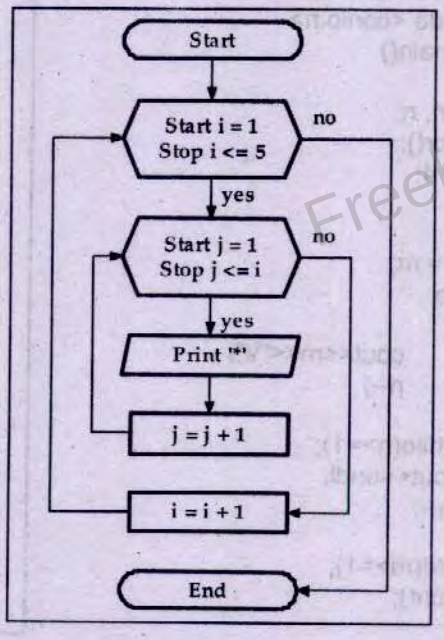
The above program uses three **for** loops. The outer loop runs five times. The first inner loop displays required number of spaces and the second inner loop displays required number of asterisks. Both inner loops use counter variable of outer loop in termination condition. In first iteration, the value of **i** is 5. The condition **s<=5-1** in first inner loop becomes **false** and no space is printed. In second inner loop, the condition **j<=i** executes loop for five time and five asterisks are displayed.

Program 6.44

Write a program that displays the following shape using nested **for** loops.

```
*
* *
* * *
* * * *
* * * * *
```

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int i, j;
  clrscr();
  for(i=1; i<=5; i++)
  {
    for(j=1; j<=i; j++)
        cout<<"*";
    cout<<endl;
  }
  getch();
}
```



Program 6.45

Write a program that displays the following output using loops.

```
#include <iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  int i,j;
  for (i = 1; i <= 6; i++)
  {
    for (j = i; j > 1; j--)
      cout << " ";
    for (j = 1; j <= 6 + 1 - i; j++)
      cout << j << " ";
```

```
1 2 3 4 5 6
  1 2 3 4 5
    1 2 3 4
      1 2 3
        1 2
          1
```

Freebooks.pk

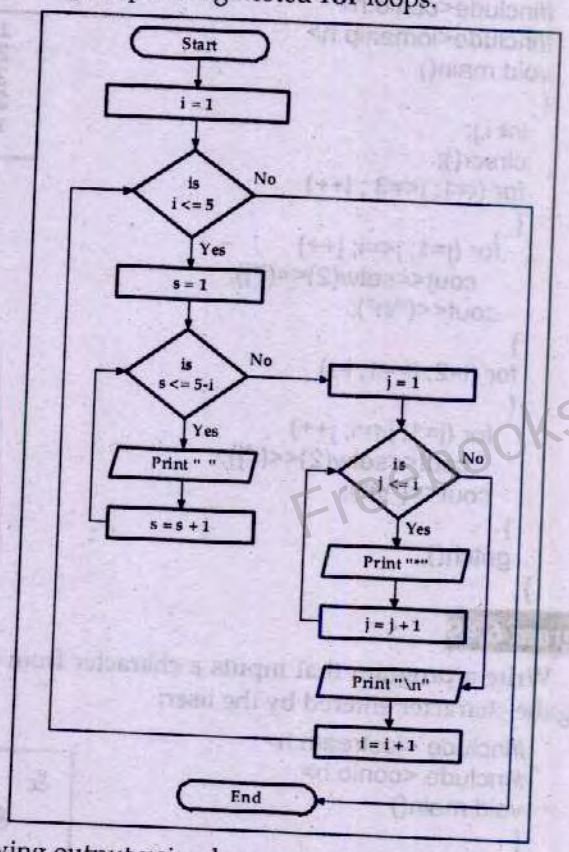
```
    cout << endl;
    }
    getch();
  }
```

Program 6.46

Write a program that displays the following shape using nested **for** loops.

```
    *
   * *
  * * *
 * * * *
* * * * *
```

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int i, j, s;
  clrscr();
  for(i=1; i<=5; i++)
  {
    for(s=1; s<=5-i; s++)
        cout<<" ";
    for(j=1; j<=i; j++)
        cout<<"*";
    cout<<endl;
  }
  getch();
}
```



Program 6.47

Write a program that displays the following output using loops.

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main()
{
  clrscr();
  int i, j, k, c = 5;
  for (i = 1; i <= 5; i++)
  {
    for (k = 1; k <= c; k++)
      cout<<(" ");
    for (j = 1; j <= i; j++)
      cout<<setw(2)<<i;
    cout<<("\n");
    c--;
  }
```

Freebooks.pk

```
    getch();
}
```

Program 6.48

Write a program that displays the following output:

```
1
2 4
3 6 9
2 4
1
```

```
#include <iostream.h>
#include<conio.h>
#include<iomanip.h>
void main()
{
  int i,j;
  clrscr();
  for (i=1; i<=3 ; i++)
  {
    for (j=1; j<=i; j++)
      cout<<setw(2)<<(i*j);
    cout<<("\n");
  }
  for (i=2; i>=1; i--)
  {
    for (j=1; j<=i; j++)
      cout<<setw(2)<<(i*j);
    cout<<("\n");
  }
  getch();
}
```

Program 6.49

Write a program that inputs a character from the user and draws the following shape using the character entered by the user:

```
&                 &
  &             &
    &         &
      &     &
        &
```

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char choice;
  int x, y, count;
  cout << "\nEnter any character: ";
  cin >> choice;
  count=1;
  for (x=0;x<5;++x)
  {
    cout << endl;
    for (y=0;y<5*2-1;++y)
        if (y==x || y==((5*2-1)-count))
          cout << choice;
        else
          cout << " ";
        ++count;
    }
    getch();
}
```

Program 6.50

Write a program that displays all possible combinations of 1, 2 and 3.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  int i, j, k;
  for (i=1; i<=3; i++)
  {
    for (j=1; j<=3; j++)
    {
      for (k=1; k<=3; k++)
        cout<<i<<j<<k<<"\t";
    }
  }
  getch();
}
```

Output:

```
111     112     113     121     122     123     131     132     133     211
212     213     221     222     223     231     232     233     311     312
313     321     322     323     331     332     333
```

Programming Exercises

1. Write a program to display the following format using **while** loop:

```
a        b
---------
1        5
2        4
3        3
4        2
5        1
---------
```

2. Write a program to display the following format using **while** loop:

```
---------
num      sum
---------
1        1
2        3
3        6
4        10
5        15
---------
```

3. Write a program that displays the sum of the following series using do-while loop.
   1 + 1/4 + 1/8 + … + 1/100
4. Write a program to display alphabets from A to Z using **for** loop.

Freebooks.pk

5. Write a program to find the largest, smallest, and average of n whole numbers. You can assume that "n" has already been set by the user.
6. Write a program that will ask the user a question with four possible answers. The question should be asked 20 times. After all the input is gathered, the program should output the number of times each answer was selected.
7. Write a program that inputs a series of 20 numbers and displays the minimum value.
8. Write a program that inputs a number from the user and displays Fibonacci series up to the number entered.
9. Write a program that inputs a number from the user and displays all Armstrong numbers up to the number entered.
10. Write a program that inputs a number from the user and displays all perfect numbers up to the number entered.
11. Write a program that inputs the number of students in the class. It then inputs the marks of these students and displays the highest and second highest marks.
12. Write a program that calculates and prints the average of several integers. Assume that the last value read is sentinel 9999. A typical input sequence might be 10 8 6 7 2 9999 indicating that average of all values preceding 9999 is required.
13. Write a program that inputs a number from the user and displays all prime numbers which are less than the input number using any loop.
14. Write a program that inputs a number from the user and displays its factorial. It asks the user whether he wants to calculate another factorial or not. If the user inputs 1, it again inputs number and calculates factorial. If user inputs 0, program terminates.
15. Write a program that inputs an integer and displays whether it is a prime number or not.
16. Write a program that continuously inputs positive integer values from the user. The user enters a zero to show that he has no more values to enter. The program should finally display the second largest number entered.
17. Write a program that takes n numbers as input. It displays total positive and negative numbers.
18. Write a program to calculate and display sum of the following series using **for** loop:
    $x + x^2 + x^3 \ldots x^n$
19. Write a program to calculate and display sum of the following series using **for** loop:
    $1! + 2! + 3! + 4! + 5!$
    Where the symbol "!" represents the factorial of the number.
20. Write a program to calculate and display the sum of the following series using **for** loop:
    $1 + 2x + 3x^2 + 4x^3 + 5x^4$
21. Write a program to calculate and display the sum of the following series using **for** loop:
    $1/2 + 2/3 + 3/4 + \ldots + 99/100$
22. Write a program to print the following sequence:
    64 32 16 8 4 2
23. Write a program to print the following sequence:
    1 3 9 27 81 ... 200
24. Write a program to print the following sequence:
    8 12 17 24 28 33 ... 100
25. Write a program to add the first seven terms of the following series using **for** loop:
    $1/1! + 2/2! + 3/3! \ldots$

Freebooks.pk

26. Write a program that sums the sequence of integers assuming that first integer read specifies the number of values remaining to be entered. The program should read one value per input statement. A typical input sequence might be 5 100 200 150 300 500. The first integer 5 indicates that subsequent five values are to be summed.
27. A person invests $1000.00 in a saving account yielding 5% interest. Assuming all interest is left deposit in the account, calculate and print the amount of money in the accounts at the end of each year for ten years. Formula: ($a = p(1+r)^n$). where
    - p Original amount invested.
    - r Annual Interest rate
    - n Number of years
    - a Amount on deposit at the end of nth years.
28. Write a program to calculate the sum of the first n odd integers.
29. Write a program that could find whether the number entered through keyboard is odd or even and should also tell that whether it is prime or not. The program should keep on taking the value till the user ends and before termination should find the total number of odds, evens and primes entered.
30. Write a loop that will calculate the sum of every third integer, beginning with i = 2 (i.e., calculate the sum 2 + 5 + 8 + 11 + ...) for all values of i that are less than 100. write the loop in each of the following ways:
    (1) Using a for loop
    (2) Using a do while loop
    (3) Using a while loop
31. Write a program to add first nine terms of following series using for and while loop: 1/3! + 5/4! + 9/5! + ... where ! indicates factorial.
32. Write a program to calculate sum of the following series using for and do while loop: 1/3 + 3/5 + 5/7 + ... + 97/99
33. Write a program to generate all possible combinations of 1, 2, 3 and 4.
34. Write a program that inputs the starting and ending numbers and displays all prime number ending with digit 7 between the given range in descending order.
35. Write a program that displays all prime numbers between 100 and 500 that are also palindrome.
36. Write a program to display the following output using nested for loop:

```
* * * * *
*       *
*       *
*       *
* * * * *
```

37. Write a program to print the following output using loop:

```
        1
      1 2
    1 2 3
  1 2 3 4
1 2 3 4 5
```

38. Write a program to print the following output using loop:

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Freebooks.pk

39. Write a program to print the following output using loop:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

40. Write a program that uses nested for loops to display multiplication table as follows:

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

41. Write a program that uses nested loops to display the following lines

```
1 2 4 6
2 2 4 6
3 2 4 6
4 2 4 6
```

42. Write a program to print the following output using loop:

```
#####*#####
####*#*####
###*###*###
##*#####*##
#*#######*#
*#########*
```

43. Write a program to print the following output using loop

```
BBBBBBBBB
. BBBBBBB
. . BBBBB
. . . BBB
. . . . B
```

44. Write a program that inputs the height of a triangle and displays it using loop. For example, if the user enters height as 5, the program should print the following:

```
& & & & & & & & &
  & & & & & & &
    & & & & &
      & & &
        &
```

45. Write a program that displays a diamond of asterisks using loop.

```
      *
     * * *
    * * * * *
   * * * * * * *
  * * * * * * * * *
   * * * * * * *
    * * * * *
     * * *
      *
```

Freebooks.pk

46. Write a program to generate the following pyramid of digits using nested loop:

```
1
232
34543
4567654
567898765
67890109876
7890123210987
890123454321098
90123456765432109
0123456789876543210
```

47. Write a program that inputs the height of triangle and displays a triangle of alphabets. For example, if the user enters 5, it displays the following:

```
A
A B
A B C
A B C D
A B C D E
```

48. Write a program to display the following output using while loop:

```
1
3 5
7 9 11
13 15 17 19
21 23 25 27 29
31 33 35 37 39 41
43 45 47 49 51 53 55
```

49. Write a program to display the following output using loop:

```
1 2 3 4 5 6 7 6 5 4 3 2 1
1 2 3 4 5 6   6 5 4 3 2 1
1 2 3 4 5       5 4 3 2 1
1 2 3 4           4 3 2 1
1 2 3               3 2 1
1 2                   2 1
1                       1
```

50. Write a program to display the following output using loop:

```
&
& &
&  &
&   &
&    &
&     &
&      &
&       &
&        &
&&&&&&&&&&&&&&&&&&&&
```

51. Write a program to display the following output using loop:

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
```

Freebooks.pk

52. Write a program that generates the following checker board by using loop.

```
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
```

Exercise Questions

Q.1. What is loop? What are two uses of loop?

A control structure that executes a statement or number of statements repeatedly is known as loop. Loops can be used to execute a statement or number of statements for a specified number of times. Loops can also be used to access a sequence of values such as 1,2,3 so on.

Q.2. What are the program control statements used to control iterations?

The program control statements used to control iterations are while loop, do-while loop and for loop.

Q.3. What is the part of the loop that contains the statements to be repeated?

The loop body is the part of the loops that contains the statements to be repeated.

Q.4. What are the different phases of loop?

- **Loop Entry:** It occurs when the control reaches the first statement in the loop body.
- **Iteration:** It is execution of the body of the loop.
- **Loop Test:** It is the point at which the test expression is evaluated to decide whether to start iteration or not.
- **Loop Exit:** It is the point at which the last iteration is complete and the control is passed to the first statement after the loop.
- **Exit Condition:** It is the condition that causes the loop to exit.

Q.5. What are the important points for Loop Design

- What is the condition that ends the loop?
- How should the condition be initialized?
- How should the condition be updated?
- What is the process being repeated?
- How should the process be initialized?
- How should the process be updated?
- What is the state of the program on exiting the loop?

Q.6. What are some important points in selecting a Loop for the program

Some important points in selecting a proper loop for the programs are as follows:

- The for loop is the best choice if the loop is a simple counter-controlled loop.
- The do while loop is appropriate if the loop is sentinel-controlled loop whose body is always executed at least once.
- The while and for loops are appropriate if the loop is an event-controlled loop and nothing is known about the first execution.

Q.7. Explain the difference between a pretest and a posttest.

A pretest is a condition that is used to test for the completion of loop at the top of loop. In this type of loop, the statements inside the loop body can never execute if the terminating condition is true the first time it is tested. A posttest is a condition that is used to test for the completion of loop at the bottom of loop. It means that the statements in the loop will always be executed at least once.

Freebooks.pk

Q.8. What is difference between while and do-while loops?

In while loop, condition comes before the body of the loop. In do-while loop, condition comes after the body of the loop. If condition is false in the beginning, while loop is never executed. do-while is executed at least once even if condition is false in the beginning.

Q.9. Differentiate between counter and conditional loops?

Counter loop depends on the value of a variable known as counter variable. The value of counter variable is incremented or decremented each time the body of the loop is executed. This loop terminates when the value of counter variable reaches a particular value.

Conditional loop depends on special value known as sentinel value. Sentinel value indicates that the loop should continue or terminate. For example, a loop may execute while the value of a variable is not –1. Here –1 is the sentinel value used to terminate loop.

Q.10. How does a programmer decide to use a While loop versus a For loop?

For loop is used when the number of iterations are known before the loop is entered. Suppose the user wants to calculate the salary of 100 employees. In this situation, For loop is the best option.

While loop is used when the number of iterations is not known at the beginning of loop. Suppose a loop inputs a number from the user and then displays its square. It then asks the user whether he wants to repeat it or not. In this situation, the iterations are decided by the user. So While loop is better.

Q.11. What is sentinel value?

Sentinel value is a value that is used to terminate a loop. It is used to control loops when the number of repetitions is unknown. A sentinel value is commonly used with while and do while loops.

Q.12. Discuss the difference between break and continue statements used in loops.

When the break statement is executed in a loop, it terminates the loop. When continue statement is executed in a loop, it terminates the current iteration of the loop and the execution moves to the next iteration of the loop.

Q.13. What are the three steps that must be performed using the loop control variable?

The three steps that must be performed using the loop control variable are initialization, test and increment/decrement.

Q.14. What is nested loop?

A loop within a loop is called nested loop. In nested loops, the inner loop is executed completely with each change in the value of counter variable of outer loop. Any loop can be used as inner loop of another loop.

Q.15. What is an infinite loop?

A loop in which the ending condition never occurs is called indefinite loop. It repeats forever until the user intervenes to stop the loop.

Q.16. Describe the syntax of while loop with example.

The syntax of **while** loop is as follows:

```
while (condition)
        statement(s);
```

Example

```
int count = 0;
while (count < 10)
{
  cout<< "C++ programming";
  count++;
}
```

Q.17. Describe the syntax of do-while loop with example.

The syntax of while loop is as follows:

```
do
{
        statement 1;
        statement 2;
        :
        statement N;
}
while (condition);
```

Example

```
int n = 4;
do
{
  n = n * 3;
}
while(n <= 100);
```

Freebooks.pk

Q.18. Describe the syntax of for loop with example.

The syntax of this loop is as follows:

```
for (initialization; condition; increment/decrement)
{
        statement 1;
        statement 2;
        :
        statement N;
}
```

Example

```
int n = 4;
for(int i = n; i >= 0; i--)
  n -= i;
```

Q.19. Trace the output of the following piece of code:

```
int i ;
for( i = 10 ; i > 0 ; i -= 1 )
{
   if ( i % 2 != 0 )
      cout<<"i = "<<i<<endl;
   else
      continue ;
}
```

Answer:

i = 9
i = 7
i = 5
i = 3
i = 1

Q.20. Trace the output of the following piece of code?

```
int j, k ;
for( j = 1 ; j < 6 ; j = j + 1 )
{
   for ( k = 0 ; k < j ; k++ )
   {
      cout<< " + ";
   }
   cout<<endl ;
}
```

Answer:

+
+ +
+ + +
+ + + +
+ + + + +

Q.21. Trace the output of the following piece of code?

```
int i, s = 0 ;
for( i = 10 ; i > 0 ; i -= 1 )
{
   s += i ;
}
cout<<"Sum is = "<<s ;
```

Answer:

Sum is = 55

Q.22. Trace the output of the following piece of code?

```
int i, j = 1 ;
for( i = 10 ; i > 0 ; i -= 1 )
{
   cout<<" \n i = "<<i<<"\t j = "<<j;
   j = j * 2;
}
```

Answer:

i = 10	j = 1
i = 9	j = 2
i = 8	j = 4
i = 7	j = 8
i = 6	j = 16
i = 5	j = 32
i = 4	j = 64
i = 3	j = 128
i = 2	j = 256
i = 1	j = 512

Q.23. Trace the output of the following piece of code:

```
int i, p = 1 ;
for( i = 1 ; i < 6 ; i += 1 )
{
   p *= 2 ;
}
cout<<"\np is = "<<p ;
```

Answer:

p is = 32

Freebooks.pk

Q.24. Trace the output of the following of code:

```
int i, j;
for(i=1; i<=3; i++)
  for(j=1; j<=4; j++)
    cout<<"\n "<<i<<"\t "<<j;
```

Answer:

1	1
1	2
1	3
1	4
2	1
2	2
2	3
2	4
3	1
3	2
3	3
3	4

Q.25. Trace the output of the following of code:

a)

```
int i,j,k;
for(i=0,j=2,k=1;i<=4;i++)
    cout<<i+j+k;
```

Answer: 34567

b)

```
int i=1;
for(;i<=4;i++)
    cout<<i;
```

Answer: 1234

c)

```
int i;
for(i=1;i<=3;i++)
    cout<<"hi ";
cout<<i;
```

Answer: hi hi hi 4

Q.26. What is the output of the following while loops?

a.

```
int n = 45;
int m = 32;
while(m > 0)
{
    cout << n / m;
    n %= m;
    m /= 2;
}
```

Answer: 101101

b.

```
int x, y;
x = -1;
y = 0;
while(x < 3)
{
        y += 2;
        x += 1;
}
cout << "x = " << x << "\n"<<"y = " << y << endl;
```

Answer:

x = 3
y = 8

c.

```
int x = 1;
while(x != 12)
{
  cout << x << "\n";
  x+=2;
}
```

Answer: infinite loop (1, 3, 5, 7, 9 ...)

d.

```
int m = 3, n = 2;
while ( n <= 74 )
{
        n = 5 * n + m;
        m ++;
        if ( m >= 5 )
                m = m/2;
        cout << n << "  " << m << endl;
}
```

Answer:

13	4
69	2
347	3

Freebooks.pk

f.

```
int x = 1;
while ( x < 9 )
{
  x ++;
  if ( x%2 == 1 )
    cout << x << "+";
}
```

Answer: 3+5+7+9+

Q.27. What is the output of the following do-while loops?

a.

```
char c = 'A';
do
{
 cout << c << " ";
 c = c + 2;
} while ( c <= 'I' );
cout << endl;
```

Answer: A C E G I

b.

```
int c = 1;
do
cout << c << " ";
while(c++ <= 4 );
cout << endl;
```

Answer: 1 2 3 4 5

c.

```
int x = 1;
do {
if ( x % 2 == 0)
x -= 2;
else
x++;
cout << x << " + ";
} while (x >= 0 || x == -2);
```

Answer: 2 + 0 + -2 + -4 +

Q.28. What is the output of the following for loops?

a.

```
int c;
  for (c=0; c<0; c--)
    cout<<c<<endl;
```

Answer: No output

b.

```
int c1, c2;
for (c1=1, c2=1;c1<=5;c2++)
  cout<<c1++<<endl;
cout<<--c2 <<endl;
```

Answer:
1
2
3
4
5
5

c.

```
for (int i = 1; i < 30; i += 3)
{
  if (i % 2 == 1)   // If i is odd.
    continue;
  else if (i == 10)
    break;

  cout << i << endl;
}
```

Answer: 4

d.

```
int c;
  for(c=0; c>=0; c++)
    cout<<c<<endl;
```

Answer:
0
1
2
3
… infinite loop

Freebooks.pk

e.

```
int count = 1;
for( ; ; count++)
        if (count < 5)
                cout << count;
        else
                break;
```

Answer: 1234

f.

```
int n = 0;
for(int m = 0; m < 10; m++)
{
   n += m++;
}
cout << n << endl;
```

Answer: 20

Q.29. What is the output of the following nested loops?

a.

```
for (int a = 0; a < 3; a++)
{
        cout << a << " ";
        for (int b = 0; b < 2; b++)
        cout << b << " ";
}
```

Answer: 0 0 1 1 0 1 2 0 1

b.

```
sum = 0;
outerCount = 1;
while (outerCount <= 5)
{
   innerCount = 1;
   while (innerCount <= outerCount)
   {
      sum = sum + innerCount;
      innerCount++;
   }
   outerCount++;
}
cout << sum << endl;
```

Answer: 35

c.

```
for (int outCount = 1 ; outCount < 2; outCount++)
        for (int inCount = 3 ; inCount > 0; inCount--)
                cout << outCount + inCount << endl;
```

Answer:
4
3
2

Q.30. What is the output of the following code if limit = 6?

```
int i, j;
for ( i = limit; i > 0; i-- )
{
   for ( j = 1; j <= i; j++ )
   {
      if ( i == 1 || i == limit || j == 1 || j == i )
      {
         cout << i;
      }
      else
      {
         cout << ' ';
      }
   }
   cout << endl;
}
```

Answer:
666666
5 5
4 4
3 3
2 2
1

Freebooks.pk

Q.31. Consider the following code:

```
int x = 3;
int a = 12;
while(x <= a)
{
        x = x + 4;
        a++;
}
```

a. How many times will the following loop be executed? 4

b. Using the above code, what will the value of x be after execution?19

c. Using the above code, what will the value of a be after execution? 16

Q.32. Rewrite the following loops as do-while loops:

a)

```
int i = 1;
while (i <= 15)
{
  cout << 'a';
  i = i + 1;
}
```

Answer:

```
int i = 1;
do
{
 cout << 'a';
 i = i + 1;
} while ( i <= 15 );
```

b)

```
int c = 1;
while(c <= 10)
{
 if( c < 5 && c != 2 )
    cout << "hello";
 i++;
}
```

Answer:

```
int c = 1;
do
{
 if ( c < 5 && c != 2)
   cout << "hello";
 i++;
} while( c <= 10 );
```

c)

```
long n = 10;
while (n < 100)
{
  cout << 'a';
  n = n + 10;
}
```

Answer:

```
long n = 10;
do
{
 cout << 'a';
 n = n + 10;
} while ( n < 100 );
```

Q.33. Rewrite the following loops as for loops:

a)

```
int c=0;
while (c<10)
{
 cout<<c;
 c=c+2;
}
```

Answer:

```
int c;
for(c=0;c<10;c=c+2)
 cout<<c;
```

b)

```
int c = 1;
while(c <= 10)
{
  if( c < 5 && c != 2 )
    cout <<"hello";
  c++;
}
```

Answer:

```
for ( int c = 1; c <= 10; c++)
  if ( c < 5 && c != 2 )
    cout << "hello";
```

Freebooks.pk

c)
```
long n = 10;
do
{
  cout << 'a';
  n = n + 10;
} while ( n < 100 );
```

Answer:
```
for( long n = 10; n < 100; n = n + 10 )
  cout << 'hello';
```

Q.34. Rewrite following loops as do while loops:
```
for( int i = 3; i <= 39; i += 6 )
{
        cout << i << "\n";
}
```

Answer:
```
int i = 3;
do
{
        cout << i << "\n";
        i += 6;
} while( i <= 39 );
```

Q.35. Rewrite the following loops as while loops:

a.
```
for(i = 0; i < num; i++)
{
  cout << i << "   Enter a value: ";
  cin >> val;
  cout << "\nYou entered " << val << endl;
}
```

Answer 21(a):
```
i = 0;
while( i < num )
{
  cout << i << "   Enter a value: ";
  cin >> val;
  cout << "\nYou entered " << val
<< endl;
  i++;
}
```

b.
```
int i, data, sum ;
sum = 0;
for (i=1 ; i<=10 ; i++)
{
        cout<<"\n Enter an integer : " ;
        cin>> data ;
        sum +=data ;
}
cout<<"\n Sum is : "<< sum<<endl;
```

Answer:
```
int sum =0;
int i = 1;
int data;
while ( i<= 10)
{
  cout<<"\n Enter an integer : ";
  cin>>data ;
  sum += data ;
  i++ ;
}
cout<<"\n The sum is: "<<sum<<endl ;
```

Q.36. What is wrong with the following code?

a.
```
int total, n;
int c = 0;
while (c < 100)
{
  cin >> n;
  total = total + n;
  c = + 1;
}
```

Answer: total must be initialized to 0.

Freebooks.pk

b.

```
for (int c = 0; c < 100; c++)
{
  cout <<"hello" << endl;
}
cout << c << " lines ";
```

Answer: c cannot be referenced outside the body of the for loop where it is defined.

c.

```
  For ( x = 50, x => 1, --x ) ;
   cout << x << endl ;
```

Answer.

For should be for. The commas (,) should be semicolons (;). The operator => should be >=. There should not be a semicolon at the end of first line. All of these errors are syntax errors.

d.

```
x = 2 ;
sum = 0 ;
while ( x <= 10 )
   sum += x ;
   x-- ;
```

Answer: The body of while loop should be enclosed in braces. x- - should be x++.

e.

```
while ( y > 0 )
{
  cout << y << endl;
  y= y + 1;
}
```

Answer: The variable y should be decremented (i.e., y=y-1;) not incremented (y=y+1;).

f.

```
while ( c <= 5 )
{
  product = product * c ;
  c= c + 1;
```

Answer: After the statement c = c + 1, closing right brace of while body should be added.

g.

This code is meant to calculate sum of all numbers entered at the keyboard, excluding the last entry, which will be –1. (Assume all appropriate prompts, variable declarations, etc.)

```
cin >> next;
while( next > 0 )
{
        sum = 0;
        sum += next;
        cin >> next;
}
```

Answer: "sum" should be initialized OUTSIDE while loop

Q.37. Write a C++ statement or a set of statements to accomplish each of the following:

a) Sum the odd integers between 1 and 99 using a for structure.

Answer:

```
Sum = 0 ;
for ( count = 1 ; count <= 99 ; count += 2 )
  Sum += count ;
```

Freebooks.pk

b) Print integers from 1 to 20 using a **while** loop and counter variable x. Assume that the variable x has been declared, but not initialized. Print only 5 integers per line. Hint: Use the calculation x % 5. When the value of this is 0, print a new line character (endl), otherwise print a tab character ('\t').

Answer:

```
int x = 1 ;
while ( x <=20)
{
  cout << x ;
  if ( x % 5 == 0 )
    cout << endl ;
  else
    cout << '\t' ;
  x++ ;
}
```

c) Output all numbers between 1 and 20 inclusive except 10. Print each number on a new line.

```
for(int i = 1; i <= 20; i++ )
{
  if( i != 10 )
    cout << i << "\n";
}
```

d) Output all odd numbers between 0 and 100, printing 5 on a line with tabs as follows:

```
1       3       5       7       9
11      13      15      17      19 .....  etc.
for(int i = 1; i < 100; i += 2 )
{
  cout << i;
  if( (i+1) % 10 == 0 )
    cout << "\n";
  else
    cout << "\t";
}
cout << "\n";
```

Multiple Choice

1. **This is a control structure that causes a statement or group of statements to repeat:**
   a. Decision statement. b. Loop c. Sequential d. None
2. **One execution of a loop is known as a(n):**
   a. Cycle. b. Duration. c. Iteration. d. Test
3. **How many types of loop structure are available in C++?**
   a. 4 b. 3 c. 2 d. 6
4. **Which of the following loop is available in C language?**
   a. while b. do-while c. for d. All
5. **A counter can be defined as:**
   a. The final value of a loop b. A variable that counts loop iterations
   c. The initial value of a loop. d. The step value of a loop.
6. **Where should be the loop control variable initialized?**
   a. Before the loop starts b. In the first statement of the loop body
   c. In the last statement of the loop body d. Anywhere in the loop body

Freebooks.pk

7. **Which loop structure always executes at least once?**
   a. do-while b. for c. while d. None
8. **In which loop the condition comes before the body of the loop?**
   a. while loop b. do-while loop c. for loop d. b and c
9. **In which loop the condition comes after the body of the loop?**
   a. while loop b. do-while loop c. for loop d. b and c
10. **Which of the following loop is called counter loop?**
    a. for b. while c. do-while d. None
11. **This loop is a good choice when you know how many times you want the loop to iterate in advance of entering the loop?**
    a. while b. do-while c. for d. nested
12. **This statement causes a loop to terminate early?**
    a. break b. terminate c. exit d. Both a and b
13. **A loop within a loop is called:**
    a. Nested loop b. Complex loop c. Infinite loop d. None
14. **The loop which never ends is called:**
    a. Infinite loop b. Running loop c. Continuous loop d. Nested loop
15. **Which statement is used to move the control to the start of loop body?**
    a. continue b. break c. switch d. None
16. **A special value that marks the end of a list of input data is called:**
    a. Terminal value b. sentinel value c. loop control value d. input value
17. **A for statement contains three expressions: initialization, test, and**
    a. Null b. validation c. increment/decrement d. None
18. **In a for statement, this expression is executed only once:**
    a. test b. validation c. initialization d. None
19. **This statement may be used to stop a loop's current iteration and begin next one:**
    a. continue b. break c. terminate d. None
20. **This means to increase a value by one:**
    a. Modulus b. increment c. decrement d. None
21. **If you want a user to enter exactly 20 values, which loop would be the best to use?**
    a. while b. do-while c. for d. infinite
22. **while loop is also called:**
    a. Conditional loop b. wend loop c. Counter loop d. None
23. **Semicolon is placed at the end of condition in:**
    a. while loop b. do-while loop c. for loop d. All
24. **Which is a loop statement?**
    a. if b. if-else c. switch d. None
25. **With respect to the loop in the following main function, what is missing?**

```
void main()
{
  int loopCount;
  while (loopCount <= 5)
  {
        cout << "Hi";
        loopCount++;
  }
  return 0;
}
```

a. The initialization of the loop control variable

Freebooks.pk

b. The testing of the loop control variable
c. The incrementation of the loop control variable
d. Nothing is missing.

26. What is the output of the following code?

```
int n = 4;
do
{
  n = n * 3;
}
while(n <= 100);
```

a. 108 b. 110 c. 107 d. 100

27. What is the output of the following code?

```
int n = 8;
for(int i=1; i <= n*3; i++)
  n++;
```

a. infinite loop b. 9 c. 12 d. 16

28. What is the output of the following code?

```
int n = 4;
for(int i = n; i >= 0; i--)
  n -= i;
```

a. -6 b. 5 c. -7 d. -5

29. What is the output of the following code?

```
int n = 5;
while ( n <= 20 )
  if ( n%2 )
        n = n-1;
  else
        n = n+3;
```

a. 21 b. 20 c. 19 d. 22

30. What is the output of the following code?

```
int x = -1, y = 0;
while(x <= 3)
{
  y = y + 2;
  x = x + 1;
}
cout << y;
```

a. 10 b. 11 c. 9 d. 8

31. What is the output of the following code?

```
int n = 0;
for(int m = 0; m < 10; m++){
  n += m++;
}
cout << n << endl;
```

a. 20 b. 21 c. 22 d. 18

32. How many times the following code prints "OPP Using C++"?

```
int count = 0;
while (count < 10)
{
  cout << " OPP Using C++";
  count++;
```

Freebooks.pk

```
}
```

a.9 b. 10 c. 0 d. 11

33. How many times the following code prints " OOP Using C++"?

```
int count = 0;
while (count++ < 10)
{
  cout << " OPP Using C++";
}
```

a. 12 b. 10 c. 9 d.0

34. How many times the following code prints " OOP Using C++"?

```
int count = 0;
do
{
  cout << " OPP Using C++";
} while (count++ < 10);
```

a. 11 b. 12 c. 9 d. 13

35. What is the value of variable s after the following loop terminates?

```
int s = 0;
int n = 0;
do
{
  n++;
  s+= n;
  if (s > 4) break;
}
while (n < 5);
```

a. 6 b. 5 c. –6 d. 0

36. What is the value of variable s after the following loop terminates?

```
int s = 0;
int n = 0;
do
{
  n++;
  s += n;
  if (s >= 4) continue;
}
while (n < 5);
```

a. 15 b. 14 c. 16 d. 18

Answers

1. c	2. c	3. b	4. d	5. b	6. a
7. a	8. a	9. b	10. a	11. c	12. a
13. a	14. a	15. a	16. b	17. c	18. c
19. a	20. b	21. c	22. a	23. b	24. d
25. a	26. a	27. a	28. a	29. a	30. a
31. a	32. b	33. b	34. a	35. a	36. a

Freebooks.pk

Fill in the Blanks

1. There are _____ types of loop in C++.
2. The loop condition controls the loop_____.
3. Repetition of statements in a program is called _____.
4. The structure that repeats the statement(s) is known as _____ structure.
5. Three types of loop structures in C++ are for loop, while loop and _____ loop.
6. A variable whose value controls the number of iterations is known as _____.
7. The compound statements that is enclosed in braces are called_____
8. In _____ loop, the loop control variable is always initialized out of body of loop and is incremented or decremented inside the loop body.
9. _____ loop executes at least once.
10. In _____ loop, condition is checked before the execution of loop body.
11. A loop that never ends is known as _____ loop.
12. In _____ loop, first the body of the loop is executed and then the test condition is checked.
13. _____ loop is also called counter-controlled loop.
14. for loop consists of three parts: initialization, _____ and increment/decrement.
15. The three expressions in _____loop statements are separated by semicolons.
16. In for loop, _____ part is executed only once when the controls enters the loop.
17. If value of conditional expression in for loop is non-zero, the body of loop is _____.
18. A loop within a loop is called _____ loop.
19. In nested loop, the inner loop is executed completely with each change in the value of _____ variable of outer loop.
20. _____ statement moves control directly to next iteration wherever it is used in loop body.
21. One or more _____ loops can be placed in the body of while or do-while loop.
22. There is no restriction on type of loops that may be placed in the body of other _____.
23. This type of loop depends on special value known as _____.
24. _____ is an end marker that follows the last item in the list of items.
25. There are _____ expressions in for loop statement.

Answers

1. Three	2. iteration	3. Loop
4. iterative	5. do-while	6. loop control variable
7. loop body	8. while	9. do while
10. while	11. Infinite	12. do-while
13. for	14. Condition	15. for
16. Initialization	17. Executed	18. Nested
19. Counter	20. Continue	21. for
22. loops	23. sentinel value	24. sentinel value
25. three		

True / False

1. The number of statements that are executed repeatedly is called loop.
2. A repeating structure in C++ is called iterative structure.
3. There are three looping structures in C++.
4. In counting loops, counter must be initialized to zero before execution of the loop body begins.
5. Loop counter variables are usually of type double.

Freebooks.pk

6. A counter is a variable that keeps a running total of a variable of interest such as the total of all the individual sales.
7. Each for loop has an initial, final and step value.
8. while loop executes the loop body even before checking the condition.
9. Loop body is executed at least once when do-while loop is used.
10. do-while is an iterative control in C++ language in which the body of while loop may not execute even once.
11. The body of for statement may not be executed at all.
12. The while loop is surely executed if the condition is false in the beginning.
13. Loop repetition condition of while or for statement can be false before loop begins to execute.
14. The loop repetition condition of for statement is tested at the end of each pass.
15. Counter-controlled loop never executes statements for a specified number of times.
16. The body of while statement must cause the loop repetition condition to become false after a finite number of passes to prevent an infinite loop.
17. The number of iterations in for loop depends on the initialization, condition and increment/decrement parts.
18. Initialization part is optional in for loop.
19. A semi-colon should be used in for loop if increment/decrement part is not given.
20. The continue statement is used in the body of loop for loop to move the control to the start of loop body.
21. The remaining statements of current iteration are not executed when 'start' statement is used in the loop body.
22. The remaining iterations of loop are skipped when 'break' statement is used in the loop body.
23. The count variable of an outer loop can be used as final value of an inner loop.
24. Only a loop of the same type can be used as an inner loop.
25. There is no difference between while and do-while loop.
26. The body of the while loop may or may not execute.
27. var++ is an example of prefix increment.
28. The condition of infinite loop never becomes true.
29. The initialization expression is optional in for loop.
30. for (m=1; m<=10 ; m++) is an infinite loop.
31. A loop is a decision making construct.
32. A while loop cannot be used in the body of for loop.
33. In type casting, a variable of one type behaves as a variable of another type temporarily.
34. The while structure and for structure are basically the same.
35. Loops are used when we need our program to make a choice between two or more things.
36. In C++, each statement must be on a separate line.

Answers

1. T	2. T	3. T	4. F	5. F	6. F
7. T	8. F	9. T	10. T	11. T	12. T
13. F	14. T	15. F	16. F	17. T	18. T
19. T	20. T	21. T	22. F	23. F	24. T
25. F	26. F	27. T	28. F	29. T	30. T
31. F	32. F	33. T	34. T	35. F	36. F

Freebooks.pk

CHAPTER 7

ARRAYS

Chapter Overview

7.1 Arrays
- 7.1.1 Advantages / Uses of Arrays
- 7.1.2 Declaring One-Dimensional Array
- 7.1.3 Array Initialization
- 7.1.4 Accessing Individual Elements of Array
- 7.1.5 Accessing Array Elements using Loops
- 7.1.6 Input and Output Values of an Array

7.2 Searching in Arrays
- 7.2.1 Sequential Search
- 7.2.2 Binary Search

7.3 Sorting Arrays
- 7.3.1 Selection Sort
- 7.3.2 Bubble Sort

7.4 Two-Dimensional Arrays
- 7.4.1 Accessing Individual Elements of 2-D Array
- 7.4.2 Entering Data in 2-D Arrays
- 7.4.3 Initializing 2-D Arrays

7.5 Multidimensional Arrays
- 7.5.1 Accessing Multidimensional Arrays

Programming Exercise
Exercise Questions
Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

7.1 Arrays

An **array** is a group of consecutive memory locations with same name and type. Simple variable is a single memory location with a unique name and a type. But an array is a collection of different adjacent memory locations. All these memory locations have one collective name and type. The memory locations in the array are known as **elements** of array. The total number of elements in the array is called its **length**.

Each element in the array is accessed with reference to its position of location in the array. This position is called **index** or **subscript**. Each element in the array has a unique index. The index of first element is 0 and the index of last element is length – 1. The value of the index is written in brackets alongwith the name of array.

Arrays are used to store a large amount of similar kind of data. Suppose the user wants to store the marks of 100 students and declares 100 variables. It is time-consuming process to use these variables individually. This process can be simplified by using array. An array of 100 elements can be declared to store these values instead of 100 individual variables.

7.1.1 Advantages / Uses of Arrays

Some advantages of arrays are as follows:

- Arrays can store a large number of values with single name.
- Arrays are used to process many values easily and quickly.
- The values stored in an array can be sorted easily.
- A search process can be applied on arrays easily.

7.1.2 Declaring One-Dimensional Array

A type of array in which all elements are arranged in the form of a list is known as **one-dimensional** array. It is also called **single-dimensional array** or **linear list**. It consists of one column or one row. The process of specifying array name, length and data type is called **array declaration**.

Syntax

The syntax of declaring one-dimensional array is as follows:

```
Data_Type Identifier[Legnth];
```

Data_Type	It indicates the data types of the values to be stored in the array.
Identifier	It indicates the name of the array.
Length	It indicates total number of elements in the array. It must be a literal constant or symbolic constant.

Example

```
int marks[5];
```

The above example declares an integer array **marks** of five elements. It allocates five consecutive locations in memory. The index of first element is 0 and index of last element is 4.

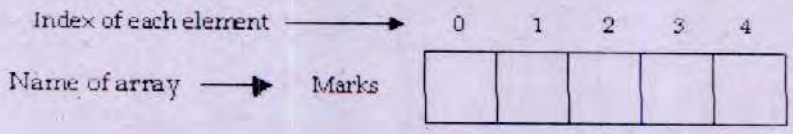


Figure 7.1: An array **marks** with five elements

Freebooks.pk

7.1.3 Array Initialization

The process of assigning values to array elements at the time of array declaration is called **array initialization**. The initialization process provides a list of initial values for array elements. The values are separated with commas and enclosed within braces. There must be at least one initial value between braces. A syntax error occurs if the values in braces are more than the length of array. If the number of initial values is less than the array size, the remaining array elements are initialized to zero.

Syntax

The syntax to initialize array is as follows:

```
Data_Type Identifier[Legnth] = { List of values };
```

Data_Type It indicates the data types of the values to be stored in the array.

Identifier It indicates the name of the array.

Length It indicates total number of elements in the array. It must be a literal constant or symbolic constant.

List of Values It indicates the values to initialize the array. These values must be constant.

Example

```
int marks[5] = {70, 54, 82, 96, 49};
```

The above statement declares an integer array **marks** with five elements. It also initializes the array with values given in the braces. In this declaration, marks[0] is initialized to 70, marks[1] is initialized to 54 and so on.

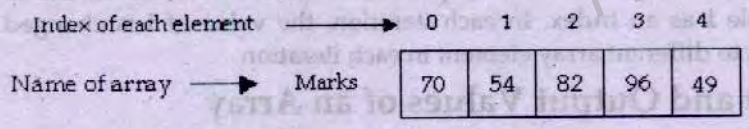


Figure 7.2: An array **Marks** with five elements

An array size can be omitted when it is initialized in the declaration as follows:

```
int age[] = {23, 56, 87, 92, 38, 12, 15, 6, 3};
```

The compiler determines the size of the **age** array according to the initialized values on the right side. The size of array in above example is 9.

7.1.4 Accessing Individual Elements of Array

Each individual element of an array is accessed by specifying the following:

- Name of array
- Index of element

Syntax

The syntax of accessing an individual element of an array as follows:

```
Array_Name[Index];
```

Array_Name It indicates the name of array.

Index It indicates the index of element to be accessed.

Example

```
int marks[5];
marks[0] = 20;
marks[1] = 50;
```

Freebooks.pk

```
marks[2] = 70;
marks[3] = 80;
marks[4] = 90;
```

The above example declares an array marks of type **int** with five elements. It accesses all elements of the array individually to store different values. **marks[0]** indicates the first element and **marks[4]** indicates the last element of the array.

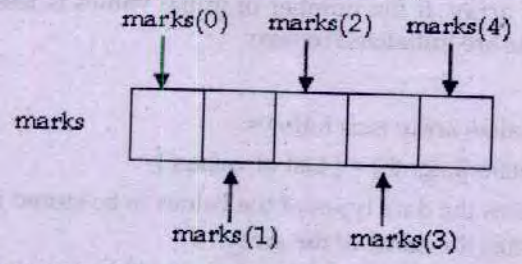


Figure 7.3: Accessing individual elements of an array

7.1.5 Accessing Array Elements using Loops

An easier and faster way of accessing array elements is using loops. The following example shows how array elements can be accessed using **for** loop.

```
int marks[5];
for(int i=0; i<5; i++)
    marks[i] = i;
```

The above example uses **for** loop to store different values in the array. It uses the counter variable **i** as an **index**. In each iteration, the value of **i** is changed. The statement **marks[i]** refers to different array element in each iteration.

7.1.6 Input and Output Values of an Array

The process of input and output with arrays is similiar to the input and output with simple variables. The **cin** object is used to input values in the arrays. The **cout** object is used to display values of arrays. However, the use of these objects with each individual elements becomes very time-consuming because each elements is treated as a simple variable.

Loops are frequently used to input and output data in an array. The use of loops with arrays makes the process of input and output faster and easier. It also reduces the code written for this purpose.

Program 7.1

Write a program that inputs five integers from the user and stores them in an array. It then displays all values in the array without using loops.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[5];
  cout<<"Enter five integers: "<<endl;
  cin>>arr[0];
  cin>>arr[1];
  cin>>arr[2];
  cin>>arr[3];
  cin>>arr[4];
```

Freebooks.pk

```
    cout<<"The values in array are: \n";
    cout<<arr[0]<<endl;
    cout<<arr[1]<<endl;
    cout<<arr[2]<<endl;
    cout<<arr[3]<<endl;
    cout<<arr[4]<<endl;
    getch();
}
```

Program 7.2

Write a program that inputs five integers from the user and stores them in an array. It then displays all values in the array using loops.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[5], i;
  for(i=0; i<5; i++)
  {
    cout<<"Enter an integer: ";
    cin>>arr[i];
  }
  cout<<"The values in array are: \n";
  for(i=0; i<5; i++)
    cout<<arr[i]<<endl;
  getch();
}
```

Output:

```
Enter an integer: 96
Enter an integer: 72
Enter an integer: 41
Enter an integer: 39
Enter an integer: 68
The values in array are:
96
72
41
39
68
```

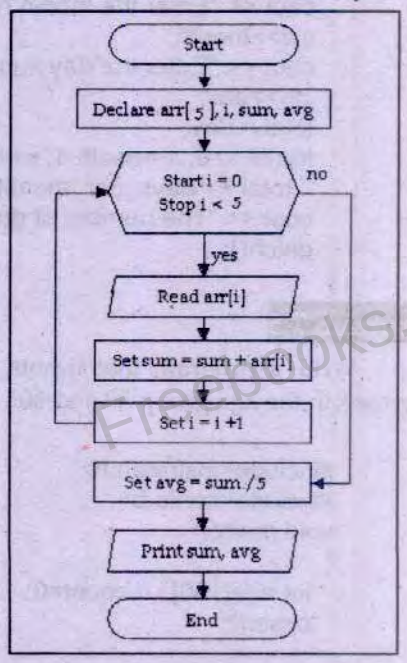
Program 7.3

Write a program that inputs five values from the user, stores them in an array and displays the sum and average of these values.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[5], i, sum = 0;
  float avg = 0.0;
  clrscr();
  for(i=0; i<5; i++)
  {
    cout<<"Enter value: ";
    cin>>arr[i];
    sum = sum + arr[i];
  }
  avg = sum/5.0;
  cout<<"Sum is "<<sum<<endl;
  cout<<"Average is "<<avg;
  getch();
}
```

Output:

```
Enter value: 3
Enter value: 7
Enter value: 5
Enter value: 9
Enter value: 6
Sum is 30
Average is 6
```



Freebooks.pk

How above Program Works?

The above example initializes the variable **Sum** to 0. The first loop gets five inputs. The second loop again visits all elements of the array. In first iteration, the value of counter variable is 0. So it adds the value of first elements and the value of **Sum** and stores the result in **Sum**. In second iteration, it adds the value of second element and **Sum** and then stores the result back to **Sum** and so on. Suppose the user enters the following values in the array.

3	7	5	9	6

The execution of the second loop to get the sum of all values will work as follows:

Iteration No.	Value of i	Statement Executed	Value of sum
1st	0	sum = sum + arr[0]	3
2nd	1	sum = sum + arr[1]	10
3rd	2	sum = sum + arr[2]	15
4th	3	sum = sum + arr[3]	24
5th	4	sum = sum + arr[4]	30

Table 7.1: Working of for loop to calculate sum

Program 7.4

Write a program that inputs current day and month from the user. It then calculates and displays the total number of days in the current year till the entered date.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int day,month,total;
  int days_per_month[12]={31,28,31,30,31,30,31,31,30,31,30,31};
  cout << "Enter the month number: ";
  cin>>month;
  cout << "Enter the day number: ";
  cin>>day;
  total = day;
  for(int x=0; x<month-1; x++)
    total += days_per_month[x];
  cout << "The number of days in this year till date = " << total << endl;
  getch();
}
```

Output:
Enter the month number: 6
Enter the day number: 6
The number of days in this year till date = 157

Program 7.5

Write a program that inputs the age of different persons and counts the number of persons in the age group 50 and 60.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int age[150],i,n,count=0;
  clrscr();
```

Output:
Enter the number of persons required 3
Enter ages of 3 persons.
51
55
20
2 persons are between 50 and 60.

Freebooks.pk

```
    cout<<"Enter the number of persons required ";
    cin>>n;
    cout<<"Enter ages of "<<n<<" persons."<<endl;
    for(i=0; i<n; i++)
    {
      cin>>age[i];
      if(age[i]>=50 && age[i]<=60)
        count = count+1;
    }
    cout<<count<<" persons are between 50 and 60.";
    getch();
}
```

Program 7.6

Write a program that uses four arrays **numbers**, **squares**, **cubes** and **sums** each consisting of 5 elements. The **numbers** array stores the values of its indexes, the **squares** array stores the squares of its indexes, the **cubes** array stores the cubes of its indexes and **sums** array stores the sum of corresponding indexes of three arrays. The program should display the values of all arrays and the total of all values in **sums** array.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
  clrscr();
  const int size = 5;
  int numbers[size];
  int squares[size];
  int cubes[size];
  int sums[size];
  // store the numbers in the arrays
  for (int i = 0; i < size; i++)
   {
     numbers[i] = i;
     squares[i] = i * i;
     cubes[i] = i * i * i;
     sums[i] = numbers[i] + squares[i] + cubes[i];
   }
  // output the sums array and add up all the sums...
  int total = 0;
  cout<<"numbers:\t";
  for(i=0; i<size; i++)
    cout<<numbers[i]<<"\t";
  cout<<endl;
  cout<<"squares:\t";
  for(i=0; i<size; i++)
    cout<<squares[i]<<"\t";
  cout<<endl;
  cout<<"cubes:\t\t";
  for(i=0; i<size; i++)
    cout<<cubes[i]<<"\t";
  cout<<endl;
```

Output:

numbers:	0	1	2	3	4
squares:	0	1	4	9	16
cubes:	0	1	8	27	64
sums:	0	3	14	39	84

Grand total: 140

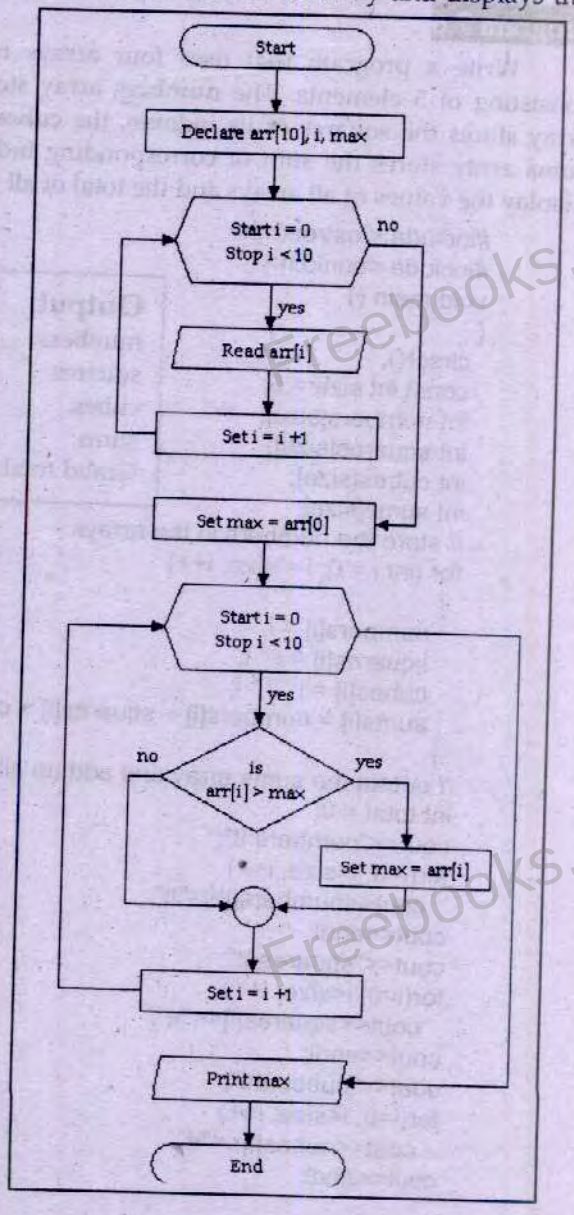
Freebooks.pk

```
    cout<<"sums:\t\t";
    for(i=0; i<size; i++)
    {
      cout<<sums[i]<<"\t";
      total = total + sums[i];
    }
    cout<<endl;
    cout << "Grand total: " << total << endl;
    getch();
  }
```

Program 7.7

Write a program that inputs ten numbers from the user in an array and displays the maximum number.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[10], i, max;
  clrscr();
  for(i=0; i<10; i++)
  {
    cout<<"Enter value: ";
    cin>>arr[i];
  }
  max = arr[0];
  for(i=0; i<10; i++)
    if(max < arr[i])
      max = arr[i];
  cout<<"Maximum value: "<<max;
  getch();
}
```



Output:

```
Enter value: 11
Enter value: 82
Enter value: 73
Enter value: 65
Enter value: 97
Enter value: 55
Enter value: 48
Enter value: 96
Enter value: 24
Enter value: 37
Maximum value: 97
```

Freebooks.pk

Program 7.8

Write a program that inputs ten numbers from the user in array and displays the minimum number.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[10], i, min;
  clrscr();
  for(i=0; i<10; i++)
  {
    cout<<"Enter value: ";
    cin>>arr[i];
  }
  min = arr[0];
  for(i=0; i<10; i++)
    if(min > arr[i])
    min = arr[i];
  cout<<"Minimum value: "<<min;
  getch();
}
```

Output:

```
Enter value: 11
Enter value: 82
Enter value: 73
Enter value: 65
Enter value: 97
Enter value: 55
Enter value: 48
Enter value: 96
Enter value: 24
Enter value: 37
Maximum value: 11
```

Program 7.9

Write a program that inputs five numbers in an array and displays them in actual and reverse order.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
  clrscr();
  int num[5], i;
  for(i=0; i<5; i++)
  {
    cout<<"Enter an integer: ";
    cin>>num[i];
  }
  cout<<"\nThe array in actual order:\n";
  for(i=0; i<5; i++)
    cout<<num[i]<<" ";
  cout<<"\nThe array in reverse order:\n";
  for(i=4; i>=0; i--)
    cout<<num[i]<<" ";
  getch();
}
```

Output:

```
Enter an integer: 35
Enter an integer: 47
Enter an integer: 83
Enter an integer: 66
Enter an integer: 72
The array in actual order:
35 47 83 66 72
The array in reverse order:
72 66 83 47 35
```

7.2 Searching in Arrays

Searching is a process of finding the required data in the array. Searching becomes more important when the length of the array is very large.

Freebooks.pk

7.2.1 Sequential Search

Sequential search is also called **linear search** or **serial search**. It is a simple way to search an array for the desired value. It follows the following steps to search a value in array:

- Visit the first element of the array and compare its value with the required value.
- If the value of array matches with the desired value, the search is complete.
- If the value of array does not match, move to next element and repeat same process.

Loops are frequently used to visit elements of array for searching a value. You can start the counter variable of loop from 0 and move it to last index of array. For example if the array to be searched has 10 elements, the loop will start the counter variable from 0 and move to 9.

Program 7.10

Write a program that initializes an array. It inputs a value from the user and searches the number in the array.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
  int i, n, loc = -1;
  clrscr();
  cout<<"Enter value to find: ";
  cin>>n;
  for(i=0; i<10; i++)
    if(arr[i] == n)
   loc = i;
  if(loc == -1)
    cout<<"Value not found in the array.";
  else
    cout<<"Value found at index "<<loc;
  getch();
}
```

Output:
Enter a value to find: 50
Value found at index 4

How above Program Works?

The above program declares and initializes an array of integers. It inputs a number from user and finds it in array. It uses variable **loc** to inidicate successful search. The variable **loc** is initialized to –1. The loop searches the array stores the value of index in **loc** if search is successful. The program checks the value of **loc** at the end. The search is successful of the value of **loc** is other than initial value. Otherwise, the search process is unsuccessful.

7.2.2 Binary Search

Binary search is a quicker method of searching for value in the array. Binary search is very quick but it can only search an sorted array. It cannot be applied on an unsorted array.

- It locates the middle element of array and compares with the search number.
- If they are equal, search is successful and the index of middle element is returned.
- If they are not equal, it reduces the search to half of the array.
- If the search number is less than the middle element, it searches the first half of array. Otherwise it searches the second half of the array. The process continues until the required number is found or loop completes without successful search.

Freebooks.pk

Program 7.11

Write a program that initializes an array of ten integers. It inputs an integer from the user and searches the value in the array using binary search.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int arr[10] = {10,20,30,40,50,60,70,80,90,100};
  int n, i, mid, start, end, loc;
  loc = -1;
  start = 0;
  end = 9;
  cout<<"Enter any number to find: ";
  cin>>n;
  while(start<=end)
  {
    mid = (start+end)/2;
    if(arr[mid] == n)
    {
  loc = mid;
  break;
    }
    else if(n<arr[mid])
  end = mid - 1;
   else
  start = mid + 1;
  }
  if(loc == -1)
   cout<<n<<" not found!"<<endl;
  else
   cout<<n<<" found at index "<<loc<<endl;
  getch();
}
```

Output:
Enter any number to find: 30
30 found at index 2

7.3 Sorting Arrays

Sorting is a process of arranging the values of array in a particular order. An array can be sorted in two orders:

1. Ascending Order

In ascending order, the smallest value is stored in the first element of array; second smallest value is stored in the second element and so on. The largest value is stored in the last element. Following figure shows an array sorted in ascending order.

12	25	33	37	48	57	86	92

Figure 7.4: Sorted array in ascending order

2. Descending Sort

In descending order, the largest value is stored in first element of array, second largest value is stored in second element and so on. The smallest value is stored in the last element.

Freebooks.pk

The following figure shows an array sorted in descending order.

92	86	57	48	37	33	25	12

Figure 7.5: Sorted array in descending order

7.3.1 Selection Sort

Selection sort is a technique that sorts an array. It selects an element in the array and moves it to its proper position. Selection sort works as follows:

1. Find the minimum value in the list
2. Swap it with the value in the first position
3. Sort the remainder of the list excluding the first value

Program 7.12

Write a program that gets five inputs from the user in an array and then sorts this array in ascending order.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int arr[5], i, j, min, temp;
  for(i=0; i<5; i++)
  {
    cout<<"Enter value: ";
    cin>>arr[i];
  }
  cout<<"The original values in array:\n";
  for(i=0; i<5; i++)
    cout<<arr[i]<<" ";

  for(i=0; i<4; i++)
  {
    min = i;
    for(j=i+1; j<5; j++)
      if(arr[j] < arr[min])
          min = j;
   if(min != i)
   {
     temp = arr[i];
     arr[i] = arr[min];
     arr[min] = temp;
   }
  }
  cout<<"\nThe sorted array:\n";
  for(i=0; i<5; i++)
    cout<<arr[i]<<" ";
  getch();
}
```

Output:

```
Enter value: 56
Enter value: 78
Enter value: 33
Enter value: 81
Enter value: 12
The original values in array:
56 78 33 81 12
The sorted array:
12 33 56 78 81
```

Freebooks.pk

How It Works?

Suppose the user enters 56, 78, 33, 81, and 12 in the array.

56	78	33	81	12

Figure 7.6: The unsorted array

This technique of sorting is known as **selection sort**. In the above example, nested loops are used to sort the array. The outer loop moves from 0 to 4 and with each iteration of outer loop, inner loop moves from **i + 1** to **4**.

The value of **i** is stored in variable **min** in each iteration. The value in index **min** is compared with all remaining elements. If any value is found less than the value in index **min**, its index is stored in the variable **min**. The value at index **i** is interchanged with the value at index **min** at the end of each pass if required. The sorting process will work as follows:

Pass 1

Iteration 1

56	78	33	81	12

i = 0 so the value of **min** is also **0**. **j = 1** so 56 is compared with 78. As 78 is not less than 56, there is no change in the value of **min**.

Iteration 2

56	78	33	81	12

min = 0 and **j = 2** so 56 is compared with 33. As 33 is less than 56, the value of **j** is stored in variable **min** so **min = 2**.

Iteration 3

56	78	33	81	12

min = 2 and **j = 3** so 33 is compared with 81. As 81 is not less than 33, there is no change in the value of **min**.

Iteration 4

56	78	33	81	12

min = 2 and **j = 4** so 33 is compared with 12. As 12 is less than 33, the value of **j** is stored in variable **min** so **min = 4**. At this point, the inner loop is completed. The value of **min** has changed in the inner loop so the value at index **i** and the value at index **min** are interchanged. The position of smallest values is now finalized and the array is as follows:

12	78	33	81	56

Freebooks.pk

Freebooks.pk

Pass 2

Iteration 1

12	78	33	81	56

i = 1 so the value of **min** is also **1**. **j = 2** so 78 is compared with 33. As 33 is less than 78, the value of **j** is stored in variable **min** so **min = 2**.

Iteration 2

12	78	33	81	56

min = 2 and **j = 3** so 33 is compared with 81. As 81 is not less than 33, there is no change in the value of **min**.

Iteration 3

12	78	33	81	56

min = 2 and **j = 4** so 33 is compared with 56. As 56 is not less than 33, there is no change in the value of **min**.

At this point, the inner loop is completed. The value of **min** has changed in the inner loop so the value at index **i** and value at index **min** are interchanged. The array is as follows:

12	33	78	81	56

Pass 3

Iteration 1

12	33	78	81	56

i = 2 so the value of **min** is also **2**. **j = 3** so 78 is compared with 81. As 81 is not less than 78, there is no change in the value of **min**.

Iteration 2

12	33	78	81	56

min = 2 and **j = 4** so 78 is compared with 56. As 56 is less than 78, the value of **j** is stored in variable **min** so **min = 4**.

At this point, the inner loop is completed. The value of **min** has changed in the inner loop so the value at index **i** and value at index **min** are interchanged. The array is as follows:

12	33	56	81	78

Freebooks.pk

Pass 4

Iteration 1

12	33	56	81	78

i = 3 so the value of **min** is also 3. j = 4 so 81 is compared with 78. As 78 is less than 81, the value of j is stored in variable **min** and **min = 4.**

At this point, the inner loop is completed. The value of **min** has changed in the inner loop so the value at index i and value at index **min** are interchanged. The array is as follows:

12	33	56	78	81

The outer loop has also completed at this point and the program has sorted the array in ascending order.

7.3.2 Bubble Sort

Bubble sort is also known as **exchange sort**. It repeatedly visits the array and compares two items at a time. It swaps these two items if they are in the wrong order. It continues to visit the array until no swaps are needed that means the array is sorted. It works as follows:

- Compare adjacent elements. If the first is greater than the second, swap them.
- Repeat this for each pair of adjacent elements, starting with the first two and ending with the last two. At this point the last element should be the greatest.
- Repeat the steps for all elements except the last one.
- Keep repeating for one fewer element each time until there are no pairs to compare.

Program 7.13

Write a program that stores five values in an array. It sorts the array using buble sort. It also displays the values of unsorted and sorted array.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int arr[5], i, j, temp;
  for(i=0; i<5; i++)
  {
    cout<<"Enter value: ";
    cin>>arr[i];
  }
  cout<<"The original values in array:\n";
  for(i=0; i<5; i++)
    cout<<arr[i]<<" ";
  for(i=0; i<5; i++)
    for(j=0; j<4; j++)
  if(arr[j]>arr[j+1])
  {
    temp = arr[j];
    arr[j] = arr[j+1];
    arr[j+1] = temp;
```

```
    }
    cout<<"\nThe sorted array:\n";
    for(i=0; i<5; i++)
      cout<<arr[i]<<" ";
    getch();
  }
```

How it Works?

The initial values stored in the array are as follows:

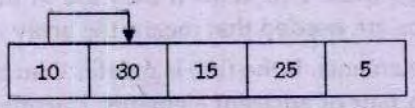
10	30	15	25	5

Figure 7.7: The unsorted array

In the above example, nested loops are used to sort the array. The outer loop moves from **0** to **4** and with each iteration of outer loop, inner loop moves from **0** to **e – (i+1)**. First of all, the value of **i** is 0 so the focus of outer loop is on the first element of the array. The sorting process will work as follows:

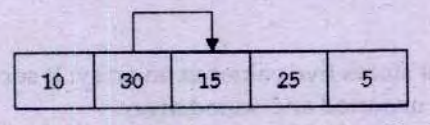
Pass 1

Iteration 1



j = 0 so the statement **arr[j] > arr[j+1]** compares 10 with 30. As 10 is not greater than 30, there will be no change in the array.

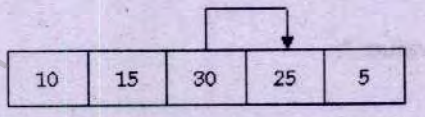
Iteration 2



j = 1 so the statement **arr[j] > arr[j+1]** compares 30 with 15. As 30 is greater than 15, both values will be interchanged and the array will be as follows:

10	15	30	25	5

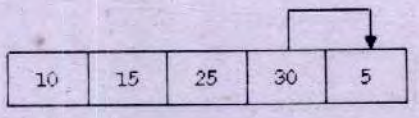
Iteration 3



j = 2 so the statement **arr[j] > arr[j+1]** compares 30 with 25. As 30 is greater than 15, both values will be interchanged and the array will be as follows:

10	15	25	30	5

Iteration 4



Freebooks.pk

j = 3 so the statement **arr[j] > arr[j+1]** compares 30 with 5. As 30 is greater than 5, both values will be interchanged and the array will be as follows:

10	15	25	5	30

At this point, the inner loop is completed and the largest value in array has moved in the last element. It means that the position of largest value is now finalized. Now the control moves to the beginning of outer loop and the value of i becomes 1.

Pass 2

Iteration 1

10	15	25	5	30

j = 0 so the statement **arr[j] > arr[j+1]** compares 10 with 15. As 10 is not greater than 15, there will be no change in the array.

Iteration 2

10	15	25	5	30

j = 1 so the statement **arr[j] > arr[j+1]** compares 15 with 25. As 15 is not greater than 25, there will be no change in the array.

Iteration 3

10	15	25	5	30

j = 2 so the statement **arr[j] > arr[j+1]** compares 25 with 5. As 25 is greater than 5, both values will be interchanged and the array will be as follows:

10	15	5	25	30

At this point, the inner loop is completed and the second largest value in array has moved in the second last element. It means that the position of second largest values is now finalized. Now the control moves to the beginning of outer loop and the value of i becomes 2.

Pass 3

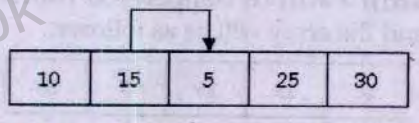
Iteration 1

10	15	5	25	30

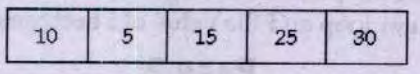
j = 0 so the statement **arr[j] > arr[j+1]** compares 10 with 15. As 10 is not greater than 15, there will be no change in the array.

Freebooks.pk

Iteration 2



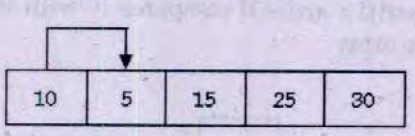
j = 1 so the statement **arr[j] > arr[j+1]** compares 15 with 5. As 15 is greater than 5, both values will be interchanged and the array will be as follows:



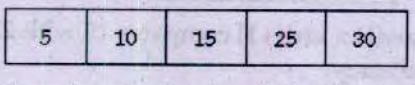
At this point, the inner loop is completed and the third largest value in the array has moved in the third last element. It means that the position of third largest value is now finalized. Now the control moves to the beginning of outer loop and the value of i becomes 3.

Pass 4

Iteration 1



j = 0 so the statement **arr[j] > arr[j+1]** compares 10 with 5. As 10 is greater than 5, both values will be interchanged and the array will be as follows:



At this point, the inner loop is completed and the fourth largest value in the array has moved in the fourth last element. It means that the position of fourth smallest value is now finalized. The position of smallest number is also automatically finalized. Now the outer loop also terminates and the array is sorted in ascending order.

7.4 Two-Dimensional Arrays

Two-dimensional array can be considered as a table that consists of rows and column. Each element in 2-D array is referred with the help of two indexes. One index is used to indicate the row and the second index indicates the column of the element.

Syntax

The syntax for declaring 2-D array is as follows:

```
Data_Type Identifier[Rows][Cols];
```

Data_Type It indicates the data types of the values to be stored in the array.

Identifier It indicates the name of the array.

Rows It indicates the number of rows in the array. It must be a literal constant or symbolic constant.

Cols It indicates the number of columns in the array. It must be a literal constant or symbolic constant.

Example

The following statement declares a 2-D array with four rows and three columns:

```
int Arr [4][3];
```

Freebooks.pk

The statement declares a two-dimensional array. The first index indicates array contains four rows. The index of first row is 0 and the index of last row is 3. The second index indicates that each row in the array contains three columns. The index of first column is 0 and the index of last column is 2. The total number of elements can be determined by multiplying rows and columns. It means that the above array contains twelve elements.

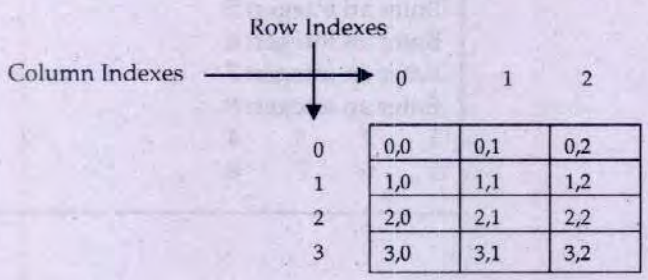


Figure 7.8: A 2-D array and its indexes

7.4.1 Accessing Individual Elements of 2-D Array

The array name and indexes of row and column are used to access an individual element of 2-D array. For example, the following statement will store 100 in the second column of first row in the array:

```
Arr[0][1] = 100;
```

The index for row or column of 2-D array can be given using variables. The following example will work similar to the line above:

```
R = 0;
C = 1;
Arr[R[[C] = 100
```

7.4.2 Entering Data in 2-D Arrays

You can enter data in any element of the array by using the name of array and index of the element. For example, the following statements enter data in the first row of a 2-D array:

```
Arr[0][0] = 10      10 stored in first column of first row
Arr[0][1] = 20      20 stored in second column of first row
Arr[0][2] = 30      30 stored in third column of first row
Arr[1][0] = 40      40 stored in first column of second row
Arr[1][1] = 50      50 stored in second column of second row
Arr[1][2] = 60      60 stored in third column of second row
```

The nested loops are frequently used to enter data in two-dimensional array. The outer loops are normally used to refer to the rows in array. Similarly, the inner loops are normally used to refer to the columns of the rows.

Program 7.14

Write a program that stores integer values in an array of 2 rows and 4 columns.

```
#include <conio.h>
#include <iostream.h>
void main()
{
  clrscr();
```

Freebooks.pk

```
int arr[2][4], i, j;
for(i=0; i<2; i++)
  for(j=0; j<4; j++)
  {
    cout<<"Enter an integer: ";
    cin>>arr[i][j];
  }
for(i=0; i<2; i++)
{
  for(j=0; j<4; j++)
    cout<<arr[i][j]<<"\t";
  cout<<endl;
}
getch();
}
```

Output:

```
Enter an integer: 1
Enter an integer: 2
Enter an integer: 3
Enter an integer: 4
Enter an integer: 5
Enter an integer: 6
Enter an integer: 7
Enter an integer: 8
1    2    3    4
5    6    7    8
```

How above Program Works?

The above program declares a two-dimensional array. It inputs and displays the values in the array using nested loops. The counter variable **i** of outer loop refers to the index of rows in the array. The counter variable **j** of inner loop refers to the index of columns. For example, the loop inputs in the first column of first row when the value of **i** is 0 and the value of **j** is also 0 and so on.

7.4.3 Initializing 2-D Arrays

The two dimensional arrays can also be initlaized at the time of declaration. The process of initialization is performed by assigning the initial values in braces separated by commas. The values of each row can further be assigned in nested braces.

Some important points to initialize two-dimensional arrays are as follows:

- The elements of each row are enclosed within braces and separated by commas.
- All rows are encolosed within the braces.
- For number arrays, if the values for all elements are not specified, the unspecified elements are initialized by zero. In this case, at least one value must be given.

Example

```
int Arr[3][4]  =  { {12, 5, 22, 84},
                    {95, 3, 41, 59},
                    {77, 6, 53, 62} };
```

The above examples declares an array of integers with three rows and four columns. The first inner braces contain the values for row 0. The second inner braces contain the values for row 1 and third inner braces contain the values for row 2.

The initialization can also be performed without using the inner braces as follows:

```
int Arr[3][4]  =  { 12, 5, 22, 84,
                    95, 3, 41, 59,
                    77, 6, 53, 62 };
```

The above initialization is same as the previous initialization. The compiler assgins the values beginning with [0][0] element and proceeds row by row to fill the array.

Both of the above initializations will initiaze the array as follows:

	0	1	2	3
0	12	5	22	84
1	95	3	41	59
2	77	6	53	62

Figure 7.9: A 2-D array and its indexes

Program 7.15

Write a program that initialzes a two dimensional array of 2 rows and 3 columns and then displays its values.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int i, j, arr[2][3] = {15,21,9,84,33,72};
  for(i=0; i<2; i++)
  {
    for(j=0; j<3; j++)
      cout<<"arr["<<i<<"]["<<j<<"] = "<<arr[i][j]<<"\t";
    cout<<endl;
  }
  getch();
}
```

Output:
arr[0][0] = 15 arr[0][1] = 21 arr[0][2] = 9
arr[1][0] = 84 arr[1][1] = 33 arr[1][2] = 72

Program 7.16

Write a program that initializes a two-dimensional array of 2 rows and 4 columns and then displays the minimum and maximum number in the array.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int i, j, max, min;
  int arr[2][4] = {{15,21,9,84},{33,72,18,47}};
  max = min = arr[0][0];
  for(i=0; i<2; i++)
    for(j=0; j<4; j++)
    {
      if(arr[i][j] > max)
        max = arr[i][j];
      if(arr[i][j] < min)
        min = arr[i][j];
    }
  cout<<"Maximum = "<<max<<endl<<"Minimum = "<<min<<endl;
  getch();
}
```

Output:
Maximum = 84
Minimum = 9

Freebooks.pk

Program 7.17

Write a program to add two matrices (two dimensional arrays). Input order of matrix (i.e. number of rows and columns). The matrices must be of same size to be added. Get the input for each element of the first matrix and then second matrix. Add the two matrices and store the values in a third matrix. Finally, display all matrices.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <process.h>
void main( )
{
 int i , j , r1 , r2 , c1 , c2 , a[20][20] , b[20][20] , c[20][20];
 cout<<"Enter rows and columns of first matrix: ";
 cin>>r1>>c1;
 cout<<"Enter rows and columns of second matrix: ";
 cin>>r2>>c2;
 if ( (r1!= r2) || (c1!=c2) )
 {
   cout<<endl<<"Matrix addition is not possible";
   exit(0);
 }
 for (i =0; i< r1; i ++)
 {
   for (j = 0; j<c1; j ++)
   {
     cout<<"Enter the "<< i <<" x "<< j <<" element of first matrix: ";
     cin>>a[ i ][ j ];
   }
 }
 for (i = 0; i<r2 ; i ++)
 {
   for (j =0; j<c2; j ++)
   {
     cout<<"Enter the "<<i<<"x"<<j<<" element of second matrix: ";
     cin>>b[ i ] [ j ];
     }
 }
 cout<<endl;
 for (i =0; i<r1; i++)
 {
   cout<<endl;
   for (j =0; j<c1; j++)
      cout<<setw(9)<<a[ i ][ j ];
 }
 cout<<"\t"<<"+";
 cout<<endl;
 for (i = 0; i<r1; i ++)
 {
   cout<<endl;
   for (j =0; j<c1; j++)
      cout<<setw(9)<<b[ i ] [ j ];
```

```
    }
    cout<<endl;
    cout<<"\t \t \t = ";
    cout<<endl;
  for (i =0; i<r1; i++)
    {
      cout<<endl;
      for (j =0; j<c1; j++)
      {
        c[ i ][ j ] = a[ i ][ j ] + b[ i ][ j ];
        cout<<setw(9)<<c[ i ][ j ];
      }
    }
    getch();
  }
```

Program 7.18

Write a program that inputs the number of rows and columns from the user. It then inputs the elements to store in the matrix. The program calculates the sum of each row and each column and displays on the screen. If it is a square matrix, it also calculates the sum of diagonal elements and displays it on screen.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int A[10][10], m, n, x, y, sum=0;
  //Create a Matrix A
  cout << "Enter number of rows and columns in Matrix A : \n";
  cin>>n>>m;
  cout << "Enter elements of Matrix A : \n";
  for(x=1;x<n+1;++x)
  for(y=1;y<m+1;++y)
    cin>>A[x][y];
  //Find sum of each row
  for(x=1;x<n+1;++x)
  {
    A[x][m+1]=0;
    for(y=1;y<m+1;++y)
      A[x][m+1]=A[x][m+1]+A[x][y];
  }
  //Find sum of each column
  for(y=1;y<m+1;++y)
  {
    A[n+1][y]=0;
    for(x=1;x<n+1;++x)
      A[n+1][y]+=A[x][y];
  }
  cout<<"\nMatrix A, Row Sum and Column Sum: \n";
  for(x=1;x<n+1;++x)
  {
    for(y=1;y<m+2;++y)
```

```
      cout << A[x][y] << "    ";
    cout << "\n";
  }
  //Print sum of each column
  x=n+1;
  for(y=1;y<m+1;++y)
    cout << A[x][y] << "    ";
  cout << "\n";
  if(m==n)
  {
    for(x=1;x<m+1;x++)
      for(y=1;y<n+1;y++)
        if(x==y)
          sum+=A[x][y];
        else
          if(y==m-(x+1))
            sum+=A[x][y];
  }
  cout << "Sum of diagonal elements is : " << sum << endl;
  getch();
}
```

Output:

```
Enter number of rows and columns in Matrix A:
3 3
Enter elements of Matrix A:
9 8 7 6 5 4 3 2 1

Matrix A, Row Sum and Column Sum:
9     8     7     24
6     5     4     15
3     2     1     6
18    15    12
Sum of diagonal elements is : 15
```

Program 7.19

Write a program that inputs integer values in a 4x4 matrix and displays the sum of diagonal elements of the matrix.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int x,y;
  int A[4][4],sum=0;
  cout << "Enter the elements of the matrix : " << endl;
  for(y=0;y<4;y++)
  for (x=0;x<4;x++)
  {
    cout << "Element " << x+1 << ", " << y+1 << " : ";
    cin>>A[x][y];
  }
  //Sum of either of the diagonal elements.
  for(x=0;x<4;x++)
    for(y=0;y<4;y++)
      if(x==y)
        sum+=A[x][y];
  cout << "Sum of the diagonal elements is : " << sum;
  getch();
}
```

7.5 Multidimensional Arrays

Multidimensional array is also called as array of arrays. Multidimensional arrays are not limited to two indices. Multidimensional arrays can have three, four or more dimensions. The amount of memory needed for an array rapidly increases with each dimension.

Syntax

The general syntax of declaring multidimensional array is as follows:

```
DataType arrayName[a] [b]........[n];
```

a, **b**, and **n** are constant expression indicating the length of different dimensions of the array.

Example

```
int arr[10] [5] [7];
```

The above statement declares a three-dimensional array **arr**. The size of the first dimension is 10. The size of the second dimension is 5. The size of the third dimension is 7. The first dimension ranges from 0 to 9. The second dimension ranges from 0 to 4 and third dimension ranges from 0 to 6. The first element in the array is designated as **arr[0][0][0]** and the last element as arr [9] [4] [6]. The total numbers of elements in the array is 10 * 5 * 7 = 350.

7.5.1 Accessing Multidimensional Arrays

The syntax to access an element of a multidimensional array is as follows:

```
ArrayName[a][b].....[n];
```

a, **b** and **n** are expressions indicating the values of different dimensions of the array.

Example

```
arr[0][0][0] = 10;
arr[0][0][1] = 10;
arr[0][0][2] = 10;
...
arr[9][4][6] = 10;
```

The above statements access individual elements of three dimensional array to store integer values. The nested loops are frequently used to enter data in multidimensional array.

```
for(i=0; i<10; i++)
  for(j=0; j<5; j++)
    for(k=0; k<7; k++)
      arr[i][j][k] = 0;
```

The above statements use nested loops to assign 0 to all elements of three dimensional array **arr**. The counter variable **i** of outer loop refers to the first dimension of the array. The counter variable **j** of second loop refers to the second dimension and the counter variable **k** of inner loop refers to the third dimension of the array.

Program 7.20

Write a program that declares a three dimensional array to store the temperatures of a month. The temperature is entered for morning, noon and evening of each day. The first dimension should be used to for three timings of a day, second dimension should be used for seven days of a week and third dimension should be used for four week of a month. The program should input the temperatures and then display the maximum, minimum and average temperature of the whole month.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
```

Freebooks.pk

```
    int i, j, k, max, min, tot=0;
    float avg;
    int temp[3][7][4];
    for(i=0; i<3; i++)
      for(j=0; j<7; j++)
        for(k=0; k<4; k++)
        {
          cout<<"Enter temperature: ";
          cin>>temp[i][j][k];
        }
    max = min = temp[0][0][0];
    for(i=0; i<3; i++)
      for(j=0; j<7; j++)
        for(k=0; k<4; k++)
        {
          tot = tot + temp[i][j][k];
          if(temp[i][j][k] > max)
            max = temp[i][j][k];
          if(temp[i][j][k] < min)
            min = temp[i][j][k];
        }
    avg = tot / 84.0;
    cout<<"Maximum temperature of month: "<<max<<endl;
    cout<<"Minimum temperature of month: "<<min<<endl;
    cout<<"Average temperature of month: "<<avg<<endl;
    getch();
}
```

How above Program Works?

The above program declares a three-dimensional array to store the temperatures of a month. It inputs values and then displays maximum, minimum and average temperatures of the month.

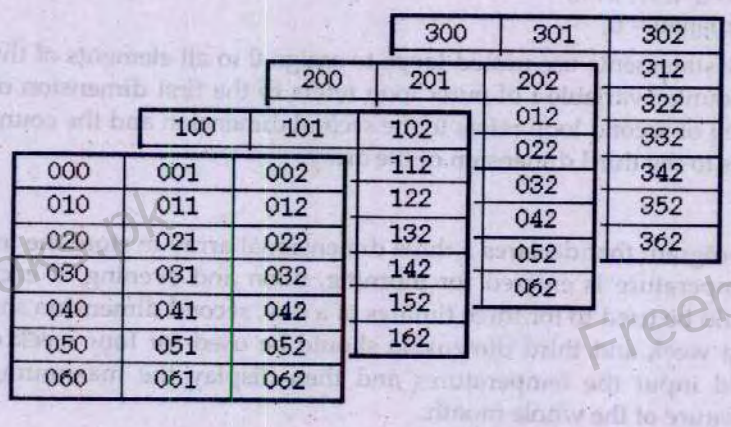


Figure 7.10: temp[3][7][4]

Freebooks.pk

Programming Exercise

1. Write a program that inputs ten integers in an array and counts all prime numbers entered by the user. The program finally displays total number of primes in array.
2. Write a program that uses two arrays to store the roll number and marks of student. It inputs roll numbers and marks of five students and stores them in corresponding elements of the arrays. (For example, if roll number 1 is stored in rno[0] then his marks must be stored in marks[0] and so on.) The program finally displays the roll number and marks of the student with highest marks.
3. Write a program that four arrays numbers, squares, cubes and sums each consisting of 10 elements. The numbers array stores the values of its indexes, the squares array stores the squares of its indexes, the cubes array stores the cubes of its indexes and sums array stores the sum of corresponding indexes of three arrays. The program should displays the values of sums array and the total of all values in sums array.
4. Write a program that inputs the names and monthly salaries of 10 employees. The program checks annual salary of each person. If annual salary is greater than or equal to Rs 2,50000/- then it prints name, salary and a message 'Tax to be paid' else it prints name, salary and a message 'No tax'.
5. Write a program that inputs ten integers in an array. It displays the number of occurrences of each number in the array as follows:

   3 is stored 4 times in the array.
   1 is stored 1 times in the array.
6. Write a program that inputs marks of ten students. The program displays the number of students in each grade. The criteria is as follows:

80 or above	A
60 to 79	B
40 to 59	C
Below 40	F

7. Write a program that uses three arrays Mango, Orange and Banana to store the number of fruits purchased by customer. The program inputs the number of mangos, oranges and bananas to be puchased by customer and stores them in corresponding arrays. The program finally displays the total bill of each customer according to the following prices:

   Rs. 20 per mango
   Rs. 10 per orange
   Rs. 5 per banana

   The output should appear as follows:

Customer No.	Mangoes	Oranges	Bananas	Total Bill
1	5	10	12	Rs. 260

8. Write a program that inputs ten floating point numbers in an array. It displays the values which are greater than the average value of the array.
9. Write a program that uses a two dimensional array to initialize the scores of students. The students are arranged in five rows with five students in each row. The program inputs the row number and student number in that row and then displays the score of the student.

Freebooks.pk

Exercise Questions

Q.1. Define an array.

An **array** is a group of consecutive memory locations with same name and type. All these memory locations have one collective name and type. The memory locations in the array are known as elements of array. The total number of elements in the array is called its **length**.

Q.2. How is an element is accessed in array?

Each element in the array is accessed with reference to its position of location in the array. This position is called **index** or **subscript**. The index of first element is 0 and the index of last element is length - 1. The value of the index is written in brackets alongwith the name of array.

Q.3. How does an array differ from simple variable?

An array can store multiple values but a simple variable can store only one value. Different types of processing can be applied on array easily. However, it is difficult to process multiple values using simple variables.

Q.4. How are individual array elements identified?

A subscript variable is used to identify individual elements in the array. The value for subscript is written in brackets [ ] with array name.

Q.5. How subscript variables are written?

A subscript variable is written in brackets [ ] with array name.

Q.6. List three uses of array.

1. Array is used to store many values with same name.
2. Array is used to process many values quickly and easily.
3. Array is used to sort many values easily.

Q.7. Distinguish between 1-D array and 2-D array.

1-D array consists of only single row or single column. Each element of this array can be accessed using one index value. 2-D array consists of multiple rows and columns. Each element of this array is accessed using two index values.

Q.8. Explain array manipulation.

The process of performing different operations on array is called array manipulation. Different operations can be performed using array such as finding an element, comparing two arrays, finding the minimum or maximum value in array and sorting an array etc.

Q.9. What is sequential search?

Sequential search is also called **linear search** or **serial search**. It is a simple way to search an array for the desired value.

Q.10. What is binary search?

Binary search is a quicker method of searching for value in the array. It takes less time than sequential search. Binary search very quick but it can search an array only if it is sorted. It cannot be applied on an unsorted array.

Q.11. What do you know about sorting? Which are two orders to sort data values?

Sorting is a process of arranging the values of array in a particular order. An array can be sorted in two orders:

- **Ascending Order:** In ascending order, smallest value is stored in first element of array, second smallest value is stored in second element. The largest value is stored in last element.
- **Descending Sort:** In descending order, the largest value is stored in the first element of array; second largest value is stored in the second element and so on. The smallest value is stored in the last element.

Q.12. What do you know about two-dimensional array?

Two-dimensional array can be considered as a table that consists of rows and column. Each element in 2-D array is referred with the help of two indexes. One index is used to indicate the row and the second index indicates the column of the element.

Freebooks.pk

Q.13. What are multidimensional arrays?

Multidimensional array is also called as array of arrays. Multidimensional arrays are not limited to two indices. Multidimensional arrays can have three, four or more dimensions. The amount of memory needed for an array rapidly increases with each dimension.

Q.14. Given the legal definition: float a[5] = {1, 2, 3};

a. How many elements are in the array? 5
b. What is the type of each element? float
c. What is the index of the first and last element? 0, 4
d. What is the value of the first and last element? 1.0, 0.0

Q.15. Consider the following array declaration and code: int A[8]; answer the following:

a. The number of cells declared in this array is: 8
b. The highest index for the array above is: 7
c. The lowest index for the array above is: 0
d. Rewrite the following code using a "while" loop instead of a "for" loop:

```
for (int i = 0; i < 8; i++)
  A[ i ] = i;
```

Answer:

```
int i=0;
while (i<8)
{
  A[i] = i;
  i++;
}
```

e. Write the code to declare an integer variable sum and store the sum of all values in the array.

```
int sum=0;
for (int k=0; k<8; k++)
    sum = sum +A[k];
```

Q.16. Trace output of the following program segments.

a)

```
int a[6], k, l;
for (k=0; k<=4; k += 2)
{
  a[k] = k;
  a[k+1] = k-1;
}
for (l=5; l>=0; l--)
  cout<<a[l]<<l<<endl;
```

Answer:

35
44
13
22
-11
00

b)

```
int a[6], b[6], k;
for (k=0; k<=4; k+=2)
{   a[k] = k;
    a[k+1] = k * k;
    b[k] = a[k+1];
    b[k+1] = a[k] + k;
}
for (k = 5; k>= 0; k--)
    cout<<k<<a[k]<<b[k]<<endl;
```

Answer:

5168
4416
344
224
100
000

c)

```
int a[10], k, l;
for (k = 9; k >= 1; k -= 2)
{
    a[k] = k * 2;
    a[k-1] = k * 3;
```

Answer:

3
2
9
6
15
10
21
14
27
18

```
    }
for (l = 0; l < 10; l++)
    cout<<a[l]<<endl;
```

d)

```
int a[10], b[10], k, d;
for (k = 0; k < 10; k++)
{
    a[k] = k;
    b[k] = 3 - k / 2;
}
d = 0;
for (k = 0; k < 10; k++)
{
    cout<<a[k]<<b[k]<<endl;
    d = d + a[k] * b[k];
}
```

Answer:

```
03
13
22
32
41
51
60
70
8-1
9-1
```

e)

```
int myArray[8];
    for (int c = 7; c > 2; c--)
            myArray[c] = c * 4;
    for (int p = 3; p < 7; p++)
            cout << myArray[p] << " ";
```

Answer: 12 16 20 24

f)

```
int arr[5] = {4, 7, 5, 3, 0};
int x = 3;
arr[x] = arr[2 * x – 4];
arr[x+1] += 6;
arr[x-1] = arr[x-2];
cout << arr[1] << arr[2] << arr[3] << arr[4];
```

Answer: 7756

Q.17. Trace and find the output of the following program segments:

a)

```
int A[5][2], k, j;
for (k = 0; k <= 4; k++)
    for (j = 0; j < 2; j++)
            A[k][j] = k + j;
    for (k = 4; k >= 0; k -=2)
            cout<<A[k][0]<<A[k][1];
    for (k = 4; k > 0; k -=2)
            cout<<A[k-1][0]<<A[k-1][1];
```

Answer: 4523013412

b)

```
int A[4][5], k, j;
for (k = 0; k < 4; k++)
    for (j = 0; j < 5; j++)
            A[k][j] = 0;
for (j = 0; j < 5; j++)
    for (k = 0; k < 4; k++)
            if (j + k < 3)
                    A[k][j] = 1;
            else if (j + k > 5)
```

Freebooks.pk

```
                    A[k][j] = 2;
            else if  (j - k < -2)
                    A[k][j] = 3;
            else if  (j - k > 1)
                    A[k][j] = 4;
for (k = 0; k < 4; k++)
    for (j = 0; j < 5; j++)
            cout<<A[k][j];
```

Answer: 11144110441000230022

c)

```
int A[4][4], k, j;
for (k = 0; k <= 3; k++)
    for (j = 0; j <= 3; j++)
            A[k][j] = (k+1) * (j+1);
    for (k = 0; k < 4; k++)
            cout<<A[k][k];
    for (k = 0; k < 4; k++)
            cout<<A[3-k][k];
```

Answer: 149164664

d)

```
void main()
{
    int a[6] = {2, 3, 4, 5, 6, 7};
    for (int i = 1; i <=3; i++)
            a[i] = i;
    for (i = 0; i < 6; i++)
            cout << a[i] << " ";
    cout << endl;
}
```

Answer: 2 1 2 3 6 7

e)

```
int A[3] = {33, 21, 34};
for (int i = 0; i < 3; i++)
{
    for (int j = i+1; j < 3; j++)
    {
            if (A[j] > A[i])
            {
                    A[i] += A[j];
                    A[j] = A[i] - A[j];
                    A[i] = A[i] - A[j];
            }
    }
}
for (i=0; i<3; i++)
    cout << A[i] << " " ;
```

Answer: 34 33 21

Q.18. Write the code required declaring an array that contains 5 rows of three columns of integers. Also initialize the array to contain all 3's.

```
int Array[5][3]={{3,3,3,3,3},{3,3,3,3,3},{3,3,3,3,3}};
```

Find errors in the following statements if any.

a)

```
int Temp = 10;
```

Freebooks.pk

```
int Arr[Temp];
```

Answer: Invalid. A variable cannot be used to specify the length of the array.

b)

```
int A[2][3] = {{2,8,1}, {4,3, 9}};
int B[4] = {3, 6, 4, 2, 10, 9};
```

Answer: Too many elements in the initialization of B

c)

```
int A[6];
for (int i=1; i<=6; i++)
{
    cout << "Enter number:";
    cin >> A[i];
}
```

Answer:

A[6] beyond end of array. Use for(int i=0; i<6; i++)

Q.19. Consider the following code:

```
int scores[10][20];
```

a) The number of elements in the scores array is: 200

b) The number of rows in the scores array is: 10

c) The number of columns in the scores array is:20

d) Write the code to initialize all the scores elements to 100:

```
for(int i=0;i<10;i++)
    for(int j=0;j<20;j++)
            scores[i][j]=100;
```

Q.20. Answer the following.

a) Why can't you use binary search on the following array?

int A[10] = {4, 5, 6, 2, 8, 1, -1, 17};

b) Assume we use ordered linear search, in ascending order starting from the smallest element, to search for the value 22 in the array {2, 15, 18, 25, 33, 45, 46, 72, 99}. What values are compared to 22 before concluding it is not there?

Answers:

a) Binary search can be applied for a sorted array only.

b) 2 15 18 25

Multiple Choice

1. **Variable that holds a large group of similar type of data is called:**
   a. Constant b. Scalar value c. Array d. Multiple variable
2. **An array:**
   a. Always begins at index 0
   b. Of two dimensions can be accessed in the following form: x[row,col]
   c. is a contiguous allocation of memory made up of elements
   d. Both a and c
3. **Which of the following statements are true?**
   a. Every element in an array has the same type
   b. The array size is fixed after it is created
   c. The array size used to declare an array must be a constant expression
   d. All
4. **An array with no elements is:**
   A. Illegal in C++ b. legal in C++ c. None

Freebooks.pk

5. **Which of the following is not a simple data type?**
   a. int b. Array c. float d. char
6. **Each element of array has its own:**
   a. Array b. Index c. upper bound d. lower limit
7. **The index of first element of an array is always**
   a. 0 b. 2 c. 4 d. 1
8. **What is the representation of the third element in an array called n?**
   a. n[2] b. n[3] c. n[1] d. n[4]
9. **What is the correct term for n[99]?**
   a. Array variable b. indexed variable- c. array variable d. None
10. **Each element in array has:**
    a. Similar index b. Unique index c. Both a and b d. None
11. **Which of the following declares an array of 10 elements and assigns the value 3.2 to the first element?**
    a. float n[9]; n[0] = 3.2; b. float n[9]; n[1] = 3.2;
    c float numbers[10]; n[0] = 3.2; d. float n[10]; n[1] = 3.2;
12. **How many elements are in the array?** int marks[5];
    a. 5 b. 6 c 4 d. 0
13. **Which line of code will declare an array and initialize it?**
    a. int a[6] = {4, 7, 8, 2, 9, 5}; b. int a[6] {4, 7, 8, 2, 9, 5};
    c. int a [6] = (4, 7, 8, 2, 9, 5); d. int a [6] (4, 7, 8, 2, 9, 5);
14. **In the statement:**
    list[4] = 3.3;
    the value 4 indicates:
    a. Size of array b. Array Element c. Subscript d. Variable
15. **To access an array element, use the array name and the element's**
    a. name b. Value c. data type d. subscript
16. **Which of the following is a valid C++ array definition?**
    a. void n[5]; b. int arr[20]; c. int arr[0]; d. None
17. **A two-dimensional array can be viewed as:**
    a. arguments,parameters b. increment, decrement c. rows, columns d. None
18. **Given the following declaration, where is 77 stored in the n array?**
    int n[ ]={83, 62, 77, 97};
    a. n[0] b.n[1] c.n[2] d.n[4]
19. **An array can easily be stepped through by using a:**
    a. Reference variable b. null value c. for loop d. None
20. **What is the output of the following code?**

```
int main()
{
  int x[5];
  int i;
   for (i = 0; i < 5; i++)
     x[i] = i;
   cout << x[i] << " ";
}
```

a. 0 1 2 3 4 b. 4 c. 0 d. None

Freebooks.pk

21. Assume int t[] = {1, 2, 3, 4}. What is t[4]?
a. 0 b. 3 c. 4 d. accessing t[4] may result in a runtime error

22. An array that will hold 5 names with 20 characters in the name would be declared using which statement?
a. char names[5][20]; b. char names[20][5]; c. char names[21][5]; d. char names[5][21];

23. Which of the following statements is valid?
a. int i[] = {3, 4, 3, 2}; b. int i[4] = {3, 4, 3, 2}; c. double d[30]; d. All

24. When an array is sorted from highest to lowest, it is said to be in:
a. Ascending order b. descending order c. Forward order d. Reverse order

25. Which of the following is adequate for searching through small arrays?
a. Linear search b. binary search c. Bubble sort d. None

26. How many elements does the following array have?
int n[1000];
a. 1000 b. 998 c. 0 d. 999

27. Which of the following correctly declare an array that can hold up to 3 rows of 5 columns of doubles?
a. double array[3],[5]; b. int array[3](int [5]);
c. double array[3][5]; d. double array[3,5];

28. what is the output of the following c++ code?
```
int n[] = {56, 90, 86, 97, 120 };
cout << n[3] << endl;
```
a. 97 b. 98 c. 99 d. 90

29. what is the output of the following c++ code?
```
int n[5];
for (int i = 1; i < 5; i++)
        n[i - 1] = i + 1;
for (int i = 0; i < 3; i++)
        cout << n[i];
```
a. 234 b. 240 c. 334 d. 221

30. What is the output of the following c++ code?
```
int n[4] = { 87, 63 };
cout << n[3] << endl;
```
a. 0 b. –1 c. 1 d. 3

Answers

1. c	2. d	3. d	4. a	5. b
6. b	7. a	8. a	9. b	10. b
11. c	12. a	13. a	14. c	15. d
16. b	17. c	18. c	19. c	20. c
21. d	22. d	23. d	24. b	25. a
26. a	27. c	28. a	29. a	30. a

Fill in the Blanks

1. An _____ is a collection of consecutive memory locations.
2. Each memory location is called an _____ of array.
3. The number of elements in an array is called the _____of array.
4. An individual element in an array is accessed with the help of an_____.
5. The process of specifying array name, length and data type is called _____.

Freebooks.pk

6. Arrays are used to store many values of similar type with _____.
7. The process of assigning values to array elements at the time of declaration is called _____.
8. Each individual element of an array is accessed by specifying _____ and _____.
9. The syntax of accessing an individual element of an array is _____.
10. An easier and faster way of accessing array elements is using _____.
11. Sequential search is also called _____.
12. _____ is a process of arranging the values of array in a particular order.
13. Two-dimensional array can be considered as a table that consists of rows and _____.
14. Each element in 2-D array is referred with the help of _____indexes.
15. An array's size declarator must be a(n)_____ with a value greater than _____.
16. Arrays may be _____ at the time they are _____.

Answers

1. Array	2. Element
3. Length	4. Index
5. Array declaration	6. Single name
7. Array initialization	8. Name of array, index element
9. Array_Name[Index];	10. Loops
11. linear search or serial search	12. Sorting
13. column	14. two
15. constant integer expression, zero	16. initialized, declared

True / False

1. The first element in an array called names is names[1].
2. The last element in the array defined by float pay[12]; is pay[12].
3. The subscript of an array must be an integer or an integer expression.
4. The first character of a string called name is name[0].
5. Arrays can contain different types of data
6. Multidimensional arrays are more efficient in that is executes faster then regular arrays.
7. Multidimensional arrays can be created for the exact same variable types as single dimensional arrays.
8. An array initialization list must be placed on one single line
9. If an array is partially initialized, the uninitialized elements will be set to zero
10. An individual array element can be processed like any other type of C++ variable.
11. The elements of an array can be stored in noncontiguous memory.
12. The sequential search is good only for very short list.
13. A binary search is much faster than a sequential search.

Answers

1. F	2. F	3. T	4. T	5. F
6. F	7. T	8. F	9. T	10. T
11. F	12. T	13. T		

Freebooks.pk

CHAPTER 8

STRUCTURES

Chapter Overview

8.1 Structures
- 8.1.1 Declaring a Structure
- 8.1.2 Defining Structure Variable
- 8.1.3 Accessing Members of Structure Variable
- 8.1.4 Initializing Structure Variables
- 8.1.5 Assigning One Structure Variable to Other
- 8.1.6 Array as Member of Structure

8.2 Array of Structures
- 8.2.1 Initializing Array of Structures

8.3 Nested Structure
- 8.3.1 Accessing Members of Nested Structure
- 8.3.2 Initializing Nested Structure

8.4 Union

8.5 Enumerations

Programming Exercise

Exercise Questions

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

8.1 Structures

A **structure** is a collection of multiple data types that can be referenced with single name. It may contain similar or different data types. The data items in a structure are called structure **elements**, **members** or **fields**.The structures are used to join simple variables together to form complex variables.

The difference between an array and a structure is that array consists of a set of variables of same data type. However, a structure may consist of different data types.

The structures are used to define new data types. The user can define a new data type that may contain different types of data. A simple variable can store only one value at a time. But a structure variable can store multiple values at the same time.

8.1.1 Declaring a Structure

A structure is declared by using the keyword **struct** followed by the structure name. The structure members are defined with their type inside the opening and closing braces { }. The closing braces is ended with a semicolon. The **declaration** tells the compiler about the details of the structure. The compiler does not allocate any memory.

The syntax for declaring a structure is as follows:

```
struct Struct_Name
{
   Data_Type1 Identifier1;
   Data_Type2 Identifier2;
          :
          :
};
```

struct	It is the keyword that is used to declare a structure.
Struct_Name	It is the name of the structure. The name of a structure is also known as **structure tag**. It can be any legal identifier.
Data_Type1	It indicates data types of first member of the structure.
Identifier1	It indicates the name of first member of the structure.
Data_Type2	It indicates data types of second member of the structure.
Identifier2	It indicates the name of second member of the structure.

The structure declaration is terminated by semicolon. The structure declaration is also called **structure specifier.**

Example

The following example declares a structure Student with four member variables:

```
struct Student
{
   int RollNo, Marks;
   float Average;
   char Grade;
};
```

The above structure contains four member variables. The above declaration has created a new data type **Student**. A variable of type **Student** can store four values at one time.

8.1.2 Defining Structure Variable

The structure variable can be defined after the declaration of a structure. The process of defining a structure variable is same as defining a variable of basic types such as **int** and **char**. The **definition** tells the compiler to allocate memory space for the variable. The compiler automatically allocates sufficient memory according to the elements of the structure.

Syntax

The syntax of defining a structure variable is as follows:

```
Struct_Name Identifier;
```

Struct_Name is the name of the structure.

Identifier is the name of the variable to be defined.

Example

```
Student Usman;
```

The above example defines a structure variable **Usman**. The variable consists of four members as specified in structure declaration. The statement allocates the memory space for all four members of the structure as follows:

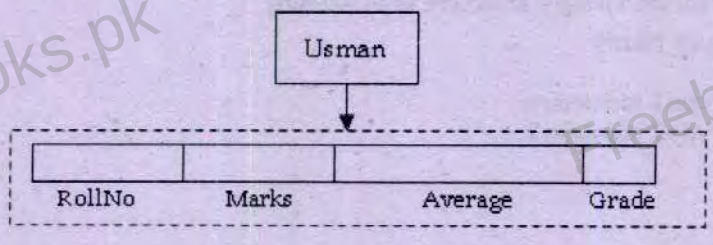


Figure 8.1: A structure variable **Usman** of type **Student**

The structure variable **Usman** will occupy 9 bytes in the memory. The member varialbe RollNo occupies 2 bytes, Marks occupies 2 bytes, Average occupies 4 bytes and Grade occupies 1 bytes. The total memory occupied by any variable of type Student will be 9 bytes.

8.1.3 Accessing Members of Structure Variable

Any member of a structure variable can be accessed by using dot operator. The name of the structure variable is written on the left side of the dot. The name of member variable is written on right side of the dot. The dot operator is also known as **member access operator**.

Syntax

The syntax of accessing a member of structure variable is as follows:

```
Struct_Var.Mem_Var;
```

Struct_Var is the name of the structure variable.

Mem_Var is the name of the member variable to be accessed.

Example

```
Student Usman;
Usman.RollNo = 10;
Usman.Marks = 786;
Usman.Average = 89.52;
Usman.Grade = 'A';
```

Freebooks.pk

The above example declares a structure variable **Usman** of type **Student**. It uses four assignment statements to store different values in member variables of **Usman**. The dot operator is used in each statement to access a separate member variable. The same method can be used to input value in member variables and display these values as follows:

```
cin>>Usman.RollNo;
cin>>Usman.Marks;
cin>>Usman.Average;
cin>>Usman.Grade;

cout<<Usman.RollNo;
cout<<Usman.Marks;
cout<<Usman.Average;
cout<<Usman.Grade;
```

Program 8.1

Write a program that declares a structure to store Roll No, Marks, Average and Grade of a student. The program should define a structure variable, inputs the values and then displays these values.

```
#include <iostream.h>
#include <conio.h>
struct Student
{
  int rno;
  int marks;
  float avg;
  char grade;
};
void main()
{
  clrscr();
  Student s;
  cout<<"Enter roll no: ";
  cin>>s.rno;
  cout<<"Enter marks: ";
  cin>>s.marks;
  cout<<"Enter average: ";
  cin>>s.avg;
  cout<<"Enter grade: ";
  cin>>s.grade;
  cout<<"You entered the following details: \n";
  cout<<"Roll No: "<<s.rno<<endl;
  cout<<"Marks: "<<s.marks<<endl;
  cout<<"Average: "<<s.avg<<endl;
  cout<<"Grade: "<<s.grade<<endl;
  getch();
}
```

Output:

```
Enter roll no: 1
Enter marks: 872
Enter average: 76.53
Enter grade: A
You entered the following details:
Roll No: 1
Marks: 872
Average: 76.53
Grade: A
```

Program 8.2

Write a program that declares a structure to store day, month and year of birth date. It inputs three values and displays date of birth in dd/mm/yy format.

```
#include <iostream.h>
```

Freebooks.pk

```
#include <conio.h>
struct birth
{
  int day;
  int month;
  float year;
};
void main()
{
  clrscr();
  birth b;
  cout<<"Enter day of birth: ";
  cin>>b.day;
  cout<<"Enter month of birth: ";
  cin>>b.month;
  cout<<"Enter year of birth: ";
  cin>>b.year;
  cout<<"\nYour date of birth is "<<b.day<<"/"<<b.month<<"/"<<b.year;
  getch();
}
```

Output:

```
Enter day of birth: 26
Enter month of birth: 5
Enter year of birth: 94

Your date of birth is 26/5/94
```

Program 8.3

Write a program that declares a structure to store BookID, price and pages of a book. It defines two structure variables and inputs values. It displays the record of most costly book.

```
#include <iostream.h>
#include <conio.h>
struct Book
{
  int id;
  int pages;
  float price;
};
void main()
{
  clrscr();
  Book b1, b2;
  cout<<"Enter id, pages and price of book: ";
  cin>>b1.id>>b1.pages>>b1.price;
  cout<<"Enter id, pages and price of book: ";
  cin>>b2.id>>b2.pages>>b2.price;
  cout<<"\nThe most costly book is as follows:\n";

  if(b1.price > b2.price)
  {
    cout<<"BookID: "<<b1.id<<endl;
    cout<<"Pages: "<<b1.pages<<endl;
    cout<<"Price: "<<b1.price<<endl;
  }
  else
  {
    cout<<"BookID: "<<b2.id<<endl;
    cout<<"Pages: "<<b2.pages<<endl;
```

Output:

```
Enter id, pages and price of book: 1  250  160
Enter id, pages and price of book: 2  485  200

The most costly book is as follows:
BookID: 2
Pages: 485
Price: 200
```

Freebooks.pk

```
        cout<<"Price: "<<b2.price<<endl;
    }
    getch();
}
```

8.1.4 Initializing Structure Variables

The process of assigning values to member variables of structure is called initialization of structure variables. The assignment operator is used to initialization. The declaration of structure variable is specified on the left side of assignment operator. The values for initialization are written on the right side of assignment operator. The values are written in braces. The values are written in the same sequence in which they are specified in structure declaration. Each value is separated by comma.

Syntax

The syntax for initializing structure variables is as follows:

```
Struct_Name Identifier = {Value1, Value2...};
```

Struct_Name It is the name of the structure.
Identifier It is the name of structure variable.
Value1, Value2 These are the values that are used to initialize the structure variable.

Example

```
Student Usman = {1, 786, 89.52, 'A'};
```

The above example declares a structure variable **Usman** of type **Student**. It initializes structure variable with the values written in braces on the right side of assignment operator.

Program 8.4

Write a program that declares a structure to store employee id and salary of an employee. It defines and initializes a structure variable and displays it.

```
#include <iostream.h>
#include <conio.h>
struct emp
{
  int eid;
  int sal;
};

void main()
{
  clrscr();
  emp e = {20, 18500};
  cout<<"Employee ID: "<<e.eid<<endl;
  cout<<"Salary: "<<e.sal<<endl;
  getch();
}
```

Output:
Employee ID: 20
Salary: 18500

Program 8.5

Write a program that uses a structure to store three parts of phone number i.e. National code, Area code, Number separately. Create two variables of structure phone. Initialize one variable and get inputs from the user in the second variable and then display both numbers.

Freebooks.pk

```
#include <iostream.h>
#include <conio.h>
struct Phone
{
  int ncode;
  int acode;
  long number;
};
void main()
{
  clrscr();
  Phone p1, p2 = {92, 41, 9220083};
  cout<<"Enter national code: ";
  cin>>p1.ncode;
  cout<<"Enter area code: ";
  cin>>p1.acode;
  cout<<"Enter phone number: ";
  cin>>p1.number;
  cout<<"Phone Number 1: +";
  cout<<p1.ncode<<"-"<<p1.acode<<"-"<<p1.number<<endl;
  cout<<"Phone Number 2: +";
  cout<<p2.ncode<<"-"<<p2.acode<<"-"<<p2.number<<endl;
  getch();
}
```

Output:

```
Enter national code: 61
Enter area code: 3
Enter phone number: 8275612
Phone Number 1: +61-3-82756125
Phone Number 2: +92-41-9220083
```

Program 8.6

Write a program that uses a structure to store employee number, name, hours worked, hourly rate and gross pay. The program inputs employee number, name, hours worked and hourly rate the user, calculates gross pay and then displays all employee data on screen.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
struct PayRoll
{
  int empNumber;
  char fname[50], sname[50];
  double hours, payRate, grossPay;
};
void main()
{
  clrscr();
  PayRoll employee;
  cout << "Enter the employee's number: ";
  cin  >> employee.empNumber;
  cout << "Enter employee's first name: ";
  cin>>employee.fname;
  cout << "Enter employee's second name: ";
  cin>>employee.sname;
  cout << "Enter number of hours worked: ";
  cin  >> employee.hours;
  cout << "Enter hourly pay rate: ";
  cin  >> employee.payRate;
```

Output:

```
Enter the employee's number: 10
Enter employee's first name: Ali
Enter employee's second name: Ahmed
Enter number of hours worked: 50
Enter hourly pay rate: 100

Employee's Payroll Data:
Name: Ali Ahmed
Number: 10
Hours worked: 50
Hourly pay rate: 100
Gross pay: Rs.  5000.00
```

```
    employee.grossPay = employee.hours * employee.payRate;
    cout << "\nEmployee's Payroll Data:\n";
    cout << "Name: " << employee.fname << " " <<employee.sname<<endl;
    cout << "Number: " << employee.empNumber << endl;
    cout << "Hours worked: " << employee.hours << endl;
    cout << "Hourly pay rate: " << employee.payRate << endl;
    cout << "Gross pay: Rs. ";
    cout.setf(ios::fixed, ios::floatfield);
    cout.setf(ios::showpoint);
    cout << setprecision(2)<< employee.grossPay << endl;
    getch();
}
```

8.1.5 Assigning One Structure Variable to Other

A structure variable can be initialized by assigning another structure variable to it by using assignment operator as follows:

```
Student Usman = {1, 786, 89.52, 'A'};
Student Abdullah = Usman;
```

The first line declares a structure variable Usman and initializes it. The second line declares another structure variable Abdullah and initializes it with the values of Usman. The values of both structure variables are now same. One structure variable can be assigned to another structure variable only if both are of same type.

Program 8.7

Write a program that declares a structure to store marks and grade of a student. It defines two structure variables. It inputs the values in one variable and assigns that variable to the second variable. It finally displays the values of both variables.

```
#include <iostream.h>
#include <conio.h>
struct Marks
{
  int m;
  char g;
};
void main()
{
  clrscr();
  Marks a, b;
  cout<<"Enter marks: ";
  cin>>a.m;
  cout<<"Enter grade: ";
  cin>>a.g;
  b = a;
  cout<<"The first record is as follows:\n";
  cout<<"Marks: "<<a.m<<endl;
  cout<<"Grade: "<<a.g<<endl;
  cout<<"The second record is as follows:\n";
  cout<<"Marks: "<<b.m<<endl;
  cout<<"Grade: "<<b.g<<endl;
  getch();
}
```

Output:

```
Enter marks: 786
Enter grade: A
The first record is as follows:
Marks: 786
Grade: A
The second record is as follows:
Marks: 786
Grade: A
```

Freebooks.pk

8.1.6 Array as Member of Structure

A structure may consist of different types of data. The member variables can be simple data types such as **int**, **char** and **float**. The member variables can also be complex like arrays.

Example

```
struct Student
{
  int RollNo;
  int Marks[5];
};
```

The above example declares a structure **Student** with two members. The first member is a simple variable of type **int**. The second member is an array of integers. It can be used to store the marks of five subjects.

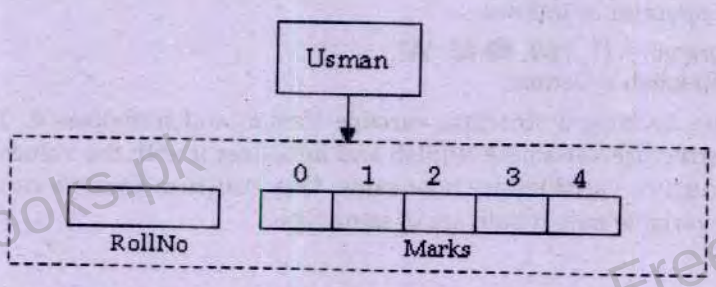


Figure 8.2: A structure variable **Usman** with Array as member

The array stored in a structure can be accessed by using both dot operator and index. The dot operator is used to refer to the array. The index is used to access the individual element of the array as follows:

```
Student Usman;
Usman.RollNo = 10;
Usman.Marks[0] = 89;
Usman.Marks[1] = 93;
Usman.Marks[2] = 83;
Usman.Marks[3] = 79;
Usman.Marks[4] = 95;
```

8.1.6.1 Initializing a Structure with Array as Member

The structure that contains an array as member variable can be initialized in the same way as initializing a simple structure variable. The values are written in braces and each value is separated by comma. Additinally, the values for the member array are written in nested braces. The following example intializes the structure variable **Usman**:

```
Student Usman = {10, {89, 93, 83, 79, 95}};
```

Program 8.8

Write a program that declares a structure to store roll no and marks of five subjects. It defines a structure variable, inputs the values and displays roll no, marks and average marks.

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
struct Test
{
  int rno;
  int m[5];
};
void main()
{
  clrscr();
  Test r;
  int i, t = 0;
  float avg = 0.0;
  cout<<"Enter roll no: ";
  cin>>r.rno;
  for(i=0; i<5; i++)
  {
    cout<<"Enter marks: ";
    cin>>r.m[i];
    t = t + r.m[i];
  }
  avg = t / 5.0;
  cout<<"Roll No: "<<r.rno<<endl;
  cout<<"Total marks: "<<t<<endl;
  cout<<"Average: "<<avg<<endl;
  getch();
}
```

Output:

```
Enter roll no: 10
Enter marks: 80
Enter marks: 72
Enter marks: 53
Enter marks: 94
Enter marks: 38
Roll No: 10
Total marks: 337
Average: 67.400002
```

8.2 Array of Structures

An array is a collection of same type of data. An array can be of simple data type such as **int**, **char** or **float** etc. Similarly, an array can be of user-defined type such as a structure. An array of structure is a type of array in which each element contains a complete structure. It can be used to store many records.

Example

```
struct Book
{
  int BookID;
  int Pages;
  float Price;
};
Book b[3];
```

The above example declares a structure **Book**. It defines an array of structures **b[3]**. The array can store the records of three books.

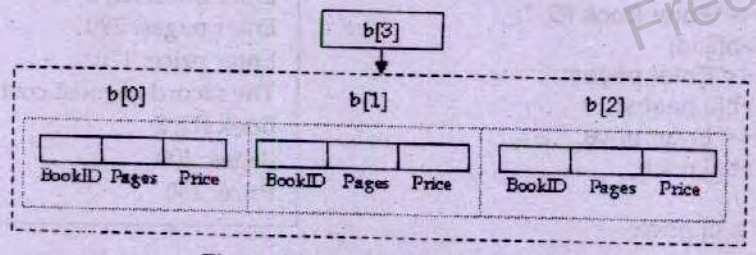


Figure 8.3: An array of structures

Freebooks.pk

The array is accessed by using its index. The structure is accessed by using using dot operator. An array of structures can be accessed as follows:

```
b[0].BookID = 1;
b[0].Pages = 230;
b[0].Price = 125;
```

8.2.1 Initializing Array of Structures

An array of structures can be initialized at the time of declaration by writing the values in braces. The values for each element of the array are written in separate inner braces.

Example

```
struct Book
{
  int BookID;
  int Pages;
  float Price;
};

Book b[3] = {{1,230,125.50},{2,480,185.75},{3,360,145.50}};
```

The above example declares an array of structures and initializes it. The values are written in braces. Each inner brace is used to initialize one element of the array.

Program 8.9

Write a program that declares a structure to store id, pages and price of a book. It defines an array of structures to store the records of five books. It inputs the records of five books and displays the record of most costly book.

```
#include <iostream.h>
#include <conio.h>
struct Book
{
  int id;
  int pages;
  int price;
};

void main()
{
  clrscr();
  Book b[5];
  int i, max, m;
  for(i=0; i<5; i++)
  {
    cout<<"Enter Book ID: ";
    cin>>b[i].id;
    cout<<"Enter pages: ";
    cin>>b[i].pages;
    cout<<"Enter price: ";
    cin>>b[i].price;
  }
  max = b[0].price;
  m = 0;
```

Output:

```
Enter Book ID: 1
Enter pages: 250
Enter price: 120
Enter Book ID: 2
Enter pages: 400
Enter price: 250
Enter Book ID: 3
Enter pages: 150
Enter price: 90
Enter Book ID: 4
Enter pages: 580
Enter price: 200
Enter Book ID: 5
Enter pages: 280
Enter price: 170
The record of most costly book:
Book ID: 2
Pages: 400
Price: 250
```

Freebooks.pk

```
    for(i=0; i<5; i++)
      if(b[i].price > max)
      {
      max = b[i].price;
      m = i;
      }
    cout<<"\nThe record of most costly book:\n";
    cout<<"Book ID: "<<b[m].id<<endl;
    cout<<"Pages: "<<b[m].pages<<endl;
    cout<<"Price: "<<b[m].price<<endl;
    getch();
  }
```

8.3 Nested Structure

A structure within a structure is known as **nested structure**. A nested structure is created when the member of a structure is itself a structure.

Example

```
struct A
{
  int n;
  float b;
};

struct B
{
  char c;
  A x;
};
```

The above example declares two structures **A** and **B**. The structure **A** contains simple member variables **n** and **b**. The structure **B** also contains two member variables. The first member is a character variable **c**. The second member is a structure variable **x**. It is an example of a nested structure.

8.3.1 Accessing Members of Nested Structure

The member variable of nested structure can be accessed using multiple dot operators. The first dot operator refers the member variable of outer structure. The second dot operator refers the inner structure and so on.

Example

```
B rec;
```

The above line creates a structure variable **rec** of type **B**. It is a nested structure that contains another structure variable **x** as its member as follows:

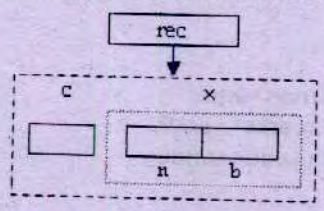


Figure 8.4: A nested structure

Freebooks.pk

The following statements can be used to store values in the above nested structure:

```
rec.c = 'A';
rec.x.n = 10;
rec.x.b = 50.25;
```

The first statement uses one dot operator as it refers to simple member variable of the structure. The last two statements use two dot operators as they refer to the members of the nested structure variables.

8.3.2 Initializing Nested Structure

The method of initializing nested structure is same as initializing an array of structure. The values of simple member variables are written in braces. The values of inner structure are written in separate inner braces. Each value is separated by comma.

Example

```
B rec = {'A', {10, 50.25}};
```

The above example declares a nested structure **rec**. It initializes it with different values. The value of simple member variable **c** is written in outer braces. The values of inner structure variable **x** are written in inner braces.

Program 8.10

Write a program that uses two structures Result and Record. The Result structure stores marks and grade, Record structure stores roll number and a Result type. The program declares a variable of type Record, inputs roll number, marks and grade. It finally displays these values on the screen.

```
#include <iostream.h>
#include <conio.h>
struct Result
{
  int marks;
  char grade;
};
struct Record
{
  int rno;
  Result r;
};
void main()
{
  Record rec;
  cout<<"Enter roll no: ";
  cin>>rec.rno;
  cout<<"Enter marks: ";
  cin>>rec.r.marks;
  cout<<"Enter grade: ";
  cin>>rec.r.grade;
  cout<<"Roll No: "<<rec.rno<<endl;
  cout<<"Marks: "<<rec.r.marks<<endl;
  cout<<"Grade: "<<rec.r.grade<<endl;
  getch();
}
```

Output:

```
Enter roll no: 1
Enter marks: 100
Enter grade: A
Roll No: 1
Marks: 100
Grade: A
```

Freebooks.pk

Program 8.11

Write a program that uses two structures date and phonebook. The date structure stores day, month and year. Phonebook structure stores name, city, telephone and a date type. The program declares a variable of type phonebook, inputs values displays the values.

```
#include <conio.h>
#include <iostream.h>
struct date
{
  int day;
  int month;
  int year;
};
struct phonebook
{
  char name[40];
  char city[40];
  int tel;
  date birthday;
};
void main()
{
  clrscr();
  struct phonebook a1;
  cout<<"Enter name: ";
  cin>>a1.name;
  cout<<"Enter city: ";
  cin>>a1.city;
  cout<<"Enter phone number: ";
  cin>>a1.tel;
  cout<<"Enter date of birth (dd-mm-yy): ";
  cin>>a1.birthday.day>>a1.birthday.month>>a1.birthday.year;
  cout<<"\nThe size of the structure variable is : "<<sizeof(a1);
  cout<<"\nThe entry made is:";
  cout<<a1.name<<"-"<<a1.city<<"-"<<a1.tel;
  cout<<"\nBirthday is on : "<<a1.birthday.day<<"-"<<a1.birthday.month<<"-
  "<<a1.birthday.year;
  getch();
}
```

How above Program Works?

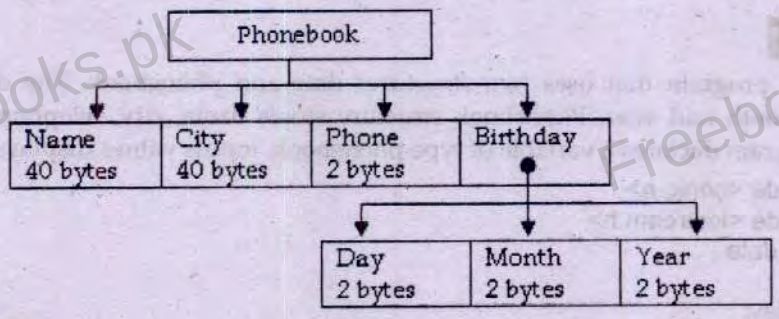
The above program declares a nested structure. The following will declare a structure variable **birthday** that belongs to the structure **date**:

```
date birthday;
```

The **date** has three member elements namely **day**, **month** and **year**. The members of **birthday** can be refered as follows:

```
a1.birthday.day
```

Freebooks.pk



Program 8.12

Write a program that uses three structures Dimension, Results and Rectangle. The Dimension structure stores length and width, Result structure stores area and perimeter and Rectangle stores two variables of Dimension and Results. The program declares a variable of type Rectangle, inputs length, width, calculates area and width and then displays the result.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
struct Dimension
{
  double length;
  double width;
};
struct Results
{
  double area;
  double perimeter;
};
struct Rectangle
{
  Dimension size;
  Results   stat;
};
void main()
{
  clrscr();
  Rectangle box;
  cout << "Enter the length of a rectangle: ";
  cin  >> box.size.length;
  cout << "Enter the width of a rectangle: ";
  cin  >> box.size.width;
  box.stat.area = box.size.length * box.size.width;
  box.stat.perimeter = 2 * box.size.length + 2 * box.size.width;
  cout.setf(ios::fixed, ios::floatfield);
  cout.setf(ios::showpoint);
  cout << setprecision(2);
  cout << "The area of rectangle is " << box.stat.area << endl;
  cout << "The perimeter of rectangle is " << box.stat.perimeter << endl;
  getch();
}
```

Output:

```
Enter the length of a rectangle: 5
Enter the width of a rectangle: 4.5
The area of rectangle is 22.50
The perimeter of rectangle is 19.00
```

Freebooks.pk

8.4 Union

Unions are similar to structures in certain aspects. Unions are used to group together variables of different data types. The individual members can be accessed using **dot operator**. The difference between structure and union is in the allocation of memory space. A structure allocates total space required for a structure variable. However, a union allocates the space required by one element that occupies the maximum size.

Syntax

The syntax of declaring a union is as follows:

```
union union_name
{
  member_type1 member_name1;
  member_type2 member_name2;
  member_type3 member_name3;
  .
  .
};
```

Example

Suppose a union consists of three variables of different types. The union will allocate space only to the variable that requires the maximum memory space.

```
union Test
{
  char ch;
  int n;
  float f;
} t;
```

The above statements declares a union **Test** and defines a variable **t**. The declaration will allocate only 4 bytes. The compiler knows that a character requires 1 byte, integer requires 2 bytes and float requires 4 bytes. Therefore, it allocates a total of only 4 bytes.

It means that when only one element will have a valid value at any time. In the above example, we can have a valid value for either **t.ch** or **t.n** or **t.f**. All elements of a union cannot have a valid value at the same time.

Program 8.13

Write a program that uses a union **shirt** to store size, chest and height. The program inputs size, chest measurement and height measurement and displays the values.

```
#include<iostream.h>
#include <conio.h>
union shirt
{
  char size;
  int chest;
  int height;
};
void main( )
{
  shirt mine;
  clrscr();
```

Freebooks.pk

```
    cout<<"\nSize of union: "<<sizeof(mine);
    cout<<"\nWhat size (S/M/L)? ";
    cin>>mine.size;
    cout<<"\nThe size is : "<<mine.size;
    cout<<"\nThe chest measurement is : "<<mine.chest;
    cout<<"\nThe height measurement is : "<<mine.height;
    cout<<"\n\nWhat is the chest measurement? ";
    cin>>mine.chest;
    cout<<"\nThe size is : "<<mine.size;
    cout<<"\nThe chest measurement is : "<<mine.chest;
    cout<<"\nThe height measurement is : "<<mine.height;
    cout<<"\n\nWhat is the height measurement? ";
    cin>>mine.height;
    cout<<"\nThe size is : "<<mine.size;
    cout<<"\nThe chest measurement is : "<<mine.chest;
    cout<<"\nThe height measurement is : "<<mine.height;
    getch();
}
```

How above Program Works?

The above program declares a union that consists of three variables of different types. It inputs different values in different variables. When the user enters the value for one variable, the values of other variables become invalid.

8.4.1 Difference between Structure and Union

The difference between structure and union is as follows:

Structure	Union
1. Every structure member is allocated memory when a structure variable is defined.	1. The memory equivalent to the largest item is allocated commonly for all members.
2. All structure variables can be initialized at a time.	2. Only one union member can be initialized at a time.
3. Structure is used when all members are to be used independently in a program.	3. Union is used when its members are not to be accessed at the same time.

8.5 Enumerations

Enumerations are used to create new data types. An enumeration consists of a list of values. Each value has a unique number that starts from 0. An enumeration may contain the values which are different from the values of fundamental data types.

Syntax

The syntax of declaring an enumeration is as follows:

```
enum identifier
{
    value1,
    value2,
    value3,
    .
    .
};
```

Freebooks.pk

enum It is the keyword that is used to declare enumerations.

identifier It is the name of enumeration to be declared.

Example

The following example will create an enumeration to store colors:

```
enum colors {black, blue, green, cyan, red, purple, yellow, white};
```

The above enumeration does not contain any fundamental data type in the declaration. The list of values in braces indicates the possible values that can be stored in a variable of type **color**.

```
colors mycolor;
mycolor = blue;
```

Each value in an enumeration is always assigned an integer value. By default, the value starts from 0 and so on. However, the values can be assigned in different way also.

```
enum week {Saturday=1,Sunday, Monday, Tuesday, Wednesday, Thursday, Friday};
```

The above example declares an enumeration **week**. The first value is assigned the 1. In this case, 1 is incremented in the initial value and the new value is assigned to next element. The value of Sunday is 2, Monday is 3 and so on.

Program 8.14

Write a program that declares an enumeration to store months of a year.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  enum year {January, February, March, April, May, June, July, August, September,
  October, November, December};
  year y;
  y = March;
  cout<<"The value of y is "<<y<<endl;
  getch();
}
```

Output:
The value of y is 2

How above Program Works?

The above program declares an enumeration with 12 values. The integer value of first element starts from 0 and so on. The program declares a variable **y** of type **year**. The program assigns the March to **y** and then displays this value.

Programming Exercises

1. Write a program that declares a structure to store the distance covered by an player along with the minutes and seconds taken to cover the distance. The program should input the records of two players and then display the record of the winner.
2. Write a program that declares a structure to store the code number, salary and grade of an employee. The program defines two structure variables, inputs records of two employees and then displays the record of the employee with more salary.

Freebooks.pk

3. Write a program that declares a structure to store income, tax rate and tax of a person. The program defines an array of structure to store the record of five persons. It inputs income and tax rate of five persons and then dispalys the tax payable.
4. Write a program that declares a structure Book to store BookID, book name and price. It declares another structure Order that contains OrderID and an array of Book of length 5. The program should define a variable of type Order and input the values from the user. The program finnally displays the values.

Exercise Questions

Q.1. Define a structure.

A **structure** is a collection of multiple data types that can be referenced with single name. It may contain similar or different data types. The data items in a structure are called structure **elements**, **members** or **fields**.The structures are used to join simple variables together to form complex variables.

Q.2. What is the main difference between array and structure?

The difference between an array and a structure is that array consists of a set of variables of same data type. However, a structure may consist of different data types.

Q.3. What is difference between declaring and defining a structure?

The **declaration** tells the compiler about the details of the structure. The compiler does not allocate any memory. The **definition** tells the compiler to allocate memory space for the variable. The compiler automatically allocates sufficient memory according to the elements of the structure.

Q.4. How are members of structure accessed?

Any member of a structure variable can be accessed by using dot operator. The name of the structure variable is written on the left side of the dot. The name of member variable is written on right side of the dot. The dot operator is also known as **member access operator**.

Q.5. What is a nested structure?

A structure within a structure is known as **nested structure**. A nested structure is created when the member of a structure is itself a structure.

Q.6. Define union. What is the difference between structure and union?

Unions are used to group together variables of different data types. The individual members can be accessed using **dot operator**. The difference between structure and union is in the allocation of memory space. A structure allocates total space required for a structure variable. However, a union allocates the space required by one element that occupies the maximum size.

Q.7. Write the code required to define a structure for the following types of data:

a. time (hours, minutes, seconds)
b. date (Name of the Month, Day, Year)
c. Student (rollno, name, class, fee)

a.

```
struct time
{
 int hours;
 int minutes;
 int seconds;
};
```

b.

```
struct date
{
    char Name[10];
    int Day;
    int Year;
};
```

Freebooks.pk

c.

```
struct student
{
    int rollno;
    char Name[20], Class[10];
    int fee;
};
```

Q.8. Declare an array of 10 structures for each of the data types you just declared:

a. time (hours, minutes, seconds) **Ans:** time TIMES[10];

b. date (Name of the Month, Day, Year) **Ans:** date DATES[10];

c. Student (includes rollno, name, class, fee etc) **Ans:** student s[10];

Q.9. Find the errors in the following code. Give the line number and error description.

```
1  struct S1 {
2          int a;
3          S2 b;
4  };
5  struct S2 {
6     int a;
7     S1 b;
8     S2 c;
9  };
10 S2 s = 3;
```

Answer:

Line 3: member b uses struct 'S2' before it is defined

Line 8: recursive definition of S2 not allowed

Line 10: assigning an integer value to a variable of type struct S2 is illegal

Q.10. Answer the following

a) Write a struct called Location which has two integer members, latitude and longitude.

```
struct Location
{
    int latitude, longitude;
};
```

b) Write a struct City with three members, a string called name, an integer called population and a Location called loc.

```
struct City
{
        char name;
        int population;
        Location loc;
};
```

c) Write a struct Country with four members, a string called name, an integer called population, an integer called n_cities (the number of cities in country) and an array of City's called "cities".

```
struct State
{
        char name[50];
        int population, n_cities;
        City cities[100];
};
```

Freebooks.pk

Multiple Choice

1. **Which of the following is required after the closing brace of the structure declaration?**
   a. Period b. colon c. semicolon d. None
2. **A structure_____ contain members of the same data type.**
   a. can b. shouldn't c. cannot d. Noe
3. **Before a structure can be used, it must be**
   a. Declared b. Initialized c. Deallocated d. None
4. **The name of the structure is referred to as its**
   a. Parameter b. data type c. argument d. None
5. **Which of the following allows you to access structure members.**
   a. Structure access operator b. dot operator c. Both a and b d. None
6. **Which of the following assigns a value to the hrsWage member of emp[2]?**
   a. emp[2]->hourlyWage = 100.00; b. emp2.hrsWage = 3.00;
   c. hrsWage[2].emp = 71.50 d. emp[2].hrsWage = 400.00;
7. **Look at the following structure declaration.**

```
struct student
{
  char name[20];
  int rollno;
};
```

   In this declaration, student is
   a. an array b. member c. tag d. None
8. **Look at the following structure declaration.**

```
struct student
{
  char name[20];
  int rollno;
};
```

   In this declaration, rollno is
   a. member b. a tag c. an array d. None

Answers

1. c	2. a	3. a	4. d
5. b	6. d	7. c	8. a

Fill in the Blanks

1. A _____ is a collection of multiple data types that can be referenced with single name
2. A structure is declared by using the keyword _____ followed by the structure name.
3. The structure declaration is terminated by _____.
4. The structure declaration is also called _____.
5. Any member of a structure variable can be accessed by using _____.
6. A structure within a structure is known as _____.
7. The name of a structure type is called the_____.
8. The dot operator also called_____.

Freebooks.pk

9. A structure value is a collection of smaller values called_____

Answers

1. Structure	2. struct	3. semicolon
4. Structure specifier	5. dot operator	6. Nested structure
7. Structure tag	8. Structure member access operator	9. member values

True/ False

1. A struct cannot contain members with varying data types.
2. The structure tag can be any legal identifier.
3. Once a structure definition has been given, the structure type can be used like predefined types int, char etc.
4. A structure variable can hold values just like any other variable can hold values.
5. Each structure type specifies a list of member names.
6. A structure value is a collection of smaller values called member values.
7. There is one member value for each member name declared in the structure definition.
8. Two or more structure types may use the same member names.
9. The assignment operator is used to specify a member variable of a structure variable.
10. It is an error if there are more initialization than structure members.

Answers

1. F	2. T	3. T	4. T	5. T
6. T	7. T	8. T	9. F	10. T

Freebooks.pk

CHAPTER 9

FUNCTIONS

Chapter Overview

9.1 Functions
- 9.1.1 Importance of Functions
- 9.1.2 Advantages of Functions

9.2 Types of Functions in C++

9.3 User Defined Functions
- 9.3.1 Function Declaration or Function Prototype
- 9.3.2 Function Definition
- 9.3.3 Function Call
- 9.3.4 Scope of Functions

9.4 Passing Parameters to Functions
- 9.4.1 Pass by Value
- 9.4.2 Pass by Reference
- 9.4.3 Difference between Call by Value and Call by Reference
- 9.4.4 Returning Value from Function

9.5 Local Variable
- 9.5.1 Scope of Local Variable
- 9.5.2 Lifetime of Local Variable

9.6 Global Variable
- 9.6.1 Scope of Global Variable
- 9.6.2 Lifetime of Global Variable
- 9.6.3 Difference between Local and Global Variable

9.7 Static Variable
- 9.7.1 Scope of Static Variable
- 9.7.2 Lifetime of Static Variable

9.8 Register Variables

9.9 Functions and Arrays
- 9.9.1 Calling a Function with Array Parameter
- 9.9.2 Passing Individual Array Element to Function
- 9.9.3 Passing Two-Dimensional Array to Function

9.10 Functions and Structures
- 9.10.1 Passing Structure by Value
- 9.10.2 Passing Structure by Reference
- 9.10.3 Returning Structure from Function

9.11 Default Parameters

9.12 Inline Functions

9.13 Command Line Parameters

9.14 Function Overloading

9.15 Recursion

Programming Exercise

Exercise Questions

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

9.1 Functions

A **function** is a named block of code that performs some action. The statements written in a function are executed when it is **called** by its name. Each function has a unique name. Functions are the building blocks of C++ programs. They encapsulate pieces of code to perform specified operations. The functions are used to accomplish the similar kinds of tasks again and again without writing the same code again. They are used to perform the tasks that are repeated many times.

The control moves in the function when a function is called. All statements of the function are executed and then the control again moves back to the point where the function was called along with possible returned value.

The functions provide a **structured programming** approach. It is modular way of writing programs. The whole program logic is divided into a number of smaller **modules** or functions. The main function **calls** these functions when they are needed to execute.

9.1.1 Importance of Functions

A program may need to repeat the same piece of code at various places. It may be required to perform certain tasks repeatedly. The program may become very large if functions are not used. The piece of code that is executed repeatedly is stored in a separate function. The real reason of using functions is to divide a program into different parts. These parts of a program can be managed easily.

9.1.2 Advantages of Functions

Some advantages of using functions in programs are as follows:

1. Easier to Code

A lengthy program can be divided into small functions. It is easier to write small functions instead of writing one long program. A function is written to solve a particular problem. A programmer can focus attention on a specific problem. It makes programming easier.

2. Easier to Modify

Each function has a unique name and is written as an independent block. If there is any error in the program, the change is made to particular function in which error exists. A small function is easier to modify than a large program.

3. Easier to Maintain

Functions are easier to maintain than long programs. Each function contains independent code. A change in the function does not affect other parts of program.

4. Reusability

The code written in functions can be reused as and when required. A function but can be executed many times. A function is written to solve a particular problem. Whenever that problem has to be solved, the function can be executed.

5. Less Programming Time

A program may consist of many functions. These functions are written as independent programs. Different programmers can work on different functions at the same time. It takes far less time to complete the program.

Freebooks.pk

9.2 Types of Functions in C++

C++ language provides the following types of functions:

1. User-defined Functions

A type of function written by programmer is known as **user-defined function**. User-defined function has a unique name. A program may contain many user-defined functions.

2. Built-in Functions

A type of function that is available as a part of language is known as **built-in function** or **library function**. These functions are ready-made programs. These functions are stored in different header files. Built-in functions make programming faster and easier. C++ language provides many built-in functions to solve different problems. For example, **clrscr** is a built-in function to clear the screen. It is part of a header file called **conio.h**.

9.3 User Defined Functions

A user-defined function consists of the following:

- Function declaration
- Function definition

9.3.1 Function Declaration or Function Prototype

Function declaration is a model of a function. It is also called **function prototype**. It provides information to compiler about the structure of the function to be used in program. It ends with a semicolon. Function prototypes are usually placed at the beginning of the source file just before the main() function. Function declaration consists of the following parts:

- Function name
- Function return type
- Number and types of parameters

Syntax

The syntax of function declaration is as follows:

```
Return-type Function-name (parameters);
```

Return-type: It indicates the type of value that will be returned by function. For example, **int** is used as return type if the function returns integer value. If the function returns no value, the keyword **void** is used.

Function-name It indicates the name of function. The rules for specifying a function name is similar to the rules for declaring a variable. Each function should have a unique name.

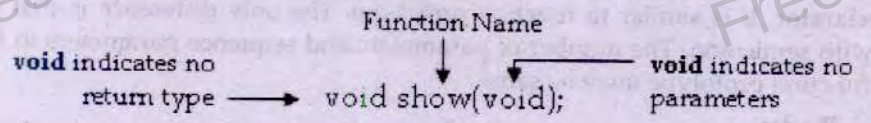
Parameters Parameters are the values that are provided to a function when the function is called. Parameters are given in parentheses. If there are many parameters, these are separated by commas. If there is no parameter, empty parentheses are used or keyword **void** is written in the parentheses.

The parameters are given in two ways:

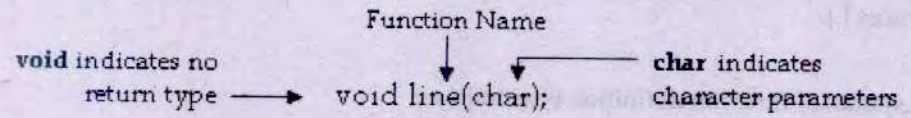
- Only data types of the parameters are written in prototype as follows:
  int add(int, int);
- Both data types and names of parameters are written in prototype as follows:
  int add(int a, int b);

Examples

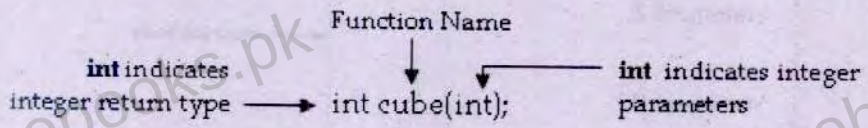
1. The following example shows that the function **show** accepts no parameter and returns no value.



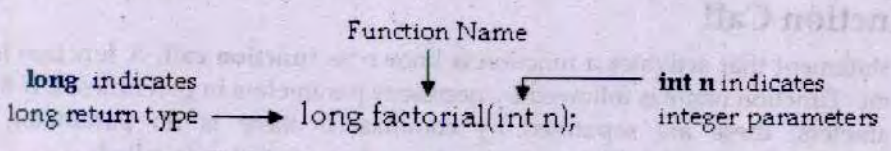
2. The following example shows that the function **line** accepts one character parameter and returns no value.



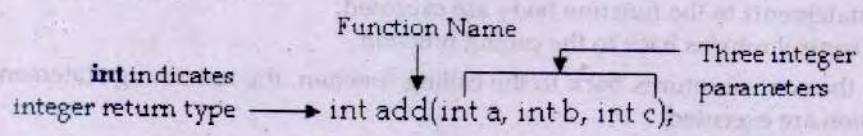
3. The following example shows that the function **cube** accepts one integer parameter and returns integer value.



4. The following example shows that the function **factorial** accepts one integer parameter and returns long value.



5. The following example shows that the function **add** accepts three integer parameters and returns integer value.



9.3.2 Function Definition

A set of statements that explains what a function does is called **function definition**. The function definition can be written at the following places:

- Before main() function
- After main() function
- In a separate file

Function declaration is not required if the function definition is written before main() function. Function declaration is compulsory if function definition is written after main() function. If function definition is written in a separate file then it can be used by including that file in the program using #include preprocessor directive.

Freebooks.pk

The function definition consists of two parts:

1. Function Header

The first line of function definition is known as function header. It is also called **function declarator**. It is similar to function prototype. The only difference is that it is not terminated with semicolon. The number of parameters and sequence parameters in function header and function prototype must be same.

2. Function Body

The set of statements which are executed inside the function is known as **function body**. The body of function appears after function declarator and the statements are written in curly braces { }.

Syntax

The syntax of function definition is as follows:

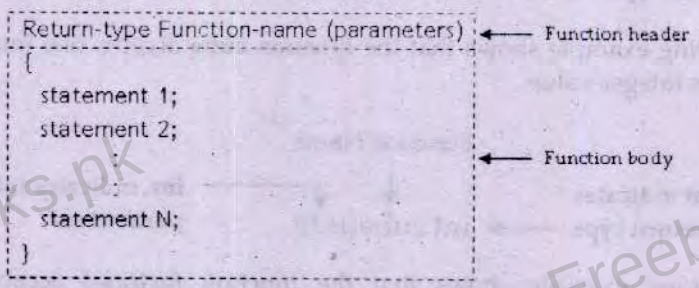


Figure 9.1: Function definition

9.3.3 Function Call

The statement that activates a function is known as **function call**. A function is called with its name. Function name is followed by necessary parameters in parentheses. If there are many parameters, these are separated by commas. If there is no parameter, empty parentheses are used. The following steps take place when a function is called:

1. The control moves to the function that is called.
2. All statements in the function body are executed.
3. The control returns back to the calling function.

When the control returns back to the calling function, the remaining statements in the calling function are executed.

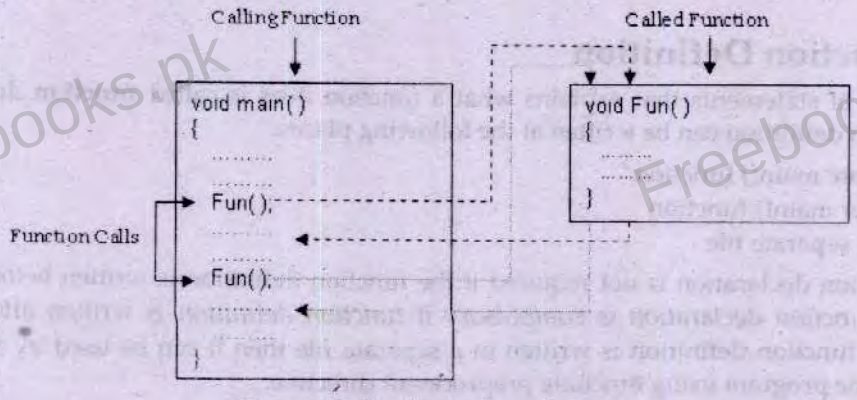


Figure 9.2: Function call mechanism

Freebooks.pk

Program 9.1

Write a program that displays a message "Programming makes life interesting" on screen using function.

```
#include <iostream.h>
#include <conio.h>
void show(void);
void main()
{
  clrscr();
  show();
  getch();
}
void show()
{
  cout<<"Programming makes life interesting.";
}
```

Output:
Programming makes life interesting.

How above Program Works?

The above program declares a function **show()** that displays the message. The main() function contains only three statement. The first statement clears the screen. The second statement calls the user-defined function **show()**. The control moves to the function and executes the single statement in it. The control moves back to **main()** function and executes the laste statement.

9.3.4 Scope of Functions

The area in which a function can be accessed is known as the scope of a function. The scope of any function is determined by the place of function declaration. Different types of functions on the basis of scope are as follows:

1. Local Functions

A type of function that is declared within another function is called **local function**. A local function can be called within the function in which it is declared.

Example

```
void main()
{
  void show(void);
  clrscr();
  show();
  getch();
}
```

2. Global Functions

A type of function that is declared outside any function is called **global function**. A global function can be called from any part of the program.

9.4 Passing Parameters to Functions

Parameters are the values that are provided to a function when the function is called. Parameters are given in the parentheses. If there are many parameters, these are separated by

Freebooks.pk

commas. If there is no parameter, empty parentheses are used. Both variables and constants can be passed to a function as parameters.

The sequence and types of parameters in **function call** must be similar to the sequence and types of parameters in function declaration.

- Parameters in function call are called **actual parameters**.
- Parameters in function declaration are called **formal parameters**.

When a function call is executed, the values of actual parameters are copied to formal parameters.

9.4.1 Pass by Value

A parameter passing mechanism in which the value of **actual parameter** is copied to **formal parameters** of called function is known as **pass by value**. If the function makes any change in formal parameter, it does not affect the values of actual parameter. It is the default mechanism for passing parameters to functions.

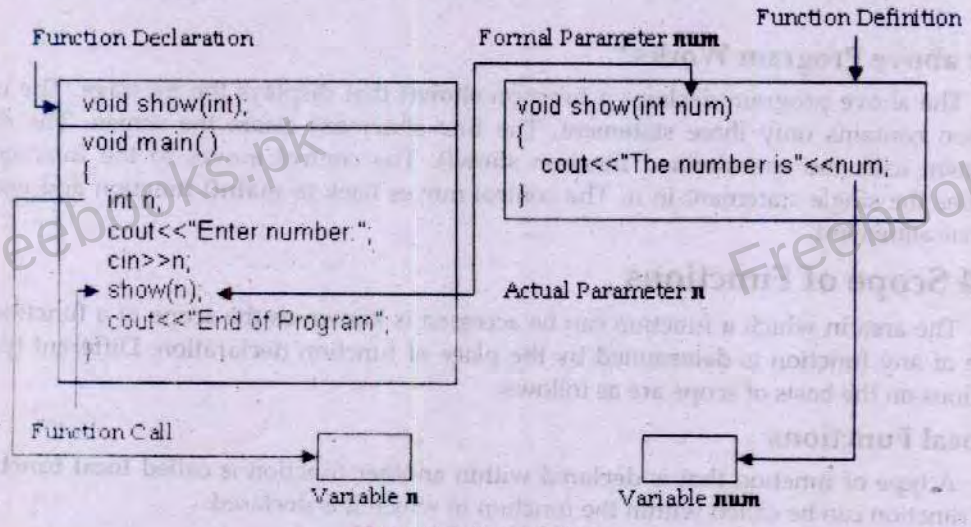


Figure 9.3: Pass by Value

Program 9.2

Write a program that inputs two numbers in main() function, passes these numbers to a function. The function displays the maximum number.

```
#include <iostream.h>
#include <conio.h>
void max(int a, int b);
void main()
{
  int x, y;
  clrscr();
  cout<<"Enter two numbers: ";
  cin>>x>>y;
  max(x, y);
  getch();
```

Output:
Enter two numbers: 20 30
Maximum number is 30

Freebooks.pk

```
void max(int a, int b)
{
  if(a > b)
    cout<<"Maximum number is "<<a;
  else
    cout<<"Maximum number is "<<b;
}
```

Program 9.3

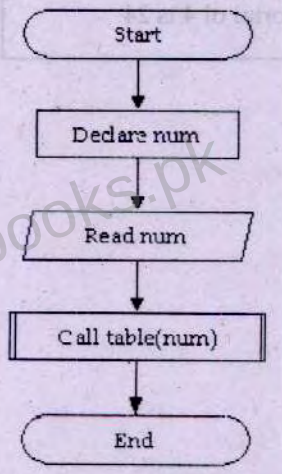
Write a program that inputs a number in main function and passes the number to a function. The function displays table of that number.

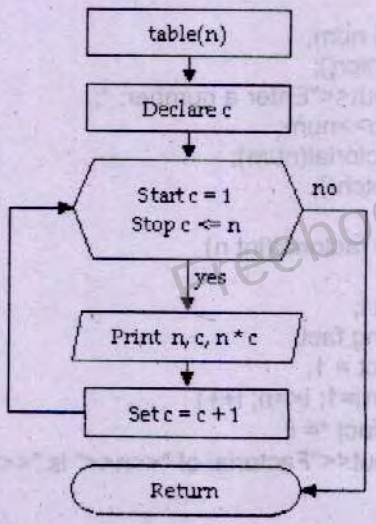
```
#include <iostream.h>
#include <conio.h>
void table(int n);
void main()
{
  int num;
  clrscr();
  cout<<"Enter a number: ";
  cin>>num;
  table(num);
  getch();
}
void table(int n)
{
  int c;
  for(c=1; c<=10; c++)
  {
    cout<<n<<" * "<<c<<" = "<<n*c<<endl;
  }
}
```

Output:

```
Enter a number: 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

Flowchart





Freebooks.pk

Program 9.4

Write a program that inputs a number and displays its precessor and successor numbers using function.

```
#include <iostream.h>
#include <conio.h>
void value(int);
void main()
{
  clrscr();
  int x;
  cout << "Enter an integer : ";
  cin>>x;
  value(x);
  getch();
}
void value(int x)
{
  int p, n;
  p = x - 1;
  n = x + 1;
  cout <<"The number before "<<x<< " is " <<p<<endl;
  cout <<"The number after "<<x<< " is " <<n<<endl;
}
```

Output:
Enter a number: 5
The number before 5 is 4
The number after 5 is 6

Program 9.5

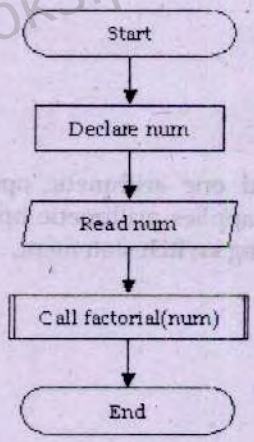
Write a program that inputs a number in main function and passes the number to a function. The function displays the factorial of that number.

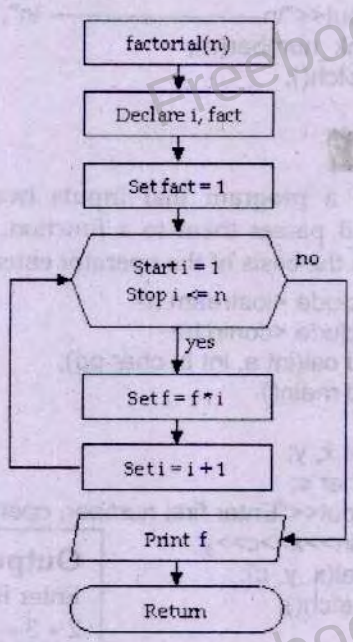
```
#include <iostream.h>
#include <conio.h>
void factorial(int n);
void main()
{
  int num;
  clrscr();
  cout<<"Enter a number: ";
  cin>>num;
  factorial(num);
  getch();
}
void factorial(int n)
{
  int i;
  long fact;
  fact = 1;
  for(i=1; i<=n; i++)
    fact *= i;
  cout<<"Factorial of "<<n<<" is "<<fact;
}
```

Output:
Enter a number: 4
Factorial of 4 is 24

Freebooks.pk

Flowchart





Program 9.6

Write a program to check whether a number is prime number, even number or odd number using function.

```
#include<iostream.h>
#include<conio.h>
void chk_number(int n)
{
  int c=0,i;
  for(i=2;i<n;i++)
  {
    if(n%i==0)
      c=1;
  }
  if(n%2==0 && c==0)
    cout<<n<<" is a prime even number: ";
  else if(n%2!=0 && c==0)
    cout<<n<<" is a odd prime number.";
  else if(n%2==0 && c!=0)
    cout<<n<<" is only an even number, not prime.";
  else if(n%2!=0)
    cout<<n<<" is only an odd number, not prime.";
  else
    cout<<" is not a prime number.";
}
void main()
{
  int n;
  cout<<"\nEnter a number: ";
  cin>>n;
```

Output:
Enter a number: 9
Nature of number
---
9 is only an odd number, not prime.

Freebooks.pk

```
    cout<<"\nNature of number \n";
    cout<<"\n------------------------ \n";
    chk_number(n);
    getch();
  }
```

Program 9.7

Write a program that inputs two numbers and one arithmetic operator in main function and passes them to a function. The function applies arithmetic operation on two numbers on the basis of the operator entered by user using **switch** statement.

```
#include <iostream.h>
#include <conio.h>
void cal(int a, int b, char op);
void main()
{
  int x, y;
  char c;
  cout<<"Enter first number, operator and second number:";
  cin>>x>>c>>y;
  cal(x, y, c);
  getch();
}
void cal(int a, int b, char op)
{
  switch(op)
  {
    case '+':
      cout<<a<<" + "<<b<<" = "<<a+b;
      break;
    case '-':
      cout<<a<<" - "<<b<<" = "<<a-b;
      break;
    case '*':
      cout<<a<<" * "<<b<<" = "<<a*b;
      break;
    case '/':
      cout<<a<<" / "<<b<<" = "<<a/b;
      break;
    case '%':
      cout<<a<<" % "<<b<<" = "<<a%b;
      break;
    default:
      cout<<"Invalid operator!";
  }
}
```

Output:
Enter first number, operator and second number: 2 + 3
2 + 3 = 5

Program 9.8

Write a program that displays a square of characters using function. The program inputs a number and a character in main function and passes them to function. For example, if the user enters 3 and @, the function displays the following 3 rows of the symbol @:

```
#include <iostream.h>
#include <conio.h>
void shape ( int , char );
void main()
{
 int num;
 char ch ;
 cout << "\nEnter a number:";
 cin >> num;
 cout <<."Enter a charcter:";
 cin >> ch;
 shape ( num, ch );
 getch();
}
void shape ( int n, char c)
{
   int i , j ;
  for ( i = 1 ; i <= n ; i++ )
  {
    cout << endl;
    for ( j = 1 ; j <= n ; j++ )
    cout << c;
  }
}
```

```
@@@
@@@
@@@
```

9.4.2 Pass by Reference

A parameter passing mechanism in which the address of **actual parameter** is passed to the called function is known as **pass by reference**. The formal parameter is not created separately in the memory. Formal parameter becomes a second name of actual parameter. It means that single memory location is shared between actual parameter and formal parameter. If the called function makes any change in formal parameter, the change is also available in calling function.

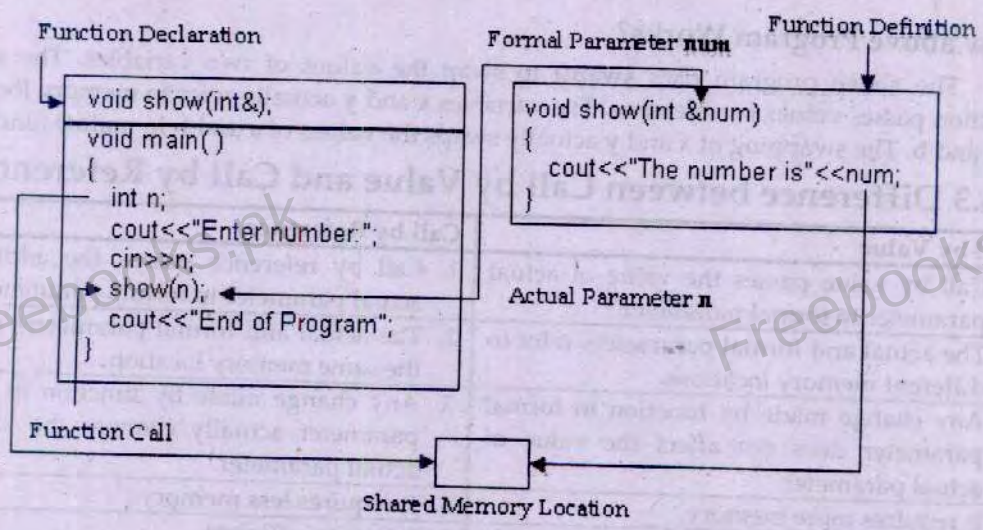


Figure 9.4: Pass by Reference

Program 9.9

Write a program that inputs two integers in main() function and passes the integers to a function by reference. The function swaps the values. The main() function should display the values before and after swapping.

```
#include <iostream.h>
#include <conio.h>
void swap(int &x, int &y);
void main()
{
  clrscr();
  int a, b;
  cout<<"Enter an integer: ";
  cin>>a;
  cout<<"Enter an integer: ";
  cin>>b;
  cout<<"Values before swapping:\n";
  cout<<"a = "<<a<<endl;
  cout<<"b = "<<b<<endl;
  cout<<"Swapping the values..."<<endl;
  swap(a, b);
  cout<<"Values after swapping:\n";
  cout<<"a = "<<a<<endl;
  cout<<"b = "<<b<<endl;
  getch();
}
void swap(int &x, int &y)
{
  int t;
  t = x;
  x = y;
  y = t;
}
```

Output:

```
Enter an integer: 10
Enter an integer: 20
Values before swapping...
a = 10
b = 20
Swapping the values...
Values after swapping:
a = 20
b = 10
```

How above Program Works?

The above program uses **swap()** to swap the values of two variables. The **main()** function passes values by reference. The variables **x** and **y** actually refer to memory locations of **a** and **b**. The swapping of **x** and **y** actually swaps the values of **a** and **b** in **main()** function.

9.4.3 Difference between Call by Value and Call by Reference

Call by Value	Call by Reference
1. Call by value passes the value of actual parameter to formal parameter.	1. Call by reference passes the address of actual parameter to formal parameter.
2. The actual and formal parameters refer to different memory locations.	2. The actual and formal parameters refer to the same memory location.
3. Any change made by function in formal parameter does not affect the value of actual parameter.	3. Any change made by function in formal parameter actually changes the value of actual parameter.
4. It requires more memory.	4. It requires less memory.
5. It is less efficient.	5. It is more efficient.

Freebooks.pk

9.4.4 Returning Value from Function

A function can return a single value. The return type in function declaration indicates the type of value returned by a function. For example, **int** is used as return type if the function returns integer value. If the function returns no value, the keyword **void** is used as return type. The keyword **return** is used to return the value back to the calling function.

When the **return** statement is executed in a function, the controls moves back to the calling function along with the returned value.

Syntax

The syntax for returning a value is as follows:

```
return expression;
```

expression: It can be a variable, constant or an arithmetic expression whose value is returned to the calling function.

The calling function can use the returned value in the following ways:

1. Assignment statement
2. Arithmetic expression
3. Output statement

1. Assignment Statement

The calling function can store the returned value in a variable and then use this variable in the program.

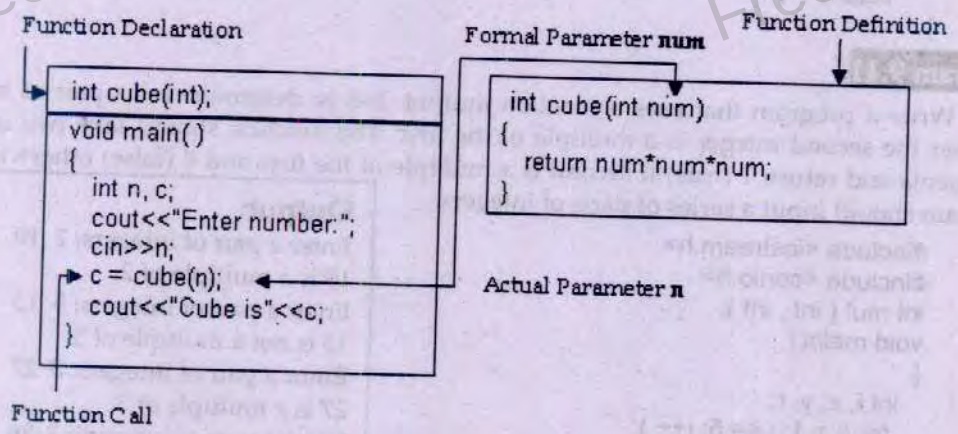


Figure 9.5: Returned Value used in Assignment Statement

The above example declares a function **cube**. The program inputs a number and passes it to the function. The function calculates its cube and returns the result. The result is stored in variable **c**. The last statement displays the value of **c**.

Program 9.10

Write a program that inputs marks in main function and passes these marks to a function. The function finds grade of student on the basis of the following criteria:

Grade A	80 or above marks
Grade B	60 to 79 marks
Grade C	40 to 59 marks
Grade F	below 40 marks

Freebooks.pk

The function returns grade back to main function where it is displayed on the screen.

```
#include <iostream.h>
#include <conio.h>
char grade(int m);
void main()
{
  int marks;
  char g;
  cout<<"Enter marks: ";
  cin>>marks;
  g = grade(marks);
  cout<<"Your grade is "<<g;
  getch();
}
char grade(int m)
{
  if(m > 80)
    return 'A';
  else if(m > 60)
    return 'B';
  else if(m > 40)
    return 'C';
  else
    return 'F';
}
```

Output:
Enter marks: 65
Your grade is B

Program 9.11

Write a program that uses a function mul(int, int) to determine for a pair of integers whether the second integer is a multiple of the first. The function should take two integers arguments and return 1 (true) if second is a mulitple of the first and 0 (false) otherwise. The program should input a series of paris of integers.

```
#include <iostream.h>
#include <conio.h>
int mul ( int , int );
void main()
{
  int i, x, y, r;
  for (i = 1; i <= 5; i++ )
  {
    cout << "Enter a pair of integers: ";
    cin >> x >> y ;
    r = mul ( x , y );
    if ( r == 1 )
      cout << y << " is multiple of " << x << endl;
    else
      cout << y << " is not multiple of " << x << endl;
  }
  getch();
}
int mul ( int a , int b )
{
  if ( b % a == 0 )
```

Output:
Enter a pair of integers: 2 10
10 is a multiple of 2.
Enter a pair of integers: 5 13
13 is not a multiple of 5.
Enter a pair of integers: 3 27
27 is a multiple of 3.
Enter a pair of integers: 4 36
36 is a multiple of 4.
Enter a pair of integers: 6 41
41 is not a multiple of 6.

Freebooks.pk

```
    return 1 ;
  else
    return 0;
}
```

Program 9.12

Write a program that inputs base and height of a triangle in main function and passes them to a function. The function finds the area of triangle and returns it to main function where it is displayed on the screen. Area = 1/2(Base * Height)

```
#include <iostream.h>
#include <conio.h>
float area(int b, int h);
void main()
{
  int base, height;
  float ar;
  clrscr();
  cout<<"Enter base: ";
  cin>>base;
  cout<<"Enter height: ";
  cin>>height;
  ar = area(base, height);
  cout<<"Area of triangle is "<<ar;
  getch();
}
float area(int b, int h)
{
  float a;
  a = 0.5*b*h;
  return a;
}
```

Output:
Enter base: 5
Enter height: 4
Area of triangle is 10

2. Arithmetic Expression

The calling function can use the returned value directly in an arithmetic expression.

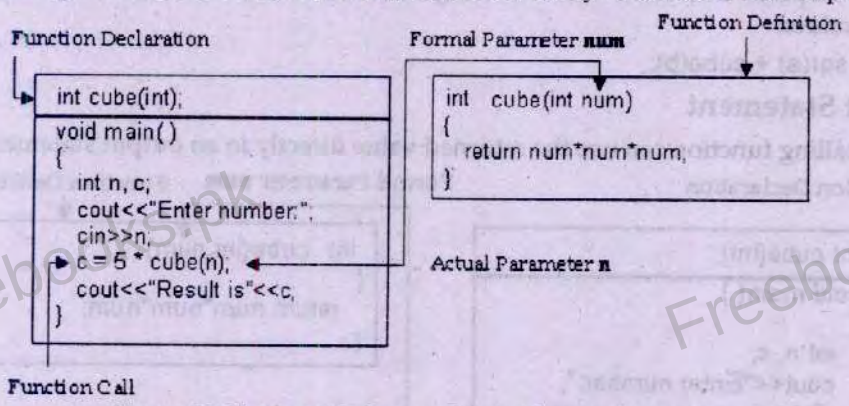


Figure 9.6: Returned Value used in Arithmetic Expression

The above example declares a function **cube**. The program inputs a number and passes it to the function. The function calculates its cube and returns the result. The result is multiplied with 5 and result is stored in variable **c**. The last statement displays the value of **c**.

Program 9.13

Write a progam that inputs two integers. It passes first integer to a function that calculates and returns its square. It passes second integer to another function that calculates and returns its cube. The **main()** functions adds both returned values and displays the result.

```
#include <iostream.h>
#include <conio.h>
int sqr(int n);
int cube(int n);
void main()
{
  int a, b, r;
  cout<<"Enter an integer: ";
  cin>>a;
  cout<<"Enter an integer: ";
  cin>>b;
  r = sqr(a) + cube(b);
  cout<<"Result = "<<r<<endl;
  getch();
}
int sqr(int n)
{
  return n*n;
}
int cube(int n)
{
  return n*n*n;
}
```

Output:
Enter an integer: 2
Enter an integer: 3
Result = 31

How above Program Works?

The above program defines two functions **sqr()** and **cube()**. The program inputs two integers and passes first integer to sqr() and second integer to cube(). Both functions calculate the required result and return values to main() function. The main() function adds both values and displays the result. The following statement calls both functions as well as adss the return values:

```
r = sqr(a) + cube(b);
```

3. Output Statement

The calling function can use the returned value directly in an output statement.

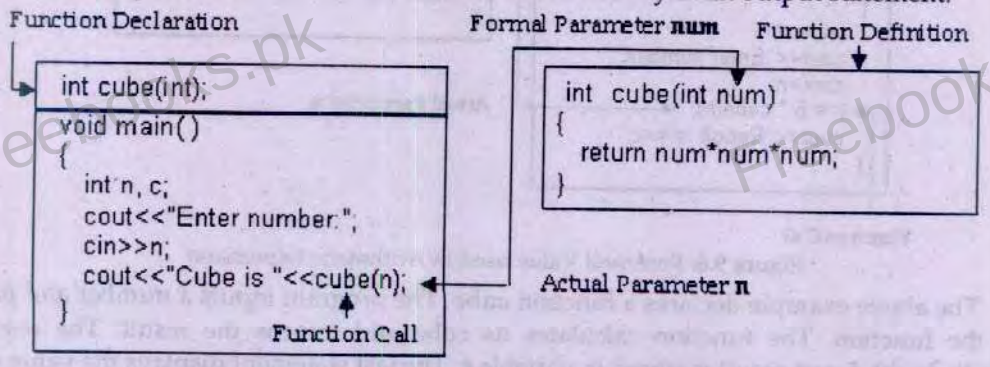


Figure 9.7: Returned Value used in Output Statement

The above example declares a function **cube**. The program inputs a number and passes it to the function. The function calculates its cube and returns the result. The result is directly displayed on the screen.

Program 9.14

Write a program that inputs two integers in main() function and passes the values to a function. The function finds and returns the greatest common divisor. The main() function then displays the returned value.

```
#include <iostream.h>
#include <conio.h>
int gcd(int x, int y);
void main()
{
  clrscr();
  int a, b;
  cout<<"Enter an integer: ";
  cin>>a;
  cout<<"Enter an integer: ";
  cin>>b;
  cout<<"Greatest common divisor is "<<gcd(a, b)<<endl;
  getch();
}
int gcd(int x, int y)
{
  int g, i, n;
  if(x < y)
    n = x;
  else
    n = y;
  for(i=1; i<=n; i++)
    if(x%i==0 && y%i==0)
  g = i;
  return g;
}
```

Output:
Enter an integer: 18
Enter an integer: 27
Greatest common divisor is 9

Program 9.15

Write a program that calculates the sum of the following series using function. The main() function inputs a number and passes it to a function. The function finds the sum from 1 to the given number and returns the result to main() function.

1+1/1!+1/2!+1/3!+1/4!.....

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
  double term;
  int n;
  unsigned long fact(int);
  clrscr();
  cout<<"\nEnter the maximum value of denominator:";
```

Output:
Enter the maximum value of denominator: 5
Sum of series is: 2.716667

Freebooks.pk

```
    cin>>n;
    double sum=1;
    for(int i=1;i<=n;i++)
    {
      term=1.0/fact(i);
      sum+=term;
    }
    cout<<"\n\nSum of Series is:"<<sum;
    getch();
  }

  unsigned long fact (int n)
  {
    unsigned long prod=1;
    int i;
    for(i=1;i<=n;i++)
      prod*=i;
    return prod;
  }
```

9.5 Local Variable

A variable declared inside a function is known as **local variable**. Local variables are also called **automatic variables**. The syntax of declaring a local variable is as follows:

```
auto data_type identifier;
```

auto	It is a keyword that indicates that the variable is automatic.
data_type	It indicates the data type of variable.
identifier	It indicates the name of variable.

The user of keyword **auto** is optional.

9.5.1 Scope of Local Variable

The area where a variable can be accessed is known as **scope of variable.** Local variable can be used only in the function in which it is declared. If a statement accesses a local variable that is not in scope, the compiler generates a syntax error.

9.5.2 Lifetime of Local Variable

The time period for which a variable exists in the memory is known as the **lifetime of variable**. Different types of variables have different lifetimes.

The lifetime of local variable starts when the control enters the function in which it is declared. Local variable is automatically destroyed when the control exits from the function and its lifetime ends. When the lifetime of a local variable ends, the value stored in this variable also becomes inaccessible.

Example

```
#include <iostream.h>
#include <conio.h>
void fun();
void main()
{
  clrscr();
```

```
    int i;
    for(i=1; i<=5; i++)
      fun();
    getch();
  }
  void fun()
  {
    int n = 0;
    n++;
    cout<<"The value of n = "<<n<<endl;
  }
```

Output:
The value of n = 1
The value of n = 1
The value of n = 1
The value of n = 1
The value of n = 1

How above Program Works?

The above program declares a function **fun()**. The function declares a local variable **n** and intializes it to 0. The main() function calls it five times using **for** loop. The control moves to the function in each function call. The local variable **n** is created in the memory and is intialized with 0. Finally, the value of **n** is displayed on screen. The variable **n** is destroyed from the memory when the control exits the function.

9.6 Global Variable

A variable declared outside any function is known as **global variable**. Global variables can be used by all functions in the program. The values of these variables are shared among different functions. If one function changes the value of a global variable, this change is also available to other functions.

9.6.1 Scope of Global Variable

Global variable can be used by all functions in the program. It means that these variables are globally accessed from any part of the program. Normally, global variables are declared before main function.

9.6.2 Lifetime of Global Variable

Global variables exist in the memory as long as the program is running. These variables are destroyed from the memory when the program terminates. These variables occupy memory longer than local variables. So, global variables should be used only when these are very necessary.

Program 9.16

Write a program that inputs a number in a global variable. The program calls a function that multiplies the value of global variable by 2. The main function then displays the value of global variable.

```
#include <iostream.h>
#include <conio.h>
int g;
void fun();
void main()
{
  clrscr();
  cout<<"Enter a number: ";
  cin>>g;
  cout<<"Value of g before function call: "<<g<<endl;
```

Output:
Enter a number: 5
Value of g before function call: 5
Value of g after function call: 10

Freebooks.pk

```
    fun();
    cout<<"Value of g after function call: "<<g<<endl;
    getch();
}

void fun()
{
    g = g * 2;
}
```

How above Program Works?

The above program declares a global variable **g**. The variable is accessible to all functions in the program. The function **fun()** doubles the value of g when **main()** function calls it. Finally, **main()** function displays the increased value of **g**.

9.6.3 Difference between Local and Global Variable

The main difference between local and global variable is as follows:

Local Variable	Global Variable
1. Local variables are declares within a function.	1. Global variables are declares outside any function.
2. Local variables can be used only in function in which they are declared.	2. Global variables can be used in all functions.
3. Local variables are created when the control enters the function in which they are declared.	3. Global variables are created when the program starts execution.
4. Local variables are destroyed when the control leaves the function.	4. Global variables are destroyed when the program terminates.
5. Local variables are used when the values are to be used within a function.	5. Global variables are used when values are to be shared among different functions.

9.7 Static Variable

A local variable declared with keyword **static** is called **static variable**. The keyword **static** is used to increase the lifetime of local variable.

9.7.1 Scope of Static Variable

The scope of static variable is similar to the scope of local variable. Static variable can be used only in the function in which it is declared.

9.7.2 Lifetime of Static Variable

The lifetime of static is similar to global variable. Static variables exist in the memory as long as the program is running. These variables are destroyed from the memory when the program terminates.

Static variables are created in the memory when the function is called for the first time. When the control exits from the function, these variables are not destroyed unlike local variables. Moreover, the initialization statement for static variable is also executed only when the function is executed for the first time.

Freebooks.pk

A function that contains static variables runs faster than the function with local variables. This is because local variables are created each time the function is called but static variables are created only once.

Program 9.17

Write a program that calls a function for five times using loop. The function uses a static variable initialized to 0. Each time the function is called, the value of static variable is incremented by 1 and is displayed on the screen.

```
#include <iostream.h>
#include <conio.h>
void fun();
void main()
{
  int i;
  clrscr();
  for(i=1; i<=5; i++)
    fun();
  getch();
}
void fun()
{
  static int n = 0;
  n++;
  cout<<"Value of n = "<<n<<endl;
}
```

Output:
Value of n = 1
Value of n = 2
Value of n = 3
Value of n = 4
Value of n = 5

How above Program Works?

The above program declares a function **fun()**. The function declares a static variable **n** and intializes it to 0. The **main()** function calls it five times using **for** loop. The control moves to the function in each function call. The local variable **n** is created and intialized to 0 only in first function call. The last statement displays the value of **n**. The variable **n** is not destroyed from memory when control exits the function. The value of **n** is different in each function call.

9.8 Register Variables

A variable declared with **register** keyword is known as **register variable**. The value of register variable is stored in registers instead of RAM. Registers are faster than RAM so the values stored in register can be accessed faster than the values stored in RAM.

Syntax

The syntax for declaring register variables is as follows:

```
register data-type variable-name;
```

register It is a keyword that indicates that it is register variable.
data_type It indicates the data type of variable.
identifier It indicates the name of variable.

Example

The following line declares a register variable:

```
register int n;
```

Freebooks.pk

9.9 Functions and Arrays

An array can be passed to a function as parameter. When an array is passed as parameter to a function, only the address of first element of the array is passed. An array is passed by reference not by value. A separate copy of the array is not created in the function.

Syntax

The following syntax is used to declare a function with array as parameter:

```
Return-type Function-name (parameter[]);
```

Return-type	It indicates the type of value that will be returned by function.
Function-name	It indicates the name of the function.
Parameter[]	Parameters are the values that are provided to a function when the function is called. The bracket with parameter indicates that the parameter will be an array.

Example

```
void display(int[]);
```

The above example declares a function **display** that will accept an array of integers as parameter.

9.9.1 Calling a Function with Array Parameter

A function with array parameter is called by giving the name of the array as actual parameter. The index of the array is not used in function call. The name of the array refers to the memory address of its first element. The memory address is passed to the function. The function then accesses the array by using the memory address.

Program 9.18

Write a program that inputs five integers in an array and passes the array to a function. The function displays the values of the array.

```
#include <iostream.h>
#include <conio.h>
void show(int arr[]);
void main()
{
  clrscr();
  int num[5], i;
  cout<<"Enter five integers: "<<endl;
  for(i=0; i<5; i++)
    cin>>num[i];
  show(num);
  getch();
}
void show(int arr[])
{
  int j;
  cout<<"The values in array:\n";
  for(j=0; j<5; j++)
    cout<<arr[j]<<"\t"
}
```

Output:

```
Enter five integers:
10
20
30
40
50
The values in array:
10  20  30  40  50
```

Freebooks.pk

How above Program Works?

The above program declares a function **show()** that accepts an array of integers as parameter. The **main()** function inputs five integers in an array and then passes the array to **show()**. The **show()** function displays the vlaues in array. The identifiers num and arr refer to the same memory locations.

Program 9.19

Write a program that inputs five integers in array and passes array to a function. The function counts even numbers in array and returns result to main function to display it.

```
#include <iostream.h>
#include <conio.h>
int even(int arr[]);
void main()
{
  int num[5], i, n;
  cout<<"Enter five integers: "<<endl;
  for(i=0; i<5; i++)
    cin>>num[i];
  n = even(num);
  cout<<"The array contains "<<n<<" even numbers."<<endl;
  getch();
}
int even(int arr[])
{
  int j, e;
  e = 0;
  for(j=0; j<5; j++)
    if(arr[j]%2==0)
   e++;
  return e;
}
```

Output:

```
Enter five integers:
52
33
81
98
33
The array contains 2 even numbers.
```

9.9.2 Passing Individual Array Element to Function

An individual element of array can also be passed to function. The data type of array and the data type of function parameter must be same. The individual element is passed to a function like a simple variable. Suppose a function is defined to accept an integer parameter. It can accept an individual element of an integer array in same way as it can accept a simple integer variable.

Program 9.20

Write a program that inputs five integers in an array. It passes all elements of the array to a function one by one. The function displays the actual value of the element and its square.

```
#include <iostream.h>
#include <conio.h>
void square(int n);
void main()
{
  clrscr();
  int num[5], i;
  cout<<"Enter five integers: "<<endl;
```

Freebooks.pk

```
    for(i=0; i<5; i++)
      cin>>num[i];
    cout<<"Calling the function..."<<endl;
    for(i=0; i<5; i++)
      square(num[i]);
    getch();
  }
  void square(int n)
  {
    cout<<n<<"\t"<<n*n<<endl;
  }
```

Output:

```
Enter five integers:
10
20
30
40
50
Calling the function...
10      100
20      400
30      900
40      1600
50      2500
```

How above Program Works?

The above program inputs five integers in array and passes each element to function. The value in each element is passed to **n**. The function displays the original value and its square on the screen.

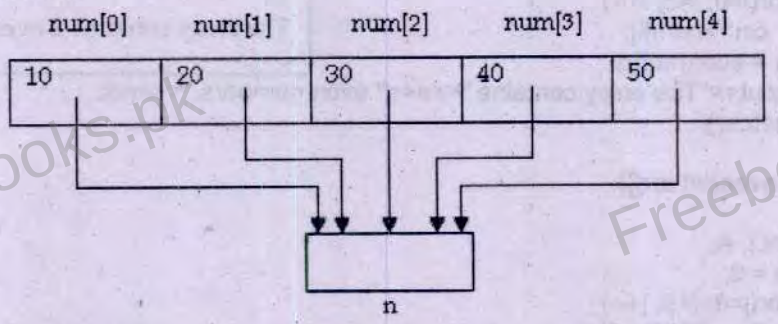


Figure 9.8: Passing individual array element to function

9.9.3 Passing Two-Dimensional Array to Function

A two dimensional array can be passed to a function in same way as one-dimensional array. The two-dimensional array is also passed to a function by reference. The specification of first dimension is not necessary. However, the second dimension is alway required.

```
void fun (int n[][]);        // Error. The second dimension is required.
void square(int n[][3])      // OK.
void square(int n[3][2])     // OK.
```

Program 9.21

Write a program that inputs values in a two-dimensional array with three rows and two columns. The program passes the array to a function. The function returns the maximum value in the array.

```
#include <iostream.h>
#include <conio.h>
int max(int arr[3][2]);
void main()
{
  clrscr();
  int num[3][2], i, j, mx;
  for(i=0; i<3; i++)
```

Output:

```
Enter value for num[0][0]: 52
Enter value for num[0][1]: 17
Enter value for num[1][0]: 83
Enter value for num[1][1]: 71
Enter value for num[2][0]: 94
Enter value for num[2][1]: 30
The maximum value is 94
```

Freebooks.pk

```
    for(j=0; j<2; j++)
    {
      cout<<"Enter value for num["<<i<<"]["<<j<<"]: ";
      cin>>num[i][j];
    }
  mx = max(num);
  cout<<"The maximum value is "<<mx<<endl;
  getch();
}
int max(int arr[3][2])
{
  int r, c, m;
  m = arr[0][0];
  for(r=0; r<3; r++)
    for(c=0; c<2; c++)
      if(arr[r][c] > m)
        m = arr[r][c];
  return m;
}
```

9.10 Functions and Structures

The structures can also be used with functions. A function can accept structures as parameters. A function can also return a structure variable. The process of passing structures to a function is similar to passing a simple variable. The structure used as parameter must be defined before using it as parameter.

9.10.1 Passing Structure by Value

A structure can be passed to a function by value. the method of passing a structure by value is same as passing a simple variable by value.

Program 9.22

Write a program that declares a structure to store marks and grade. It defines structure variable and inputs values. It passes the variable to a function that shows its contents.

```
#include <iostream.h>
#include <conio.h>
struct Test
{
  int marks;
  char grade;
};
void show(Test p);
void main()
{
  Test t;
  cout<<"Enter marks: ";
  cin>>t.marks;
  cout<<"Enter grade: ";
  cin>>t.grade;
  show(t);
  getch();
}
```

Output:
Enter marks: 92
Enter grade: A
Marks: 92
Grade: A

Freebooks.pk

```
void show(Test p)
{
  cout<<"Marks: "<<p.marks<<endl;
  cout<<"Grade: "<<p.grade<<endl;
}
```

Program 9.23

Write a program that declares a structure to store country name and population in millions. It defines two structure variable and inputs values. It passes both variables to a function that shows the record of the country with more population.

```
#include <iostream.h>
#include <conio.h>
struct Pop
{
  char name[50];
  float pop;
};
void fun(Pop x, Pop y);
void main()
{
  Pop a, b;
  cout<<"Enter name and population (in millions): ";
  cin>>a.name>>a.pop;
  cout<<"Enter name and population (in millions): ";
  cin>>b.name>>b.pop;
  fun(a, b);
  getch();
}
void fun(Pop x, Pop y)
{
  cout<<"The country with more population: "<<endl;
  if(x.pop > y.pop)
  {
    cout<<"Name: "<<x.name<<endl;
    cout<<"Population: "<<x.pop<<" millions."<<endl;
  }
  else
  {
    cout<<"Name: "<<y.name<<endl;
    cout<<"Population: "<<y.pop<<" millions."<<endl;
  }
}
```

Output:

```
Enter name and population (in millions): Pakistan 1.6
Enter name and population (in millions): China 112.5
The country with more population:
Name: China
Population: 112.5 millions.
```

Program 9.24

Write a program that declares a structure **time** to store hours, minutes and seconds. The program declares an array of this structure and inputs two times from the user. It adds both times and displays the total time in **hh mm ss** format.

```
#include<iostream.h>
#include<conio.h>
```

Freebooks.pk

```
struct time
{
  int hours;
  int minutes;
  int seconds;
} time1[2];
void addtime (time,time);

void main()
{
  int sec;
  clrscr();
  for( int i=0;i<=1;i++)
  {
    cout<<"\nEnter the time"<<i+1<<"\n";
    cout<<"Hours: ";
    cin>>time1[i].hours;
    cout<<"Mintues: ";
    cin>>time1[i].minutes;
    cout<<"Seconds: ";
    cin>>time1[i].seconds;
  }
  addtime(time1[0], time1[1]);
  getch();
}

void addtime (time t1, time t2)
{
  int sec, min, hr, x;
  sec= t1.seconds + t2.seconds;
  x= sec / 60;
  sec %= 60;
  min= t1.minutes + t2.minutes + x;
  hr = t1.hours + t2.hours + min / 60;
  min %= 60;
  cout<<"\nTotal time in hh mm ss format is: \n";
  cout<<"\nHours: "<<hr;
  cout<<"\nMinutes: "<<min;
  cout<<"\nSeconds: "<<sec;
}
```

Output:

```
Enter the time1
Hours: 5
Minutes: 40
Seconds: 50
Enter the time2
Hours: 5
Minutes: 40
Seconds: 50

Total time in hh mm ss format is:

Hours: 11
Minutes: 21
Seconds: 40
```

Program 9.25

Write a program that inputs the following record of cricketers:

- Player's Name
- Runs
- Innings
- Times Not out

The program should declare a structure to store the above data values. The program should declare an array of structure of five elements to input the records of five cricketers and then display them in tabular form.

Freebooks.pk

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void line();
void star();
struct cricketer
{
  char name[15];
  int runs;
  int innings;
  int tno;
  float avg;
};
void main()
{
  int runs,innings, i;
  float avg;
  cricketer rec[5];
  cout<<"\nEnter 5 records including following details\n";
  cout<<"1)      Player's Name\n";
  cout<<"2)      Runs\n";
  cout<<"3)      Innings\n";
  cout<<"4)      Times Not out\n\n";
  for(i=0;i<5;i++)
  {
    cout<<"\nEnter Player Name:-";
    cin>>rec[i].name;
    cout<<"Enter Runs:-";
    cin>>rec[i].runs;
    cout<<"Enter Innings:-";
    cin>>rec[i].innings;
    cout<<"Enter Time not out:-";
    cin>>rec[i].tno;
    rec[i].avg = float(rec[i].runs)/rec[i].innings;
  }
  clrscr();
  cout<<"\n\n\n";
  cout<<setw(49)<<"CRICKETER'S  TABLE\n";
  line();
  cout<<setw(15)<<"Player's Name"<<setw(15)<<"Runs"<<setw(15)<<"Innings";
  cout<<setw(20)<<"Times not out"<<setw(12)<<"Average\n";
  line();
  for(i=0;i<5;i++)
  {
    cout<<setw(15)<<rec[i].name<<setw(15)<<rec[i].runs<<setw(12)<<rec[i].innings;
    cout<<setw(18)<<rec[i].tno<<setw(16)<<rec[i].avg<<endl;
  }
  line();
  cout<<endl<<endl<<endl;
  star();
  cout<<setw(43)<<"Finish\n";
  star();
```

```
    getch();
  }

  void line()
  {
    for(int i=1;i<41;i++)
      cout<<"--";
    cout<<"\n";
  }
  void star()
  {
    for(int i=1;i<41;i++)
      cout<<"**";
    cout<<"\n";
  }
```

Output

CRICKETER'S TABLE

Player's Name	Runs	Innings	Times not out	Average
Inzamam	5223	51	8	102.411766
Razzaq	2000	35	12	57.142857
Yousuf	3200	31	10	103.225807
Younas	5123	55	21	93.145454
Shoaib	4227	81	42	52.111111

****

Finish

****

9.10.2 Passing Structure by Reference

A structure variable can also be passed to a function by reference. The method of passing a structure by reference is same as passing a simple variable by reference.

Program 9.26

Write a program that declares a structure to store price and author name of a book. It defines a structure variable and inputs values. It passes the variable to a function by reference that doubles the value of price. The main() function finally displays the values.

```
#include <iostream.h>
#include <conio.h>
struct Book
{
  char author[50];
  float price;
};
void dbl(Book &x);
void main()
{
  clrscr();
```

Output:
Enter author name and price: Tasleem Mustafa
250
Author Name: Tasleem Mustafa
Price: 500

Freebooks.pk

```
    Book b;
    cout<<"Enter author name and price: ";
    cin.get(b.author, 50);
    cin>>b.price;
    dbl(b);
    cout<<"Author Name: "<<b.author<<endl;
    cout<<"Price: "<<b.price<<endl;
    getch();
}
void dbl(Book &x)
{
  x.price = x.price * 2;
}
```

Program 9.27

Write a program that uses a structure to simulate the time in hours, minutes and seconds. Write a function to set the time, another function to increment the time by a number of seconds and a function to display the time.

```
#include <iostream.h>
#include <conio.h>
struct Time
{
  int Hours;
  int Minutes;
  int Seconds;
};
void SetTime(int H, int M, int S, Time & time);
void IncrementTime(int S, Time & time);
void DisplayTime(Time time);

void SetTime(int H, int M, int S, Time & time)
{
  time.Hours=H;
  time.Minutes=M;
  time.Seconds=S;
}
void IncrementTime(int S, Time & time)
{
  time.Seconds+=S;
  if (time.Seconds/60 > 0)
  {
    time.Minutes+=time.Seconds/60;
    time.Seconds %= 60;
    if (time.Minutes/60 > 0)
    {
        time.Hours+=time.Minutes/60;
        time.Minutes %= 60;
        time.Hours%=24;
    }
  }
}
```

```
void DisplayTime(Time time)
{
  cout<<time.Hours<<":"<<time.Minutes<<":"<<time.Seconds<<endl;
}
void main()
{
  Time Now;
  SetTime(23,58,30, Now);
  cout<<"Initial time: ";
  DisplayTime(Now);
  IncrementTime(30, Now);
  cout<<"Time after increment of 30 seconds: ";
  DisplayTime(Now);
  IncrementTime(65, Now);
  cout<<"Time after increment of 65 seconds: ";
  DisplayTime(Now);
  getch();
}
```

Output:
Initial time: 23:58:30
Time after increment of 30 seconds: 23:59:0
Time after increment of 65 seconds: 0:0:5

9.10.3 Returning Structure from Function

A function can also return a structure variable. The method of returning a structure variable from a function is same as returning a simple variable. The return type of the function must be the type of structure to be returned.

Program 9.28

Write a program that declares a structure to store author name, pages and price of a book. It declares two structure variables and inputs values. It passes these variables to a function. The function returns the variable with more price. The program finally displays the values of the returned structure variable.

```
#include <iostream.h>
#include <conio.h>
struct Book
{
  int pages;
  float price;
};
Book check(Book x, Book y);
void main()
{
  Book a, b, r;
  cout<<"Enter pages and price: ";
  cin>>a.pages>>a.price;
  cout<<"Enter pages and price: ";
  cin>>b.pages>>b.price;
  r = check(a, b);
  cout<<"The more costly book is:\n";
  cout<<"Pages: "<<r.pages<<endl;
  cout<<"Price: "<<r.price<<endl;
  getch();
}
```

Output:
Enter pages and price: 400 250
Enter pages and price: 680 325
The more costly book is:
Pages: 680
Price: 325

Freebooks.pk

```
Book check(Book x, Book y)
{
  if(x.price > y.price)
    return x;
  else return y;
}
```

9.11 Default Parameters

The parameters initialized during function declaration are called **default parameters**. The default parameters allow user to call a function without giving required parameters. The initialized values of default parameters are used if the user does not specify any value for the parameters in function call. Default values can be constants, global variables or function calls.

The user can also specify the values in function call even if the default parameters are initialized. The values of default parameters are not used if the user specifies the values in function call. The default values are used only if values in function call are not specified. The user can specify some parameters with default values and some parameters without them. In this case, the parameters with default values must be rightmost parameters in parameters list.

Example

```
#include <iostream.h>
#include <conio.h>
void Test(int n=100);
void main()
{
  Test();
  Test(2);
  Test(75);
  getch();
}
void Test(int n)
{
  cout<<"n = "<<n<<endl;
}
```

Output:

```
100
2
75
```

How above Program Works?

The above program declares a function **Test** with one parameter **n**. The default value of **n** is specified as 100 in function declaration. The program calls the function three times. The first function call contains no parameter value. Therefore, the default value 100 is used. The second and third function calls also provide values for the parameter. That is why the values given in function call are displayed and the default value is ignored.

Example

```
#include <conio.h>
#include <iostream.h>
void add (int var1=5, int var2=10)
{
  int var3;
  var3 = var1 + var2;
  cout<<"The sum of the two numbers is "<<var3<<endl;
}
```

Freebooks.pk

```
void main( )
{
  add();
  add(1,2);
  add(1);
  getch();
}
```

Output:
The sum of the two numbers is 15
The sum of the two numbers is 3
The sum of the two numbers is 11

Program 9.29

Write a function that receives an integer array, its size and character + or –. By default, the character should be '+'. The function returns the sum of positive numbers stored in the array for the character '+'. It returns the sum of negative numbers for the character –.

```
#include <iostream.h>
#include <conio.h>
int funct(int a[], int n, char s = '+')
{
  int sum = 0;
  for(int i = 0; i< n ; i++ )
  {
    if(s == '+')
    {
      if(a[i] > 0 )
        sum += a[i];
    }
    else
    {
      if(a[i]<0)
        sum+=a[i];
    }
  }
  return sum;
}

void main()
{
  int arr[20], dn;
  cout<<"Enter dimension :";
  cin>> dn;
  for(int i = 0; i< dn ; i++)
  {
    cout<<"Enter any interger (positive / negetive): ";
    cin>> arr[i];
  }
  int s = funct( arr, dn);
  int s2 = funct(arr, dn, '-');
  cout<< "\n The sum of the positive integers : " << s;
  cout<< "\n The sum of all negative integers : " << s2;
  getch();
}
```

Output:
Enter dimension : 5
Enter any integer (positive/negative): 10
Enter any integer (positive/negative): 20
Enter any integer (positive/negative): -5
Enter any integer (positive/negative): -7
Enter any integer (positive/negative): 35
The sum of the positive integers : 65
The sum of the negative integers : -12

Freebooks.pk

9.12 Inline Functions

The function declared with keyword **inline** is known as **inline function**. The inline function is declared before main function. Only very small functions are declared as inline functions. When a program is compiled, the compiler generates special code at each function call to move the control to the called function. It also generates special code at the end of function definition to move the control back to the calling function. The shifting of control from calling function to called function and back to calling function takes time during program execution. This time can be eliminated by using inline functions.

The code of function body is inserted at each function call if the function is declared as inline function. It allows the program to execute the function statements without shifting the control from calling function. It saves time and increases program's efficiency.

The code of function body is repeated at each function call. It takes more memory than normal function call. That is why the inline functions must only be used if the function body contains a small number of statements.

Example

```
#include <iostream.h>
#include <conio.h>
inline int cube(int n)
{
  return n*n*n;
}
void main()
{
  clrscr();
  cout<<cube(3)<<endl;
  cout<<cube(5)<<endl;
  cout<<cube(9)<<endl;
  getch();
}
```

Output:
27
125
729

9.13 Command Line Parameters

An executable file with **.exe** extension is created by the compiler when the program is compiled. The executable file can be executed directly in **DOS** by writing the name of the file on command prompt. The parameters passed to a function from DOS prompt are known as **command-line parameters**. The command-line parameters can be passed to main function by writing the name of the executable file followed by the parameters. The parameters can be processed in main functions in the same way as other parameters.

Syntax

The syntax of using command line parameters is as follows:

```
void main (int argc, const char* argv[]);
```

argc	It indicates the number of parameters passed to the program.
argv[]	It indicates an array of string constants that represent the parameters.

Example

```
#include <iostream.h>
#include <conio.h>
```

Freebooks.pk

```
void main(int n, char *str[])
{
  clrscr();
  int i;
  cout<<"You provided the following parameters:"<<endl;
  for(i=1; i<n; i++)
    cout<<str[i]<<endl;
  getch();
}
```

Output:
D:\TC\BIN>fun17 3 Hello World
You provided the following parameters:
3
Hello
World

How above Program Works?

The above program is designed to accept command-line parameters. The output shows that the executable file of the program is stored in D:\TC\BIN directory. The name of program is **fun17**. The value 3 indicates the number of command-line parameters. The program simply displays the command-line parameters provided by the user.

Note: The first index of *str[ ] always refers to the name of the program itself. The following statement will display "D:\TC\BIN\Fun17" on the screen:

```
cout<<str[0];
```

9.14 Function Overloading

The process of declaring multiple functions with same name but different parameters is called **function overloading**. The functions with same name must differ in one of the following ways:

- Number of parameters
- Type of parameter
- Sequence of parameters

Function overloading allows the programmer to assign same name to all functions with similar tasks. It helps the user in using different functions easily. The user does not need to remember many names for similar types of tasks. For example, the programmer can declare the following functions to calculate average:

```
float Average(int, int);
float Average(float, float);
float Average(int, float);
```

The above three functions are used to calculate average but these functions accept different types of parameters. The user can use the same name **Average** and pass different types of parameters. The user may also think that one function is working in different ways. When the user calls a function, the compiler checks the parameters and their types. The corresponding function is executed according to the types, number and sequence of the parameters in function call.

Program 9.30

Write three versions of function line. The first version takes no parameter and displays a line of 10 asterisks. The second version takes an integer parameter and displays a line of n asterisks. The third version takes an integer and a character as parameters and displays a line of given character of n length.

Freebooks.pk

```
#include <iostream.h>
#include <conio.h>
void line();
void line(int n);
void line(int n, char c);
void main()
{
  clrscr();
  line();
  line(3);
  line(5, '@');
  getch();
}
void line()
{
  int i;
  for(i=1; i<=10; i++)
    cout<<"*";
  cout<<endl;
}
void line(int n)
{
  int i;
  for(i=1; i<=n; i++)
    cout<<"*";
  cout<<endl;
}
void line(int n, char c)
{
  int i;
  for(i=1; i<=n; i++)
    cout<<c;
  cout<<endl;
}
```

Output:

```
**********
***
@@@@@
```

How above Program Works?

The above program declares three functions with same name **line**. The program calls all three functions. The first function call contains no parameters. It displays a line of ten asterisks. The second function call contains one integer parameter. It displays three asterisks. The last function call contains both parameters and displays five '@'.

Program 9.31

Write two versions of an overloaded function sum(). This first version takes one parameter of integer array and returns the sum of all elements of the array. The second version takes two parameters of integer array and character 'E' or 'O'. If the character is 'E', it returns the sum of even elements of the array and if the character is 'O', it returns the sum of odd elements. In case of any other character it returns 0 (zero).

```
#include<iostream.h>
#include<conio.h>
int sum( int a[]);
int sum(int a[], char ch);
```

```
void main()
{
  int num[10];
  cout<< "\n Enter 10 integers : ";
  for(int i = 0;  i < 10 ;  i++ )
     cin >> num[i];
  int s = sum(num);
  cout<< "\n The sum of all integers = " << s;
  s = sum(num, 'E');
  cout<< "\n The sum of all even integers = " << s;
  s = sum(num, 'O');
  cout<< "\n The sum of all odd integers = " << s;
  s = sum(num , 'X');
  cout<< "\n No sum for wrong choice . The result  = " << s;
  getch();
}
int sum(int arr[])
{
  int s = 0;
  for ( int i = 0; i <10 ; i++ )
     s = s+ arr[i];
  return s;
}
int sum(int arr[], char a)
{
  int se = 0, so = 0, s = 0;
  switch(a)
  {
    case 'E':
    case 'e':
    for ( int i = 0; i <10 ; i++ )
    {
   if(arr[i]%2 == 0)
     se = se+ arr[i];
    }
    s = se;
    break;
    case 'O':
    case 'o':
    for ( i = 0; i <10 ; i++ )
    {
   if(arr[i]%2 != 0)
     so = so + arr[i];
    }
    s = so;
    break;
    default :
     s = 0;
  }
  return s;
}
```

Output:

Enter 10 integers: 1 2 3 4 5 6 7 8 9 10

The sum of all integers: 55
The sum of all even integers: 30
The sum of all odd integers: 25
No sum for wrong choice. The result = 0

Freebooks.pk

9.15 Recursion

A programming technique in which a function calls itself is known as **recursion**. A function that calls itself is called **recursive function**. The recursion is a powerful technique.

Example

The calculation of factorial of a number is a simple example of recursive technique. Suppose the factorial of 5 is to be calculated.

The recursive solution of this probelm can be shown as follows:

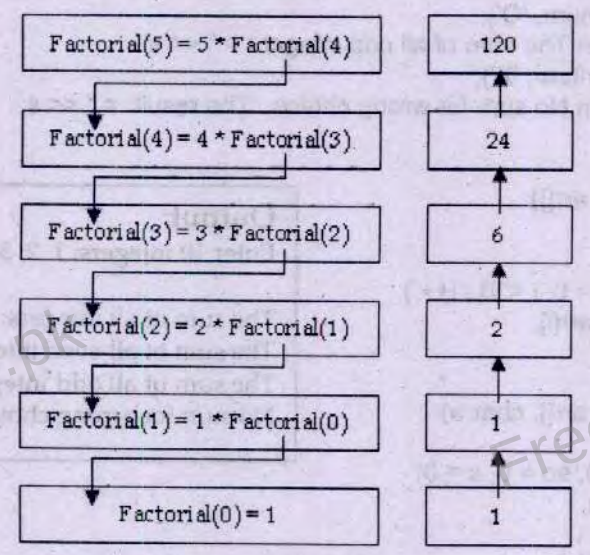


Figure 9.9: Recursive calculation of factorial

The above figure shows that factorial can be calculated by decomposing the problem. The factorial of 5 can be calculated by multiplying 5 with the factorial of 4. The factorial of 4 can be calculated by multiplying 4 with the factorial of 3 and so on.

The above procedure can be implemented as follows:

```
int Factorial (unsigned int n)
{
  if(n == 0)
    return 1;
  else
    return n * Factorial(n-1);
}
```

The following table provides a trace of different calls to Factorial function:

Call	n	n == 0	n * Factorial(n-1)	Returns
First	5	0	5 * Factorial(4)	120
Second	4	0	4 * Factorial(3)	24
Third	3	0	3 * Factorial(2)	6
Fourth	2	0	2 * Factorial(1)	2
Fifth	1	0	1 * Factorial(0)	1
Sixth	0	1		1

Freebooks.pk

A recursive function must have at least one **termination condition** that can be satisfied. Otherwise, the function will call itself indefinitely. The **Factorial** function has the termination condition **n == 0** that causes the recursive calls to move back.

Program 9.32

Write a program that inputs a number from user and calculates its factorial recursively.

```
#include <iostream.h>
#include <conio.h>

long int fact(int n)
{
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
void main()
{
  clrscr();
  int num;
  cout<<"Enter an integer: ";
  cin>>num;
  cout<<"Factorial of "<<num<<" is "<<fact(num);
  getch();
}
```

Output:
Enter an integer: 5
Factorial of 5 is 120

Program 9.33

Write a program that inputs two integers and calculates first number raised to the power of second number recursively.

```
#include <iostream.h>
#include <conio.h>
int power(int x, int n)
{
  if(n <= 0)
    return 1;
  else if(n == 1)
    return x;
  else
    return x * power(x, n-1);
}
void main()
{
  clrscr();
  int a, b;
  cout<<"Enter two integers: ";
  cin>>a>>b;
  cout<<a<<" ^ "<<b<<" = "<<power(a, b);
  getch();
}
```

Output:
Enter two integers: 2 5
2 ^ 5 = 32

Freebooks.pk

Programming Exercises

1. Write a function that inputs a decimal number and converts it to binary digits.
2. Write a function that converts binary to decimal.
3. Write a program that counts the number of zeros, odd, and even numbers.
4. Write a function that Print a triangle of stars.
5. Write a program that enters an number from the user and displays Fibonacci numbers from 1 to given number using function.
6. Write a program that accepts two integers. Create a function that tells whether or not the first integer is a multiple of the second.
7. Write a program that inputs two numbers in main function and passes them to a function. The function displays first number raised to the power of second number. For example, if the user enters 2 and 4, it displays 16.
8. Write a program that uses a function EQ() to find whether four integers a,b,c,d passed to the function satisfy the equation $a^3+b^3+c^3 = d^3$ or not. The function returns 0 if the above equation is satisfied and returns –1 otherwise.
9. Write a program that prompts the user to enter a number and reverse it. Write a function Reverse() to reverse the number. For example, if the user enters 2765, the function should reverse it so that it becomes 5672. The function should accept the number as an input parameter and return the reverse number.
10. Write a program that inputs a number in main() function and passes it to a function. The function displays whether number is prime or not.
11. Write a function LCM() that receives two integer arguments and returns LCM.
12. Write a program that calls two functions Draw_Horizontal and Draw_Vertical to construct a rectangle. Also write functions Draw_Horizontal to draw two parallel horizontal lines and the function Draw_Vertical to draw two vertical lines.
13. Write a function that returns the smallest of three floating point numbers.
14. Write a program that inputs five numbers and passes them to a function one at a time. The function returns **true** if the integer is even and false otherwise.
15. Write a program that prompts the user for Cartesian coordinates of two points x1, y1 and x2, y2 and displays the distance between them. Write a function Distance() with four input parameters to compute the distance. The function uses the following formula to compute distance and return result to calling function.?
16. Write a program that declares a function accepting two parameters. The first parameter is a floating pointer number and the second parameter is an integer. The program should multiply the floating-point number by itself the number of times indicated by the integer. The function should return the result to the main function. The main function should ask the user for the floating point number and integer. It should then call the function and store the result in a variable. Finally the main function should display the returned.
17. Write a function that accepts a salary and returns the tax according to following rules:

    No tax for first Rs.1000
    5% for second Rs.1000
    4% for third Rs.1000
    3% for remain untaxed salary

    For example, if the salary is Rs.4000, then the tax is Rs.120.

Freebooks.pk

18. Write a program that calculates greatest common divisor (gcd) of two numbers using recursive function.
19. Write a program that calculates Fibonacci series of a number using recursive function.
20. Write a program that calculates and displays the sum of the following series using function:

    x – (x ^ 2) / 2! + (x ^ 3) / 3! – (x ^ 4) / 4! ... + (x ^ 16) / 16!
21. Write a function AVG which calculates and displays the average of a player. Call this function in main function. The program inputs the runs given and balls delivered in main function. The average may be calculated by the formula:

    Average = (Total runs given * 60) / (Total number of balls delivered)
22. Write a program inputs an integer and passes it to a function. The function should return the number of digits in the integer. For example, if the integer is 35 the function should return 2, if it is 3572 the function should return 4.
23. Write a program that inputs five integers in a one-dimensional array and passes the array to a function. The function finds the maximum value in the array and returns to main function where it is displayed.
24. Write a program that gets two numbers, one should be passed by value and the other should be passed by reference and then check the original variables whether their values have been changed or not.
25. Write a program that inputs a positive integer and passes it to a function that displays the prime factors of this number. For example, prime factors of 24 are 2, 2, 2, 3 and prime factors of 35 are 5 and 7.
26. Write a function that takes two times as two integer arguments of hours and minutes. It returns the number of minutes between two times.
27. Write a program to generate a Pascal's triangle using function as follows:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

28. Write a program to use two functions Large() and Sum(). The Large() function gets two integer arguments by reference and sets the larger number to its square. The Sum() function gets an integer argument by value and returns the sum of the individual digits of the number. The main() function inputs two integers from user and prints the sum of individual digits and square of larger number.
29. Write a program that inputs five integers in a one-dimensional array and passes the array to a function. The function finds the minimum value in the array and returns to main function where it is displayed.
30. Write a program that inputs the name and population of two cities in structure variables and passes them to a function. The function displays the record of the city that has less population.
31. Write a program that inputs a float array having 10 elements. The program uses reverse() function to reverse this array. The main() function displays the original and reversed array.
32. Write a function Change() that accepts an array of integers and its size as parameters. It divides all array elements by 5 that are divisible by 5 and multiplies other array elements by 2.

Freebooks.pk

33. Write a program that inputs five integers in two arrays each. It declares a function that accepts four parameters. The first parameter is the first array, second parameter is the second array, third parameter is the third array and fourth parameter is the length of the arrays. The function adds the corresponding values of first two arrays and stores the result in the corresponding element of third array. The main function finally displays the values of all arrays. **Note:** The length of all three arrays must be same.
34. Write a program that inputs values in a 2-D array of 5 columns and 5 rows. It displays these values using a function display(). It passes the array to a function times2() that doubles the values stored in all elements of array. The program then again displays the changed values of the array using display() function.
35. Write an inline function MAX(double x, double y), which returns the maximum value of x and y. Test the function by reading values from the keyboard.
36. Write a program that inputs radius of circle and uses an inline function Area() to calculate and return the area of circle.

Exercise Questions

Q.1. What is difference between unstructured and structured programming languages?

In unstructured programming languages, the entire logic of the program is implemented in a single module or function. The programs written in these languages are error prone, difficult to understand, modify and debug.

In structured programming languages, the entire logic of the program is divided into number of smaller modules or function. Each module is a piece of code that implements a different functionality. The main module calls other modules when they are needed to execute.

Q.2. What is a function? List some benefits of using functions

A function is a named block of code that performs some action. The statements written in a function are executed when it is called by its name. Each function has a unique name. Functions are the building blocks of C++ programs. Functions are easier to write, modify and maintain. Functions can be reused and reduces programming time.

Q.3. What is difference between user-defined functions and built-in functions?

A type of function written by the programmer is known as user-defined function. User-defined function has a unique name. A program may contain many user-defined functions. These functions are written according to the exact need of the user.

A type of function that is available as a part of language is known as built-in function or library function. These functions are ready-made programs. These functions are stored in different header files. Built-in functions make programming faster and easier.

Q.4. What is a function header?

The first line of a function definition is known as function header. The function header consists of return type, function name and parameters.

Q.5. What do you know about function body?

The set of statements which are executed inside the function is known as function body. The body of function appears after function header. The statements are written in curly braces { }. The variable declaration and program logic are implemented in function body.

Q.6. What is function declaration or function prototype?

Function declaration is a model of a function. It is also known as function prototype. It provides information to compiler about the structure of the function to be used in program. It consists of function name, function return type and number and types of parameters. It is terminated with semicolon.

Freebooks.pk

Q.7. What is difference between function definition and function declaration?

A set of statements that explains what a function does is called **function definition**. The function definition consists of function header and function body. It must be defined before or after the main () function. It cannot be written inside the main() function.

Function declaration is a model of a function. It is also known as function prototype. It provides information to compiler about the structure of the function to be used in program. It consists of function name, function return type and number and types of parameters. It is terminated with semicolon.

Q.8. What is a function call?

The statement that activates a function is known as function call. A function is called with its name. Function name is followed by necessary parameters in parentheses. If there are many parameters, these are separated by commas.

Q.9. How a function returns value?

A function can return a single value. The return type in function declaration indicates the type of value returned by function. The keyword return is used to return the value back to the calling function.

Q.10. What is difference between local variable and global variable?

A variable declared inside a function is known as **local variable**. Local variable can be used only in the function in which it is declared. The lifetime of local variable starts when control enters the function in which it is declared. Local variable is automatically destroyed when control exits function

A variable declared outside any function is known as global variable. Global variables can be used by all functions in the program. Global variables exist in the memory as long as the program is running. These variables are destroyed from the memory when the program terminates.

Q.11. Explain the difference between call by reference and call by value. When would you use one over the other?

The call by value makes a copy of the parameter. The formal parameters of a function are initialized by the values of actual parameters given in function call.

The call by references makes an alias between the parameter and the parameters in function call. The values of actual parameter is not copied to formal parameters. The name of formal parameter actually refers to the memory location of actual parameter.

The call by reference is used when the change in formal parameter is also desired in actual parameters. It saves memory and provides efficiency as the values are not copied to formal parameters.

Give two main reasons for using reference parameters.

a) To change the value of the actual parameter

b) More efficient than copying

c) To return more than one value

Q.13. Write a prototype for each of the following descriptions:

a. A function called Function1 that has no parameters and returns an integer.
b. A function called Function2 that has no parameters and no return value.
c. A function called ReturnInteger that has an integer parameter called Value followed by a floating point value called Number. The function also returns an integer.
d. A function called NewFunction that accepts a floating point number called Number and an integer called Number2. The function also returns a floating point number.
e. Function swap which takes two integers x, y by reference, and returns a character.
f. Function intToDouble that takes an integer number and returns a double.

Answer:

a. int Function1();
b. void Function2();
c. int ReturnInteger(int Value, float Number);
d. float NewFunction(float Number, int Number2);
e. char swap(int x&, int& y);
f. double intToDouble(int number);

Freebooks.pk

Q.14. Does the following code compile and run without error?

```
#include <stdlib.h>
#include <iostream.h>
void F1();
void F2();
void main()
{
  F1()
}
void F1()
{
  F2()
}
void F2()
{
  cout<<"WELCOME";
}
```

No, the two calls to F1 and F2 must have semicolons after them.

Q.15. What is the output of the following C++ program?

```
int my_function(int, int);
void main()
{
        int temp = 0;
        int var1 = 3;
        int var2 = 5;
        temp = my_function(var1, var2);
}
int my_function(int x, int y)
{
        return x + y;
}
```

What is the final value of the variable *temp*?

Answer: 8

Q.16. What is the output of the following C++ program?

```
#include <iostream.h>
void showDub(int);
int main()
{
  int x = 2;
  showDub(x);
  cout << x << endl;
  return 0;
}
void showDub(int num)
{
  cout << (num * 2) << endl;
}
```

Answer:

4

2

Freebooks.pk

Q.17. What is the output of the following code, given the function definition below?

```
void main()
{
        int a = 7, b = 12;
        tester (a, b);
        cout << a << " " << b;
}
void tester (int m, int &n)
{
        n = n - 2 * m;
        m = 2 * m;
}
```

Answer: 7 -2

Q.18. What is the output of the following C++ program?

```
#include <iostream.h>
void doSomething(int&);
void main()
{
  int x = 2;
  cout << x << endl;
  doSomething(x);
  cout << x << endl;
}
void doSomething(int& num)
{
  num = 0;
  cout << num << endl;
}
```

Answer:

2
0
0

Q.19. What is the output of the following code?

```
void main()
{
        int a = 4, b = 10;
        a = aFunction (a, b);
        cout << a << " " << b;
}
int aFunction (int j, int &k)
{
        while (j < k)
        {
                j++;
                k -= 2;
        }
        return j;
}
```

Answer: 6 6

Freebooks.pk

Q.20. What is the output of the following C++ program?

```
void main()
{
        void numbers(int  x, int & y);
        int a,b,c;
        a = 22;
        b = 90;
        c = 14;
        numbers(a,a);
        numbers(a,b);
        numbers(b,a);
        cout <<a <<" " <<b  <<" " <<c <<endl;
}
void numbers (int x, int & y)
{
        int b;
        x +=6;
        y += 11;
        b = 55;
        cout <<b <<" " <<x <<" " <<y <<endl;
}
```

Answer:

55 28 33
55 39 101
55 107 44
44 101 14

Q.21. What is the output of the following C++ program?

```
int square(int& a)
{
        a=a*a;
        return a;
}
void main()
{
        int a=2;
        cout << "Is " << square(a=square(a)) << " equal to ";
        cout << a << "^4 ?" << endl;
}
```

Answer:

Is 16 equal to 16^4

Q.22. What is the output of the following program?

```
int test(int n1, int n2)
{
        cout << n2 << n1 << endl;
        return n2*n1;
}
void main()
{
        int n1=2, n2=3, n3=4;
        n2 = test(test(n1, n3), n2);
        cout << n1 << n3 << n2 << endl;
}
```

Answer:

42
38
2424

Freebooks.pk

Q.23. What is the output of the following C++ program?

```
int compute(int, int, const int[], const int[]);
void main()
{
        int c[6] = {2, 3, 5, 7, 93, 4};
        int d[7] = {23, 4, 9, 3, 40, 33, 1};
        cout << compute(2, 4, d, c) << endl;;
}
int compute(int start, int end, int const a[], int const b[])
{
        int sum = 0;
        for (int i = start; i <= end; i++)
                sum += b[i] - a[i];
        return sum;
}
```

Answer: 53

Q.24. Show the output of the following program?.

```
int i = 0;
const int j = 5;
void first()
{
        i = i+j;
        int j = ++i;
        int i = j;
}
void main()
{
        first();
        cout<<i<<" "<<j<<endl;
}
```

Answer:

6 5

Q.25. What is wrong with the following recursion program?

```
int f(int n)
{
        int result = 3+2*f(n-1);
        return (result);
}
void main()
{
        int r, n;
        cout << "Enter value for n:" << endl;
        cin >> n;
        r = f(n);
        cout << "Result is:" << r << endl;
}
```

Answer:

No stopping condition, or infinite loop

Multiple Choice

1. **Which of the following is type of function available in C++?**
   a. User-defined b. Built-in c. Subprogram d. Both a and b
2. **Another name for built-in function is:**
   a. User-defined function b. Library function c. Arithmetic function d. Both a & b
3. **A type of function that is available as part of language is known as:**
   a. User-defined function b. Library function c. Sub-program d. Both a and b
4. **Function prototype for built-in functions are specified in:**
   a. Source file b. Header file c. Object files d. Image files
5. **Global variables are created in:**
   a. RAM b. ROM c. Hard disk d. Cache
6. **Memory is allocated to a local variable at the time of its:**
   a. Declaration b. Destruction c. Definition d. First reference
7. **The name of actual and formal parameters:**
   a. May or may not be same b. Must be same
   c. Must be different d. Must be in lower case
8. **Formal arguments are also called:**
   a. Actual arguments b. Dummy arguments
   c. Original arguments d. Referenced arguments
9. **The pow() is a:**
   a. Built-in function b. User-defined function c. Local function d. Keyword
10. **A built-in function:**
    a. can not be redefined b. can be redefined c. can not return a value d. should be redefined
11. **Function declaration consists of:**
    a. Function name b. Return type c. Number and types of parameters d. All
12. **In a C++ program, two functions can have:**
    a. Same name b. Same parameters
    c. Same name and same parameters d. Same name but different parameters
13. **Which of the following is NOT a valid function declaration?**
    a. int ave3(int a, int b, int c); b. int 3ave(int a, int b, intc);
    c. int ave3(int, int, int); d. int ave_3(int a1, int a2, int a3);
14. **Function definition can be written:**
    a. Before main() function b. After main() function c. In a separate file d. All
15. **A value that can be sent to a function is known as:**
    a. User-defined function. b. Function c. Argument. d. None.
16. **The process of sending an argument to a function is called:**
    a. Sending b. Filtering c. Delivering d. Passing
17. **The function definition consists of:**
    a. Function header b. Function body c. Both a and b d. None
18. **The first line of function definition is known as:**
    a. Function header b. Function body c. Arguments d. None
19. **The statement that activates a function is known as:**
    a. Function call b. Function output c. Invoking a function d. None
20. **Which of the following steps takes place when function is called?**
    a. The control moves to the function that is called
    b. All statements in the function body are executed
    c. The control returns back to the calling function
    d. All of these

Freebooks.pk

21. **What is the variable name that is used by a function to receive passed values?**
a. Function b. Parameter c. Expression d. Constant
22. **Which of the following is incorrect?**
a. A function can call another function.
b. A function can be called many times in a program.
c. A function can return values input by the user.
A function must have at least one value parameter
23. **The parameters in function declaration are called:**
a. Formal parameters b. Actual parameters c. Both a and b d. None
24. **Multiple arguments to a function are separated by:**
a. Comments b. Semicolons c. Colons d. Commas
25. **Which statement is used by a function use to return a value?**
a. Function statement b. Return statement c. Continue statement d. None
26. **The scope of a variable refers to:**
a. Length of variable b. Name of variable
c. Accessibility of variable d. Data type of variable
27. **A variable declared inside a function is known as:**
a. Local variable b. Global variable c. Automatic variable d. a and c
28. **A variable declaration outside any function is known as:**
a. Global variable b. Local variable c. External variable d. Static
29. **Which of the following is true about a function call?**
a. Stops the execution of the program b. Transfer control to the called function
c. Transfer control to the main function d. Resumes the execution of the program
30. **Which of the following looks for the prototypes of functions used in a program?**
a. Linker b. Loader c. Compiler d. Parser
31. **The declaration "void foo (int x, int & y);" is called a:**
a. Function body b. function type c. Function stereotype d. function prototype
32. **The function "foo" from question 31 returns the following type as result:**
a. integer b. no result is returned c. float d. none of the above
33. **An example of a passed-by-value parameter in declaration of void foo (int x, int & y);" is:**
a. x b. y c. c d. foo
34. **An example of passed-by-reference parameter in declaration of "void foo (int x, int & y);" is:**
a. x b. y c. c d. foo
35. **The default parameter passing mechanism in C++ is:**
a. Call by reference b. Call by name c. Call by value d. Call by global variable
36. **Parameters are preferable to global variables because:**
a. Function is considerably shorter b. Function is considerably longer
c. Function can be reused in other programs d. Function is more efficient in terms of run time
37. **In parameter passing by value:**
a. Actual and formal parameters must be similar types.
b. Parameter passing by value van be used both for input and output purpose.
c. Both a and b
d. None
38. **Functions can communicate data by:**
a. Global variables b. Returning value under function name
c. using parameters d. All
39. **Reference parameters:**
a. are specified by using an & in the formal parameter list
b. efficiently share complex objects with a function
c. are used to pass back values from a function d. All

Freebooks.pk

40. When you pass an array to a function, the function receives _____.

a. Copy of array b. Reference of array c. Length of array d. None

41. Which of the following function declarations will accept an array declared int x[30][40]?

a. void doStuff(int x[]) ; b. void doStuff(int x[][]) ;

c. void doStuff(int x[30][]) ; d. void doStuff(int x[][40]) ;

42. How is an array passed into function?

a. always by value b. always by reference c. by value or by reference d. None

43. Which of the following function declarations will accept following two-dimensional array?

int pages[10][30];

a. void f1(int pages[][], int size); b. void f1(int pages[][30], int size);

c. void f1(int pages[10][], int size); d. void f1(int& pages, int size);

44. Consider the C++ function ppp:

```
int ppp(int k)
{
  if (k < 1)
        return k;
  else
        return k + ppp(k/2);
}
```

The value returned by the call ppp(9) is:

a. 13 b. 15 c. 16 d. 17

45. Consider the C++ function qqq:

```
int qqq(int k)
{
        if (k < 1)        return k;
        else              return qqq(k-3) + qqq(k/2);
}.
```

The value returned by the call qqq(7)is:

a. –7 b. –6 c. –5 d. –2

46. What is the value of x after the following function call, given the function definition below?

```
int x;
aFunction (3, x);
void aFunction (int a, int &b)
{
        b = a * 5 + 1;
}
```

a. 16 b. 32 c. 14 d. 0

47. What value would the following function return when called with an argument of 3?

```
int factorial(int n)
{
        int product=0;
        while(n > 0)
        {
                product = product * n;
                n--;
        }
        return product;
}
```

a. 0 b. 6 c. 10 d. 12

Freebooks.pk

48. **What is the value of b after the following function call?**

```
int b = 3;
mystery (b);
void mystery  (int &val)
{
        for (int c = 0; c < 5; c++)
                val += 2;
}
```

a. 13 b. 16 c. 12 d. 11

Answers

1. d	2. b	3. b	4. b	5. a	6. c
7. a	8. b	9. a	10. a	11. d	12. d
13. b	14. d	15. c	16. d	17. c	18. a
19. a	20. d	21. b	22. d	23. a	24. d
25. b	26. c	27. d	28. a	29. b	30. c
31. d	32. b	33. a	34. b	35. c	36. c
37. a	38. d	39. d	40. b	41. d	42. b
43. b	44. c	45. a	46. a	47. a	48. a

Fill in the Blanks

1. A _____ is a self-contained piece of code.
2. _____ allows to reuse the existing code as and when required.
3. Another name for parameters is_____.
4. _____ of a variable refers to the region of the program where it can be referenced
5. The parameters specified in the function header are called_____ parameter.
6. The parameters passed to a function in the function call are called_____ parameters.
7. Functions help to achieve _____ programming.
8. Two types of functions in C++ language are built-in functions and _____ functions.
9. _____ Functions are predefined functions that provide convenient ways to perform variety of task.
10. Predefined functions are packaged in_____.
11. A _____ provide basic information about the function to the compiler.
12. A function in C++ consists of _____ that identifies the function followed by body of the function between curly braces.
13. The first line of function definition is called _____.
14. A function with no parameters specifies keyword _____ as its parameter list.
15. The _____ statement is used to specify the value returned by the function.
16. _____ is a mechanism that is used to invoke a function to perform a specific task.
17. The duration for which a variable exists in memory is called its_____.
18. _____ of a variable refers to the part of program where it can be referenced.
19. _____ variables are declared outside all blocks.
20. A function cannot return more than_____ value through return statement.
21. The parameters specified in the function header are called_____ parameter.
22. The parameters passed to a function in function call are called_____ parameters.
23. The formal parameters are also called _____.

Freebooks.pk

24. Function prototype consists of function name, function _____ and function parameters.
25. A function with no return type is indicated by keyword _____.
26. **float fun (int);** indicates that the result of the function is of _____ type.
27. A set of statements that explains what a function does is called _____.
28. Statement that executes a function is known as function_____.
29. _____are the values provided to a function when it is called.
30. The sequence and types of parameters in function call must be similar to the sequence and types of parameters in function _____.
31. Actual parameters are the parameters in function _____.
32. Parameters in the function declaration are called_____ parameters.
33. When a function call is executed, values are copied to _____parameters from_____ parameters.
34. sqrt() function is found in _____ header file of C++.
35. The control moves back to the calling program along with the returned value when _____statement is executed in a function.
36. A variable declared inside a function is known as_____ variable.
37. Local variables can be used only in the function in which they are_____.
38. Local variables are automatically destroyed when _____ exits from function.
39. A variable declared outside any function is known as_____.
40. getch() and getche( ) function is found in _____ header file of C++.
41. _____ statement is used in C++ to return the result of function to calling program.
42. _____ is a mechanism that is used to invoke a function to perform a specific task.
43. The_____ statement is used to specify the value returned by a function.

Answers

1. Function	2. Functions	3. Functions
4. scope	5. formal	6. actual
7. modular	8. User-defined	9. Built-in
10. Libraries	11. function prototype	12. function header
13. function header	14. void	15. return
16. Function call	17. life time of variable	18. scope of a variable
19. global variables	20. one	21. Formal arguments or formal parameters
22. Actual arguments or actual parameters	23. dummy arguments	24. Return type
25. void	26. float	27. Function Definition
28. Call	29. Parameters	30. Declaration
31. Call	32. Formal	33. Formal, Actual
34. One	35. return	36. Local
37. Declared	38. Control	39. Global variable
40. conio.h	41. return	42. function call
43. return		

Freebooks.pk

True/ False

1. In C++, arguments can be passed to a function only by value.
2. There can be multiple main functions in a C++ program.
3. A function can be called anywhere in the program.
4. In C++, every function must return a value.
5. a user-defined function cannot be called in another user-defined function.
6. A function can be called only once in a program.
7. Scope of local variable is the block in which it is defined.
8. Global variables exist in memory till the execution of the program.
9. An unstructured program is more difficult to debug than a structured program.
10. Function body is an optional part of the function.
11. A block of code for a specific purpose is called function.
12. Function statements are executed when it is declared.
13. No function can be written as an independent block.
14. Reusability is one of the main features of functions.
15. Function written by the programmer is called built-in function.
16. User-defined function may also be called library function.
17. A user-defined function can call library functions or user-defined functions.
18. Ready-made functions are part of the language and are called built-in functions.
19. Function prototype is a model for the function.
20. Compiler gets information about the structure of a function from function definition.
21. Function declaration consists of three parts: function name, function return type and number & types of function parameters.
22. Every function prototype must include at least one formal parameter.
23. The keyword void is used if the function returns a string value.
24. A set of statements that explains what a function does is called function definition.
25. Function definition may be written in a separate file.
26. The first line of function declaration is known as function header.
27. A set of statements executed inside the function is called function body.
28. Parameters are the values passed to a function when it is declared.
29. The keyword void is used in parenthesis if there is no parameter.
30. Statement that activates a function is called function call.
31. An expression cannot be used as an actual argument in a function call.
32. The sequence and types of parameters in function call must be similar to the sequence and types of parameters in function declaration.
33. An actual argument of type int cannot be passed to corresponding formal parameter of type double.
34. Formal parameters are the values that those used in function call.
35. A function normally can return two values.
36. The control moves back to calling function with returned value when 'back' statement is executed.
37. The control is transferred to the next defined function after the last statement of a function executes.
38. The calling function may use the returned value in an assignment statement.
39. A user-defined function can call library functions or user-defined functions.
40. A function that takes no arguments is more versatile than a function that requires arguments.
41. A variable declared inside a function is called automatic variable.
42. The scope of a variable is actually the area in which the variable can be accessed.
43. The duration for which a variable exists in CPU is called lifetime of variable.
44. Local variables are destroyed when control exits from the function.
45. Local variable declared with the keyword static is called constant variable.

Freebooks.pk

46. Local variables are also called automatic variables.
47. Local variable may be created several times in a single program.
48. Referencing an identifier outside its scope will cause a run-time error.
49. clrscr () function can be used without including conio.h header file.
50. math.h header file contains sqrt() function.
51. Global variables are declared inside main(), at the very beginning.
52. The scope of a variable is the code block surrounding the variable.
53. A function may return more than one item.
54. Functions may have multiple return statements.
55. The keyword void is used in a function declaration or definition to specify that the function does not return a value.
56. the parameters listed in the function declaration are considered global variables.
57. The keyword void is used in a function declaration or definition to specify that the function does not return a value
58. It is possible to have a function that has no parameters.
59. It is illegal for one function to call other functions.
60. The scope of a variable declared within a function is "all functions in the same source code file".
61. Using pass-by-reference parameters in a function allows the function to modify variables passed to it as arguments.
62. A reference parameter requires that the actual parameter be an lvalue. A function's local variables are accessible by other functions.
63. A prototype specifies a function's interface
64. A function cannot modify the members of a structure.
65. Structure variables may be passed as arguments to functions.
66. A variable of struct type can be passed to a function as an argument.
67. Anything that can be solved using pass-by-value, can be solved using pass-by-reference.
68. Expressions shown in the function definition are called actual parameters.
69. After a function is executed, execution jumps back to the main function.
70. A function must always be defined before it can be called.
71. All functions have input parameters.
72. If there is only one declaration for a variable, it is always visible within its scope.
73. Arrays cannot be function return types.
74. The definition of a function consists of the function name, an optional function type, the list of required info (if any), and a compound statement.
75. Every function must have a prototype

Answers

1. F	2. F	3. T	4. F	5. F	6. F
7. T	8. T	9. T	10. F	11. T	12. F
13. F	14. T	15. F	16. F	17. T	18. T
19. T	20. F	21. T	22. F	23. F	24. T
25. T	26. F	27. T	28. F	29. T	30. T
31. F	32. T	33. F	34. F	35. F	36. F
37. F	38. T	39. T	40. F	41. T	42. T
43. F	44. T	45. F	46. T	47. T	48. F
49. F	50. T	51. F	52. T	53. F	54. T
55. T	56. F	57. T	58. T	59. F	60. F
61. T	62. F	63. T	64. F	65. T	66. T
67. T	68. F	69. F	70. F	71. F	72. T
73. F	74. F	75. F			

Freebooks.pk

CHAPTER 10

BUILT-IN FUNCTIONS

Chapter Overview

10.1 Built-in Functions
10.2 The 'conio.h' Header File
10.3 The 'stdio.h' Header File
10.4 Math Functions (math.h)
10.5 Type Functions / Character Functions (ctype.h)

Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

10.1 Built-in Functions

A type of function that is available as a part of language is known as **built-in function** or **library function**. These functions are ready-made programs. These functions are stored in different header files. Built-in functions make programming faster and easier. C++ language provides many built-in functions to solve different problems.

Different built-in functions are defined in different header files. Bult-in functions can be accessed in programs by including the header files in which these functions are defined.

10.2 The 'conio.h' Header File

The word 'conio' stands for **console input/output**. The header file 'conio.h' contains the built-in function that are used for input and output. Some important function defined in this header file are as follows:

clrscr()

The word 'clrscr' stands for **clear screen**. The function is used to clear the contents of the screen. When this function is executed, the screen is cleared and the cursor moves to the start of first line on the screen.

Syntax: clrscr();

Example

```
#include <conio.h>
void main()
{
  clrscr();
}
```

clreol()

The word 'clreol' stands for **clear end of line**. The function is used to clear current line from current cursor position to the end of line. The remaining text on screen is not cleared.

Syntax: clreol();

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  cout<<"Hello World of C++ \r";
  clreol();
  cout<<"Pakistan";
}
```

First of all, the above example displays "Hello World of C++" on the screen. The '\r' escape sequence moves the cursor at the beginning of the line. The clreol() function clears the whole line. The last statement dispalys "Pakistan".

getch()

The word 'getch' stands for **get character**. The function is used to input single character from the keyboard. The character typed by the user does not appear on the screen. It can be stored in a variable by using assignment statement. The user does not need to press **Enter** key to complete the input. The function is frequently used to pause program execution.

Syntax: variable = getch();

The use of variable is optional.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char ch;
  cout<<"Enter any character: ";
  ch = getch();
  cout<<"You entered "<<ch;
  getch();
}
```

The above example inputs a character in variable **ch** by using getch() function and displays the character. The last statement again uses getch() function to pause the program execution. The user can see the output and then press any key to end the program.

getche()

The word 'getche' stands for **get character echo**. The function is used to input single character from the keyboard. The character typed by the user also appears on the screen. It can be stored in a variable by using assignment statement. The user does not need to press **Enter** key to complete the input.

Syntax: variable = getche();

The use of variable is optional.

kbhit()

The word 'kbhit' stands for **keyboard hit**. The function is used to check if the user hit the keyboard or not. It returns a non-zero value if the user hits any key on the keyboard and returns zero otherwise. The value of the key can also be stored in a variable by using assignment statement.

Syntax: variable = kbhit ();

The use of variable is optional.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  while(!kbhit())
    cout<<"Pakistan Zindabad\t";
}
```

The above example repeatedly displays "Pakistan Zindabad" on the screen until the user presses any key on the keyboard.

gotoxy()

The gotoxy() function is used to move the cursor to a specified location on the screen. It moves the cursor with respect to x-axis and y-axis. It is very useful when the output is to be displayed at a specific location on the screen.

Syntax: gotoxy(x, y);

The variable x indicates x-axis. Its value can be from 0 to 79. The variable y indicates the y-axis. Its value can be from 0 to 24.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  gotoxy(0,0);
  cout<<"Pakistan Zindabad";
  gotoxy(1,50);
  cout<<"Pakistan Zindabad";
  getch();
}
```

The above example displays "Pakistan Zindabad" on the top and bottom of the screen.

10.3 The 'stdio.h' Header File

The word 'stdio' stands for **standard input/output**. The 'stdio.h' header files contains different function that are used in the tasks of input and output. Some important function defined in this header file are as follows:

getchar()

The word 'getchar' stands for **get character**. The function works similar to getch() and getche() functions of 'conio.h' header file. It is used to input single character from keyboard. The character can be stored in a variable by using assignment statement. The user needs to press **Enter** key to complete input. The character typed by the user also appears on screen.

Syntax: variable = getchar();

The use of variable is optional.

putchar()

The word 'putchar' stands for **put character**. The function is used to display single character on the screen.

Syntax: putchar(ch);

The parameter **ch** is the variable or constant whose value is to be displayed on screen.

Example

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
  char ch;
  cout<<"Enter a character:";
  cin>>ch;
  putchar(ch);
  putchar('*');
}
```

Freebooks.pk

How above Program Works?

The above program inputs a character from the user in **ch** variable. It displays the value of the variable using 'putchar' function. The last statement displays a constant value '*' on the screen using 'putchar' function.

gets()

The word 'gets' stands for **get string**. The function is used to input a string value from keyboard. The value entered by the user can be stored in an array of characters. The string may consist of any characters and space.

The user also needs to press **Enter** key after typing the string to complete the input. A NULL character is added at the end of string automatically when the user presses **Enter** key.

Syntax: gets(variable);

The parameter **variable** is the variable that is used to store the string value.

puts()

The word 'puts' stands for **put string**. The function is used to display a string value on the screen. It can display both string varialbe as well as string constant. The string constant is enclosed in double quotation marks.

Syntax: puts(variable);

The parameter **variable** is the variable whose value is to be displayed on the screen.

Example

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
  clrscr();
  char string[] = "This is an example output string\n";
  puts(string);
  getch();
}
```

10.4 Math Functions (math.h)

The header file 'math.h' contains the built-in function that are used to perform arithmetic calculations. Some important function defined in this header file are as follows:

abs()

The abs() function is used to find the absolute value of an integer.

Syntax: abs(x);

x It indicates the integer whose absolute value is to be found.

fabs()

The fabs() function is used to find the absolute value of a floating point number.

Syntax: fabs(x);

x It indicates the floating point number whose absolute value is to be found.

Freebooks.pk

Program 10.1

Write a program that inputs an integer number and a float number. It dispalys the absolute values of both numbers.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{
  clrscr();
  int n;
  float f;
  cout<<"Enter an integer: ";
  cin>>n;
  cout<<"Enter a float: ";
  cin>>f;
  cout<<"The absolute value of "<<n<<" is "<<abs(n)<<endl;
  cout<<"The absolute value of "<<f<<" is "<<fabs(f);
  getch();
}
```

Output:
Enter an integer: -5
Enter a float: -3.8
The absolute value of -5 is 5
The absolute value of -3.8 is 3.8

ceil()

The ceil() function rounds a float or double value to the next integer and returns it.

Syntax: ceil(x);

x It indicates the float or double value that is to be rounded.

floor()

The floor() function rounds a float or double value to an integer and returns it. The rounded value is not greater than the original value.

Syntax: floor(x);

x It indicates the float or double value that is to be rounded.

Example

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{
  clrscr();
  double d;
  cout<<"Enter a double: ";
  cin>>d;
  cout<<"The rounded value with ceil() = "<<ceil(d)<<endl;
  cout<<"The rounded value with floor() = "<<floor(d);
  getch();
}
```

Output:
Enter a double: 14.2
The rounded value with ceil() = 15
The rounded value with floor() = 14

cos()

The cos() function is used to find the trignometric cosine of the specified angle. The angle is given in radians as floating point value.

Freebooks.pk

Syntax: cos(x);

x It indicates the angle in radians.

sin()

The sin() function is used to find the trignometric sine of the specified angle. The angle is given in radians as floating point value.

Syntax: sin(x);

x It indicates the angle in radians.

tan()

The tan() function is used to find the trignometric tangent of the specified angle. The angle is given in radians as floating point value.

Syntax: tan(x);

x It indicates the angle in radians.

log()

The log() function is used to find the natural logarithm of a given floating point value.

Syntax: log(x);

x It indicates the value whose log is to be found.

log10()

The log10() function is used to find base 10 logarithm of a given floating point value.

Syntax: log10(x);

x It indicates the value whose log is to be found.

Program 10.2

Write a program that inputs a floating point number and displays cosine, sine, tangent, log and base 10 log of the given number.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{
  clrscr();
  float n;
  cout<<"Enter a floating point value: ";
  cin>>n;
  cout<<"Trigonometric cosine of "<<n<<" is = "<<cos(n)<<endl;
  cout<<"Trigonometric sine of "<<n<<" is = "<<sin(n)<<endl;
  cout<<"Trigonometric tangent of "<<n<<" is = "<<tan(n)<<endl;
  cout<<"Natural Log of "<<n<<" is = "<<log(n)<<endl;
  cout<<"Base 10 Log of "<<n<<" is = "<<log10(n)<<endl;
  getch();
}
```

Output:

```
Enter a floating point number: 5.5
Trigonometric cosine of 5.5 = 0.70867
Trigonometric sine of 5.5 = 0.70554
Trigonometric tangent of 5.5 = -0.995584
Natural Log of 5.5 = 1.704748
Base 10 Log of 5.5 = 0.740363
```

Freebooks.pk

fmod()

The fmod() function is used to find remainder by dividing two floating point values.

Syntax: fmode(x, y);

x It indicates a floating point value to be used as nominator. It can be a variable or constant value.

y It indicates a floating point value to be used as denominator. It can be a variable or constant value.

Program 10.3

Write a program that inputs two floating point numbers, divides these numbers and displays the remainder of the division operation using built-in function.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{
  clrscr();
  double a, b, r;
  cout<<"Enter first double value: ";
  cin>>a;
  cout<<"Enter second double value: ";
  cin>>b;
  r = fmod(a, b);
  cout<<"The result of fmod()= "<<r;
  getch();
}
```

Output:
Enter first double value: 5.3
Enter second double value: 2.5
The result of fmod() = 0.3

pow()

The pow() function is used to find the result of one integer raised to the power of second integer (n^m).

Syntax: pow(x, y);

x It indicates the first integer whose power is to be calculated.

y It indicates the second integer to be used as exponent of the number.

Program 10.4

Write a program that inputs two integer numbers and displays the result of first number raise to the power of second number using built-in function.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{
  clrscr();
  int a, b;
  cout<<"Enter first integer : ";
  cin>>a;
  cout<<"Enter second integer: ";
  cin>>b;
```

Output:
Enter first integer: 3
Enter second integer: 2
The result of pow() = 9

Freebooks.pk

```
    cout<<"The result of pow(a,b) = "<<pow(a,b);
    getch();
  }
```

sqrt()

The sqrt() function is used to calculate the square root of a floating point number.

Syntax: sqrt(x);

x It indicates the floating point number whose square root is to be calculated.

Program 10.5

Write a program that inputs a floating point number and displays its square root using built-in function.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{
  clrscr();
  float n;
  cout<<"Enter a floating point number: ";
  cin>>n;
  cout<<"The square root of "<<n<<" is = "<<sqrt(n);
  getch();
}
```

Output:
Enter a floating point number: 3.5
The square root of 3.5 is = 1.870829

10.5 Type Functions / Character Functions (ctype.h)

The header file 'ctype.h' contains the built-in function that are used to find the types of different values. The character functions are extremely useful for testing and transforming characters. Some important function defined in this header file are as follows:

isxxxx Functions

The **isxxxx** functions are used to test an integer value of a character and return a non-zero value if integer satisfies the condition. It returns 0 if integer does not satisfy condition. These functions use ASCII character set. Different functions are as follows:

Function	Description
isalnum	It returns a non-zero value if the parameter is a letter or digit.
isalpha	It returns a non-zero value if the parameter is a letter.
iscntrl()	It returns nonzero if character is a control character otherwise returns 0.
isdigit	It returns a non-zero value if the parameter is a digit (0–9.
islower	It returns a non-zero value if the parameter is a lower-case character.
isupper	It returns a non-zero value if the parameter is an upper-case character.
isprint	It returns a non-zero value if the parameter is a printing character including space.
ispunct	It returns a non-zero value if the parameter is a punctuation character.
isspace	It returns a non-zero value if the parameter is a tab, linefeed, form feed, return or space.
isxdigit	It returns a non-zero value if the parameter is one of the characters from 0 to 9, A to F or a to f.

Freebooks.pk

Syntax: isxxxx(p);

x It indicates the parameter whose value is to be tested.

Example

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>
void main ()
{
  clrscr();
  char ch;
  cout<<"Enter a single character: ";
  cin>>ch;
  cout<<ch<<" is: "<<endl;
  if(isalnum(ch) != 0)
    cout<<"\t an alphnumeric."<<endl;
  else
    cout<<"\t not an alphnumeric."<<endl;
  if(isalpha(ch) != 0)
    cout<<"\t an alphbet."<<endl;
  else
    cout<<"\t not an alphabet."<<endl;
  if(isdigit(ch) != 0)
    cout<<"\t a digit."<<endl;
  else
    cout<<"\t not a digit."<<endl;
  if(islower(ch) != 0)
    cout<<"\t lower character."<<endl;
  else
    cout<<"\t not lower character."<<endl;
  if(isupper(ch) != 0)
    cout<<"\t upper character."<<endl;
  else
    cout<<"\t not upper character."<<endl;
  if(isprint(ch) != 0)
    cout<<"\t a printing character."<<endl;
  else
    cout<<"\t not a printing character."<<endl;
  if(ispunct(ch) != 0)
    cout<<"\t a punctuation character."<<endl;
  else
    cout<<"\t not a punctuation character."<<endl;
  if(isspace(ch) != 0)
    cout<<"\t a space."<<endl;
  else
    cout<<"\t not a space."<<endl;
  if(isxdigit(ch) != 0)
    cout<<"\t a character between 0-9, A-F or a-f."<<endl;
  else
    cout<<"\t not a character between 0-9, A-F or a-f."<<endl;
  getch();
}
```

Freebooks.pk

toascii()

The toascii() function is used to convert a character to an ASCII character. If the character is already a valid character, it is returned unchanged.

Syntax: toascii(ch);

ch It indicates the parameter whose value is to be converted.

tolower()

The tolower() function is used to convert a character to lower case character. If the character is already in lower case, it is returned unchanged.

Syntax: tolower(ch);

ch It indicates the parameter whose value is to be converted.

toupper()

The toupper() function is used to convert a character to an uppercase character. If the character is already in uppercase, it is returned unchanged.

Syntax: toupper(ch);

ch It indicates the parameter whose value is to be converted.

Program 10.6

Write a program that inputs a sentence in lowercase and displays it in uppercase.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
void main ()
{
  int i = 0;
  char ch[50];
  cout<<"Enter a sentence: ";
  gets(ch);
  cout<<"The sentence in lowercase is: ";
  while(ch[i]!='\0')
  {
    ch[i] = tolower(ch[i]);
    i++;
  }
  puts(ch);
  i = 0;
  cout<<"\nThe sentence in uppercase is: ";
  while(ch[i]!='\0')
  {
    ch[i] = toupper(ch[i]);
    i++;
  }
  puts(ch);
  getch();
}
```

Note: The Built-in functions for string handling are covered in chapter 12 "String Handling".

Freebooks.pk

Multiple Choice

1. **Which of the following function is used to clear the contents of the screen?**
   a. clrscr() b. clscr() c. clreol d. None
2. **Which of the following function is used to find the result of one integer raised to the power of second integer?**
   a. pow() b. power() c.pwr() d. None
3. **Which of the following function is used to test whether a character is a numeric digit character?**
   a. isdigit- b. isnumeric c. isnumber d. None
4. **Which of the following function is used to test whether a character is a printable character?**
   a. isprint- b. isalpha c. isprintok d. isprintable
5. **Which of the following function is used to determine whether a character entered is a letter of the alphabet**
   a. isdigit b. isalpha c. alphaok d. None
6. **Which of the following function is used to determine whether a character is whitespace**
   a. isspace b. isblank c. iswhite d. None
7. **Which of the following function is used to change a character argument from lower to upper case?**
   a. itolarge b. tosmall c. toupper d. toolower

Answers

1. a	2. a	3. a	4. a
5. b	6. a	7. c	8.

Fill in the Blanks

1. The_____ function is used to check if the user hit the keyboard or not.
2. _____ function is used to move the cursor to a specified location on the screen
3. The header file _____ contains the built-in function that are used to perform arithmetic calculations
4. The _____ function is used to calculate the square root of a floating point number
5. The _____ function is used to convert a character to lower case character
6. The_____ function is used to find the absolute value of an integer.
7. The_____ function rounds a float or double value to the next integer and returns it.

Answers

1. kbhit	2. gotoxy()	3. math.h	4. sqrt()
5. tolower()	6. abs()	7. ceil()	8.

Freebooks.pk

CHAPTER 11

POINTERS

Chapter Overview

11.1 Memory and References
11.2 Pointers
- 11.2.1 Pointer Declaration
- 11.2.2 The 'void' Pointer
- 11.2.3 Dereference Operator
- 11.2.4 Pointer Initialization

11.3 Operations on Pointers
- 11.3.1 Pointer Addition
- 11.3.2 Pointer Subtraction

11.4 Pointers and Arrays
- 11.4.1 Accessing Array Elements with Pointers

11.5 Pointers and Strings
11.6 Array of Pointers
11.7 Pointers and Functions
11.8 Pointers and Structures
- 11.8.1 Passing Structure to Function using Pointers

11.9 Memory Management with Pointers
- 11.9.1 Dynamic Variables
- 11.9.2 The new Operator
- 11.9.3 The delete Operator

Programming Exercises
Exercise Questions
Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

11.1 Memory and References

Computer memory is a collection of different consecutive memory locations. These memory location are numbered sequentially. Each variable is created at a unique location in memory known as its **address**.

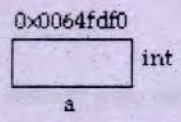
A program may declare many variables for different tasks. A variable declaration reserves specific amount of space in memory for a particular variable. The variable name is used to refer to that memory location. It allows the user to access a value in the memory. The computer refers to the memory using an address. A variable declaration associates following three attributes to a variable:

- Variable name
- Variable type
- Variable memory address

The following statement declares an integer variable:

```
int a;
```

The name of variable is **a** and the type of variable is **int**. The address of the variable is unknown. Computer creates the variable at any available location in the memory. The memory address is a hexadecimal number that refers to a particular location in the memory. The variable is created in the memory as follows:



The above box represents the memory location allocated for the variable **a**. The hexadecimal value above the box is its assumed address. The variable will occupy 2 bytes or 4 bytes in memory depending on the type of computer.

The actual address of the variable can be displayed by using **reference operator &**. It is also known as **address operator**. The reference operator is used to access the memory address of a variable. The following statement will display the address of **a**:

```
cout<<&a;
```

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int n = 10;
  cout<<"The value of n: "<<n<<endl;
  cout<<"The address of n: "<<&n<<endl;
  getch();
}
```

Output:
The value of n: 10
The address of n: 0x8fb1fff4

How above Example Works?

The above program declares an integer variable **n** and initializes it to 10. The program displays the value and address of **n**. The address of operator is used to display the memory address of **n**. The memory address may be different each time the program is executed.

11.2 Pointers

A **pointer** is a variable that is used to store a memory address. The reference operator is used to access the memory address of a variable and store it in a pointer.

11.2.1 Pointer Declaration

The method of declaring a pointer is same as declaring a simple variable. An asterisk '*' is used in the declaration that indicates that the variable is a pointer variable.

Syntax

The syntax of declaring a pointer is as follows:

```
DataType *var;
```

DataType	It is the type of variable pointed by the pointer variable.
*	It indicates that the variable is a pointer variable
var	It is the name of the pointer variable.

Example

```
int *p;
```

The above statement declares a pointer variable **p**. The data type **int** indicates that it can store the memory address of an integer variable. The data type of a pointer must be same as the data type of the variable whose memory address is to be stored in the pointer.

It is also possible to declare many pointer variables in one statement as follows:

```
float *p1, *p2;
```

Program 11.1

Write a program that inputs a number in an integer variable. It stores the address of the variable in a pointer and then displays the value and address of the variable.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  int n;
  int *ptr;
  cout<<"Enter an integer: ";
  cin>>n;
  ptr = &n;
  cout<<"The value of n: "<<n<<endl;
  cout<<"The address of n: "<<ptr<<endl;
  getch();
}
```

Output:
Enter an integer: 25
The value of n: 25
The address of n: 0x8facfff4

How above Program Works?

The above program declares an integer variable **n** and an integer pointer **ptr**. It inputs a value from the user and stores it in **n**. The address of **n** is assigned to **ptr** using address of operator. Finally, the value and address of the variable is displayed.

11.2.2 The 'void' Pointer

The type of pointer variable and the type of variable it refers must be same. It restricts the use of a pointer variable to specific type of variables. A pointer variable can store the address of any type of variable if it is declared as **void**.

Freebooks.pk

The keyword 'void' is used as the data type of the pointer as follows:

```
void *p;
```

The above statement declares a pointer variable **p**. The data type of the pointer is **void**. It means that it can store the memory address of any type of variable.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int n = 10;
  float f = 25.18;
  char c = '$';
  void *ptr;
  ptr = &n;
  cout<<"The value of n: "<<n<<endl;
  cout<<"The address of n: "<<ptr<<endl;
  ptr = &f;
  cout<<"The value of f: "<<f<<endl;
  cout<<"The address of f: "<<ptr<<endl;
  ptr = &c;
  cout<<"The value of c: "<<c<<endl;
  cout<<"The address of c: "<<ptr<<endl;
  getch();
}
```

Output:
The value of n: 10
The address of n: 0x8f72fff4
The value of f: 25.18
The address of f: 0x8f72fff0
The value of c: $
The address of c: 0x8f72ffef

How above Program Works?

The above program declares and initializes three variables of type **int**, **float** and **char**. It also declares a **void** pointer variable. The pointer can refer to different types of variables. The program stores the memory addresses of the variables **n**, **f** and **c** one by one and displays the values and memory addresses of these variables.

11.2.3 Dereference Operator

The **dereference operator** is used to access the value of the variable whose memory address is stored in pointer. It is denoted by asterisk '*'. It is also called **indirection operator**. It can also be used to input a value in the variable and process the data stored in the variable.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int a, b, s, *p1, *p2;
  p1 = &a;
  p2 = &b;
  cout<<"Enter an integer: ";
  cin>>*p1;
  cout<<"Enter an integer: ";
  cin>>*p2;
  s = *p1 + *p2;
```

Output:
Enter an integer: 25
Enter an integer: 50
25 + 50 = 75

Freebooks.pk

```
    cout<<*p1<<" + "<<*p2<<" = "<<s<<endl;
    getch();
}
```

How above Example Works?

The above program declares three integer variables and two pointers. The pointer **p1** refers to the memory address of **a** and **p2** refers to **b**. The program uses indirection operator '*' with pointers to access the memory addresses of integer variables. It inputs integer values, adds the values and displays the result.

11.2.4 Pointer Initialization

The process of assigning a memory address to a pointer at the time of declaration is called point initialization. C++ does not initialize variables automatically. Therefore, a pointer variable should be initialized so that it may not point to anything invalid. The pointer can be intialized to any valid memory address. It can also be intialized to a NULL or 0 value.

Syntax

The syntax to intialize pointers is as follows:

```
DataType *P = &Variable;
```

DataType	It is the type of variable pointed by the pointer variable.
***P**	It indicates that the pointer variable to be initialized.
&	It is the address operator that is used to access memory address of a variable.
Variable	It is the name of variable whose memory address is assigned to the pointer.

Example

```
int n = 100;
int *p1 = &n;
int *p2 = NULL;
```

The pointer **p1** is initialized to the memory address of variable **n**. The pointer **p2** is initialized to **NULL**.

Example

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int a;
  int *ptr = &a;
  cout<<"Enter an integer: ";
  cin>>*ptr;
  cout<<"You entered "<<*ptr;
  getch();
}
```

Output:
Enter an integer: 100
You entered 100

11.3 Operations on Pointers

The arithmetical operations on pointers work differently than normal integer data types. Only addition and subtraction operations can be performed on pointers. The effect of both addition and subtraction depends on the size of the data type of the pointer.

Freebooks.pk

11.3.1 Pointer Addition

The addition operation on pointer is used to move the pointer reference forward in the memory. The change of memory address referenced by the pointer depends on the data type of pointer. If the increment operator is used with an integer pointer, it will change the reference by 2 bytes. If the increment operator is used with a character pointer, it will change the reference by 1 bytes.

Example

Suppose there are three pointers as follows:

```
char *pChar;
short *pShort;
long *pLong;
```

Suppose the above pointers are pointing to the memory locations 1000, 2000 and 3000 respectively. The use of increment operator on the pointer will change the reference of the pointers. The result of increment operator on each of the pointers is as follows:

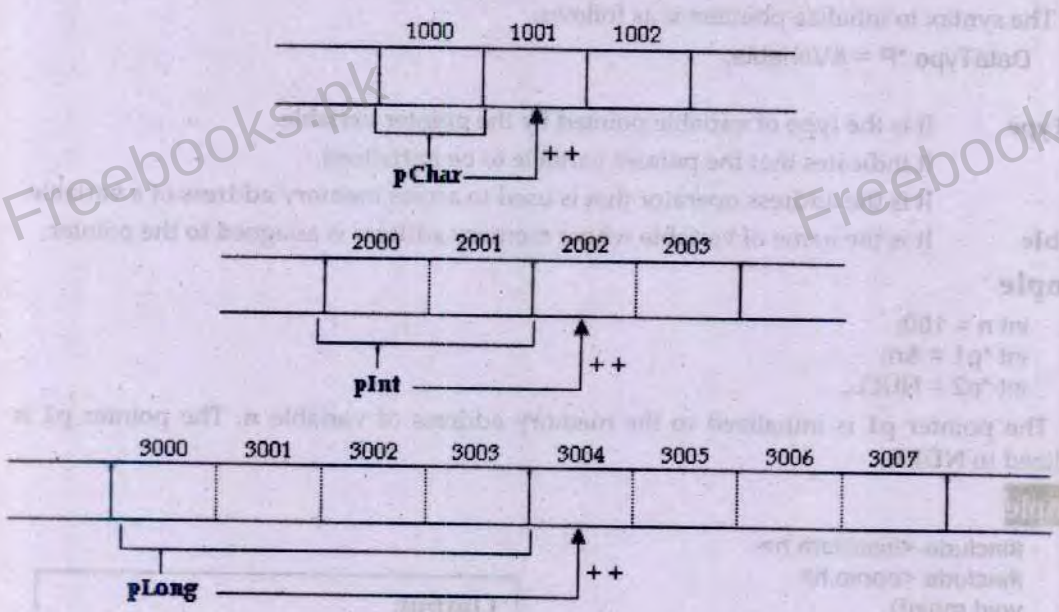


Figure 11.1: The result of increment operator on pointers

The above figure shows that the increment operator works differently on each pointer. The type of **pChar** pointer is **char**. The increment operator moves the refernece of **pChar** by 1 byte. The type of **pInt** pointer is **int**. The increment operator moves the refernece of **pInt** by 2 bytes. The type of **pLong** pointer is **long**. The increment operator moves the refernece of **pLong** by 4 bytes.

11.3.2 Pointer Subtraction

The subtraction operation on pointer is used to move the pointer reference backward in the memory. The change of memory address referenced by the pointer depends on the data type of pointer. If the decrement operator is used with an integer pointer, it will change the reference by 2 bytes. If the decrement operator is used with a character pointer, it will change the reference by 1 bytes.

11.4 Pointers and Arrays

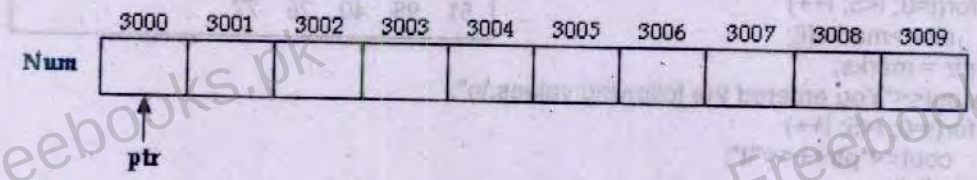
The pointers can also be used with arrays. An array is a collection of many elements of same type. All elements of the array are stored at consecutive memory locations. A pointer can access all elements of an array if the address of first element is assigned to it. The name of array represents the address of its first element. The address of first element can be assigned to a pointer by assigning the name of the array to pointer. The pointer then can access the remaining elements as they are stored consecutively in the memory.

Example

The following example uses a pointer to access an array:

```
int Num[10];
int * ptr;
ptr = Num;
```

The first statement declares an array of integers **Num** with 20 elements. The second statement declares a pointer variable **ptr** of type integer. The last statement assigns the address of first element of **Num** to pointer **ptr** as follows:



11.4.1 Accessing Array Elements with Pointers

The array elements can be accessed with pointers by moving the pointer to the desired element. The contents of an array elements can be accessed using dereference operator '*'. The pointer reference can be moved forward and backward by using increment operator ++ and decrement operators – –.

Example

The following example displays the values of first two array elements using pointer:

```
int Num[5] = {10, 20, 30, 40, 50};
int * ptr = Num;
cout<<*ptr;
ptr++;
cout<<*ptr;
```

The first line declares an array **Num** and initializes it with five values. The second declares a pointer **ptr** and initializes it to the first element of the array **Num**. The third statement displays the value of first element that is 10. The next statement increments the pointer reference of the array. The pointer now refers to the second element. The last statement then displays the value of second element that is 20.

The array elements can also be accessed without moving the pointer reference permanently as follows:

```
cout<<*ptr;
cout<<*(ptr+1);
cout<<*(ptr+2);
```

Freebooks.pk

The first statement displays the value of first element as the pointer refers to the first element of the array. The second statement displays the value of second element. The pointer refers to the first element. The 'ptr+1' adds two bytes to that address resulting the address of second element. Here, 1 represents one byte. Similarly, the last statement displays the value of third element. Here, the value 2 represents two bytes. Two bytes are added to the first element of the array that results in the third element of the array.

Program 11.2

Write a program to input five integers in an array and display them using a pointer.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int marks[5], i;
  int *ptr;
  cout<<"Enter five marks: ";
  for(i=0; i<5; i++)
    cin>>marks[i];
  ptr = marks;
  cout<<"You entered the following values:\n";
  for(i=0; i<5; i++)
    cout<<*ptr++<<"\t";
  getch();
}
```

Output:

```
Enter five marks: 51
98
40
26
77
You entered the following values:
51   98   40   26   77
```

Program 11.3

Write a program that inputs five floating-point values in an array and displays the values in reverse order.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  float arr[5], *ptr;
  int i;
  cout<<"Enter five floating-point values: ";
  for(i=0; i<5; i++)
    cin>>arr[i];
  ptr = &arr[4];
  cout<<"The values in reverse order: \n";
  for(i=0; i<5; i++)
    cout<<*ptr--<<endl;
  getch();
}
```

Output:

```
Enter five floating-point values:
1.1
2.2
3.3
4.4
5.5
The values in reverse order:
5.5
4.4
3.3
2.2
1.1
```

How above Program Works?

The above program inputs five values in an array. It stores the address of last element in a pointer **ptr** using the following statement:

```
ptr = &arr[4];
```

Freebooks.pk

The name of array always refers to its first element. The address of operator & is not used with the name of array. However, arr[4] is not the name of array. It refers to the last element of the array. Therefore, the address of operator is used to access its address. Finally, the loop displays the value of the array element referred by **ptr** and decrements the pointer to the previous element by using decrement operator.

11.5 Pointers and Strings

A string is an array of characters. A pointer of type **char** can refer to a string. The **char** pointer can be used in different ways to refer to a string.

```
char name[] = "Usman Khalil";
cha *ptr = name;
```

The first statement declares and initializes a string. The second statement declares a pointer variable and initializes it to the string **name**.

```
char *ptr = "Usman Khalil";
```

The above statement declares and initializes a string in memory and stores its address in the pointer **ptr**. The string is stored in an unnamed memory location.

Program 11.4

Write a program that inputs a string value from the user and displays it using pointer.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char name[20], *ptr;
  cout<<"Enter your name: ";
  cin.get(name, 20);
  ptr = name;
  cout<<"You name is "<<ptr<<endl;
  getch();
}
```

Output:
Enter your name: Muhammad Abdullah
Your name is Muhammad Abdullah

How above Program Works?

The above program declares an array of characters **name** and a pointer **ptr**. It inputs name from the user and stores it in the array. It assigns the address of array to the pointer and then displays the string using that pointer. The pointer displays the values stored in each element of the array **name** until it finds the null character **\0**.

M	u	h	a	m	m	a	d		A	b	d	u	l	l	a	h	\0		

Program 11.5

Write a program that declares and initializes a string. It inputs a character from the user and searches the character in the array.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char str[] = "hello programming";
  char ch, *ptr, s;
```

Output:
Enter any character: p
The character is found in the array.

Freebooks.pk

```
    s = 'n';
    ptr = str;
    cout<<"Enter any character to find: ";
    cin>>ch;
    while(*ptr++ != '\0')
      if(*ptr == ch)
        s = 'y';
    if(s == 'y')
      cout<<"The character is found in the array."<<endl;
    else
      cout<<"The character is not in the array."<<endl;
    getch();
}
```

11.6 Array of Pointers

An array of pointers is an array in which each element is a pointer. Each element in the array can store a memory address. The array can store the memory addresses of different objects of same type.

Example

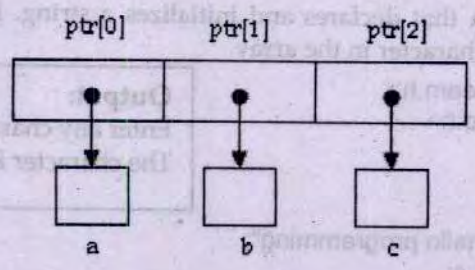
```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int *ptr[3], a, b, c;
  int i;
  ptr[0] = &a;
  ptr[1] = &b;
  ptr[2] = &c;
  cout<<"Enter three integers: "<<endl;
  cin>>a>>b>>c;
  cout<<"You entered the following values:\n";
  for(i=0; i<3; i++)
    cout<<*ptr[i]<<endl;
  getch();
}
```

Output:

```
Enter three integers:
10
20
30
You enter the following values:
10
20
30
```

How above Example Works?

The above program declares an array of integer pointers and three integer variables. Each element of pointer array refers to different variable. The program inputs the values in the variables and displays them using array of pointers.



Freebooks.pk

11.7 Pointers and Functions

A parameter can be passed to function using pointers. The address of actual parameter is passed to the formal parameter if the formal parameters are defined as pointers. It is similar to passing parameters to a function by reference.

The main difference is that the formal parameter in pass by reference is an alias of the actual parameters. On the other hand, the formal parametr in pass by pointer is a pointer variable that stores the memory address of actual parameters. However, any change made in formal parametr by function actually changes the value of actual parameter in both cases.

Program 11.6

Write a program that inputs two integers and passes them to a function using pointers. The function exchanges the values and the program finally displays the values.

```
#include <iostream.h>
#include <conio.h>
void exchange(int*,int*);
void main()
{
  clrscr();
  int n1, n2;
  cout<<"Enter two integers: ";
  cin>>n1>>n2;
  cout<<"Values before swapping: \n";
  cout<<"n1 = "<<n1<<endl;
  cout<<"n2 = "<<n2<<endl;
  exchange(&n1, &n2);
  cout<<"Values after swapping: \n";
  cout<<"n1 = "<<n1<<endl;
  cout<<"n2 = "<<n2<<endl;
  getch();
}
void exchange(int *m, int *n)
{
  int temp;
  temp = *m;
  *m = *n;
  *n = temp;
}
```

Output:
Enter two integers: 10 20
Values before swapping:
a = 10
b = 20
Values after swapping:
a = 20
b = 10

Program 11.7

Write a program that declares an integer in main() function. It also declares two functions. One function inputs a value in the variable defined in the main() function. The second function doubles the value of that variable.

```
#include <iostream.h>
#include <conio.h>
void get(int*);
void dbl(int*);
void main()
{
  clrscr();
  int num;
```

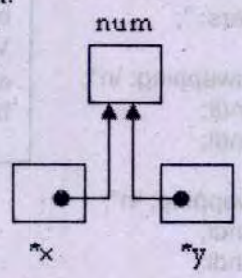
Freebooks.pk

```
    get(&num);
    cout<<"You entered "<<num<<endl;
    dbl(&num);
    cout<<"It's double is "<<num<<endl;
    getch();
}
void get(int *x)
{
    cout<<"Enter an integer: ";
    cin>>*x;
}
void dbl(int *y)
{
    *y = *y * 2;
}
```

Output:
Enter an integers: 50
You entered 50
It's double is 100

How above Program Works?

The above program declares an integer variable **num** and two functions. It passes the variable to both function using pointers. The first function inputs the value and second function doubles its value. The formal parameters of both functions actually refer to the same memory address of the variable **num**.



11.8 Pointers and Structures

The pointers can refer to a structure variable in the same way as simple variables. However, the dot operator is not used to access the members of a structure when pointers are used to refer to a structure variable. The **selection operator ->** is used instead of dot operator to access the structure member. It consists of **minus** sign and **greater than** sign.

Program 11.8

Write a program that declares a structure to store the record of a book. It defines a structure variable, inputs the values and displays them using pointer.

```
#include <iostream.h>
#include <conio.h>
struct Book
{
    char author[30];
    int pages;
    int price;
};
void main()
{
    clrscr();
    Book rec, *ptr;
```

```
    ptr = &rec;
    cout<<"Enter author name: ";
    cin.get(ptr->author, 30);
    cout<<"Enter pages: ";
    cin>>ptr->pages;
    cout<<"Enter price: ";
    cin>>ptr->price;
    cout<<"Author: "<<ptr->author<<endl;
    cout<<"Pages: "<<ptr->pages<<endl;
    cout<<"Price: "<<ptr->price<<endl;
    getch();
}
```

Output:
Enter author name: Haroon Yahya
Enter pages: 490
Enter price: 350
Author: Haroon Yahya
Pages: 490
Price: 350

11.8.1 Passing Structure to Function using Pointers

A structure variable can be passed to function as parameter using pointer. The process is same as passing a simple variable to a function using pointer.

Program 11.9

Write a program that decalres a structure to store the record of a student. It declares a structure variable. It passes the structure variable to a function using pointer to input data. It again passes it to another function using pointer to display the data.

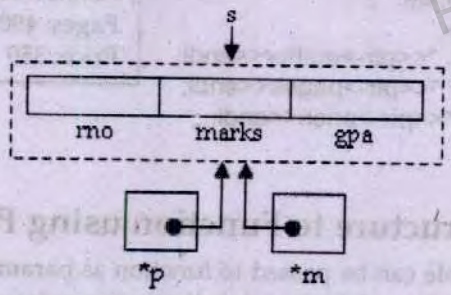
```
#include <iostream.h>
#include <conio.h>
struct Student
{
  int rno, marks;
  float gpa;
};
void input(Student*);
void output(Student*);
void main()
{
  Student s;
  input(&s);
  output(&s);
  getch();
}
void input(Student *p)
{
  cout<<"Enter roll no: ";
  cin>>p->rno;
  cout<<"Enter marks: ";
  cin>>p->marks;
  cout<<"Enter gpa: ";
  cin>>p->gpa;
}
void output(Student *m)
{
  cout<<"Roll no: "<<m->rno<<endl;
  cout<<"Marks: "<<m->marks<<endl;
  cout<<"GPA: "<<m->gpa<<endl;
}
```

Output:
Enter roll no: 10
Enter marks: 922
Enter gpa: 3.55
Roll No: 10
Marks: 922
GPA: 3.55

Freebooks.pk

How above Program Works?

The above program declares a structure variable s and two functions. It passes the variable to both function using pointers. The first function inputs the record and second function displays it. The formal parameters of both functions actually refer to the same memory address of the variable s.



Program 11.10

Write a program that decalres a structure to store hours, minutes and seconds. It declares a structure variable. It inputs number of seconds from the user and passes seconds to function by value and structure variable using pointer. The function converts the seconds into hours, minutes and seconds and stores it in the structure variable. The main() function finally displays the time on screen.

```
#include<iostream.h>
#include<conio.h>
struct time
{
 int hours;
 int minutes;
 int seconds;
} time1;
void main()
{
 long sec;
 clrscr();
 void convert(time*, long);
 cout<<"Enter total number of seconds: ";
 cin>>sec;
 convert(&time1,sec);
 cout<<"\nThe time in hh mm ss format is:";
 cout<<"\nHours: "<<time1.hours;
 cout<<"\nMinutes: "<<time1.minutes;
 cout<<"\nSeconds: "<<time1.seconds;
 getch();
}
void convert(time* temp, long sec)
{
 temp->seconds = sec % 60;
 long x = sec/60;
 temp->minutes = x % 60;
 temp->hours = x / 60;
}
```

Output:
Enter total number of seconds: 5300
Hours: 1
Minutes: 28
Seconds: 20

Freebooks.pk

11.9 Memory Management with Pointers

The process of allocating and deallocating memory is known as **memory management**. The operating system sets up different areas of memory according to the requirement when a program starts execution. Some important areas in the memory are as follows:

- **Global name space:** It is used to store global variables.
- **Registers:** These are special memory units built into CPU.
- **Code space:** It contains the program code.
- **Stack:** It contains local variables and function parameters. It is also called **static memory**.
- **Free store:** It is the remaining memory. It is also known as **heap** or **dynamic memory**. It is used for dynamically allocating memory blocks during program execution.

11.9.1 Dynamic Variables

A variable that is created during program execution is called **dynamic variable**. The dynamic variables can be created in C++ using pointers. C++ provides two operators **new** and **delete** operators for dynamic variables. The **new** operator is used to create dynamic variables and **delete** operator is used to delete dynamic variables during program execution.

11.9.2 The new Operator

The **new** operator is used to allocate memory dynamically. It is followed by the type of object for which the memory is to be allocated. The compiler allocates the amount of memory according to the type of object. The **new** operator returns a memory address. The returned address must be assigned to a pointer. The pointer is then used to access the memory location and process the values stored in that address. The **new** operator can be used to create simple variable, an object or an array of objects.

Syntax

The following syntax is used to create one variable:

```
new Datatype;
```

The following syntax is used to create an array dynamically:

```
new Datatype[length];
```

Example

```
int *ptr;
ptr = new int;
```

The first statement declares an integer pointer variable. The second statement uses the **new** operator to create an integer variable in the memory dynamically. The address of the allocated memory is stored in the pointer **ptr**.

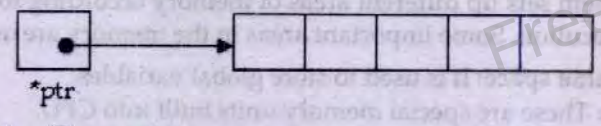
The above figure shows that the newly created memory has no name. It can only be accessed using the pointer.

Example

```
int *ptr;
ptr = new int[5];
```

Freebooks.pk

The first statement declares an integer pointer variable. The second statement uses the **new** operator to create an array of integers in the memory dynamically. The address of first element of the array is stored in the pointer **ptr**.



11.9.3 The delete Operator

The **delete** operator deallocates the memory and returns the allocated memory back to the free store. It takes a pointer as parameter and releases the memory referred by the pointer. The dynamically created object should be deleted when it is nor more required.

The pointer declared in a function is treated as a local variable. The pointer goes out of scope and is lost when the control exists from the function in which the pointer is declared. However, the memory allocated with **new** operator is not freed automatically. That memory becomes unavailable. This situation is known as **memory leak**. This memory canot be recovered until the program ends.

Syntax

The following syntax is used to delete one variable dynamically:

```
delete variable;
```

The following syntax is used to delete an array dynamically:

```
delete[] variable;
```

Example

```
delete ptr;
```

The above statement will deallocate the memory referred by the pointer **ptr**. The use of delete operator is slightly different when it refers to an array. The following statement is used if the pointer ptr refers to an array:

```
delete [] ptr;
```

Program 11.11

Write a program that uses new operator to create an integer, inputs value in it and then displays that value.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  int *ptr;
  ptr = new int;
  cout<<"Enter an integer: ";
  cin>>*ptr;
  cout<<"You entered "<<*ptr<<endl;
  cout<<"It is stored at "<<ptr<<endl;
  delete ptr;
  getch();
}
```

Output:
Enter an integer: 50
You entered 50
It is stored at 0x8fad0ede

Freebooks.pk

Program 11.12

Write a program that inputs length from the user and declares an array of integers of the length specified by the user. The program then inputs values in array and displays them.

```
#include <iostream.h>
#include <conio.h>
void get(int*, int);
void main()
{
  clrscr();
  int n, j, *ptr;
  cout<<"Enter the length of array: ";
  cin>>n;
  ptr = new int[n];
  get(ptr, n);
  cout<<"You entered the following values:\n";
  for(j=0; j<n; j++)
    cout<<*ptr++<<endl;
  delete[] ptr;
  getch();
}
void get(int *p, int l)
{
  int i;
  for(i=0; i<l; i++)
  {
    cout<<"Enter an integer: ";
    cin>>*p++;
  }
}
```

How above Program Works?

The above program declares a pointer variable **ptr**. It inputs the length of array and create an array of integers dynamically according to the given length. The program passes the address of array and its length to the function. The loop in the function inputs values in the array. The loop is executed according to the length of the array. Finally, the program displays the values in the array and deletes the array using **delete** operator.

Freebooks.pk

Programming Exercise

1. Write a program that sorts an array of integers using pointers.
2. Write a program that inputs two integers and passes them to a function using pointers. The function swaps the values of both integers. The main() function should display the values before and after calling the function.
3. Write a program that declares five integer variables a, b, c, d and e. It also declares an array of pointer with five elements. The first element refers to a, the second element refers to b and so on. The program should use the array to input values in the variables and then display the maximum value.
4. Write a program that declares structure to store account ID and amount. It inputs the number of account holders from the user and creates a dynamic array of structures to store the records of accounts. The program should declare two functions i.e. one for getting input from the user and the other for showing records to the user.
5. Write a program that inputs the number of students in a class from user. It then declares a dynamic array with same number of elements. It inputs the marks of students and finally displays the average marks of the whole class.
6. Write a program to swap two values by passing pointers as arguments to function.
7. Write a program to input any integer value and then to find out if the given value is a prime number or not by passing a number to a function as pointer argument.

Exercise Questions

Q.1. What is reference operator?

The actual address of the variable can be displayed by using reference operator &. It is also known as address operator. The reference operator is used to access the memory address of a variable.

Q.2. What is a pointer?

A pointer is a variable that is used to store a memory address. The reference operator is used to access the memory address of a variable and store it in a pointer.

Q.3. What is 'void' pointer?

The type of pointer variable and the type of variable it refers must be same. It restricts the use of a pointer variable to specific type of variables. A variable can store the address of any type of variable if the pointer is declared as void. The keyword 'void' is used as the data type of the pointer as follows:void *p;

Q.4. What is dereference operator?

The dereference operator is used to access the value of the variable whose memory address is stored in a pointer. It is denoted by an asterisk '*'. It is also known as indirection operator. The dereference operator can also be used to input a value in the variable and process the data stored in the variable.

Q.5. What type of operations performed on pointers?

Only addition and subtraction operations can be performed on pointers. The arithmetic operations on pointers work differently than normal integer data types. The effect of both addition and subtraction depends on the size of the data type of the pointer.

Q.6. How can a pointer be used with arrays?

An array is a collection of many elements of same type. All elements of the array are stored at consecutive memory locations. A pointer can access all elements of an array if the address of first element is assigned to it. The name of array represents the address of its first element. The address of first element can be assigned to a pointer by assigning the name of the array to pointer.

Q.7. How an array element is accessed with pointer?

The array elements can be accessed with pointers by moving the pointer to the desired element. The contents of an array elements can be accessed using dereference operator '*'. The pointer reference can be moved forward and backward by using increment operator ++ and decrement operators - -.

Q.8. How can a pointer be used with structures?

The pointers can refer to a structure variable in the same way as simple variables. However, the dot operator is not used to access the members of a structure when pointers are used to refer to a structure variable. The selection operator -> is used instead of dot operator to access the structure member. It consists of minus sign and greater than sign.

Q.9. What is the difference between new and delete operator?

The new operator is used to allocate memory dynamically. The compiler allocates the amount of memory according to the type of object. The new operator returns a memory address that is assigned to a pointer. The delete operator deallocates the memory and returns the allocated memory back to the free store. It takes a pointer as parameter and releases the memory referred by the pointer.

Q.10. Given the following lines of codes:

```
int ival = 1024;
int ival2 = 2048;
int* pi1 = &ival;
int* pi2 = &ival2;
```

Are the following statements legal or illegal?

a. pi2 = *pi1;
b. ival = pi1;
c. pi3 = &pi2;

Answer:
a. illegal
b. illegal
c. legal

Q.11. Given the following declarations:

```
char c;
double d;
char *pc;
double *pd;
```

Indicate whether the following statements are correct or incorrect?

a. c = C;	Incorrect
b. pc = &c;	Correct
c. d = 23.9;	Correct
d. pd = &d;	Correct

Q.12. Show the output from the following program segment:

```
int A[3][4] = {{1, 2, 3, 4},{5, 6, 7, 8},{9, 10, 11, 12}};
int (*p)[4] = &A[1];
int (*q)[4] = A;
cout << (*p)[0] << " " << (*p)[2] << endl;
cout << (*q)[0] << " " << (*q)[1] << endl;
cout << *(*(A+2)+1) << " " << *(*(A)+1) << endl;
```

Answer:
5 7
1 2
10 2

Q.13. Show the output from the following program segment:

```
int num[5] = {3, 4, 6, 2, 1};
int *p = num;
int *q = num + 2;
int *r = &num[1];
cout << num[2] << " " << *(num+2) << endl;
cout << *p << " " << *(p+1) << endl;
cout << *q << " " << *(q+1) << endl;
cout << *r << " " << *(r+1) << endl;
```

Answer:
6 6
3 4
6 2
4 6

Q.14. What is the output of the following program?

```
int a=2, b=3, c=4;
int *x = &a;
int *y;
```

Freebooks.pk

```
*x = 11;
y = &c;
c++;
x = &b;
b--;
cout << a << " "<< b << endl;
cout << *x << " " << *y << endl;
```

Answer
11 2
2 5

Q.15. What is the output of the following program?

```
int a[]={1,2,3,4,5,6,7,8,9};
int *b;
int *c = &(a[1]);
b=a;
b[2] = 0;
cout << b[1] << c[0] << c[1] << a[2] << endl;
```

Answer
2200

Q.16. What is the output of the following program?

```
int a[3] = {1, 3, 5 }, b = 23, c = 48;
int *p1 = a,
*p2 = &b,
*p3 = 0;
p1++;
(*p1)++;
p3 = p2;
*p3 = p1[1];
for(int i = 0; i < 3; i++)
        cout << a[i];
cout << endl;
cout << b << endl;
```

Answer:
145
5

Q.17. What is the output of the following program?

```
void swap(int* left, int* right) {
    int* p;
    p = left;
    left = right;
    right = p;
}
void main( )
{
    int a = 3, b = 5;
    swap(&a, &b);
    cout << a << " " << b << endl;
}
```

Answer
3 5

Q.18. Write the code to declare a struct type person with two fields, name and age, and a pointer called aPointer to a struct type variable.

```
struct person
{
        char name;
        int age;
};
person *aPointer;
```

Q.19. Write declaration of a pointer called ptr to store memory address of integer variable.

Answer: int *ptr;

Q.20. Write a command that assigns a memory address for a variable of type integer to an integer pointer called ptr (already defined).

Answer: ptr = new int;

Freebooks.pk

Multiple Choice

Freebooks.pk

1. **Which statement below is true about pointers?**
   a. A pointer is a keyword b. A pointer is a datatype
   A pointer holds a hexadecimal address d. Both b and c are true
2. **Which of the following can be used as pointers?**
   a. Array names b. Numeric constants c. Punctuation marks d. None
3. **After the following statements are executed, what can be said about variables x and p?**
   char x = 'M';
   char* p = &x;
   *p = 'W';
   a. p holds the value 'M'.
   b. p holds the value 'W'.
   c. p points to the variable x, which has the value 'M'
   d. p points to the variable x, which has the value 'W'
4. **Which of the following statements is not valid ?**
   a. int p = &total; b. int p = int *total;
   c. float total = &p2; d. All of these are invalid
5. **If you assign int x = 0.75; what actually will be stored in memory?**
   a. 1 b.0
   c. 75 d. The compiler will issue an error.
6. **The C++ operator _____ returns memory to the system for reuse.**
   a. Null b. Delete c. Remove d. All
7. **If p1 and p2 are pointers to two different variables v1 and v2 of the same type, with p1 pointing to v1 and p2 pointing to v2, then which of these statements is correct?**
   a. *p1 = *p2; makes the value of p1 equal to the value of p2.
   b. *p1 = *p2; makes the value of v2 equal to the value of v1.
   c. *p1 = *p2; makes the value of p2 equal to the value of p1.
   d. *p1 = *p2; makes the value of v1 equal to the value of v2.
8. **A pointer variable is designed to store**
   a. floating-point b. Memory address. d. An integer d. None
9. **If double *p_total = &total; what will be printed by the statement: Cout<<&p_total**
   a. Address of total b. Value of total c. Value of p_total d. Address of p_total
10. **Suppose you have a pointer variable p and you execute the statement delete p;. What happens?**
    a. p is deleted from the heap.
    b. p is deleted from the stack.
    c. The variable which p points to is deleted but p still remains.
    d. Both the variable which p points to, as well as p itself, are deleted.
11. **Which answer below is the most accurate for the statement: a=*ptr_m**
    a. The variable a must also be a pointer
    b. The value of variable that ptr_m is pointing to is being assigned to the variable a-
    c. The value of the pointer ptr_m is being assigned to the variable a.
    d. The address of the pointer ptr_m is being assigned to the variable a.
12. **A pointer variable may be initialized with**
    a. Any non-zero integer value. b. Any address in the memory.
    c. both a and b d. Neither a nor b
13. **Every byte in the computer's memory is assigned a unique**
    a. Pointer b. Address c. name d. None

14. What does the following statement do? double *total;

a. Declares a pointer variable named total. b. Initializes a variable named *total

c. Declares a double variable named num2. d. None

15. The statement int *p; has the same meaning as

a. int p; b. int* p; c. *int ptr; d. None

16. Given the code below, which code will correctly print the value of the variable dbl?

```
double *p_dbl;
double dbl;
p_dbl = &dbl;
*p_dbl = 6.5;
```

a. cout << p_dbl; b. cout << &dbl;

c. cout << *p_dbl; d. Two of these are correct

17. Which of the following assigns the address of value to the pointer p1?

a. *p1 = &value; p1 = value; p1 = &value; &p1 = *value;

18. C++ enables dynamic allocation of memory by providing the operator:

a. create b. allocate c. dynamic d. new

19. Dynamic memory allocation occurs

a. When a new variable is created by the compiler

b. when a new variable is created at runtime

c. Both a and b

d. None of these

20. What is the output of the following code fragment?

int v1 = 2; int v2 = -1; int* p1; int* p2;

p1 = &v1;

p2 = &v2;

p2 = p1;

cout << *p2 << endl;

a. 2 b. –1 c. –2 d. 1

21. The syntactically correct way (in standard C++) to declare and initialize, in a single step, a pointer to a previously declared variable s, of type double, is

a. double* p = &s; b. double &p = *s; c. double* p = s; d. double p = *s;

22. Suppose that p1 and p2 are both pointers to the same integer variable x; y is another integer variable. After the execution of which of the statements below is the pointer variable p1 considered a dangling pointer?

a. x = 0; b. p1 = &y; c. delete x; d. delete p2;

23. Given that p1 is an integer pointer variable, and a1 is an integer array, which of the following statements are not legal code?

a. p1= a1; b. cout << p1[0]; c. cin >> p1[0]; d. a1 = p1;

24. Which of the following statements correctly returns the memory from the dynamic array pointer p1 to the freestore?

a. delete [] p1; b. delete p1[]; c. delete *p1; d. delete p1;

25. If two pointer variables point to the same memory location, what happens when one of the pointers is freed?

a. The other pointer should be considered to be un-initialized

b. The other pointer still points to a valid memory address

c. If you attempt to free the other pointer a run-time error will occur.

d. A and C

26. Which of the following assigns the fifth element of an array pointed to by a pointer p1 to the value 6?

a. (p1+4) = 6 ; b. (*p1) + 4 = 6 ; c. *(p1+4) = 6 ; d. *(p1[4]) = 6 ;

Freebooks.pk

27. **Which of the following correctly reclaims an array of integers pointed at by a pointer p1?**
a. delete p1 ; b. delete *p1 ; c. delete [] p1 ; d. delete [] *p1 ;

28. **What is the value of pointer p after the following assignment?**
p = new char;
a. 0 b. x12 c. char d. cannot be determined

29. **What is the output of the following segment of code?**
```
int *p;
p = new int;
*p = 5;
cout << *p;
```
a. 0 b. 5 c. -5 d. None

30. **You may use a pointer to a structure as a**
a. structure member. b. function return type.
c. function parameter d. All

31. **After the following statements have been executed, which statements are true?**
```
char x = 'a' ;
char* p = &x ;
*p = 'b' ;
```
a. p now points to the value 'a' b. p holds the value 'b'
c. p points to the variable x d. p holds the value 'x'

Answers

1. d	2. b	3. d	4. d	5. b	6. b
7. d	8. b	9. d	10. c	11. b	12. b
13. b	14. a	15. b	16. c	17. c	18. d
19. b	20. a	21. a	22. d	23. d	24. a
25. d	26. c	27. c	28. d	29. b	30. d
31. c					

Fill in the Blanks

1. A memory cell that stores the address of a variable is called a(n) _____
2. Each variable is created at a unique location in memory known as its _____.
3. The actual address of the variable can be displayed by using _____.
4. A variable can store the address of any type of variable if the pointer is declared as____.
5. The process of assigning a memory address to a pointer at the time of declaration is called _____.
6. The pointer displays the values stored in each element of the array name until it finds the _____.
7. The address of _____parameter is passed to the formal parameter if the formal parameters are defined as pointers.
8. The process of allocating and deallocating memory is known as _____
9. The _____operator is used to allocate memory dynamically.
10. The _____operator deallocates the memory and returns the allocated memory back to the free store.

Freebooks.pk

Answers

1. Pointer	2. address	3. reference operator &.
4. void	5. point initialization	6. null character \0
7. actual	8. memory management	9. new
10. delete		

True/ False

1. Pointer is a construct that gives you more control of the computer CPU.
2. A pointer is the memory address of a variable.
3. There must be asteric before each of the pointer variable.
4. A pointer is a value of type int or any other numeric type.
5. The * operator is also called dereferencing operator.
6. You cannot assign the value of one pointer variable to another pointer variable.
7. The & operator is simply called the address of operator.
8. The new operator creates a new dynamic variable of a specified type and returns to pointer that points to this new variable.
9. Dereferencing allows access to the pointer's value.
10. A pointer is an object that contains the location of another object.
11. The null address is the literal 0.
12. The address of an object can be determined using the * operator.
13. It is possible for a structure to contain as a member a pointer to its own structure type.

Answers

1. F	2. T	3. T	4. F
5. T	6. F	7. T	8. T
9. F	10. T	11. T	12. F
13. T			

Freebooks.pk

CHAPTER 12

STRING HANDLING

Chapter Overview

12.1 String
- 12.1.1 String Declaration
- 12.1.2 String Initialization

12.2 String Input
- 12.2.1 The cin Object
- 12.2.2 cin.getline()
- 12.2.3 cin.get()

12.3 Array of Strings
- 12.3.1 Initializing Array of Strings
- 12.3.2 Input/Output with Array of Strings

12.4 String Functions (string.h)
- 12.4.1 memchr()
- 12.4.2 memcmp()
- 12.4.3 memcpy()
- 12.4.4 memmove()
- 12.4.5 memset()
- 12.4.6 strcat()
- 12.4.7 strncat()
- 12.4.8 strchr()
- 12.4.9 strrchr()
- 12.4.10 strcmp()
- 12.4.11 stricmp()
- 12.4.12 strncmp()
- 12.4.13 strcoll()
- 12.4.14 strcpy()
- 12.4.15 strncpy()
- 12.4.16 strcspn()
- 12.4.17 strlen()
- 12.4.18 strstr()
- 12.4.19 strpbrk()
- 12.4.20 strspn()
- 12.4.21 strtok()
- 12.4.22 strerror()
- 12.4.23 strxfrm()
- 12.4.24 strrev()
- 12.4.25 strset()
- 12.4.26 strnset()
- 12.4.27 strlwr()
- 12.4.28 strupr()
- 12.4.29 strdup()
- 12.4.30 stpcpy()

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

12.1 String

A collection of characters written in double quotations is called string or string constant. It may consist of any alphabetic characters, digits and special symbols.

A string is stored as an array of characters. The array is terminated by a special symbol known as **null character**. It is denoted by the escape sequence \0 and is used to indicate the end of string. It consists of back slash and zero.

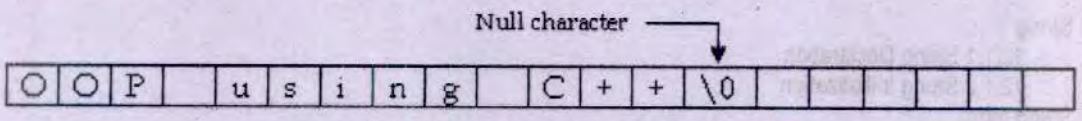


Figure 12.1: Representation of string in memory

The values stored of an array of characters can be manipulated individually using the index of the array. The user can input, process and displays the individual characters of string in the same way as an array.

Examples

Some examples of stings are as follows:

"Pakistan" "123" "99-Mall Road, Lahore"

12.1.1 String Declaration

C++ stores a string as an array of characters. An array is a group of contiguous memory location that can store same type of data.

Syntax

The syntax of declaring a string in C++ is as follows:

```
char array_name [length];
```

char It indicates the type of array used as string.

array_name It indicates the name of array.

length It indicates the number memory locations in the array.

Example

An example of string declaration is as follows:

```
char book[20];
```

The above statement declares a variable **book**. The value 20 in brackets indicates the length of string that can be stored in the variable. The above declaration allocates twenty memory locations and each memory location can store one character as follows:

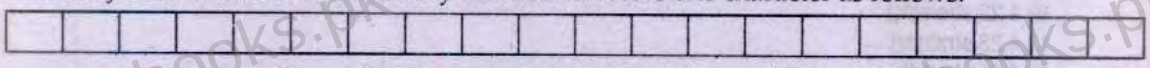


Figure 12.2: Memory allocation for string variable **book**

12.1.2 String Initialization

The syntax of initializing a string in C++ is as follows:

```
char array_name [length] = value;
```

char It indicates the type of array used as string.

array_name It indicates the name of array.

length It indicates the number memory locations in the array.

value It indicates the value initialized in string. It is enclosed in double quotes.

Freebooks.pk

Example

The string variable can be initialized with a string value as follows:

```
char book[20] = "OOP using C++";
```

The above statement will store the string as follows:

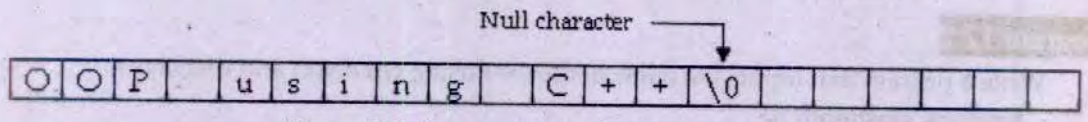


Figure 12.3: Representation of string in memory

The last character \0 in the memory is known as **null character**. It consists of back slash and zero. It indicates the end of string. It is added at the end of string automatically.

A string variable can also be declared without indicating the length as follows:

```
char name[] = "Pakistan";
```

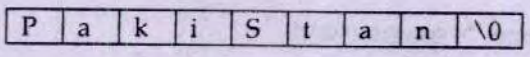


Figure 12.4: String variable **name**

The above statement automatically declares a string according to the length of value.

Program 12.1

Write a program that displays the value stored in a string variable **name**.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char name[ ] = "OOP using C++";
  clrscr();
  cout<<"Name is "<<name;
  getch();
}
```

Output:
Name is OOP using C++

12.2 String Input

A string value can be input from the user using different functions. C++ prvides the following functions for string input:

C++ Routine	Description
cin	General-purpose keyboard input
cin.getline()	Input a string of characters from keyboard
cin.get()	Input a single character from keyboard

Figure 12.5: C++ routines for string input

12.2.1 The 'cin' Object

The **cin** object is used to input a string value without any blank space in it. It does not support a string with spaces. If the user inputs a string value that contains any space, the value before the space will be used. The remaining string value will be ignored.

Freebooks.pk

Syntax

The syntax of getting input using **cin** object is as follows:

```
cin>>str;
```

>> It is the extraction operator that is used with **cin** object to input values.

str It indicates the name of string variable in which the value is to be stored.

Program 12.2

Write a prgram that inputs the name of the user using **cin** object and displays it.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char name[50];
  cout<<"Enter your name: ";
  cin>>name;
  cout<<"Your Name: "<<name<<endl;
  getch();
}
```

Output:
Enter your name: Usman Khalil
Your Name: Usman

How above Program Works?

The above program uses **cin** object to input the name of the user. The user inputs "Usman Khalil" in the above figure. The **cin** object cannot input a string with blank space. It only stores the value before space. The output displays only "Usman".

12.2.2 cin.getline()

The **getline()** function of **cin** object is used to input any string value including blank spaces. The user can input any type of string value.

Syntax

The syntax of using this function is as follows:

```
cin.getline(str, len);
```

str It indicates the name of string variable in which the value is to be stored.

len It indicates the length of string variable.

Program 12.3

Write a program that inputs the name of user using **cin.getline()** function and displays it on the screen.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char name[50];
  cout<<"Enter your name: ";
  cin.getline(name,50);
  cout<<"Your Name: "<<name<<endl;
  getch();
}
```

Output:
Enter your name: Usman Khalil
Your Name: Usman Khalil

Freebooks.pk

12.2.3 cin.get()

The **get()** function of **cin** object is used input a single character. The syntax of using this function is as follows:

```
cin.get(ch);
```

ch It indicates the name of character variable in which the value is to be stored.

Program 12.4

Write a program that inputs a character from the user using **cin.get()** function and displays it on screen.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char c;
  cout<<"Enter any character: ";
  cin.get(c);
  cout<<"You entered "<<c<<endl;
  getch();
}
```

Output:
Enter any character: *
You entered *

Program 12.5

Write a program that inputs a string from the user and displays its length.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char str[50];
  int i = 0;
  cout<<"Enter a string: ";
  cin.getline(str,50);
  while(str[i] != '\0')
    i++;
  cout<<"The length of string is "<<i<<endl;
  getch();
}
```

Output:
Enter a string: OOP using C ++
The length of string is 13

How above Program Works?

The above program inputs a string and counts its length. The **while** loop uses counter variable **i** to go through the string until the null character is found. The loop is terminated when the null character is found. The value of counter variable indicates the number of characters in the string.

Program 12.6

Write a program that inputs a string from the user and then copies it another string.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  char s1[50], s2[50];
```

Freebooks.pk

```
    int i = 0;
    cout<<"Enter a string: ";
    cin.getline(s1,50);
    while(s1[i] != '\0')
    {
      s2[i] = s1[i];
      i++;
    }
    s2[i] = '\0';
    cout<<"The value of s1: "<<s1<<endl;
    cout<<"The value of s2: "<<s2<<endl;
    getch();
  }
```

Program 12.7

Write a program that inputs a string from the user and counts the number of vowels in the sting.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char str[50];
  int i, v;
  i = v = 0;
  cout<<"Enter a string: ";
  cin.getline(str,50);
  while(str[i] != '\0')
  {
    switch(str[i])
    {
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u':
     v++;
    }
    i++;
  }
  cout<<"The string has "<<v<<" vowels."<<endl;
  getch();
}
```

Output:
Enter a string: object oriented programming
The string has 9 vowels in it.

Program 12.8

Write a program that inputs a sentence and displays the number of uppercase and lowercase consonants, uppercase and lowercase vowels in sentence.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
```

Freebooks.pk

```
char line[80];
int uc, lc, uv, lv;
uc = lc = uv = lv = 0;
cout << "Enter a sentence:\n";
cin.getline(line,80);
for(int x=0; line[x]!='\0';x++)
{
  if(line[x]=='A'||line[x]=='E'||line[x]=='I'||line[x]=='O'||line[x]=='U')
    uv++;
  else if(line[x]=='a'||line[x]=='e'||line[x]=='i'||line[x]=='o'||line[x]=='u')
    lv++;
  else if(line[x]>+65&&line[x]<=90)
    uc++;
  else if (line[x]>=97&&line[x]<=122)
    lc++;
}
cout << "Uppercase Consonants = " << uc << endl;
cout << "Lowercase Consonants = " << lc << endl;
cout << "Uppercase Vowels = " << uv << endl;
cout << "Lowercase Vowels = " << lv << "." << endl;
getch();
}
```

Output:
Enter a string: Object Oriented Programming
Uppercase consonants: 1
Lowercase consonants: 15
Uppercase vowels: 2
Lowercase vowels: 7

Program 12.9

Write a program that inputs first name and second name in two strings, concatenate both strings in third string and then displays it.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char first[30], second[30], full[60];
  int i, j;
  i = j = 0;
  cout<<"Enter your first name: ";
  cin>>first;
  cout<<"Enter your second name: ";
  cin>>second;
  while(first[i] != '\0')
  {
    full[i] = first[i];
    i++;
  }
  full[i++] = ' ';
  while(second[j] != '\0')
  {
    full[i] = second[j];
    i++;
    j++;
  }
  full[i] = '\0';
  cout<<"Your first name is "<<first<<endl;
```

Output:
Enter your first name: Usman
Enter your second name: Khalil
Your first name is Usman
Your second name is Khalil
Your full name is Usman Khalil

Freebooks.pk

```
    cout<<"Your second name is "<<second<<endl;
    cout<<"Your full name is "<<full<<endl;
    getch();
}
```

12.3 Array of Strings

An array of strings is actually a two dimensional array of characters. Each row of the array represents one string. Each character in an array of strings is stored in a separate index of a two dimensional array.

Syntax

The syntax of declaring an array of strings is as follows:

```
char str[rows][cols];
```

str	It indicates the name of two dimensional array.
rows	It indicates the number of rows in the array.
cols	It indicates the number of columns in each row of the array.

Example

```
char names[3][50];
```

The above example declares a two dimensional array of characters with three rows and fifty columns in each row. It can be considered as an array of three strings.

12.3.1 Initializing Array of Strings

An array of strings can be initialized in different ways. It can be initialized by assigning individual characters to each index in the array. It can also be initialized by assigning complete strings to each row in the array.

Example

The following example initializes an array of strings by assigning individual characters

```
char str[3][5] = { 'a', 'b', 'c', 'd', 'e',
                   'f', 'g', 'h', 'i', 'j',
                   'k', 'l', 'm', 'n', 'o' };
```

The above example declares a two dimensional array of characters and initializes it with the given values as follows:

	0	1	2	3	4
0	a	b	c	d	e
1	f	g	h	i	j
2	k	l	m	n	o

The above figure shows that the array contains fifteen characters. The characters in one row can be considered as single string. It means that the above array contains three strings. A string must be terminated by the null character. The above array does not contain any null character that may create some problem in the output. Another method of initializing an array of strings is as follows:

```
char str[3][10] = { "Ali",
                    "Abdullah",
                    "Usman" };
```

Freebooks.pk

The above example declares a two dimensional array of characters with three rows and ten columns in each row. There are three strings values on the right side of assignment operator. Each string is separated by comma. The string values are assigned to different rows in the array as follows:

	0	1	2	3	4	5	6	7	8	9
0	A	l	i	\0						
1	A	b	d	u	l	l	a	h	\0	
2	U	s	m	a	n	\0				

The above figure shows that each string is automatically terminated with the null character. The array contains three string values in three rows.

12.3.2 Input/Output with Array of Strings

Each row of a two-dimensional array of characters represents a complete string. The values can be input to an array of string by referencing the complete row of a two dimensional array of characters as follows:

```
char str[3][10];
cin>>str[0];          // represents the first row in the array
cin>>str[1];          // represents the second row in the array
cin>>str[2];          // represents the third row in the array
```

Similarly, the values can be displayed by referencing the complete row of a two dimensional array of characters as follows:

```
char str[3][10];
cout<<str[0];         // represents the first row in the array
cout<<str[1];         // represents the second row in the array
cout<<str[2];         // represents the third row in the array
```

Program 12.10

Write a program that inputs the names of five cities and then displays them.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char city[5][30];
  int i;
  for(i=0; i<5; i++)
  {
    cout<<"Enter city name: ";
    cin>>city[i];
  }
  for(i=0; i<5; i++)
    cout<<city[i]<<endl;
  getch();
}
```

Output:

```
Enter city name: Islamabad
Enter city name: Lahore
Enter city name: Karachi
Enter city name: Peshawar
Enter city name: Quetta
Islamabad
Lahore
Karachi
Peshawar
Quetta
```

Freebooks.pk

How above Program Works?

The above program declares a two-dimensional array of characters with five rows and thirty columns. It can store five string values. The program uses the row index to input five string values. Each value is stored in different row. For example, the value of counter variable i is 0 in first iteration. Therefore, the first string is stored in first row and so on. The second for loop displays the string values in the same way using only row index.

Program 12.11

Write a program that inputs the names of five countries. It only displays the countries whose name starts with a vowel.

```
#include <iostream.h>
#include <conio.h>
void main()
{
  clrscr();
  char country[5][30], ch;
  int i;
  for(i=0; i<5; i++)
  {
    cout<<"Enter country name: ";
    cin>>country[i];
  }
  for(i=0; i<5; i++)
  {
    ch = country[i][0];
    switch(ch)
    {
      case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U':
        cout<<country[i]<<endl;
    }
  }
  getch();
}
```

Output:

```
Enter country name: Pakistan
Enter country name: Iran
Enter country name: Bangladesh
Enter country name: England
Enter country name: Afghanistan
Iran
England
Afghanistan
```

12.4 String Functions (string.h)

The header file 'string.h' contains the built-in function that are used to process string values. Some important function defined in this header file are as follows:

12.4.1 memchr()

The **memchr()** function is used to search a byte with a particular value in buffer. The search continues for count bytes or until the value is found. It returns a pointer to the first location of the value in the buffer. The function returns **NULL** value if the value is not found.

Syntax: memchr(buffer, ch, size);

memchr	It is the name of the function.
buffer	It indicates the name of the buffer to be searched for the value.

Freebooks.pk

ch It indicates the value to be searched from the buffer.
size It indicates the size of the buffer in bytes.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char test[] = "Test String";
  char search;
  void *result;
  cout<<"Enter a single character to search:";
  cin>>search;
  cout<<"The test string is : ";
  puts(test);
  cout<<"memchr() is looking for "<<search<<" ... \n";
  result = memchr(test, search, 12);
  if (result != NULL)
        cout<<"The character found in the string.";
  else
        cout<<"The character not found in the string.";
  getch();
}
```

How above Program Works?

The above program initializes an array of characters **test** with a string value. It inputs a character and searches it in **test** buffer using memchr() function. The first parameter of the function is **test** in which the search will be performed. The second parameter **search** contains the character to be searched. The last parameter 12 indicates the size of the buffer **test** in bytes. The function returns a pointer to the byte that contains the value to be searched in the string. It returns NULL value if the character is not found in the string.

12.4.2 memcmp()

The **memcmp()** function is used to compare each successive byte referenced by first pointer with the corresponding byte referenced by second pointer. The comparison continues both pointers do not have same value or the number of specified bytes have been compared. The function returns an integer value as follows:

Returned Value	Description
Less than zero	It returns a value less than zero if the last byte compared in the buffer referenced by first pointer is less than the corresponding byte referenced by second pointer.
Zero	It returns zero if the last byte compared in the buffer referenced by first pointer is equal to the corresponding byte referenced by second pointer.
Greater than zero	It returns a value greater than zero if the last byte compared in the buffer referenced by first pointer is greater than the corresponding byte referenced by second pointer.

Freebooks.pk

Syntax: memcmp(buffer1, buffer2, size);

memcmp	It is the name of the function.
buffer1	It indicates the name of the first buffer.
buffer2	It indicates the name of the second buffer.
size	It indicates the size of the buffer in bytes.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
   char test1[] = "This is test string 1";
   char test2[] = "This is test string 2";
   int result;
   cout<<"Test string 1 is "<<test1<<endl;
   cout<<"Test string 2 is "<<test2<<endl;
   cout<<"Comparing test1 and test2 for 20 bytes...\n";
   result = memcmp(test1, test2, 20);
   if (result == 0)
         cout<<"test1 and test2 are same for 20 bytes."<<endl;
   else
         cout<<"test1 and test2 are not same for 20 bytes."<<endl;

   cout<<"Comparing 1 and 2 for 21 bytes...\n";
   result = memcmp(test1, test2, 21);
   if (result == 0)
         cout<<"test1 and test2 are same for 21 bytes."<<endl;
   else
         cout<<"test1 and test2 are not same for 21 bytes."<<endl;
}
```

How above Example Works?

The above program initializes two arrays of characters **test1** and **test2** with a string values. Each string consists of 21 characters taking 21 bytes in memory. The program compares these strings twice using memcmp() function. The first comaprison is limited to 20 bytes. It compares first 20 characters which are same. The result of first comparison is 0. The second comparison compares 21 bytes. It compares all 21 characters. The last characters of both strings are different. The result of second comparison is less than zero.

12.4.3 memcpy()

The **memcpy()** function is used to copy the number of specified characters from first buffer to second buffer and returns the first buffer. It is faster than memmove() function. It does not ensure that source bytes in overlapping region are copied before being overwritten.

Syntax: memcpy(buffer2, buffer1, size);

memcpy	It is the name of the function.
buffer2	It indicates the name of buffer in which characters are to be copied.
buffer1	It indicates the name of buffer whose value is to be copied to buffer2.
size	It indicates the size of the buffer in bytes.

Freebooks.pk

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char test1[] = "Sample String.";
  char test2[15];
  cout<<"The value of test1 is: ";
  puts(test1);
  cout<<"Copying test1 to test2...\n";
  memcpy (test2, test1, 15);
  cout<<"The value of test2 is: ";
  puts(test2);
  getch();
}
```

How above Example Works?

The above program initializes an array of characters **test1** with a string value. It copies the value of **test1** to **test2** using memcpy() function and then displays the value of **test2**.

12.4.4 memmove()

The **memmove()** function is also used to copy the number of specified characters from first buffer to second buffer and returns the first buffer. It is slower than memcpy() function. It can handle the overlapping moves.

Syntax: memmove(buffer2, buffer1, size);

memmove	It is the name of the function.
buffer2	It indicates the name of buffer in which characters are to be copied.
buffer1	It indicates the name of buffer whose value is to be copied to buffer2.
size	It indicates the size of the buffer in bytes.

12.4.5 memset()

The memset() function is also used to set the first count characters refernced by buffer to the specified value. It returns the buffer.

Syntax: memset(buffer, char, number);

memset	It is the name of the function.
buffer	It indicates the name of buffer in which characters are to be set.
char	It indicates the character to be set in the buffer.
number	It indicates the number for which the **char** is set in the buffer.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
```

Freebooks.pk

```
    clrscr();
    char buffer[15] = "A sample string";
    cout<<"buffer before memset: ";
    puts(buffer);
    memset (buffer, 'x', 8);
    cout<<"buffer after memset: ";
    puts(buffer);
    getch();
}
```

How above Program Works?

The above program initializes an array of characters **buffer** with a string value and displays its value. The memset() function copies the value 'x' in first 8 bytes of the buffer. The program finally displays the changed value of the **buffer** on the screen.

12.4.6 strcat()

The **strcat()** function is used to append a copy of one string to the end of second string. It also terminates the resulting string with null character. The string in which the value is copied should have enough space to hold the result.

Syntax: strcat(str1, str2);

strcat It is the name of the function.
str1 It indicates the name of string in which the value is to be appended.
str2 It indicates the name of string whose value is to be appended to str1.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char str1[50] = "The more you work...";
  char str2[50] = "The more you gain...";
  cout<<"String 1: ";
  puts(str1);
  cout<<"String 2: ";
  puts(str2);
  cout<<"Concating the strings...\n";
  strcat(str1,str2);
  cout<<"String 1 after concatenation: ";
  puts(str1);
  getch();
}
```

12.4.7 strncat()

The **strncat()** function is used to append the specified number of characters of one string to the end of second string. It also terminates the resulting string with null character. The string in which the value is copied should have enough space to hold the result.

Syntax: strncat(str1, str2, n);

strncat	It is the name of the function.
str1	It indicates the name of string in which the value is to be appended.
str2	It indicates the name of string whose value is to be appended to str1.
n	It indicates the number of characters to be appended.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char first[25] = "Hello";
  char second[] = " World of Programming";
  strncat(first, second, 6);
  cout<<first<<endl;
  getch();
}
```

12.4.8 strchr()

The **strchr()** function is used to find the first occurrence of a character in string and returns a pointer to this character. It retuns NULL value if the character is not found.

Syntax: strchr(str, ch);

strchr	It is the name of the function.
str	It indicates the string in which the character is searched.
ch	It indicates the character to be searched in the string.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char str[15] = "string example";
  char *res, ch;
  cout<<"Enter any character to search: ";
  cin>>ch;
  cout<<"Looking for "<<ch<<" in the string...\n";
  res = strchr(str, ch);
  if(res == NULL)
        cout<<ch<<" not found in the string.";
  else
        cout<<ch<<" found in the string at index "<<res -str;
  getch();
}
```

Freebooks.pk

12.4.9 strrchr()

The **strrchr()** function is used to find the last occurrence of a character in string and returns a pointer to this character. It retuns NULL value if the character is not found.

Syntax: strchr(str, ch);

strrchr It is the name of the function.
str It indicates the string in which the character is searched.
ch It indicates the character to be searched in the string.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  char string[20] = "This is a string";
  char *ptr, c = 'r';
  ptr = strrchr(string, c);
  if(ptr)
    cout<<"The character "<<c<<" is at position: "<<ptr-string<<endl;
  else
    cout<<"The character was not found."<<endl;
  getch();
}
```

Output:
The character r is at position 12

12.4.10 strcmp()

The **strcmp()** function is used to compare two strings character by character. The comparison is case sensitive. It returns integer value as follows:

Returned Value	Description
Less than zero	It returns a value less than zero if the first string is less than second string.
Zero	It returns zero if the first string is equal to second string.
Greater than zero	It returns a value greater than zero if the first string is greater than second string.

Syntax: strcmp(str1, str2);

strcmp It is the name of the function.
str1 It indicates the first string in comparison.
str2 It indicates the second string in comparison.

Program 12.12

Write a program that inputs two strings from the user and compares them using **strcmp** function. The program displays whether the first string is less than, equal to or greater than second string.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
```

Freebooks.pk

```
void main()
{
  char str1[50], str2[50];
  int r;
  clrscr();
  cout<<"Enter first string: ";
  cin.getline(str1,50);
  cout<<"Enter second string: ";
  cin.getline(str2,50);
  r = strcmp(str1, str2);
  if(r < 0)
    cout<<"String1 is less than string2.";
  else if(r == 0)
    cout<<"String1 is equal to string2.";
  else
    cout<<"String1 is greater than string2.";
  getch();
}
```

Output:
Enter first string: Pakistan
Enter second string: Zindabaad
String1 is less than string2.

12.4.11 stricmp()

The **stricmp()** function is used to compare two strings character by character. The comparison is case insensitive. It returns integer value as follows:

Returned Value	Description
Less than zero	It returns a value less than zero if the first string is less than second string.
Zero	It returns zero if the first string is equal to second string.
Greater than zero	It returns a value greater than zero if the first string is greater than second string.

Syntax: stricmp(str1, str2);

stricmp It is the name of the function.
str1 It indicates the first string in comparison.
str2 It indicates the second string in comparison.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char test1[] = "this is a test string";
  char test2[] = "THIS IS A TEST STRING";
  int result;
  cout<<"Test string 1 is: ";
  puts(test1);
  cout<<"Test string 2 is: ";
  puts(test2);
```

Output:
Test string 1 is: this is a test string
Test string 2 is: THIS IS A TEST STRING
Comparing string1 and string2 using strcmp()...
string 1 is greater than string 2.
Comparing string1 and string2 using stricmp()...
string 1 is equal to string 2.

```
    cout<<"Comparing string1 and string2 using strcmp()...\n";
    result = strcmp(test1, test2);
    if (result == 0)
        cout<<"string 1 is equal to string 2. \n";
    else if (result < 0)
        cout<<"string 1 is less than string 2. \n";
    else
        cout<<"string 1 is greater than string 2. \n";
    cout<<"Comparing 1 and 2 using stricmp()...\n";
    result = stricmp(test1, test2);
    if (result == 0)
        cout<<"string 1 is equal to string 2. \n";
    else if (result < 0)
        cout<<"string 1 is less than string 2. \n";
    else
        cout<<"string 1 is greater than string 2. \n";
    getch();
}
```

12.4.12 strncmp()

The **strncmp()** function is used to compare the specified number of characters in two strings. The comparison is case sensitive. It returns integer value as follows:

Returned Value	Description
Less than zero	It returns a value less than zero if the specified characters of first string are less than the specified characters in second string.
Zero	It returns zero if the specified characters of first string are equal to the specified characters in second string.
Greater than zero	It returns a value greater than zero if the specified characters of first string are greater than the specified characters in second string.

Syntax: strncmp(str1, str2, n);

strncmp It is the name of the function.
str1 It indicates the first string in comparison.
str2 It indicates the second string in comparison.
n It indicates the number of characters to be compared in each string.

Program 12.13

Write a program that inputs two strings from the user and compares them using **strncmp** function. It also inputs the number of characters to be compared. The program displays whether the first string is less than, equal to or greater than second string.

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char str1[50], str2[50];
  int ch, r;
```

Output:
Enter string1: Pakistan
Enter string2: My country
Enter number of characters to compare: 8
String1 is greater than String2.

```
    cout<<"Enter string1: ";
    cin.getline(str1,50);
    cout<<"Enter string2: ";
    cin.getline(str2,50);
    cout<<"Enter number of characters to compare: ";
    cin>>ch;
    r = strncmp(str1, str2, ch);
    if(r > 0)
      cout<<"String1 is greater than String2."<<endl;
    else if(r == 0)
      cout<<"String1 is equal to String2."<<endl;
    else
      cout<<"String1 is less than String2.";
    getch();
  }
```

12.4.13 strcoll()

The **strcoll()** function is used to compare two string values using the collating sequence specified by setlocale function. It returns a value indicating the relationship between strings.

Returned Value	Description
Less than zero	It returns a value less than zero if the first string is less than second string.
Zero	It returns zero if the first string is equal to second string.
Greater than zero	It returns a value greater than zero if the first string is greater than second string.

Syntax: strcoll(str1, str2);

strcoll It is the name of the function.
str1 It indicates the first string in comparison.
str2 It indicates the second string in comparison.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char two[] = "International";
  char one[] = "Borland";
  int check;
  check = strcoll(one, two);
  if(check == 0)
    cout<<"The strings are equal."<<endl;
  if(check < 0)
    cout<<one<<" comes before "<<two<<endl;
  if(check > 0)
    cout<<two<<" comes before "<<one<<endl;
  getch();
}
```

Output:
Borland comes before International

Freebooks.pk

12.4.14 strcpy()

The word 'strcpy' stands for **string copy**. The function is used to copy one string to another including the terminating null character.

Syntax: strcpy(str1, str2);

strcpy	It is the name of the function.
str1	It indicates the first string in which the string value is to be copied.
str2	It indicates the second string whose value is to be copied. It can be string variable or string constant.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char str1[50] = "Programming makes the life Hell";
  char str2[50] = "Built-in functions make it Heaven";
  cout<<"str1 = ";
  puts(str1);
  cout<<"str2 = ";
  puts(str2);
  strcpy (str1, str2);
  cout<<"\nAfter strcpy(str1, str2)...\n";
  cout<<"str1 = ";
  puts(str1);
  cout<<"str2 = ";
  puts(str2);
  getch();
}
```

Output:
str1 = Programming makes the life Hell
str2 = Built-in functions make it Heaven

After strcyp(str1, str2)...
str1 = Programming makes the life Hell
str2 = Built-in functions make it Heaven

12.4.15 strncpy()

The **strncpy()** function is used to copy one string to another including the terminating null character. It copies only specifed number of characters in one string to another.

Syntax: strncpy(str1, str2, n);

strncpy	It is the name of the function.
str1	It indicates the first string in which the string value is to be copied.
str2	It indicates the second string whose value is to be copied. It can be string variable or string constant.
n	It indicates the number of characters to be copied from str2 to str1. it can a variable or constant.

12.4.16 strcspn()

The **strcspn()** function is used to search a string for the first occurrence of a character belonging to the set of characters in second string. It returns the index of this character. This value is equivalent to the length of initial substring of first string consisting entirely of charcters not in second string.

Freebooks.pk

The function returns the length of initial segment of first string that consists of the characters not found in second string. It returns 0 if first string begins with a character from second string. If no character in second string appears in first string, then total length of first string is returned.

Syntax: strcspn(str, search);

strcspn It is the name of the function.
str It indicates the first string in which the string value is to be searched.
search It indicates the character set to be searched.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char str[25] = "C++ makes life spicy";
  char srch[10];
  int res;
  cout<<"The search string is: ";
  puts(str);
  cout<<"Enter the set of characters for strcspn(): ";
  gets(srch);
  res = strcspn (str, srch);
  if (res == strlen (str))
        cout<<"\nNone of the characters found.\n";
  else
        cout<<"\nCharacter found at index "<<res;
  getch();
}
```

Output:
The search string is: C++ makes life spicy
Enter the set of characters for strcspn: yia

Character found at index 5

12.4.17 strlen()

The word 'strlen' stands for **string legnth**. The function is used to find the length of a string. The length includes all characters as well spaces in the string. It does not count the null character. It returns an integer value indicating the length of the string.

Syntax: strlen(str);

strlen It is the name of the function.
str It indicates the string whose length is to be found.

Program 12.14

Write a program that inputs a string and displays the number of characters in it.

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
```

Freebooks.pk

```
    char str[25];
    int len;
    cout<<"Enter a string: ";
    gets(str);
    len = strlen(str);
    cout<<"The string contains "<<len<<" characters.";
    getch();
}
```

Output:
Enter a string: Hello World
The string contains 11 characters.

Program 12.15

Write a function that checks whether two arrays of characters are identical or not.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <process.h>
void main()
{
   char sen1[50], sen2[50];
   clrscr();
   void chk_eq(char str1[], char str2[]);
   cout<< "Enter String1: ";
   gets(sen1);
   cout<<"Enter String 2: ";
   gets(sen2);
   chk_eq(sen1, sen2);
   getch();
}

void chk_eq(char str1[], char str2[])
{
  int l1,l2;
  l1 = strlen(str1);
  l2 = strlen(str2);
  if(l1 != l2)
  {
    cout<<" \nTwo strings are NOT equal.";
    getch();
    exit(1);
  }
  for( int i = 0; i<= l1 - 1 ; i++)
  {
    if(str1[i] != str2[i])
    {
      cout<<" \n Two strings are NOT equal.";
      getch();
      exit(1);
    }
  }
  cout<< "\nTwo strings are EQUAL.";
}
```

Output:
Enter String1: Hello World
Enter String2: Hello World
Two strings are EQUAL

Freebooks.pk

12.4.18 strstr()

The **strstr()** function is used to find the the first occurrence of second string within first string. It returns a pointer to the first occurrence of second string in first string. It returns NULL if no occurrence was found. It returns first string if second string is of 0 length.

Syntax: strstr(str1, str2);

strstr	It is the name of the function.
str1	It indicates the first string.
str2	It indicates the second string.

Example

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
{
  clrscr();
  char str1[40] = "Programming makes life exciting.";
  char str2[20];
  char *res;
  cout<<"The string to be searched is: ";
  puts(str1);
  cout<<"Enter the search string: ";
  gets(str2);
  res = strstr (str1, str2);
  if (res == NULL)
        cout<<"\nThe search string was not found.\n";
  else
  {
        cout<<"\nMatching string found at "<<res -str1;
        cout<<"\nResult points to the string: ";
        puts(res);
  }
  getch();
}
```

Output:
The string to be searched is: Programming makes life exciting
Enter the search string: makes
Matching string found at: 12
Result points to the string: makes life exciting

12.4.19 strpbrk()

The **strpbrk()** function locates the first occurrence in the string pointed to by **s1** of any character from the string pointed to by **s2**. It returns a pointer to the character found. It returns a null pointer if no character from **s2** occurs in **s1**.

Syntax: strpbrk(const char *s1, const char *s2);

strpbrk	It is the name of the function.
s1	It indicates the pointer that references first string.
s2	It indicates the pointer that references second string.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
```

Freebooks.pk

```
void main()
{
  clrscr();
  char *string1 = "abcdefghijklmnopqrstuvwxyz";
  char *string2 = "onm";
  char *ptr;
  ptr = strpbrk(string1, string2);
  if (ptr)
    cout<<"strpbrk found first character: "<<*ptr<<endl;
  else
    cout<<"strpbrk didn't find character in set"<<endl;
  getch();
}
```

Output:
strpbrk found first character: m

12.4.20 strspn()

The **strspn()** function computes the length of the maximum initial segment of the string pointed to by **s1** which consists entirely of characters from the string pointed to by **s2**. It returns the length of the segment.

Syntax: strspn (str1, str2);

strspn	It is the name of the function.
str1	It indicates the first string.
str2	It indicates the second string.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  char *string1 = "1234567890";
  char *string2 = "123DC8";
  int length;
  length = strspn(string1, string2);
  cout<<"Character where strings differ is at position "<<length;
  getch();
}
```

Output:
Character where strings differ is at position 3

12.4.21 strtok()

The **strtok()** function scans the first string for the first token that is not contained in second string. It sequentially truncates string if delimiter is found. If **string** is not NULL, the function scans it for the first occurrence of any character included in **delimiters**. If it is found, the function overwrites the delimiter in *string* by a null-character and returns a pointer to the token i.e. the part of the scanned string previous to the delimiter. After a first call to **strtok**, the function may be called with NULL as string parameter, and it will follow by where the last call to strtok found a delimiter. delimiters may vary from a call to another.

Syntax: strtok (const char * string, const char * delimiters);

strtok	It is the name of the function.
string	It indicates the string to be scanned.
delimiters	It indicates the delimiters to be used in scanning.

Freebooks.pk

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char str[16] = "abc,d";
  char *p;
  /* strtok places a NULL terminator
     in front of the token, if found */
  p = strtok(str, ",");
  if(p)
     cout<<p<<endl;
  /* A second call to strtok using a NULL
     as the first parameter returns a pointer
     to the character following the token    */
  p = strtok(NULL, ",");
  if(p)
     cout<<p<<endl;
  getch();
}
```

Output:
abc
d

12.4.22 strerror()

The function **strerror()** takes an error number as parameter and returns a pointer to the error message associated with that error number. The funcion can be called with global variable **errno** declared in **errno.h**.

Syntax: strerror (int errnum);

strerror It is the name of the function.
errnum It indicates the error number whose error message is to be returned.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <errno.h>
void main()
{
  clrscr();
  char *buffer;
  buffer = strerror(errno);
  cout<<"Error: "<<buffer<<endl;
  getch();
}
```

Output:
Error: Error 0

12.4.23 strxfrm()

The **strxfrm()** function transforms the string pointed to by second string and places it in the array pointed to by first string. It transforms the entire string if possible but places no more than the number of bytes specified by **Number** parameter in the array pointed to by first string. It returns the length of transformed string not including the terminating null byte.

Syntax: strxfrm (str1, str2, n);

strxfrm	It is the name of the function.
str1	It indicates the first string.
str2	It indicates the second string.
n	It indicates the number of characters to be transformed.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char target[100];
  char source[] = "Hello World";
  int length;
  length = strxfrm(target, source, 80);
  cout<<target<<" has the length "<<length<<endl;
  getch();
}
```

Output:
Hello World has the length 11

12.4.24 strrev()

The **strxfrm()** function reverses all characters in a string except the null character.

Syntax: strrev(str);

strrev	It is the name of the function.
str	It indicates the string to be reversed.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char str[] = "string";
  cout<<"Before strrev(): "<<str<<endl;
  strrev(str);
  cout<<"After strrev(): "<<str<<endl;
  getch();
}
```

Output:
Before strrev(): string
After strrev(): gnirts

12.4.25 strset()

The **strset()** function sets all characters in a string to the specified character except the null character.

Syntax: strset(str, ch);

strset	It is the name of the function.
str	It indicates the string whose character are to be set.
ch	It indicates the character to be used for setting the string.

Freebooks.pk

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char str[10] = "123456789";
  char ch = '*';
  cout<<"Before strset(): "<<str<<endl;
  strset(str, ch);
  cout<<"After strset:  "<<str<<endl;
  getch();
}
```

Output:
Before strset(): 123456789
After strset(): ********

12.4.26 strnset()

The **strnset()** function sets specified number of characters at the start of a string to the specified character except the null character.

Syntax: strnset(str, ch, n);

strnset It is the name of the function.
str It indicates the string whose character are to be set.
ch It indicates the character to be used for setting the string.
n It indicates the number of characters to be set.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char str[] = "Hello World";
  char ch = '@';
  cout<<"string before strnset: "<<str<<endl;
  strnset(str, ch, 5);
  cout<<"string after  strnset: "<<str<<endl;
  getch();
}
```

Output:
string before strnset(): Hello World
string after strnset(): @@@@@ World

12.4.27 strlwr()

The **strlwr()** function converts all characters of a string to lower case.

Syntax: strlwr(str);

strlwr It is the name of the function.
str It indicates the string whose character are to be converted to lowercase.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
```

Freebooks.pk

```
void main()
{
  clrscr();
  char str[] = "Hello World";
  cout<<"string before to strlwr: "<<str<<endl;
  strlwr(str);
  cout<<"string after strlwr:   "<<str<<endl;
  getch();
}
```

Output:
string before strlwr(): Hello World
string after strlwr(): hello world

12.4.28 strupr()

The **strupr()** function converts all characters of a string to upper case.

Syntax: strupr(str);

strupr It is the name of the function.
str It indicates the string whose character are to be converted to uppercase.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char str[] = "Hello World";
  cout<<"string before to strlwr: "<<str<<endl;
  strupr(str);
  cout<<"string after strlwr:   "<<str<<endl;
  getch();
}
```

Output:
string before strlwr(): Hello World
string after strlwr(): HELLO WORLD

12.4.29 strdup()

The **strdup()** function duplicates a string in a newly created space.

Syntax: strdup(str);

strdup It is the name of the function.
str It indicates the string whose character are to be converted to uppercase.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  clrscr();
  char *dup, str[] = "Hello World";
  cout<<"Original string: "<<str<<endl;
  dup = strdup(str);
  cout<<"Duplicate string: "<<dup<<endl;
  getch();
}
```

Output:
Original string: Hello World
Duplicate string: Hello World

Freebooks.pk

12.4.30 stpcpy()

The stpcpy() function copies one string to other.

Syntax: strdup(str1, str2);

stpcpy	It is the name of the function.
str1	It indicates the string where the string will be copied.
str2	It indicates the string whose value will be copied.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main()
{
  char string[10];
  char str1[] = "String Handling";
  stpcpy(string, str1);
  cout<<string<<endl;
  getch();
}
```

Output:
String Handling

Programming Exercise

1. Write a program that inputs a string from user and checks the string is a palindrome or not. A palindrome is a string that reads the same backwards as forwards such as MADAM and MOM.
2. Write a program that inputs a string and displays number of words and number of characters without spaces in the string.
3. Write a program that inputs a string and displays it in reverse order.
4. Write a program that inputs a string and displays it as triangle. For example, if the user enters World, the program displays the following:

```
W
W o
W o r
W o r l
W o r l d
```

5. Write a program that inputs a string and a character. It converts each occurrence of the given character in the string to opposite case.

Freebooks.pk

Multiple Choice

1. **How can you initialize an array of two characters to 'a' and 'b'?**
   a. char[] charArray = {'a', 'b'}; b. char charArray[] = {'a', 'b'};
   c. char charArray[2] = {'a', 'b'}; d. Both b and c
2. **Which of the following statements are correct?**
   a. char charArray[2][2] = {{'a', 'b'}, {'c', 'd'}}; b. char charArray[][] = {'a', 'b'};
   c. char charArray[2][] = {{'a', 'b'}, {'c', 'd'}}; d. char charArray[][] = {{'a', 'b'}, {'c', 'd'}};
3. **Which of the following correctly declares an array of 3 rows of 5 columns?**
   a. char array[3],[5]; b. char array[3](char [5]);
   c. char array[3][5]; d. char array[3,5];
4. **Which of the following correctly instantiates and initializes an array of characters?**
   a. char str[] = "Initialize" ; b. char str[] = { 'I', 'n', 'i', 't', 'i', 'a', 'l', 'i', 'z', 'e' } ;
   c. char str[] = { "Initialize" } ; d. All
5. **Which is an appropriate statement to define an array to store names up to 20 characters:**
   a. char Name[20]; b. char Name[21]; c. string lastName[22]; d. None
6. **Two-dimensional array of characters can contain**
   a. strings of the same length b. uninitialized elements
   c. strings of different lengths d. None
7. **Which of the following is automatically appended to a character array when it is initialized with a string constant.**
   a. Null terminator b. number of element c. array name d. None
8. **Which of the following function accepts a pointer to a string as an argument, and it returns the length of the string not including the null terminator.**
   a. strlength b. strlen c.strcpy d.strcat
9. **The strcpy function's arguments are:**
   a. three pointers b. two pointers c. two addresses d. None
10. **Which of the following function that can find one string inside another is:**
    a. strstr b.strcmp c. strcat d. None
11. **Which of the following function is used to copy one string to another string.**
    a. strcpy b. strcmp c. strcat d. None
12. **Which of the following function is used to input data into string?**
    a. gets b. puts c. both a and b d. None
13. **Which function is used to concatenate the contents of one string with another**
    a. strcat b. strcopy c. strcpy d. Non
14. **Which of the following function is used to compare the specified number of characters in two strings**
    a. strncmp b. strchr c. strcompare d. None

Answers

1. d	2. a	3. c	4. d	5. b	6. d
7. a	8. b	9. c	10. a	11. a	12. a
13. a	14. a				

Freebooks.pk

Fill in the Blanks

1. The function _____ is used to input single character from the keyboard.
2. _____ function is used to display single character on the screen.
3. The _____ function is used to append the specified number of characters of one string to the end of second string
4. The _____ function is used to find the first occurrence of a character in string and returns a pointer to this character
5. The _____ function is used to compare two strings character by character
6. The _____ function is used to compare the specified number of characters in two strings.
7. The _____function is used to search a string for the first occurrence of a character belonging to the set of characters in second string.

Answers

1. getch	2. putchar
3. strncat()	4. strchr()
5. strcmp	6. strncmp
7. strcspn()	

True/False

1. A string is stored as an array of characters
2. A string value can be input from the user using different functions.
3. The strlen function returns a string's length and adds one for \0f
4. An individual array element can be processed like any other type of C++ variable

Answers

1. T	2. T	3. F	4. T

Freebooks.pk

CHAPTER 13

BASICS OF OBJECT-ORIENTED PROGRAMMING

Chapter Overview

13.1 Object-Oriented Programming
- 13.1.1 Features of Object-Oriented Programming

13.2 Objects
- 13.2.1 Properties of Object
- 13.2.2 Functions of Object

13.3 Classes
- 13.3.1 Declaring a Class
- 13.3.2 Access Specifiers

13.4 Creating Objects
- 13.4.1 Executing Member Functions
- 13.4.2 Defining Member Functions outside Class

13.5 Constructors
- 13.5.1 Passing Parameters to Constructors
- 13.5.2 Constructor Overloading

13.6 Default Copy Constructor

13.7 Destructors

13.8 Objects as Function Parameters
- 13.8.1 Returning Objects from Member Functions

13.9 Static Data Member

13.10 Friend Functions

13.11 Friend Classes

13.12 Static Functions

Programming Exercise

Exercise Questions

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

13.1 Object-Oriented Programming

Object-Oriented programming (OOP) is a programming technique in which programs are written on the basis of objects. An **object** is a collection of data and functions. Object may represent a person, thing or place in real world. In OOP, data and all possible functions on data are grouped together. Object oriented programs are easier to learn and modify.

Object-Oriented programming is a powerful technique to develop software. It is used to analyze and design the applications in terms of objects. It deals with data and the procedures that process the data as a single object.

Some of the object-oriented languages that have been developed are:

- C++
- Smalltalk
- Eiffel
- CLOS
- Java

13.1.1 Features of Object-Oriented Programming

Following are some features of object-oriented programming:

1. Objects

OOP provides the facility of programming based on objects. Object is an entity that consists of data and functions.

2. Classes

Classes are designs for creating objects. OOP provides the facility to design classes for creating different objects. All properties and functions of an object are specified in classes.

3. Real-world Modeling

OOP is based on real-world modeling. As in real world, things have properties and working capabilities. Similarly, objects have data and functions. Data represents properties and functions represent working of objects.

4. Reusability

OOP provides ways of reusing the data and code. **Inheritance** is a technique that allows a programmer to use the code of existing program to create new programs.

5. Information Hiding

OOP allows the programmer to hide important data from the user. It is performed by encapsulation.

6. Polymorphism

Polymorphism is an ability of an object to behave in multiple ways.

13.2 Objects

An **object** represents an entity in the real world such as a person, thing or concept etc. An object is identified by its name. An object consists of the following two things:

- **Properties:** Properties are the characteristics of an object.
- **Functions:** Functions are the action that can be performed by an object.

Freebooks.pk

Examples

Some examples of objects of different types are as follows:

- Physical Objects
  - Vehicles such as car, bus, truck etc.
  - Electrical components
- Elements of Computer-User Environment
  - Windows
  - Menus
  - Graphics objects
  - Mouse and Keyboard
- User-defined Data Types
  - Time
  - Angles

13.2.1 Properties of Object

The characteristics of an objects are known as its **properties** or **attributes**. Each object has its own properties. These properties can be used to describe the object. For example, the properties of an object **Person** can be as follows:

- Name
- Age
- Weight

The properties of an object **Car** can be as follows:

- Color
- Price
- Model
- Engine power

The set of values of the attributes of a particular object is called its **state**. It means that the state of an object can be determined by the values of its attributes. The following figure shows the values of the attributes of an object **Car**:

Car	
Model	Honda City
Color	Silver
Price	12,00,000
Engine Power	1600 CC

Figure 13.1: Properties of an object Car

13.2.2 Functions of Object

An object can perform different tasks and actions. The actions that can be performed by an object are known as **functions** or **methods**. For example, the ojbect **Car** can perform the following functions:

- Start
- Stop
- Accelerate
- Reverse

Freebooks.pk

The set of all functions of an object represents the **behavior** of the object. It means that the overall behavior of an object can be determined by the list of functions of that object.

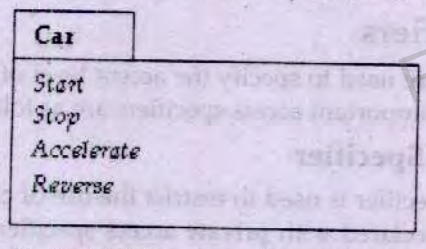


Figure 13.2: Functions of an object Car

13.3 Classes

A collection of objects with same properties and functions is known as **class**. A class is used to define the characteristics of the objects. It is used as a model for creating different objects of same type. For example, a class Person can be used to define the characteristics and functions of a person. It can be used to create many objects of type Person such as Ali, Usman, Abdullah etc. All objects of **Person** class will have same characteristics and functions. However, the values of each object can be different. The values are of the objects are assigned after creating an object.

Each object of a class is known as an **instance** of its class. For example, Ali, Usman and Abdullah are three instances of a class **Person**. Similarly, myBook and yourBook can be two instances of a class **Book**.

13.3.1 Declaring a Class

A class is declared in the same way as a structure is declared. The keyword **class** is used to declare a class. A class declaration specifies the variables and functions that are common to all objects of that class. The variables declared in a class are known as **member variables** or **data members**. The functions declared in a class are called **member functions**.

Syntax

The syntax of declaring a class is as follows:

```
class identifier
{
  body of the class
};
```

class It is the keyword that is used to declare a class.

identifier It is the name of the class. The rules for class name are same as the rules for declaring a variable.

The class declaration always ends with semi colon. All data members and member functions are declared within the braces known as body of the class.

Example

```
class Test
{
  int n;
  char c;
  float x;
};
```

Freebooks.pk

The above class only contains data members in it. A class may contain both data members and member functions.

13.3.2 Access Specifiers

The commands that are used to specify the access level of class members are known as **access specifers**. Two most important access specifiers are as follows:

1. The 'private' Access Specifier

The **private** access specifier is used to restrict the use of class member within the class. Any member of the class declared with **private** access specifier can only be accessed within the class. It cannot be accessed from outside the class. It is also the default access specifier.

The data members are normally declared with **private** access specifier. It is because the data of an object is more sensitive. The **private** access specifier is used to protect the data member from direct access from outside the class. These data members can only be used by the functions declared within the class.

2. The 'public' Access Specifier

The **public** access specifier is used to allow the user to access a class member within the class as well as outside the class. Any member of the class declared with **public** access specifier can be accessed from anywhere in the program.

The members functions are normally declared with **public** access specifier. It is because the users access functions of an object from outside the class. The class cannot be used directly if both data members and member functions are declared as **private**.

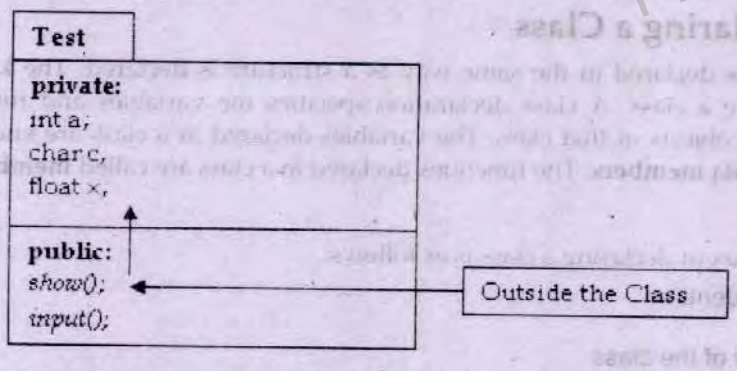


Figure 13.3: The **private** and **public** access specifiers in class **Test**

The above figure shows that data members **a**, **c** and **x** of class **Test** cannot be accessed from outside the class because they are declared with **private** access specifier. The member functions **show()** and **input()** are accessible from outside the class because they are declared with public access specifier. The data members are accessed and manipulated indirectly through member functions.

13.4 Creating Objects

A class is simply a model or prototype for creating objects. It is like a new data type that contains both data and functions. An object is created in the same way as other variables are created. When an object of a class is created, the space for all data members defined in the class is allocated in the memory according to their data types. An object is also known as **instance**. The process of creating an object of a class is also called **instantiation**.

Syntax

The syntax of creating an object of a class is as follows:

```
class_name object_name;
```

class_name: It is the name of the class whose type of object is to be created.

object_name: It is the name of the object to be created. The rules for object name are same as the rules for declaring a variable.

Example

```
Test obj;
```

The above example declares an object **obj** of type **Test**. The object contains all data members that are defined in class **Test**.

13.4.1 Executing Member Functions

An object of a particular class contains all data members as well as member functions defined in that class. The data members contains the value related to the object. The member functions are used to manipulate data members. The member functions can be executed only after creating an object.

Syntax

The syntax of executing member functions is as follows:

```
object_name.function();
```

object_name: It is the name of the object whose member function is to be executed.

function: It is the name of the member function to be executed. Any required parameters are also passed to the member function in parenthesis.

The object name and member function are separated by **dot operator**.

Example

```
Test obj;
obj.input();
```

Program 13.1

Write a program that declares a class with one integer data member and two member functions in() and out() to input and output data in data member.

```
#include <iostream.h>
#include <conio.h>
class Test
{
  private:
  int n;
  public:
  void in()
  {
    cout<<"Enter a number:";
    cin>>n;
  }
  void out()
  {
    cout<<"The value of n = "<<n;
  }
```

Output:
Enter a number: 10
The value of n = 10

Freebooks.pk

```
}
void main()
{
  clrscr();
  Test obj;
  obj.in();
  obj.out();
  getch();
}
```

Program 13.2

Write a class Marks with three data members to store three marks. Write three member functions in() to input marks, sum() to calculate and return the sum and avg() to calculate and return the average marks.

```
#include <iostream.h>
#include <conio.h>
class Marks
{
  private:
  int a, b, c;
  public:
  void in()
  {
    cout<<"Enter three marks:";
    cin>>a>>b>>c;
  }
  int sum()
  {
    return a+b+c;
  }
  float avg()
  {
    return (a+b+c)/3.0;
  }
};
void main()
{
  Marks m;
  int s;
  float a;
  m.in();
  s = m.sum();
  a = m.avg();
  cout<<"Sum = "<<s<<endl;
  cout<<"Average = "<<a;
  getch();
}
```

Output:

```
Enter three marks: 50
40
50
Sum = 140
Average = 46.666668
```

Program 13.3

Write a class circle with one data member radius. Write three member functions get_radius() to set radius value with parameter value, area() to display radius and circum() to calculate and display circumference of circle.

```
#include<iostream.h>
#include<conio.h>
class circle
{
  private :
    float radius;
  public :
  void get_radius(float r)
  {
    radius = r;
  }
void area( )
{
  cout<< "\nArea of circle: " << 3.14 * radius * radius ;
}

void circum()
{
  cout<< "\nCircumference of circle: "<< 2 *3.14 * radius ;
}
};

void main()
{
  clrscr();
  circle c1;
  float rad;
  cout<< "\nEnter radius: ";
  cin >> rad;
  c1.get_radius(rad);
  c1.area();
  c1.circum();
  getch();
}
```

Output:
Enter radius: 5
Area of circle: 78.5
Circumference of circle: 31.4

Program 13.4

Write a class **Book** with three data members **BookID**, **Pages** and **Price**. It also contains the following member function:

- The **get()** function is used to input values
- The **show()** function is used to display values
- The **set()** function is used to set the values of data members using parameters
- The **getPrice()** function is used to return the value of **Price**.

The program should create two objects of the class and input values for these objects. The program displays the details of the most costly book.

```
#include <iostream.h>
#include <conio.h>
class Book
{
  private
    int BookID, Pages;
    float Price;
```

```
    public:
    void get()
    {
      cout<<"Enter Book ID:";
      cin>>BookID;
      cout<<"Enter pages:";
      cin>>Pages;
      cout<<"Enter price:";
      cin>>Price;
    }
    void show()
    {
      cout<<"BookID = "<<BookID<<endl;
      cout<<"Pages = "<<Pages<<endl;
      cout<<"Price = "<<Price<<endl;
    }
    void set(int id, int pg, float pr)
    {
      BookID = id;
      Pages = pg;
      Price = pr;
    }
    float getPrice()
    {
      return Price;
    }
  };
  void main()
  {
    clrscr();
    Book b1, b2;
    b1.get();
    b2.set(2, 320, 150.75);
    cout<<"\nThe detail of most costly book is as follows:"<<endl;
    if(b1.getPrice() > b2.getPrice())
      b1.show();
    else
      b2.show();
    getch();
  }
```

Output:

```
Enter Book ID: 1
Enter page: 250
Enter price: 125.20

The detail of most costly book is as follows:
BookID = 2
Pages = 320
Price = 150.75
```

How above Program Works?

The above program creates two objects of class Book. When two or more objects of the same class are created, each object contains its own values in the memory. The data members of each object are separate to store the values of a particular object. However, the member functions of the class are created only once in the memory. These member functions are shared by all objects of the class.

Freebooks.pk

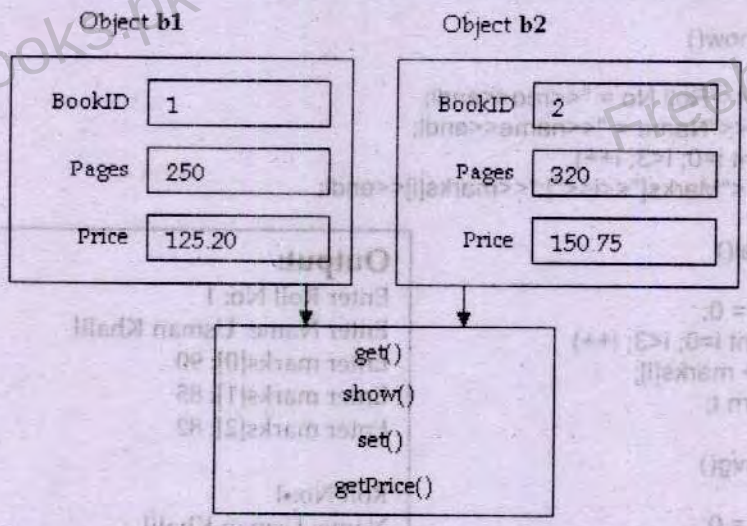


Figure 13.4: The creation of objects in memory

The above figure shows that all objects have their own memory addresses to store the values of data members. However, all objects share the same copy of member functions.

Program 13.5

Write a class **Result** that contains roll no, name and marks of three subjects. The marks are stored in an array of integers. The class also contains the following member functions:

- The **input()** function is used to input the values in data members
- The **show()** function is used to displays the value of data members
- The **total()** function returns the total marks of a student.
- The **avg()** function returns the average marks of a student.

The program should create an object of the class and call the member functions.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
class Result
{
  private:
  int rno, marks[3];
  char name[50];
  public:
  void input()
  {
    cout<<"Enter Roll No:";
    cin>>rno;
    cout<<"Enter name:";
    gets(name);
    for(int i=0; i<3; i++)
    {
      cout<<"Enter marks["<<i<<"]:";
      cin>>marks[i];
    }
  }
```

```
    void show()
    {
      cout<<"Roll No = "<<rno<<endl;
      cout<<"Name = "<<name<<endl;
      for(int i=0; i<3; i++)
     cout<<"Marks["<<i<<"]:"<<marks[i]<<endl;
    }
    int total()
    {
      int t = 0;
      for(int i=0; i<3; i++)
     t = t + marks[i];
      return t;
    }
    float avg()
    {
      int t = 0;
      for(int i=0; i<3; i++)
     t = t + marks[i];
      return t/3.0;
    }
  };
  void main()
  {
    Result r;
    r.input();
    r.show();
    cout<<"\nTotal marks = "<<r.total()<<endl;
    cout<<"Average marks = "<<r.avg()<<endl;
    getch();
  }
```

Output:

```
Enter Roll No: 1
Enter Name: Usman Khalil
Enter marks[0]: 90
Enter marks[1]: 85
Enter marks[2]: 82

Roll No: 1
Name: Usman Khalil
Marks[0]: 90
Marks[1]: 85
Marks[2]: 82
Total Marks = 257
Average marks = 85.666664
```

13.4.2 Defining Member Functions outside Class

The member function of a class can also be defined outside the class. The declaration of member functions is specified within the class and function definition is specified outside the class. The scope **resolution operator ::** is used in function declarator if the function is defined outside the class.

Syntax

The syntax of defining member function outside the class is as follows:

```
return_type class_name :: function_name(parameters)
{
  function body
}
```

return_type It indicates the type of value to be returned by the function.

class_name It indicates the name of the class to which the function belongs.

:: It is the scope resolution operator to define member function outside class.

function_name It is the name of the member function to be defined.

function_body It is the body of the function.

Freebooks.pk

Program 13.6

Write a class Array that contains an array of integers to store five values. It also contains the following member functions:

- The **fill()** function is used to fill the array with values from the user.
- The **display()** function is used to display the values of array.
- The **max()** function shows the maximum value in the array.
- The **min()** function shows the minimum value in the array.

All member function should be defined outside the class.

```
#include <iostream.h>
#include <conio.h>
class Array
{
   private:
   int a[5];
   public:
   void fill();
   void display();
   int max();
   int min();
};
void Array::fill()
{
   for(int i=0; i<5; i++)
   {
      cout<<"Enter a["<<i<<"]: ";
      cin>>a[i];
   }
}
void Array::display()
{
   for(int i=0; i<5; i++)
      cout<<"a["<<i<<"]: "<<a[i]<<endl;
}
int Array::max()
{
   int m = a[0];
   for(int i=0; i<5; i++)
      if(m<a[i])
     m = a[i];
   return m;
}

int Array::min()
{
   int m = a[0];
   for(int i=0; i<5; i++)
      if(m>a[i])
     m = a[i];
   return m;
}
```

Output:

```
Enter a[0]: 52
Enter a[1]: 31
Enter a[2]: 97
Enter a[3]: 68
Enter a[4]: 28
You entered the following values:
a[0]: 52
a[1]: 31
a[2]: 97
a[3]: 68
a[4]: 28
Maximum value = 97
Minimum value = 28
```

Freebooks.pk

```
void main()
{
  Array arr;
  arr.fill();
  cout<<"You entered the following values:"<<endl;
  arr.display();
  cout<<"Maximum value = "<<arr.max()<<endl;
  cout<<"Minimum value = "<<arr.min();
  getch();
}
```

13.5 Constructors

A type of member function that is automatically executed when an object of that class is created is known as **constructor**. The constructor has no return type and has same name that of class name. The constructor can work as a normal function but it cannot return any value. It is normally defined in classes to initialize data member.

Syntax

The syntax of declaring constructor is as follows:

```
name()
{
  constructor body
}
```

name It indicates the name of the constructor. The name must be same as the name of the class in which the constructor is declared.

Program 13.7

Write a class that displays a simple message on the screen whenever an object of that class is created.

```
#include <iostream.h>
#include <conio.h>
class Hello
{
  private:
  int n;
  public:
  Hello()
  {
    cout<<"Object created..."<<endl;
  }
};
void main()
{
  Hello x, y, z;
  getch();
}
```

Output:
Object created…
Object created…
Object created…

How above Program Works?

The above program declares a constructor that displays a message on the screen. The program creates three objects in **main()** function. The constructor is executed each time an object of the class is created in the memory.

Freebooks.pk

Program 13.8

Write a class that contains two integer data members which are initialized to 100 when an object is created. It has a member function **avg** that displays the average of data members.

```
#include <iostream.h>
#include <conio.h>
class Number
{
  private:
  int x, y;
  public:
  Number()
  {
    x = y = 100;
  }
  void avg()
  {
    cout<<"x = "<<x<<endl;
    cout<<"y = "<<y<<endl;
    cout<<"Average = "<<(x+y)/2<<endl;
  }
};
void main()
{
  clrscr();
  Number n;
  n.avg();
  getch();
}
```

Output:
x = 100
y = 100
Average = 100

13.5.1 Passing Parameters to Constructors

The method of passing parameters to a constructor is same as passing parameters to normal functions. The only difference is that the parameters are passed to the constructor when the object is declared. The parameters are written in parenthesis along with the object name in declaration statement.

Sytnax

The syntax of passing parameters to constructor is as follows:

```
type object_name(parameters);
```

type It is the name of a class and indicates the type of object to be created.

object_name It indicates the name of the object to be created.

parameters It indicates the list of parameters passed to the constructor.

Program 13.9

Write a class that has marks and grade as data members. A constructor with two parameters initializes data members with the given values and member function show displays the values of data members. Create two objects and display the values.

```
#include <iostream.h>
#include <conio.h>
```

```
class Student
{
  private:
  int marks;
  char grade;
  public:
  Student(int m, char g)
  {
    marks = m;
    grade = g;
  }
  void show()
  {
    cout<<"Marks = "<<marks<<endl;
    cout<<"Grade = "<<grade<<endl;
  }
};
void main()
{
  clrscr();
  Student s1(730,'A'), s2(621,'B');
  cout<<"Record of Student 1:"<<endl;
  s1.show();
  cout<<"Record of Student 2:"<<endl;
  s2.show();
  getch();
}
```

Output:
Record of Student 1:
Marks = 730
Grade = A
Record of Student 2:
Marks = 621
Grade = B

Program 13.10

Write a class **TV** that contains attributes of Brand Name, Model and Retail Price. Write a method to display all attributes and a method to change the attributes. Also write a constructor to initialize all the attributes.

```
#include <conio.h>
#include <iostream.h>
#include <string.h>
class TV
{
  public:
  TV(char Brand[], char Mod[], float Price);
  void Change(char Brand[], char Mod[], float Price);
  void Display();
private:
 char BrandName[20];
 char Model[10];
 float RetailPrice;
};
TV :: TV(char Brand[], char Mod[], float Price)
{
  strcpy(BrandName, Brand);
  strcpy(Model, Mod);
  RetailPrice=Price;
}
```

Freebooks.pk

```
void TV :: Change(char Brand[], char Mod[], float Price)
{
  strcpy(BrandName, Brand);
  strcpy(Model, Mod);
  RetailPrice = Price;
}

void TV :: Display()
{
  cout<<"Brand Name: "<<BrandName<<endl;
  cout<<"Model: "<<Model<<endl;
  cout<<"Price: "<<RetailPrice<<endl;
}

void main()
{
  clrscr();
  TV Test("SONY", "HDTV", 25000);
  cout<<"Displaying the object..."<<endl;
  Test.Display();
  Test.Change("Toshiba", "STDV", 22000);
  cout<<"Displaying object after change..."<<endl;
  Test.Display();
  getch();
}
```

Output:
Displaying the object...
Brand Name: SONY
Model: HDTV
Price: 25000
Displaying object after change...
Brand Name: Toshiba
Model: SDTV
Price: 22000

13.5.2 Constructor Overloading

The process of declaring multiple constructors with same name but different parameters is known as **constructor overloading**. The constructors with same name must differ in one of the following ways:

- Number of parameters
- Type of parameter
- Sequence of parameters

Program 13.11

Write a class that has **num** and **ch** as data members. A constructor with no parameter initializes **num** to 0 and **ch** to 'x'. A constructor with two parameters initializes data members with the given values and member function show displays the values of data members.

```
#include <iostream.h>
#include <conio.h>
class Over
{
  private:
  int num;
  char ch;
  public:
  Over()
  {
    num = 0;
    ch = 'x';
  }
```

```
    Over(int n, char c)
    {
      num = n;
      ch = c;
    }
    void show()
    {
      cout<<"num = "<<num<<endl;
      cout<<"ch = "<<ch<<endl;
    }
  };
  void main()
  {
    Over first, second(100, 'p');
    cout<<"The contents of first:"<<endl;
    first.show();
    cout<<"The contents of second:"<<endl;
    second.show();
    getch();
  }
```

Output:
The contents of first:
num = 0
ch = x
The contents of second:
num = 100
ch = p

13.6 Default Copy Constructor

A type of constructor that is used to initialize an object with another object of the same type is known as **default copy constructor**. Its name is "default copy constructor" because it is available by default in all classes. The user does not need to write this constructor. It accepts a single object of the same type as parameter. The parameter for default copy constructor can be given in parenthesis or using assignment operator.

Syntax

The syntax of using default copy constructor is as follows:

```
class_name object_name (parameter);   OR
class_name object_name = parameter;
```

class_name It is the name of a class and indicates the type of object to be created.

object_name It indicates the name of the object to be created.

parameter It indicates the name of parameter that is passed to default copy constructor. The values of data members of parameter object are copied to data members of the new object. The type of new object and parameter object must be same.

Program 13.12

Write a class **Book** that has attributes for pages, price and title. It has two functions to input the values and displays the values. Create three objects of the class and input values.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
class Book
{
  private:
  int pg, pr;
  char title[50];
  public:
```

```
    void get()
    {
      cout<<"Enter title: ";
      gets(title);
      cout<<"Enter pages: ";
      cin>>pg;
      cout<<"Enter price: ";
      cin>>pr;
    }
    void show()
    {
      cout<<"Title: "<<title<<endl;
      cout<<"Pages: "<<pg<<endl;
      cout<<"Price: "<<pr<<endl;
    }
};
void main()
{
  Book b1;
  b1.get();
  Book b2(b1);
  Book b3 = b1;
  cout<<"\nThe detail of b1:"<<endl;
  b1.show();
  cout<<"\nThe detail of b2:"<<endl;
  b2.show();
  cout<<"\nThe detail of b3:"<<endl;
  b3.show();
  getch();
}
```

Output:

```
Enter title: Visual Programming
Enter pages: 637
Enter price: 200

The detail of b1:
Title: Visual Programming
Pages: 637
Price: 200

The detail of b2:
Title: Visual Programming
Pages: 637
Price: 200

The detail of b3:
Title: Visual Programming
Pages: 637
Price: 200
```

How above Program Works?

The above program declares an object **b1** of type **Book**. It accepts the values from user for **b1** using **get()** member function. The following statements declare two objects **b2** and **b3** using default copy constructor:

```
Book b2(b1);
Book b3 = b1;
```

The data members of **b2** and **b3** are initialized to the values of **b1**.

13.7 Destructors

A type of member function that is automatically executed when an object of that class is destroyed is known as **destructor**. The destructor has no return type and its name is same as class name. The destructor cannot return any value. It also cannot accept any parameters. The destructor name is preceded by tilde sign ~.

Syntax

The syntax of declaring destructor is as follows:

```
~name()
{
  destructor body
}
```

Freebooks.pk

~name It indicates the name of the destructor. The name must be same as the name of the class in which the constructor is declared.

Example

```
#include <iostream.h>
#include <conio.h>
class Test
{
  private:
  int n;
  public:
  Test()
  {
    cout<<"Object created..."<<endl;
  }
  ~Test()
  {
    cout<<"Object destroyed..."<<endl;
  }
};
void main()
{
  clrscr();
  Test a, b;
}
```

Output:
Object created...
Object created...
Object destroyed...
Object destroyed...

How above Program Works?

The above program creates a constructor and a destructor in the class. Both display simple messages on screen. The message "Object created..." will appear when the program is executed. The message "Object destroyed..." will appear on the screen when the program is terminated and all objects are destroyed from memory. Press **Alt+F5** to view the output after terminating the program.

13.8 Objects as Function Parameters

Objects can also be passed as parameters to member functions. The method of passing objects to a functions as parameters is same as passing other simple variables.

Program 13.13

Write a class **Travel** that has the attributes of kilometers and hours. A constructor with no parameter initializes both data members to 0. A member function get() inputs the values and function show() displays the values. It has a member function add() that takes an object of type Travel to add the kilometers and hours of calling object and the parameter.

```
#include <iostream.h>
#include <conio.h>
class Travel
{
  private:
  int km, hr;
  public:
```

Freebooks.pk

```
    Travel()
    {
      km = hr = 0;
    }
    void get()
    {
      cout<<"Enter kilometers traveled: ";
      cin>>km;
      cout<<"Enter hours traveled: ";
      cin>>hr;
    }
    void show()
    {
      cout<<"You traveled "<<km<<" in "<<hr<<" hours"<<endl;
    }
    void add(Travel p)
    {
      Travel t;
      t.km = km + p.km;
      t.hr = hr + p.hr;
      cout<<"Total traveling is "<<t.km<<" kilometers in "<<t.hr<<" hours."<<endl;
    }
};
void main()
{
  clrscr();
  Travel my, your;
  my.get();
  my.show();
  your.get();
  your.show();
  my.add(your);
  getch();
}
```

Output:

```
Enter kilometers traveled: 100
Enter hours traveled: 2
You traveled 100 kilometers in 2 hours.
Enter kilometers traveled: 300
Enter hours traveled: 3
You traveled 300 kilometers in 3 hours.
Total traveling is 400 kilometers in 5 hours.
```

How above Program Works?

The above program declares two objects of class **Travel** and inputs data in both objects. The **add()** member function accepts an object of type Travel as parameter. It adds the values of data members of the parameter object and the values of calling object's data members and displays the result. The working of member function add() is as follows:

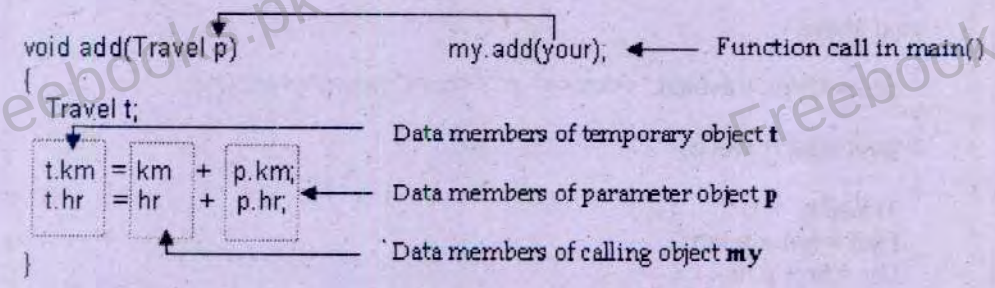


Figure 13.5: Working of member function **add()**

The above figure shows that the **add()** function receives the contents of **your** object in parameter **p**. It means that the values of **p** represent the values of **your** object. The function adds both values and stores the result in a temporary object **t** using the following statements:

```
t.km = km + p.km;
t.hr = hr + p.hr;
```

In the above statements, **t.km** and **t.hr** are the data members of the temporary object **t**, **p.km** and **p.hr** are the data members of parameter object **p**. The data members without dot operator are the data members of calling object **my**. It means that any data member that is not preceded by object name in a member function represents the data member of **calling object**.

13.8.1 Returning Objects from Member Functions

The method of returning an object from member function is same as returning a simple variable. If a member function returns an object, its return type should be the same as the type of object to be returned.

Program 13.14

Write a class **Travel** that has the attributes of kilometers and hours. A constructor with no parameter initializes both data members to 0. A member function get() inputs the values and function show() displays the values. It has a member function add() that takes an object of type Travel, adds the kilometers and hours of calling object and the parameter and returns an object with added values.

```
#include <iostream.h>
#include <conio.h>
class Travel
{
  private:
  int km, hr;
  public:
  Travel()
  {
    km = hr = 0;
  }
  void get()
  {
    cout<<"Enter kilometers traveled: ";
    cin>>km;
    cout<<"Enter hours traveled: ";
    cin>>hr;
  }
  void show()
  {
    cout<<"You traveled "<<km<<" in "<<hr<<" hours"<<endl;
  }
  Travel add(Travel p)
  {
    Travel t;
    t.km = km + p.km;
    t.hr = hr + p.hr;
    return t;
  }
```

```
};
void main()
{
  clrscr();
  Travel my, your, r;
  my.get();
  my.show();
  your.get();
  your.show();
  r = my.add(your);
  cout<<"Total traveling is as follows:"<<endl;
  r.show();
  getch();
}
```

Output:

```
Enter kilometers traveled: 100
Enter hours traveled: 2
You traveled 100 kilometers in 2 hours.
Enter kilometers traveled: 300
Enter hours traveled: 3
You traveled 300 kilometers in 3 hours.
Total traveling is as follows:
You traveled 400 kilometers in 5 hours.
```

How above Program Works?

The **add()** function in the above program accepts a parameter object of type **Travel**. It adds the contents of parameter and calling object and stores the result in a temporary object. The function then returns the whole object back to main() function that is stored in **r**. The program finally displays the result using **r.show()** statement.

13.9 Static Data Member

A type of data member that is shared among all objects of class is known as **static data member**. The static data member is defined in the class with **static** keyword. When a data member is defined as static, only one variable is created in the memory even if there are many objects of that class.

The characteristics of a static data member are same as normal static variable. It is visible only in the class in which it is defined but its lifetime starts when the program starts its execution. Its lifetime ends when the entire program is terminated. It is normally used to share some data among all objects of a particular class.

The main difference between normal data member and static data member is that each object has its own variable of normal data member. On the other hand, static data member is shared among all objects of the class. Only one memory location is created for static data member that is shared among all objects.

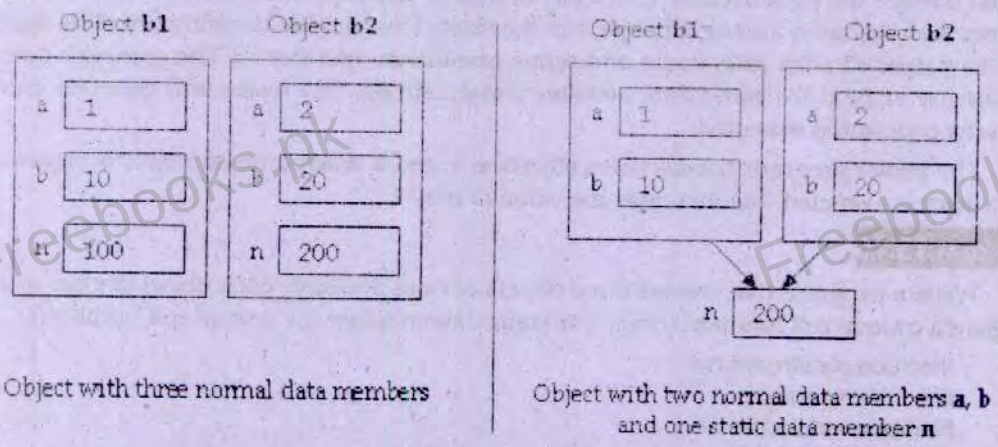


Figure 13.6: Difference between normal data member and static data member

Program 13.15

Write a program that counts the number of objects created of a particular class.

```
#include <iostream.h>
#include <conio.h>
class yahoo
{
  private:
  static int n;
  public:
  yahoo()
  {
    n++;
  }
  void show()
  {
    cout<<"You have created "<<n<<" objects so far."<<endl;
  }
};
int yahoo::n = 0;
void main()
{
  clrscr();
  yahoo x, y;
  x.show();
  yahoo z;
  x.show();
  getch();
}
```

Output:
You have created 2 objects so far.
You have created 3 objects so far.

How above Program Works?

The above program declares a static data member **n** to count the number of objects that have been created. The following statement defines the variable:

```
int yahoo::n = 0;
```

The above statement defines the variable and initializes it to 0 value. The variable is defined outside the class because it will be not part of any object. It is created only once in the memory and is shared among all objects of the class. The variable definition outside the class must be preceded with class name and scope **resolution operator : :**. The compiler does not display any error if the static data member is not defined. The linker will generate an error when the program is executed.

The above program creates three objects **x**, **y** and **z**. Each time an object is created, the constructor is executed that increases the value of **n** by 1.

Program 13.16

Write a program that creates three objects of class **Student**. Each object of class must be assigned a unique roll number. (Hint: Use static data member for unique roll number.)

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
class Student
{
```

Freebooks.pk

```
    private:
    static int r;
    int rno, marks;
    char name[50];
    public:
    Student()
    {
      r++;
      rno = r;
    }
    void in()
    {
      cout<<"Enter name: ";
      gets(name);
      cout<<"Enter marks: ";
      cin>>marks;
    }
    void show()
    {
      cout<<"Roll No: "<<rno<<endl;
      cout<<"Name: "<<name<<endl;
      cout<<"Marks: "<<marks<<endl;
    }
};
int Student::r = 0;
void main()
{
  Student s1, s2, s3;
  s1.in();
  s2.in();
  s3.in();
  s1.show();
  s2.show();
  s3.show();
  getch();
}
```

Output:

```
Enter name: Usman Khalil
Enter marks: 730
Enter name: Abdullah
Enter marks: 786
Enter name: Ali
Enter marks 521

Roll No: 1
Name: Usman Khalil
Marks: 730
Roll No: 2
Name: Abdullah
Marks: 786
Roll No: 3
Name: Ali
Marks: 521
```

How above Program Works?

The above program uses a static data member **r** to assign unique roll numbers to each object of the class **Student**. The static data member is initialized to 0. The constructor increments its value by 1 when an object is created and then assigns the updated value of **r** to the data member **rno**. It ensures that each object gets a unique roll number.

13.10 Friend Functions

A type of function that is allowed to access the private and protected members of a particular class from outside the class is called **friend function**. Normally, the private and protected members of any class cannot be accessed from outside the class. In some situations, a program may require to access these members. The use of friends functions allows the user to access these members.

A function that is declared in a class with **friend** keyword becomes the friend function of that class. It enables that function to access the private and protected members of the class.

Freebooks.pk

Example

Suppose a friend function accepts two objects of different classes as parameters. It has to process the private data members of these classes and then displays the result. It is not possible to perform the task because a function that is not a member of a class cannot access the private or protected members of the class.

The problem can be solved by declaring that function as friend function of both classes. It will enable the function to access the private and protected members of both classes.

Example

```
#include <iostream.h>
#include <conio.h>
class B;
class A
{
  private:
  int a;
  public:
  A()
  {
    a = 10;
  }
  friend void show(A, B);
};
class B
{
  private:
  int b;
  public:
  B()
  {
    b = 20;
  }
  friend void show(A,B);
};
void show(A x, B y)
{
  int r;
  r = x.a + y.b;
    cout<<"The value of class A object = "<<x.a<<endl;
    cout<<"The value of class B object = "<<y.b<<endl;
    cout<<"The sum of both values = "<<r<<endl;
}
void main()
{
  clrscr();
  A obj1;
  B obj2;
  show(obj1,obj2);
  getch();
}
```

Output:
The value of class A object = 10
The value of class B object = 20
The sum of both values = 30

Freebooks.pk

How above Program Works?

The above class declares two classes A and B. Each class contains one data member. The program declares a separate function **show()** that accepts the objects of both classes and displays the sum of the data members in these objects. The function must be a friend function of both classes in order to perform this task. The program declares the following statement in both classes:

```
friend void show(A, B);
```

The above statement declares tells the compiler that the function is friend function of the classes. It is allowed to access the private and protected data members of these classes. The program also specifies the prototype of the class B before class A as follows:

```
class B;
```

The above declaration is important to specify because a class cannot be referenced before is has been declared. The class B is referenced in the declaration of friend function **show()** in class A. Therefore, the above class declaration is necessary. Otherwise the compiler will generate an error.

13.11 Friend Classes

A type of class all of whose member functions are allowed to access the private and protected members of a particular class is called **friend class**. Normally, the private and protected members of any class cannot be accessed from outside the class. In some situations, a program may require to access these members. The use of friends classes allows a class to access these members of another class. A class that is declared in another class with **friend** keyword becomes the friend of that class.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  private:
  int a, b;
  public:
  A()
  {
    a = 10;
    b = 20;
  }
  friend class B;
};
class B
{
  public:
  void showA(A obj)
  {
    cout<<"The value of a: "<<obj.a<<endl;
  }
  void showB(A obj)
  {
    cout<<"The value of b: "<<obj.b<<endl;
```

Output:
The value of class a: 10
The value of class b: 20

Freebooks.pk

```
    }
};
void main()
{
    A x;
    B y;
    y.showA(x);
    y.showB(x);
    getch();
}
```

How above Program Works?

The above program declares two classes A and B. The class B is a friend class of A. It means that all member functions of class B can access private and protected data members of class A. The class B declares two member functions **showA()** and **showB()**. Both functions accepts an object of class A as parameter and displays the value of one of its data member.

13.12 Static Functions

A type of member function that can be accessed without any object of the class is called **static function**. Normally, a member function of any class cannot be accessed or executed without creating an object of that class. In some situations, a member function has to be executed without referencing any object.

The static data members of a class are not created for each object. The class creates only one data member for all objects. The static data member is defined when the program is executed. The program may require to access a static data member before creating an object. The static member functions can be used to access a static data member.

Example

```
#include <iostream.h>
#include <conio.h>
class Test
{
    private:
    static int n;
    public:
    static void show()
    {
        cout<<"n = "<<n<<endl;
    }
};
int Test::n = 10;
void main()
{
    Test::show();
    getch();
}
```

Output:
n = 10

How above Program Works?

The above program declares a class Test with a static data member **n**. The following statement defines the data member with an initial value of 10:

```
int Test::n = 10;
```

The static data member exists in the memory even before creating any object. The program also declares a static member function **show()** that displays the value of **n**. The program calls the static member function without creating any object of the class as follows:

```
Test::show();
```

Program 13.17

Write a program that counts the number of objects created of a particular class. The program must be able to display the result even if no object is created so far.

```
#include <iostream.h>
#include <conio.h>
class yahoo
{
  private:
  static int n;
  public:
  yahoo()
  {
    n++;
  }
  static void show()
  {
    cout<<"You have created "<<n<<" objects so far."<<endl;
  }
};
int yahoo::n = 0;
void main()
{
  clrscr();
  yahoo::show();
  yahoo x, y;
  x.show();
  yahoo z;
  x.show();
  getch();
}
```

Output:
You have created 0 objects so far.
You have created 2 objects so far.
You have created 3 objects so far.

How above Program Works?

The above program declares a static data member **n** to count the number of objects that have been created. The following statement defines the variable:

```
int yahoo::n = 0;
```

The above statement defines the variable and initializes it to 0 value. The above program creates three objects **x**, **y** and **z**. Each time an object is created, the constructor is executed that increases the value of **n** by 1.

The program also declares a static member function **show()** to displays the value of data member. It can be accessed directly by using class name followed by **scope resolution operator : :** and function name as follows:

```
yahoo::show();
```

Freebooks.pk

Programming Exercises

1. Write a class Player that contains attributes for the player's name, average and team. Write three functions to input, change and display these attributes. Also write a constructor that asks for input to initialize all the attributes.
2. Define a class for a bank account that includes the following data members:

- Name of the depositor
- Account Number
- Type of account
- Balance amount in the account

The class also contains the following member functions:

- A constructor to assign initial values
- Deposit function to deposit some amount. It should accept the amount as parameter.
- Withdraw function to withdraw an amount after checking the balance. It should accept the amount as parameter.
- Display function to display name and balance.

3. Create two classes DM and DB to store the value of distances. DM stores distances in metres and centimetres and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. **Hint:** Use friend function.
4. Write a class Run that contains the following data members:

- The name of the runner
- The distance covered by a runner.

The class has the following member functions:

- Get function to input runner name and distance.
- Show function to displays runner name and distance.

The user should be able to show the name of the runner who has covered the longest distance at any point of time. **Hint:** Use static data members.

5. Write a class Car that contains the following attributes:

- The name of car
- The direction of car (E, W, N, S)
- The position of car (from imaginary zero point)

The class has the following member functions:

- A constructor to initialize the attributes.
- Turn function to change the direction of car to one step right side (e.g. if the direction is to E, it should be changed to S and so on.)
- Overload the Turn function to change the direction to any side directly. It should accept the direction as parameter.
- Move function to change the position of car away from zero point. It should accept the distance as parameter.

Freebooks.pk

Exercise Question

Q.1. Define the terms class and object. How are they related to each other?

A class is the set-up or the definition for an object. A class describes a type of state and behavior. An object is a particular instance of a class. For example, a Telephone class can describe that a telephone has a volume that could be changed, a color, and a telephone can be turned on and off. A Telephone object is something that could actually be used and be turned on and off.

Q.2. Briefly describe the role of each of the following within a class:

a) Instance variables

Answer: They describe the state or properties of an object.

b) Instance methods

Answer: They provide the behavior of an object – things that can be done with or to that object.

c) Constructors

Answer: They initialize an object's instance variables.

Q.3. What is abstraction?

Abstraction is a process of examining certain aspects of a problem. The user can focus on an object and its functionality before deciding how it should be implemented. An abstract specification tells us what an object does independent of how it works.

Q.4. What is data abstraction?

Data abstraction is a process of identifying properties related to a particular entity as relevant to the application. Data abstraction is helpful in grouping related information. This can help define operations that can be performed on the data. Data abstraction ignores the way data is represented in memory and helps us think in terms of what operations can be performed on data.

Q.5. What is procedure abstraction?

A procedural abstraction describes what a procedure does without saying anything about how it does it. The abstraction ignores irrelevant details and makes it easier to understand program. It means that implementation of procedure can be ignored and that focus is on arguments and return value of procedure. Replacing an implementation of a procedure does not affect other component of program.

Q.6. What do you know about Encapsulation?

Encapsulation is the grouping of properties and methods within class. It allows selective hiding of properties and methods in a class. The advantage of encapsulation is that a class can have many properties and methods but only some of these are exposed to the user.

Q.7. Consider this list of terms: private, public, static, final, instance variable, instance method, void, argument. For each of the following sentences, give the term that the sentence best describes. Each term can be used once, more than once, or not at all.

1. Indicates that a method does not operate on a particular object.
   static
2. A variable passed into a method when the method gets called.
   argument
3. Represents a characteristic of an object.
   instance variable
4. Indicates that a method can be called from any class.
   public
5. A variable that's declared inside a class, but outside any methods.
   instance variable
6. Indicates that a method doesn't return a value.
   void
7. Indicates that a variable can only be accessed from within the class in which it's declared.
   private
8. The three keywords found in the declaration of the main() method.
   public, static, void

Freebooks.pk

Q.8. Provide a brief description of three access control keywords for classes.

Public: member objects and functions are accessible by any function..

Protected:member objects and functions are accessible by member functions and friends of the class in which it is declared. They are also accessible by member functions and friends of classes derived from the class.

Private:member objects and functions are accessible only by member functions of the class in which they are declared.

Q.9. What is the default for member access, if none is specified?

The default member access is private.

Q.10. How can one calculate the amount of storage taken up by a class definition?

The amount of storage taken up by a class definition cannot be calculated because class definitions do not use any storage.

Q.11. How many destructors can be defined in a class?

A class can have only one destructor.

Q.12. What is the return type of a destructor?

There is no return type, not even void.

Q.13. When would a class destructor be useful?

The destructor is useful for releasing memory used by the object, closing files, printing diagnostic messages etc.

Q.14. When are friend functions necessary?

They are necesssary when a nonmember function needs direct access to the private and protected members of a class.

Q.15. If class A is a friend of class B, which member functions in class A have access to the private members of B?

All member functions in class A have access to the private members of B.

Q.16. What is difference between constructor and destructor?

Constructor is used to construct the object of a class. It is automatically called when an object is created. Destructor is also automatically called when an object is destroyed. It is usually used to release the memory occupied by the objects etc.

Q.17. What is difference between function and member function?

A block of code that is defined with a certain name is called function. It can be executed by calling it by its name. A function that is defined within the scope of a class is called member function.

Q.18. What is friend function and its advantages?

A function that can access the non-public members of a class is called friend function. The advantage of friend function is that it can be used with multiple classes to perform specific task.

Multiple choice

1. **A programming technique in which programs are written on the basis of objects is called:**
   a. Procedural programming. b. Object-oriented programming.
   c. Nonprocedural programming. d. None
2. **A C++ class is similar to:**
   a. Structure b. Header file c. Library function d. None
3. **A category of objects is called:**
   a. Attribute. b. Instance. c. Class. d. Method.
4. **The name for the values of an object's data members is its:**
   a. State b. Status c. Condition d. None

Freebooks.pk

5. **A data item that is contained within an object is called:**
   a. Attribute. b. Instance. c. Class. d. Method
6. **A specific occurrence of an object is called:**
   a. Attribute. b. Instance. c. Class. d. Method
7. **A routine that implements the operations and actions that an object performs is called:**
   a. Attribute. b. Instance. c. Class. d. Method
8. **The feature by which a subclass includes all the properties and methods of the class from which it is derived is called:**
   a. Encapsulation. b. Polymorphism. c. Inheritance. d.None
9. **C++ class contains data members and:**
   a. Methods b. Clients c. interfaces d. None
10. **Class definition.**
   a. Must have a constructor specified b. Must end with a semicolon
   c. Provides the class interface d. Both b and c
11. **Members of a class object are accessed with the:**
   a. dot operator. b. extraction operator d. Insertion operator d. None
12. **Which of the class's members are available to anyone:**
   a. Public b. protected c. private d. final
13. **If you do not declare an access specification, the default for members of a class is:**
   a. Public b. global c. private d. final
14. **Which of the class's members can only be used by its own methods and friends?**
   a. Public b. protected c. private d. final
15. **Examples of access specifiers are the keywords:**
   a. Open and close b. column and row c. private and public d. None
16. **Which of the following provide a mechanism for declaring and initializing objects?**
   a. Constructor functions b. Member functions c. Friend functions d. None
17. **Access to public functions of a class is available to:**
   a. Only public members of class b. any place the class is visible
   c. Private member functions d. only member functions
18. **The constructor for a class is called when:**
   a. A function is called b. An object needs a destructor
   c. possible d. An object is created
19. **The destructor for a class is called automatically whenever:**
   a. An object goes out of scope b. An object comes into scope
   c. new operator is called d. An object becomes a memory leak
20. **Which of the following is a valid header for a Person constructor?**
   a. Person::(int age) b. Person::Person(int age) c. Person.Person(int age) d. Both b and c
21. **The constructor function's return type is**
   a. int b. char c. float d. None
22. **The destructor function's return type is:**
   a. int b. tilde c. float d. destructor have no return type
23. **A class may have this many default constructor(s).**
   a.only one b.more than one c. Any Number d. None
24. **Which of the following is true about a constructor?**
   a. A constructor cannot have parameters b. A constructor has the same name as class
   c. A class can only have a single constructor d. None
25. **This is automatically called when an object is destroyed.**
   a. Constructor function b. destructor function d. Output function d. None

Freebooks.pk

26. **When a member function is defined outside a class declaration, in its header the function name is preceded by the class name and:**
    a. Extraction operator b. constructor function c. Scope resolution operator d. None
27. **Which type of member variable may be accessed before creating any object of the class?**
    a. Inline b. public c. static d. None
28. **Which type of function is not a member of a class, but can access its private members?**
    a. Constructor b. Destructor c. friend d. Static
29. **When the body of a member function is defined inside a class declaration, it is said to be:**
    a. inline b. static c. Conditionally d. None
30. **Which sections of a class can a member function of that class access?**
    a. Private b. Protected c. Public d. All
31. **Which sections of a class can a non-member function access?**
    a. Private b. Protected c. Public d. All
32. **const member functions:**
    a. Mutator functions b. must have a void return type
    c. cannot change its object's data members d. can invoke other member functions in its class

Answers

1. b	2. a	3. c	4. a	5. a	6. b
7. d	8. c	9. a	10. d	11. a	12. a
13. c	14. c	15. c	16. a	17. b	18. d
19. a	20. b	21. d	22. d	23. a	24. b
25. b	26. c	27. c	28. c	29. a	30. d
31. c	32. c				

Fill in the Blanks

1. _____is a programming technique in which programs are written on the basis of objects
2. An _____is a collection of data and functions
3. _____are the action that can be performed by an object
4. The characteristics of an objects are known as its _____.
5. A collection of objects with same properties and functions is known as_____.
6. Each object of a class is known as an _____of its class
7. The keyword _____is used to declare a class.
8. The variables declared in a class are known as _____.
9. The functions declared in a class are called _____.
10. The class declaration always ends with _____.
11. All data members and member functions are declared in the braces known as _____.
12. The commands that are used to specify access level of class members are known as _____.
13. The_____access specifier is used to restrict the use of class member within the class
14. The _____access specifier is used to allow the use of class member within the class as well as outside the class
15. Any member of the class declared with _____access specifier can be accessed from anywhere in the program
16. The process of creating an object of a class is also called _____.
17. The object name and member function are separated by _____.
18. The _____is used in function declarator if the function is defined outside the class

Freebooks.pk

19. A type of member function that is automatically executed when an object of that class is created is known as _____.
20. The process of declaring multiple constructors with same name but different parameters is known as _____.
21. A type of constructor used to initialize an object with another object of same type is called ___.
22. A type of member function that is automatically executed when an object of that class is destroyed is known as _____.
23. The constructor name is preceded by _____.
24. A type of data member that is shared among all objects of class is known as_____.
25. The static data member is defined in the class with _____ keyword
26. A type of function that is allowed to access the private and protected members of a particular class from outside the class is called_____.
27. A function declared in a class with _____keyword becomes friend function of that class.
28. A type of class all of whose member functions are allowed to access the private and protected members of a particular class is called _____.
29. A type of member function that can be accessed without any object of class is called _____.

Answers

1. Object-Oriented programming (OOP)	2. object	3. Functions
4. properties or attributes	5. class	6. instance
7. class	8. member variables or data members.	9. member functions
10. semi colon	11. body of the class	12. access specifers
13. private	14. public	15. public
16. instantiation	17. dot operator	18. scope resolution operator ::
19. constructor	20. constructor overloading	21. default copy constructor
22. destructor	23. tilde sign ~.	24. static data member
25. static	26. friend function	27. friend
28. friend class	29. static function	

True/ False

1. In C++, the scope resolution operator is a colon (:).
2. The constructor function may not accept arguments.
3. The data members of a class are usually placed in the private section of a class t
4. A destructor function can have zero to many parameters.
5. More than one constructor function may be defined for a class.
6. By default, all members in a class are public
7. More than one destructor function may be defined for a class.
8. You must declare all data members of a class before you declare member functions.
9. Only the private section of a class is accessible from the outside
10. A public data member may be declared a friend of a private function.
11. A user can (usually) only access the data of a class by calling its member functions
12. A constructor cannot specify a return type
13. It is possible to declare an entire class as a friend of another class.
14. All member functions must be in the public section of a class

Freebooks.pk

15. A non-static member function may not access a static member variable.
16. The semicolon after the closing brace of a class definition is optional
17. A destructor in a class must be called explicitly before exiting from a program.
18. To access a class member, we use the dot (.) operator.
19. Public function in a class can be called only by functions outside the class.
20. Private data in a class can be accessed only by private functions in the class.
21. We cannot initialize data variables in a class declaration.
22. Private data members of object are only viewable by private member functions of same object.
23. Public data members of object can be changed by any function with access to that object.
24. An object is an instance of a class.
25. Any member of a class is accessible to other members of the class.

Answers

1. F	2. F	3. T	4. F	5. T
6. F	7. F	8. F	9. F	10. F
11. T	12. T	13. T	14. F	15. F
16. F	17. F	18. T	19. F	20. F
21. T	22. F	23. T	24. T	25. T

Freebooks.pk

CHAPTER 14

OPERATOR OVERLOADING

Chapter Overview

14.1 Operator Overloading
- 14.1.1 Overloading an Operator

14.2 Overloading Unary Operators
- 14.2.1 Overloading ++ Operator
- 14.2.2 Operator Overloading with Returned Value
- 14.2.3 Overloading Postfix Increment Operator

14.3 Overloading Binary Operators
- 14.3.1 Arithmetic Operators
- 14.3.2 Overloading Comparison Operator
- 14.3.3 Overloading Arithmetic Assignment Operators

Programming Exercise
Exercise Questions
Fill in the Blanks

Freebooks.pk

14.1 Operator Overloading

The process of defining additional meanings of operators is known as **operator overloading**. It enables an operator to perform different operations depending on the type of operands. It also enables the operators to process the user-defined data types.

The basic arithmetic operators such as +, –, * and / normally work with basic types such as **int**, **float** and **long** etc. The application of these operators with basic types are already defined in the language. However, an error will occur if these operators are used with user-defined objects.

Example

The addition operator is used to add two numeric values. Suppose **a**, **b** and **c** are three integer variables. The following statement will add the contents of **a** and **b** and store the result in variable **c**:

```
c = a + b;
```

The addition operator already knows how to process integer operands. But it does not know how to process two user-defined objects. The above statement will generate error if **a**, **b** and **c** are objects of a class. However, operator overloading can enable the addition operator to manipulate two objects.

14.1.1 Overloading an Operator

An operator can be overloaded by declaring a special member function in the class. The member function uses the keyword **operator** with the symbol of operator to be overloaded.

Syntax

The syntax of overloaing an operator is as follows:

```
return_type operator op ( )
{
  function body;
}
```

return_type It indicates the type of value returned by the member function.

operator It is the keyword that indicates that the member function is used to overload an operator.

op It is the symbol of operator to be overloaded.

Example

```
void operator ++ ( )
{
  function body;
}
```

14.2 Overloading Unary Operators

A type of operator that works with single operand is called **unary operator**. The unary operators are overloaded to increase their capabilities.

The unary operators that can be overloaded in C++ are as follows:

```
+     –     *     !     ~     &

++    – –   ( )   ->    new   delete
```

Freebooks.pk

14.2.1 Overloading ++ Operator

The increment operator ++ is a unary operator. It works with single operand. It increases the value of operand by 1. It only works with numerical values by default. It can be overloaded to enable it to increase the values of data members of an object in the same way.

Program 14.1

Write a program that overloads increment operator to work with user-defined objects.

```
#include <iostream.h>
#include <conio.h>
class Count
{
  private:
  int n;
  public:
  Count()
  {
    n = 0;
  }
  void show()
  {
    cout<<"n = "<<n<<endl;
  }
  void operator ++()
  {
    n = n + 1;
  }
};
void main()
{
  Count obj;
  obj.show();
  ++obj;
  obj.show();
  getch();
}
```

Output:
n = 0
n = 1

How above Program Works?

The above program overloads the increment operator ++ to work with the objects of all user-defined class **Count**. It increases the value of data member **n** by 1. The user can use the same format to increase the value of object as used with integers. The above overloading only works in prefix notation.

14.2.2 Operator Overloading with Returned Value

The increment operator can be used in assignment statement to store the incremented value in another variable. Suppose a and b are two integers. the following statement will increment the value of **a** by 1 and then assign the new value to **b**:

```
b = ++a;
```

The above functionality can be assigned to the operator by returning the new value from the member function. The returned value can be stored in another object of the same type on the left side of the assignment operator.

Freebooks.pk

Program 14.2

Write a program that overloads increment operator to work with user-defined objects. The overloaded function should return an object after incrementing the data member.

```
#include <iostream.h>
#include <conio.h>
class Count
{
  private:
  int n;
  public:
  Count()
  {
    n = 0;
  }
  void show()
  {
    cout<<"n = "<<n<<endl;
  }
  Count operator ++()
  {
    Count temp;
    n = n + 1;
    temp.n = n;
    return temp;
  }
};
void main()
{
  clrscr();
  Count x, y;
  x.show();
  y.show();
  y = ++x;
  x.show();
  y.show();
  getch();
}
```

Output:
n = 0
n = 0
n = 1
n = 1

How above Program Works?

The above program overloads increment operator. The member function increments the value of data member **n**. It stores the increment value in temporary object and returns the object. It allows the user to use the increment operator in an assignment operator.

14.2.3 Overloading Postfix Increment Operator

The increment operator works in two notation i.e. prefix notation and postfix notation. The operator has to be overloaded separately for both notations.

Program 14.3

Write a program that overloads postfix increment operator to work with user-defined objects.

Freebooks.pk

```
#include <iostream.h>
#include <conio.h>
class Count
{
  private:
   int n;
   public:
   Count()
   {
     n = 0;
   }
   void show()
   {
     cout<<"n = "<<n<<endl;
   }
   Count operator ++()
   {
     Count temp;
     n = n + 1;
     temp.n = n;
     return temp;
   }
   Count operator ++(int)
   {
     Count temp;
     n = n + 1;
     temp.n = n;
     return temp;
   }
};
void main()
{
  Count x;
  x.show();
  ++x;
  x++;
  x.show();
  getch();
}
```

Output:
n = 0
n = 2

How above Program Works?

The above program overloads increments operator for both prefix and postfix notation. The keyword **int** in following statement indicates that operator is overloaded for postfix:

Count operator ++(int)

The use of **int** in parenthesis is not an integer parameter. It is simply a flag to compiler that indicates that the operator is overloaded for postfix notation.

14.3 Overloading Binary Operators

A type of operator that works with two operands is called **binary operator**. The binary operators are overloaded to increase their capabilities.

Freebooks.pk

The binary operators that can be overloaded in C++ are as follows:

+	–	*	/	%	&	|	^	<<	>>
==	+=	– =	/=	%=	&=	|=	^=	<<=	>>=
>	<	<=	>=	&&	||	[]	( )		

14.3.1 Arithmetic Operators

The binary arithmetic operators such as +, –, *, / are binary operators. They work with two operands. These operators can be overloaded to enable them to work with user-defined object in the same way as they work with basic types such as **int**, **long** and **float** etc.

Program 14.4

Write a program that overloads binary addition operator +.

```
#include <iostream.h>
#include <conio.h>
class Add
{
  private:
  int a, b;
  public:
  Add()
  {
    a = b = 0;
  }
  void in()
  {
    cout<<"Enter a: ";
    cin>>a;
    cout<<"Enter b: ";
    cin>>b;
  }
  void show()
  {
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
  }
  Add operator +(Add p)
  {
    Add temp;
    temp.a = p.a + a;
    temp.b = p.b + b;
    return temp;
  }
};
void main()
{
  Add x, y, z;
  x.in();
  y.in();
  z = x + y;
  x.show();
  y.show();
```

Output:

```
Enter a: 10
Enter b: 20
Enter a: 5
Enter b: 3
a = 10
b = 20
a = 5
b = 3
a = 15
b = 23
```

Freebooks.pk

```
    z.show();
    getch();
  }
```

How above Program Works?

The above program overloads the binary addition operator. When the binary operator is used with objects, the left operand acts as calling object and the right operand acts as parameter passed to the member function.

```
z = x + y;
```

In the above statement, **x** is the calling object and **y** is passed as parameter. The member function adds the values of calling object and parameter object. It stores the result in a temporary object and then returns the temporary object. The retuned object is copied to the object on the right side of assignment operator in **main()** function.

Program 14.5

Write a program that overloads arithmetic addition operator + for concatenating two string values.

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
class String
{
  private:
  char str[50];
  public:
  String()
  {
    str[0] = '\0';
  }
  void in()
  {
    cout<<"Enter string: ";
    gets(str);
  }
  void show()
  {
    cout<<str<<endl;
  }
  String operator + (String s)
  {
    String temp;
    strcpy(temp.str, str);
    strcat(temp.str, s.str);
    return temp;
  }
};
void main()
{
  String s1, s2, s3;
  s1.in();
```

Output:

```
Enter string: Hello
Enter string: World
s1 = Hello
s2 = World
s3 =
Concatenating s1 and s2 in s3...
s1 = Hello
s2 = World
s3 = HelloWorld
```

Freebooks.pk

```
    s2.in();
    cout<<"s1 = ";
    s1.show();
    cout<<"s2 = ";
    s2.show();
    cout<<"s3 = ";
    s3.show();
    cout<<"Concatenating s1 and s2 in s3..."<<endl;
    s3 = s1 + s2;
    cout<<"s3 = ";
    s3.show();
    getch();
}
```

14.3.2 Overloading Comparison Operator

The comparison operators are binary operators. These operators are frequently used to compare the values of basic data types. Suppose **a** and **b** are two integers and the following statement is executed:

```
a == b;
```

The above statement will return **true** if the values of **a** and **b** are equal and **false** otherwise. The comparison operator == cannot work with user-defined objects. But it can be overloaded in order to use it with user-defined objects.

Program 14.6

Write a program that overloads the comparison operators == to work with **String** class. The result of comparison must be 1 if two strings are of same length and 0 otherwise.

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
class String
{
  private:
  char str[50];
  public:
  String()
  {
    str[0] = '\0';
  }
  void in()
  {
    cout<<"Enter string: ";
    gets(str);
  }
  void show()
  {
    cout<<str<<endl;
  }
  int operator == (String s)
  {
    if(strlen(s.str) == strlen(str))
```

Output:
Enter string: Hello
Enter string: World
s1 = Hello
s2 = World
Both strings are of equal length.

Freebooks.pk

```
        return 1;
        else
        return 0;
    }
};
void main()
{
    clrscr();
    String s1, s2;
    s1.in();
    s2.in();
    cout<<"s1 = ";
    s1.show();
    cout<<"s2 = ";
    s2.show();
    if(s1==s2)
      cout<<"Both strings are of equal length.";
    else
      cout<<"Both strings are of different length.";
    getch();
}
```

14.3.3 Overloading Arithmetic Assignment Operators

The arithmetic assignment operators are used to increase the value of a variable. Suppose **a** and **b** are two integers and the following statement is executed:

```
a += b;
```

The above statement will add the values of **a** and **b** and assign the result to **a**. The arithmetic assignment operator += cannot work with user-defined objects. But it can be overloaded in order to use it with user-defined objects.

Program 14.7

Write a program that overloads arithmetic assignment operator to work with user-defined objects.

```
#include <iostream.h>
#include <conio.h>
class Read
{
  private:
  int days, pages;
  public:
  Read()
  {
    days = pages = 0;
  }
  void in()
  {
    cout<<"How many days have you read? ";
    cin>>days;
    cout<<"How many pages have you read? ";
    cin>>pages;
  }
```

```
    void show()
    {
      cout<<"You have read "<<pages<<" pages in "<<days<<" days."<<endl;
    }
    void operator += (Read r)
    {
      days = days + r.days;
      pages = pages + r.pages;
    }
};
void main()
{
    clrscr();
    Read r1, r2;
    r1.in();
    r2.in();
    cout<<"\nReading number 1 ..."<<endl;
    r1.show();
    cout<<"\nReading number 2 ..."<<endl;
    r2.show();
    cout<<"\nAdding r1 to r2 using += operator..."<<endl;
    r2 += r1;
    cout<<"\nTotal reading is as follows:"<<endl;
    r2.show();
    getch();
}
```

Output:

```
How many days have you read? 10
How many pages have you read? 300
How many days have you read? 3
How many pages have you read? 150

Reading number 1 ...
You have read 300 pages in 10 days.

Reading number 2 ...
You have read 150 pages in 3 days.

Adding r1 to r2 using += operator...

Total reading is as follows:
You have read 450 pages in 13 days.
```

How above Program Works?

The above program overloads arithmetic assignment operator += to work with objects of **Read** class. The overloading member function accepts an object as a parameter and adds the values of parameter to the calling object as follows:

```
days = days + r.days;
pages = pages + r.pages;
```

In the above statements, **days** and **pages** are the data members of calling object **r2** and **r.days** and **r.pages** are the data members of the parameter object.

Programming Exercise

1. Write a class Time that has three data member hour, minutes and seconds. The class has the following member functions:

- A constructor to initialize the time.
- Show function to show the time.
- Overload ++ operator to increase the time by 1 minute.
- Overload – – operator to decrease the time by 1 minute.

2. Define a class for a bank account that includes the following data members:

- Name of the depositor
- Account Number
- Type of account
- Balance amount in the account

Freebooks.pk

The class also contains the following member functions:

- A constructor to assign initial values
- Deposit function to deposit some amount. It should accept the amount as parameter.
- Withdraw function to withdraw an amount after checking the balance. It should accept the amount as parameter.
- Display function to display name and balance.
- Overload binary + operator that adds the balance of one account to another account. It should accept an object as parameter and add the values of the parameter to the calling object.

3. Write a class Array that contains an array of integers as data member. The class contains the following member functions:

- A constructor that initializes the array elements to -1.
- Input function to input the values in the array.
- Show function to display the values of the array.
- Overload == operator to compare the values of two objects. The overloaded function returns 1 if all values of both objects are same and returns 0 otherwise.

Exercise Questions

Q.1. What is operator overloading?

The process of defining additional meanings of operators is known as operator overloading. It enables an operator to perform different operations depending on the type of operands. It also enables the operators to process the user-defined data types

Q.2. Name at least three examples of unary operators that can be overloaded.

Examples are: unary +, unary -, ++, --, *.

Q.3. When overloading binary + operator, is it possible to declare three parameters?

No, the number and order of parameters cannot be changed from the defaults already in effect for an operator.

Q.4. In "X + Y" expression, which operand invokes the overloaded + operator function?

The variable X invokes the overloaded + operator because the first operand in binary expression is always the one that invokes the function

Fill in the Blanks

1. The process of defining additional meanings of operators is known as
2. The member function uses the keyword _____with the symbol of operator to be overloaded.
3. A type of operator that works with single operand is called _____.
4. The increment operator works in _____ notation.
5. A type of operator that works with two operands is called_____.
6. The member function uses the keyword _____with the symbol of operator to be overloaded.

Answers

1. operator overloading	2. operator	3. unary operator
4. two	5. binary operator	6. operator

Freebooks.pk

CHAPTER 15

INHERITANCE

Chapter Overview

15.1 Inheritance
- 15.1.1 Advantages of Inheritance
- 15.1.2 Categories of Inheritance
- 15.1.3 Protected Access Specifier

15.2 Specifying a Derived Class

15.3 Accessing Members of Parent Class
- 15.3.1 Accessing Constructors of Parent Class

15.3.2 Accessing Member Functions of Parent Class

15.4 Function Overriding

15.5 Types of Inheritance
- 15.5.1 Public Inheritance
- 15.5.2 Protected Inheritance
- 15.5.3 Private Inheritance

15.6 Multilevel Inheritance
- 15.6.1 Multilevel Inheritance with Parameters

15.7 Multiple Inheritance
- 15.7.1 Constructors in Multiple Inheritance
  - 15.7.1.2 Constructors with Parameters
- 15.7.2 Ambiguity in Multiple Inheritance
  - 15.7.2.1 Removing Ambiguity

15.8 Containership

Programming Exercise

Exercise Questions

Multiple Choices

Fill in the Blanks

True/False

Freebooks.pk

15.1 Inheritance

A programming technique that is used to reuse an existing class to build a new class is known as **inheritance**. The new class inherits all the behavior of the original class. The existing class that is reused to create a new class is known as **super class, base class** or **parent class**. The new class that inherits the properties and functions of an existing class is known as **subclass, derived class** or **child class**. The inheritance relationship between the classes of a program is called a **class hierarchy**.

Inheritance is one of the most powerful features of object-oriented programming. The basic principle of inheritance is that each subclass shares common properties with the class from which it is derived. The child class inherits all capabilities of the parent class and can add its own capabilities.

Example

Suppose we have a class named **Vehicle**. The subclasses of this class may share similar properties such as **wheels** and **motor** etc. Additionnaly, a subclass may have its own particular characteristics. For example, a subclass **Bus** may have seats for people but another subclass **Truck** may have space to carry goods.

A class of animals can be divided into sub classes like mammals, amphibians, insects, reptiles. A class of vehicles can be divided into cars, trucks, buses, and motorcycles. A class of shapes can be divided into the subclasses lines, ellipses, boxes.

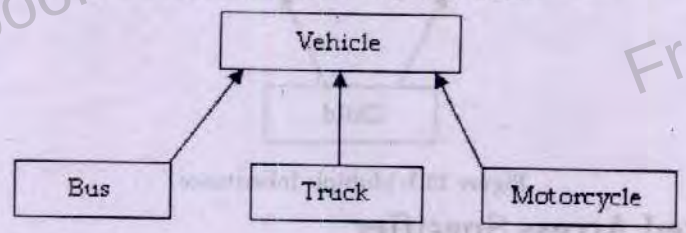


Figure 15.1: Super class and sub classes in inheritance

The above figure shows that **Vehicle** is parent class and **Bus, Truck** and **Motorycycle** are three sub classes. The upward arrows indicate that the subclasses are derived from the parent **Vehicle** class.

15.1.1 Advantages of Inheritance

Some important advantages of inheritance are as follows:

1. Reusability

Inheritance allows the developer to reuse existing code in many situations. A class can be created once and it can be reused again and again to create many sub classes.

2. Saves Time and Effort

Inheritance saves a lot of time and effort to write the same classes again. The reusability of existing classes allows the program to work only on new classes.

3. Increases Program Structure and Reliability

A super class is already compiled and tested properly. This class can be used in a new application without compiling it again. The use of existing class increases program reliability.

Freebooks.pk

15.1.2 Categories of Inheritance

There are two categories of inheritance:

1. Single Inheritance

A type of inheritance in which a child class is derived from single parent class is known as **single inheritance**. The child class in this inheritance inherits all data members and member functions of the parent class. It can also add further capabilities of its own.

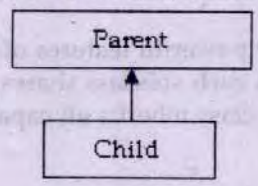


Figure 15.2: Single Inheritance

2. Multiple Inheritance

A type of inheritance in which a child class is derived from multiple parent classes is known as **multiple inheritance**. The child class in this inheritance inherits all data members and member functions of all parent classes. It can also add further capabilities of its own.

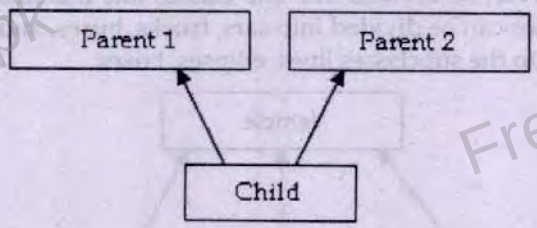


Figure 15.3: Multiple Inheritance

15.1.3 Protected Access Specifier

The **private** data members of a class are only accessible in the class in which they are declared. The **public** data members are accessible from anywhere in the program. The **protected** access specifier is different from **private** and **public** access specifiers. It is specially used in inheritance. It allows a **protected** data member to be acccessed from all derived classes but not from anywhere else in the program. It means that child class can access all protected data members of its parent class.

The difference between **private**, **public** and **protected** access specifiers is as follows:

Access Specifier	Accessible from own class	Accessible from derived class	Accessible from objects outside class
public	Yes	Yes	Yes
protected	Yes	Yes	No
private	Yes	No	No

Table 15.1: Difference between different access specifiers

15.2 Specifying a Derived Class

The process of specifying derived class is same as specifying simple class. Additionally, the reference of parent is specified along with derived class name to inherit the capabilities of parent class.

Syntax

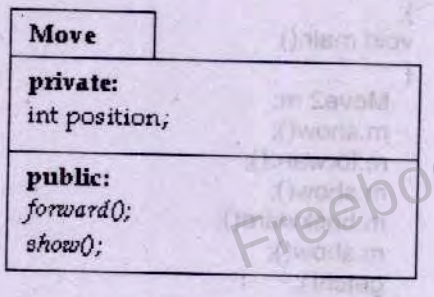
The syntax of specifying a derived class is as follows:

```
class sub_class : specifier parent_class
{
   body of the class
};
```

class	It is the keyword that is used to declare a class.
sub_class	It is the name of derived class.
:	It creates a relationship between derived class and super class.
specifier	It indicates the type of inheritance. It can be private, public or protected.
parent_class	It indicates the name of parent class that is being inherited.

Suppose there is a class **Move** with the following members:

```
class Move
{
   private:
   int position;
   public:
   Move()
   {
      position = 0;
   }
   void forward()
   {
      position++;
   }
   void show()
   {
      cout<<"Position = "<<position<<endl;
   }
};
```



The above class has only one data member **position**. The member function **forward()** increments the value of **position** by 1. The member function **show()** displays its value.

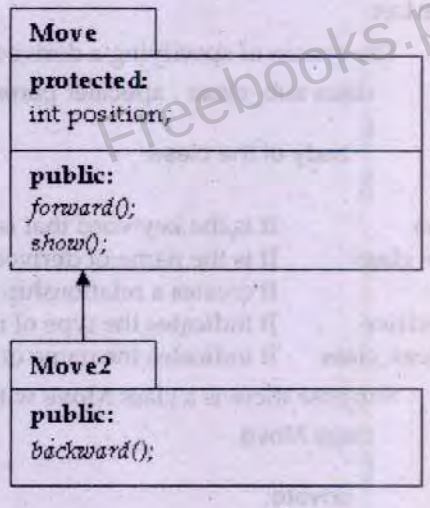
Suppose that a new class is required. The new class should contain a member function to decrement the value of **position**. In this situation, the class **Move** can be inherited in the new class. The new class will inherit the data member and member functions of **Move**. It will then add another member function **backward()** to decrement the value of position variable.

Example

```
#include <iostream.h>
#include <conio.h>
class Move
{
   protected:
   int position;
   public:
   Move()
   {
      position = 0;
   }
```

Freebooks.pk

```
    void forward()
    {
      position++;
    }
    void show()
    {
      cout<<"Position = "<<position<<endl;
    }
  };
  class Move2 : public Move
  {
    public:
    void backward()
    {
      position--;
    }
  };
  void main()
  {
    Move2 m;
    m.show();
    m.forward();
    m.show();
    m.backward();
    m.show();
    getch();
  }
```



Output:
Position = 0
Position = 1
Position = 0

How above Program Works?

The above program declares two classes **Move** and **Move2**. The class **Move** is parent class with data member **position** that is declared as **protected** so that it may be accessed in the dervied class also. The class **Move2** derives **Move** and also declares a member function **backward()** that decrements the value of **position**.

The program declares an object **m** of type **Move2** that is the derived class. The object contains one data member **position** and three member functions **show()**, **forward()** and **backward()**. The data member and two member functions are derived from the parent class and the only member function **bakcward()** is decared in the derived class.

15.3 Accessing Members of Parent Class

An important issue in inheritance is the accessibility of base class members by the objects of derived class. It is known as **accessibility**. The objects of derived class can access certain members of parent class.

15.3.1 Accessing Constructors of Parent Class

The objects of derived class can access certain constructors of parent class. It will use an appropriate constructor from parent class if no constructor is declared in derived class. The program in previous example declares an object **m** of derived class. There is no constructor in derived class. The compiler automatically uses the constructor of parent class. The objects of derived class can automatically access the constructors of parents class if derived class declares no construcotr and parent class has a constructor with no parameter. The derived class can also access constructors of parent class with parameters by passing values to them.

Freebooks.pk

Syntax

The syntax of accessing the constructor of parent class in derived class is as follows:

```
child_con : parent_con(parameters)
{
  body of constructor
}
```

child_con It is the name of constructor of derived class.

parent_con It is the name of constructor of parent class.

parameters It is the list of parameters passed to the constructor of parent class.

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  protected:
  int n;
  public:
  Parent()
  {
    n = 0;
  }
  Parent(int p)
  {
    n = p;
  }
  void show()
  {
    cout<<"n = "<<n<<endl;
  }
};
class Child : public Parent
{
  private:
  char ch;
  public:
  Child() : Parent()
  {
    ch = 'x';
  }
  Child(char c, int m) : Parent(m)
  {
    ch = c;
  }
  void display()
  {
    cout<<"ch = "<<ch<<endl;
  }
};
```

Output:

```
Obj1 is as follows:
n = 0
ch = x

Obj2 is as follows:
n = 100
ch = @
```

Freebooks.pk

```
void main()
{
  clrscr();
  Child obj1, obj2('@', 100);
  cout<<"Obj1 is as follows:\n";
  obj1.show();
  obj1.display();
  cout<<"\nObj2 is as follows:\n";
  obj2.show();
  obj2.display();
  getch();
}
```

How above Program Works?

The above program declares two classes **Parent** and **Child**. Both classes declare two constructors each. One constructor uses no parameter and the second constructor takes parameters for data members. Each constructor in child class also calls the corresponding constructor of the parent class.

The program declares two objects of child class. The **obj1** uses the constructor with no parameter and **obj2** uses the constructor with parameters. The obj2 takes two parameters for data members. The first parameter '@' is used for the data member **ch** declared in child class. The second parameter 100 is used for the data member **n** declared in parent class.

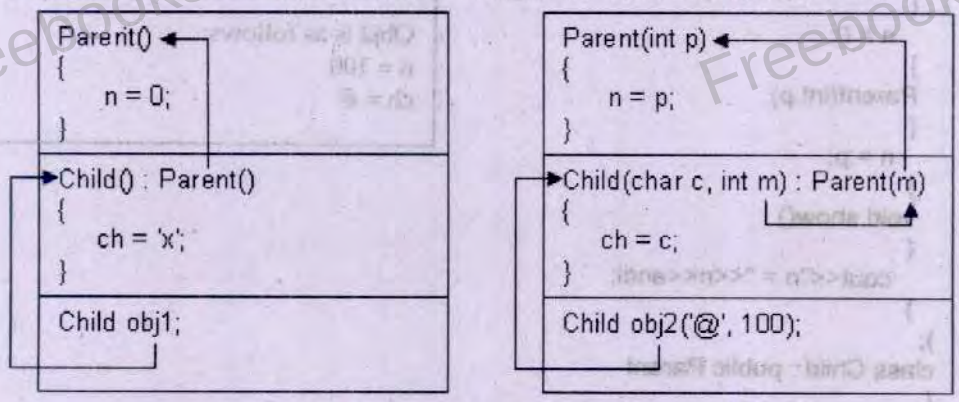


Figure 15.4: Accessing Constructors in Inheritance

15.3.2 Accessing Member Functions of Parent Class

The objects of derived class can access all member functions of parent class that are declared as **protected** or **public**. The object m in the above program contains three member functions **show()**, **forward()** and **backward()**. The first two member functions are declared in parent class and only **backward()** function is declared in derived class.

Program 15.3

Write a class Person that has the attributes of id, name and address. It has a constructor to initialize, a member function to input and a member function to display data members. Create another class Student that inherits Person class. It has additional attributes of roll number and marks. It also has member function to input and display its data members.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
```

```
class Person
{
  protected:
  int id;
  char name[50], address[100];
  public:
  Person()
  {
    id = 0;
    name[0] = '\0';
    address[0] = '\0';
  }
  void GetInfo()
  {
    cout<<"Enter your id: ";
    cin>>id;
    cout<<"Enter your name: ";
    gets(name);
    cout<<"Enter your address: ";
    gets(address);
  }
  void ShowInfo()
  {
    cout<<"\nYour personal information is as follows:\n";
    cout<<"id = "<<id<<endl;
    cout<<"Name = "<<name<<endl;
    cout<<"Address = "<<address<<endl;
  }
};
class Student : public Person
{
  private:
  int rno, marks;
  public:
  Student()
  {
    Person::Person();
    rno = marks = 0;
  }
  void GetEdu()
  {
    cout<<"Enter your roll no: ";
    cin>>rno;
    cout<<"Enter your marks: ";
    cin>>marks;
  }
  void ShowEdu()
  {
    cout<<"\nYour educational information is as follows:\n";
    cout<<"Roll No = "<<rno<<endl;
    cout<<"Marks = "<<marks<<endl;
  }
```

Output:

```
Enter your id: 1
Enter your name: Usman Khalil
Enter your address: Faisalabad
Enter your Roll No: 10
Enter your marks: 786

Your personal information is as follows:
Id = 1
Name = Usman Khalil
Address = Faisalabad

Your education information is as follows:
Roll No = 10
Marks = 786
```

Freebooks.pk

```
};
void main()
{
  clrscr();
  Student s;
  s.GetInfo();
  s.GetEdu();
  s.ShowInfo();
  s.ShowEdu();
  getch();
}
```

How above Program Works?

The above program declares two classes. The **Person** class is a parent class and **Student** class is its derived class. The **Person** class is designed to store and process the information of any person. The **Student** class is designed to store and process the information of a student. A student is also a person, so the **Student** class can inherit **Person** class.

The program declares an object **s** of **Student** class. The object has five data members and four member functions. When the object calls **GetInfo()** member function, the compiler looks in the derived class. The derived class contains no such function, so that compiler automatically uses the function from parent class.

15.4 Function Overriding

The process of declaring member function in derived class with same name and same signature as in parent class is known as **function overriding**.

Function overriding allows the user to use same names for calling the member functions of different class. When a member function is overridden in the derived class, the object of derived class cannot access the function of parent class. However, the function of parent class can be accessed by using scope resolution operator.

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  protected:
  int n;
  public:
  Parent(int p)
  {
    n = p;
  }
  void show()
  {
    cout<<"n = "<<n<<endl;
  }
};
class Child : public Parent
{
  private:
  char ch;
```

```
    public:
    Child(char c, int m) : Parent(m)
    {
      ch = c;
    }
    void show()
    {
      Parent::show();
      cout<<"ch = "<<ch<<endl;
    }
};
void main()
{
  clrscr();
  Child obj('@', 100);
  obj.show();
  getch();
}
```

How above Program Works?

The above program declares two classes. Both classes declares a member function **show()** to display the value of data member. The derived class overrides **show()** function. The object of derived class cannot access this function directly. The object calls the function declared in derived class. The member function in derived class then calls the function of parent class with the following statement:

```
Parent::show();
```

Program 15.5

Write a program that declares two classes. The parent class is called **Simple** that has two data members **a** and **b** to store two numbers. It also has four member functions:

- The **add()** function adds two numbers and displays the result.
- The **sub()** function subtracts two numbers and displays the result.
- The **mul()** function multiplies two numbers and displays the result.
- The **div()** function divides two numbers and displays the result.

The child class is called **Complex** that overrides all four functions. Each function in the child class checks the values of data members. It calls the corresponding member function in the parent class if the values are greater than 0. Otherwise it displays error message.

```
#include <iostream.h>
#include <conio.h>
class Simple
{
  protected:
  int a, b;
  public:
  Simple()
  {
    a = b = 0;
  }
  void in()
  {
```

Output:

```
Invalid values.
Enter a: 10
Enter b: 2
a + b = 12
a – b = 8
a * b = 20
a / b = 5
```

```
    cout<<"Enter a: ";
    cin>>a;
    cout<<"Enter b: ";
    cin>>b;
  }
  void add()
  {
    cout<<"a + b = "<<a+b<<endl;
  }
  void sub()
  {
    cout<<"a - b = "<<a-b<<endl;
  }
  void mul()
  {
    cout<<"a * b = "<<a*b<<endl;
  }
  void div()
  {
    cout<<"a / b = "<<a/b<<endl;
  }
};
class Complex : public Simple
{
  public:
  void add()
  {
    if(a<=0 || b<=0)
   cout<<"Invalid values."<<endl;
    else
   Simple::add();
  }
  void sub()
  {
    if(a<=0 || b<=0)
   cout<<"Invalid values."<<endl;
    else
   Simple::sub();
  }
  void mul()
  {
    if(a<=0 || b<=0)
   cout<<"Invalid values."<<endl;
    else
   Simple::mul();
  }
  void div()
  {
    if(a<=0 || b<=0)
   cout<<"Invalid values."<<endl;
    else
   Simple::div();
  }
```

```
};
void main()
{
  clrscr();
  Complex obj;
  obj.add();
  obj.in();
  obj.add();
  obj.sub();
  obj.mul();
  obj.div();
  getch();
}
```

Program 15.5

Write a base class **Computer** that contains data members of wordSize (in bits), memorySize (in megabytes), storageSize (in megabytes) and speed (in megahertz). Derive a Laptop class that is a kind of Computer but also specifies the object's length, width, height and weight. Member functions for both classes should include a default constructor, a constructor to initializes all components and a function to display data members.

```
#include<iostream.h>
#include<conio.h>
class Computer
{
public:
  Computer() { }
  Computer( int, int, double, int );
  void Show();
protected:
  int    wordSize;     // bits
  int    memorySize;   // megabytes
  double storageSize;  // megabytes
  int    speed;        // megahertz
};
class Laptop: public Computer
{
public:
  Laptop() { }
  Laptop( int, int, double, int, double, double, double, double );
  void Show();
private:
  double length, width, height;
  double weight;
};
Computer::Computer( int wdSiz, int memSiz, double storSiz, int spd )
{
  wordSize = wdSiz;
  memorySize = memSiz;
  storageSize = storSiz;
  speed = spd;
}
```

```
void Computer::Show()
{
   cout<<"Word size: "<<wordSize<<endl;
   cout<<"Memory size: "<<memorySize<<endl;
   cout<<"Speed: "<<speed<<" Mhz"<<endl;
}
Laptop::Laptop( int wdSiz, int memSiz, double storSiz, int spd,
         double len, double wid, double ht, double wt )
   :Computer( wdSiz, memSiz, storSiz, spd )
{
   length = len;
   width = wid;
   height = ht;
   weight = wt;
}
void Laptop::Show()
{
   Computer::Show();
   cout<<"Length: "<<length<<endl;
   cout<<"Width: "<<width<<endl;
   cout<<"Height: "<<height<<endl;
   cout<<"Weight: "<<weight<<endl;
}
void main()
{
   clrscr();
   Computer comp(4, 512, 20, 2);
   Laptop lap(8, 1024, 50, 2, 15, 19, 14, 2);
   cout<<"Computer specification:"<<endl;
   comp.Show();
   cout<<"Laptop specification:"<<endl;
   lap.Show();
   getch();
}
```

15.5 Types of Inheritance

A parent class can be inherited using **private, protected** or **private** type of inheritance. The type of inheritance defines the access status of parent class members in the derived class. Different types of inheritance are as follows:

15.5.1 Public Inheritance

In public inheritance, the access status of parent class members in the derived class remains the same. The **public** members of parent class become **public** members of derived class. The **protected** members of parent class become **protected** members of derived class. The **private** members of parent class become **private** members of derived class.

Syntax

The syntax of defining public inheritance is as follows:

```
class child_class : public parent_class
{
   body of the class
}
```

Freebooks.pk

class It is the keyword that is used to declare a class.
child_class It is the name of derived class.
public It is the keyword that is used to define a public inheritance.
parent_class It is the name of the parent class that inherited.

The accessibility of derived class in public inheritance is as follows:

- The derived class can access the **public** members of parent class.
- The derived class can access the **protected** members of parent class.
- The derived class cannot access the **private** members of parent class.

The accessibility of an object of derived class is as follows:

- The object of derived class can access the **public** members of parent class.
- The object of derived class cannot access the **protected** members of parent class.
- The object of derived class cannot access the **private** members of parent class.

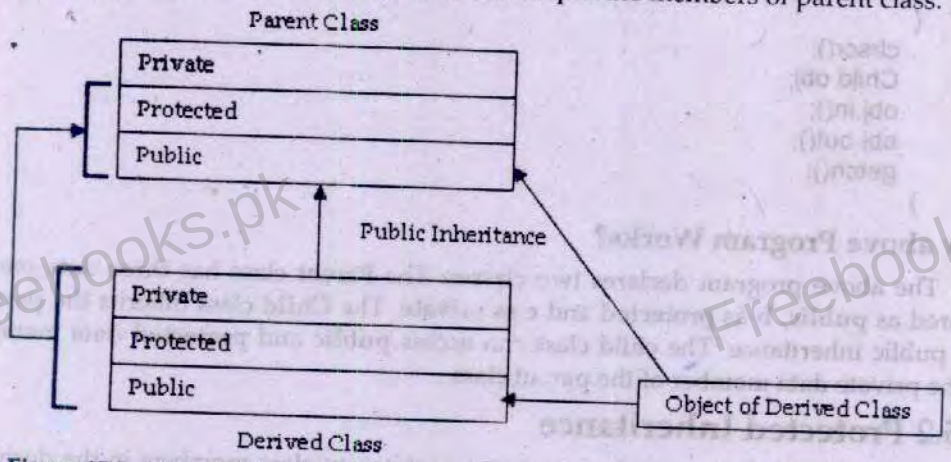


Figure 15.5: Accessibility of Derived Class & Derived Class Object in Public Inheritance

The above figure shows that private, protected and public members of derived class can access the protected and public members of parent class. However, the object of derived class can only access public members of both classes directly.

Program 15.6

Write a program that declares two classes and defines a relationship between them using public inheritance.

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  public:
  int a;
  protected:
  int b;
  private:
  int c;
};
class Child : public Parent
{
  public:
```

```
    void in()
    {
      cout<<"Enter a: ";
      cin>>a;
      cout<<"Enter b: ";
      cin>>b;
    }
    void out()
    {
      cout<<"a = "<<a<<endl;
      cout<<"b = "<<b<<endl;
    }
};
void main()
{
  clrscr();
  Child obj;
  obj.in();
  obj.out();
  getch();
}
```

Output:
Enter a: 10
Enter b: 20
a = 10
b = 20

How above Program Works?

The above program declares two classes. The **Parent** class has three data members a declared as **public**, b as **protected** and c as **private**. The **Child** class inherits the parent class with public inheritance. The child class can access **public** and **protected** data members but not the private data member of the parent class.

15.5.2 Protected Inheritance

In protected inheritance, the access status of parent class members in the derived class is restricted. The **public** members of parent class become **protected** members of derived class. The **protected** members of parent class become **protected** members of derived class. The private members of parent class become **private** members of derived class.

Syntax

The syntax of defining protected inheritance is as follows:

```
class child_class : protected parent_class
{
  body of the class
}
```

class It is the keyword that is used to declare a class.
child_class It is the name of derived class.
protected It is the keyword that is used to define a protected inheritance.
parent_class It is the name of the parent class that inherited.

The accessibility of derived class in protected inheritance is as follows:

- The derived class can access the **public** members of parent class.
- The derived class can access the **protected** members of parent class.
- The derived class cannot access the **private** members of parent class.

The accessibility of an object of derived class is as follows:

- The object of derived class cannot access the **public** members of parent class.

- The object of derived class cannot access the **protected** members of parent class.
- The object of derived class cannot access the **private** members of parent class.

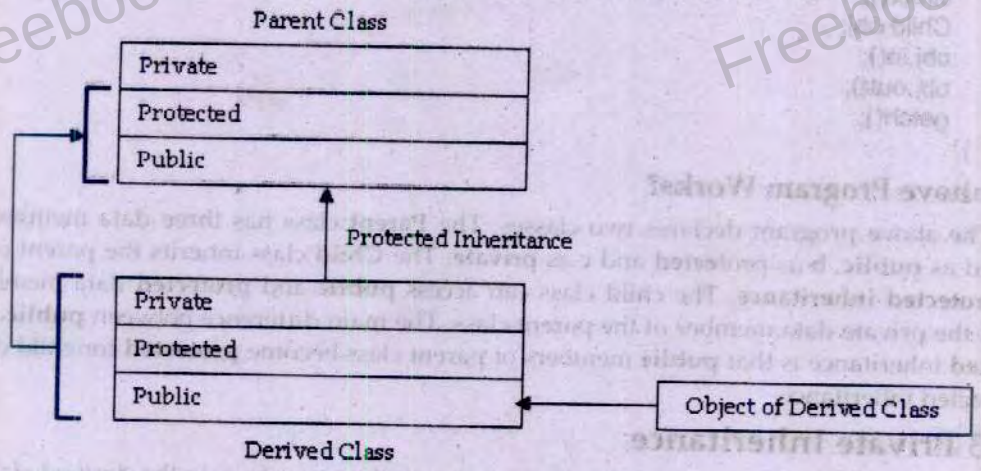


Figure 15.6: Accessibility of Derived Class & Derived Class Object in Protected Inheritance

The above figure shows that private, protected and public members of derived class can access the protected and public members of parent class. However, the object of derived class can only access public members of derived classes directly. The object of parent class cannot access any member of parent class directly.

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  public:
  int a;
  protected:
  int b;
  private:
  int c;
};
class Child : protected Parent
{
  public:
  void in()
  {
    cout<<"Enter a: ";
    cin>>a;
    cout<<"Enter b: ";
    cin>>b;
  }
  void out()
  {
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
  }
```

Output:
Enter a: 50
Enter b: 100
a = 50
b = 100

Freebooks.pk

```
};
void main()
{
  clrscr();
  Child obj;
  obj.in();
  obj.out();
  getch();
}
```

How above Program Works?

The above program declares two classes. The **Parent** class has three data members **a** declared as **public**, **b** as **protected** and **c** as **private**. The **Child** class inherits the parent class with **protected inheritance**. The child class can access **public** and **protected** data members but not the private data member of the parent class. The main difference between **public** and **protected** inheritance is that **public** members of parent class become **protected** for child class in protected inheritance.

15.5.3 Private Inheritance

In private inheritance, the access status of parent class members in the derived class is restricted. The **private**, **protected** and **public** members of parent class all become the **private** members of derived class.

Syntax

The syntax of defining private inheritance is as follows:

```
class child_class : private parent_class
{
  body of the class
}
```

class	It is the keyword that is used to declare a class.
child_class	It is the name of derived class.
private	It is the keyword that is used to define a private inheritance.
parent_class	It is the name of the parent class to be inherited.

The accessibility of derived class in private inheritance is as follows:

- The derived class can access the **public** members of parent class.
- The derived class can access the **protected** members of parent class.
- The derived class cannot access the **private** members of parent class.

The accessibility of an object of derived class is as follows:

- The object of derived class cannot access the **public** members of parent class.
- The object of derived class cannot access the **protected** members of parent class.
- The object of derived class cannot access the **private** members of parent class.

The following figure shows that private, protected and public members of derived class can access the protected and public members of parent class. However, the object of derived class can only access public members of derived classes directly. The object of parent class cannot access any member of parent class directly.

Freebooks.pk

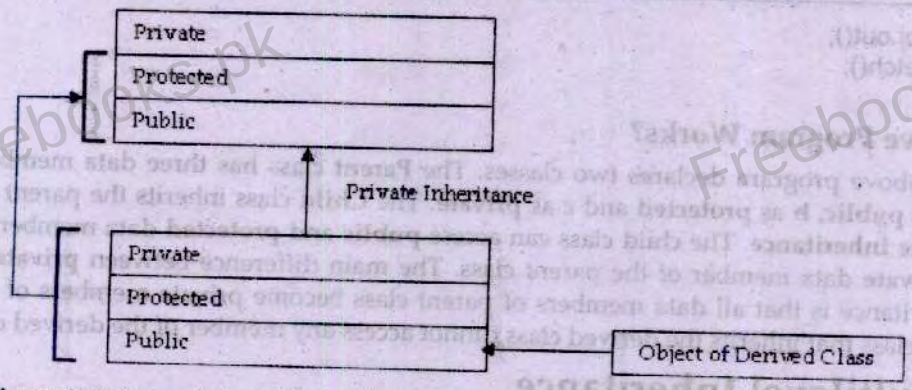


Figure 15.7: Accessibility of Derived Class & Derived Class Object in Private Inheritance

The main difference between public inheritance and private inheritance is that private inheritance is mostly used to restrict the access to base class by the classes which are derived from the child class. In private inheritance, all public and protected members of base class become private members in child class objects. They are not accessible by the functions of the classes which are derived from the child class.

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  public:
  int a;
  protected:
  int b;
  private:
  int c;
};
class Child : private Parent
{
  public:
  void in()
  {
    cout<<"Enter a: ";
    cin>>a;
    cout<<"Enter b: ";
    cin>>b;
  }
  void out()
  {
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
  }
};
void main()
{
  Child obj;
  obj.in();
```

```
    obj.out();
    getch();
}
```

How above Program Works?

The above program declares two classes. The **Parent** class has three data members **a** declared as **public**, **b** as **protected** and **c** as **private**. The **Child** class inherits the parent class with **private inheritance**. The child class can access **public** and **protected** data members but not the **private** data member of the parent class. The main difference between **private** and other inheritance is that all data members of parent class become private members of child class. Any class that inherits the derived class cannot access any member of the derived class.

15.6 Multilevel Inheritance

A type of inheritance in which a class is derived from another derived class is called **multilevel inheritance**. In multilevel inheritance, the members of parent class are inherited to the child class and the members of child class are inherited to the grand child class. In this way, the members of parent class and child class are combined in grand child class.

Example

```
class A
{
  body of the class
};
class B : public A
{
  body of the class
};
class C : public B
{
  body of the class
};
```

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  private:
  int a;
  public:
  void in()
  {
    cout<<"Enter a: ";
    cin>>a;
  }
  void out()
  {
    cout<<"The value of a is "<<a<<endl;
  }
};
```

Freebooks.pk

```
class B : public A
{
  private:
  int b;
  public:
  void in()
  {
    A::in();
    cout<<"Enter b: ";
    cin>>b;
  }
  void out()
  {
    A::out();
    cout<<"The value of b is "<<b<<endl;
  }
};
class C : public B
{
  private:
  int c;
  public:
  void in()
  {
    B::in();
    cout<<"Enter c: ";
    cin>>c;
  }
  void out()
  {
    B::out();
    cout<<"The value of c is "<<c<<endl;
  }
};
void main()
{
  clrscr();
  C obj;
  obj.in();
  obj.out();
  getch();
}
```

Output:

```
Enter a: 10
Enter b: 20
Enter c: 30
The value of a is 10
The value of b is 20
The value of c is 30
```

How above Program Works?

The above program declares three classes. The class A is the parent class and class B is the child class of A. The class C is the child class of B. The program declares an object **obj** of class C. When **obj** calls **in()** function, the control moves to **in()** function declared in class C. The following statement declared in **in()** function of C moves the control to class B:

```
B::in();
```

Similarly, following statement declared in **in()** function of B moves control to class B:

```
A::in();
```

Freebooks.pk

The member function **out()** also executes in the same way. Both **in()** and **out()** member functions of all three classes are overloaded functions.

Program 15.10

Write a class Person that has attributes of id, name and address. It has a constructor to initialize, a member function to input and a member function to display data members. Create 2nd class Student that inherits Person class. It has additional attributes of roll number and marks. It also has member functions to input and display its data members. Create 3rd class Scholarship that inherits Student class. It has additional attributes of scholarship name and amount. It also has member functions to input and display its data members.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
class Person
{
  protected:
  int id;
  char name[50], address[100];
  public:
  Person()
  {
    id = 0;
    name[0] = '\0';
    address[0] = '\0';
  }
  void input()
  {
    cout<<"Enter your id: ";
    cin>>id;
    cout<<"Enter your name: ";
    gets(name);
    cout<<"Enter your address: ";
    gets(address);
  }
  void output()
  {
    cout<<"\nPersonal Information:\n";
    cout<<"id = "<<id<<endl;
    cout<<"Name = "<<name<<endl;
    cout<<"Address = "<<address<<endl;
  }
};
class Student : public Person
{
  private:
  int rno, marks;
  public:
  Student()
  {
    Person::Person();
    rno = marks = 0;
  }
```

Output:

```
Enter your id: 1
Enter your name: Usman Khalil
Enter your address: Faisalabad
Enter your roll no: 10
Enter your marks: 898
Enter scholarship name: HEC Scholarship
Enter scholarship amount: 100000

Personal Information:
id = 1
Name = Usman Khalil
Address = Faisalabad

Educational Information:
Roll No = 10
Marks = 898

Scholarship Information:
Scholarship Name: HEC Scholarship
Scholarship Amount: 100000
```

Freebooks.pk

```
    void input()
    {
      Person::input();
      cout<<"Enter your roll no: ";
      cin>>rno;
      cout<<"Enter your marks: ";
      cin>>marks;
    }
    void output()
    {
      Person::output();
      cout<<"\nEducational Information:\n";
      cout<<"Roll No = "<<rno<<endl;
      cout<<"Marks = "<<marks<<endl;
    }
};
class Scholarship : public Student
{
  private:
  char sname[50];
  long amount;
  public:
  void input()
  {
    Student::input();
    cout<<"Enter scholarship name: ";
    gets(sname);
    cout<<"Enter scholarship amount: ";
    cin>>amount;
  }
  void output()
  {
    Student::output();
    cout<<"\nSchilarship Information:\n";
    cout<<"Scholarship Name: "<<sname<<endl;
    cout<<"Scholarship amount: Rs. "<<amount<<endl;
  }
};
void main()
{
  clrscr();
  Scholarship obj;
  obj.input();
  obj.output();
  getch();
}
```

15.6.1 Multilevel Inheritance with Parameters

The member functions in multilevel inheritance can pass the value to the member functions of parent classes. When a function is overridden, the member function in child class calls the member function in the parent class. It can also send parameter values to the parent class functions.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  private:
  int a;
  public:
  void set(int x)
  {
    a = x;
  }
  void out()
  {
    cout<<"The value of a is "<<a<<endl;
  }
};
class B : public A
{
  private:
  int b;
  public:
  void set(int m, int n)
  {
    A::set(m);
    b = n;
  }
  void out()
  {
    A::out();
    cout<<"The value of b is "<<b<<endl;
  }
};
class C : public B
{
  private:
  int c;
  public:
  void set(int g, int h, int k)
  {
    B::set(g, h);
    c = k;
  }
  void out()
  {
    B::out();
    cout<<"The value of c is "<<c<<endl;
  }
};
void main()
{
  clrscr();
```

Freebooks.pk

```
    C obj;
    obj.set(1,2,3);
    obj.out();
    getch();
}
```

How above Program Works?

The above program declares three classes with multilevel inheritance. The member function **set** in class A accepts one integer to set the value of data member **a**. The member function **set** in class B is overloaded and accepts two integers as parameter. The first parameter is passed to the **set** function of class A and second parameter is used to set the value of data member **b**. The member function **set** in class C is again overloaded and accepts three integers as parameters. The first two parameters are passed to the **set** function of class B and third parameter is used to set the value of data member **c**.

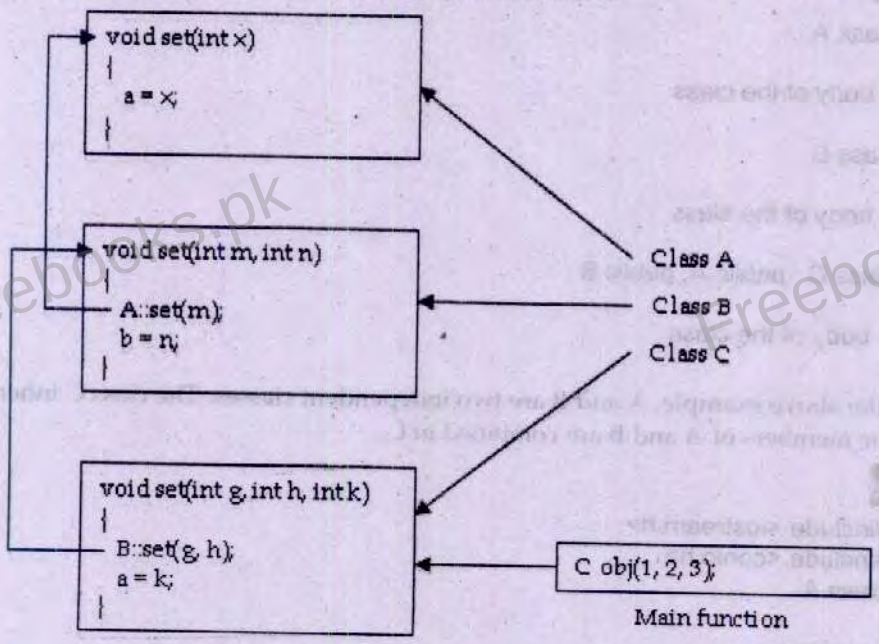


Figure 15.8: Parameter passing in overloaded functions in multilevel inheritance

15.7 Multiple Inheritance

A type of inheritance in which a derived class inherit multiple base classes is known as **multiple inheritance**. In multiple inheritance, the derived class combines the members of all base classes.

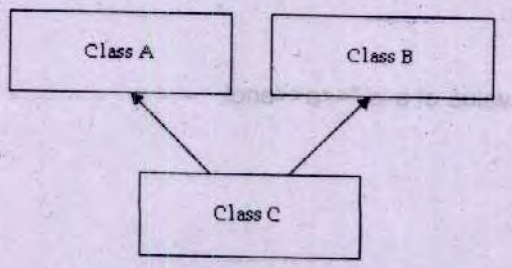


Figure 15.9: Multiple Inheritance

Freebooks.pk

Syntax

The syntax of multiple inheritnace is as follows:

```
class child_class : spec parent_class1, spec parent_class2 ...
{
  body of the class
}
```

class	It is the keyword that is used to declare a class.
child_class	It is the name of derived class.
spec	It is the access specifier that is used to define the type of inheritance.
parent_class1	It is the name of first parent class to be inherited.
parent_class2	It is the name of second parent class to be inherited.

Example

```
class A
{
  body of the class
};
class B
{
  body of the class
};
class C : public A, public B
{
  body of the class
};
```

In the above example, A and B are two independent classes. The class C inherits both A and C. The members of A and B are combined in C.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  private:
  int a;
  public:
  void in()
  {
    cout<<"Enter a: ";
    cin>>a;
  }
  void out()
  {
    cout<<"The value of a is "<<a<<endl;
  }
};
class B
{
  private:
  int b;
  public:
```

Freebooks.pk

```
    void input()
    {
      cout<<"Enter b: ";
      cin>>b;
    }
    void output()
    {
      cout<<"The value of b is "<<b<<endl;
    }
};
class C : public A, public B
{
  private:
  int c;
  public:
  void get()
  {
    A::in();
    B::input();
    cout<<"Enter c: ";
    cin>>c;
  }
  void show()
  {
    A::out();
    B::output();
    cout<<"The value of c is "<<c<<endl;
  }
};
void main()
{
  clrscr();
  C obj;
  obj.get();
  obj.show();
  getch();
}
```

Output:

```
Enter a: 1
Enter b: 2
Enter c: 3
The value of a is 1
The value of b is 2
The value of c is 3
```

How above Program Works?

The above program declares three classes. The classes A and B are independent classes and C is the child class that inherits both A and B. The program declares an object of class C. The program calls **get()** and **show()** functions of class C. The **get()** function of C calls **in()** function of A and **input()** function of B. Similarly, **show()** function of C calls **out()** function of A and **output()** function of B.

15.7.1 Constructors in Multiple Inheritance

In multiple inheritance, the constructors in base classes and child classes are executed when an object of derived class is created. The constructors may accept parameters or they can be without parameters.

Freebooks.pk

15.7.1.1 Constructors without Parameters

The constructors without parameters can be called from derived class by writing a colon after derived class constructor and the name of constructor in parent class.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  public:
  A()
  {
    cout<<"Constructor of class A..."<<endl;
  }
};
class B
{
  public:
  B()
  {
    cout<<"Constructor of class B..."<<endl;
  }
};
class C : public A, public B
{
  public:
  C() : B(), A()
  {
    cout<<"Constructor in class C..."<<endl;
  }
};
void main()
{
  clrscr();
  C obj;
  getch();
}
```

15.7.1.2 Constructors with Parameters

The constructors with parameters can be called from derived class by writing a colon after derived class constructor and the name of constructor in parent class. The required values for the parameters are also provided in call to the parent class constructors.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  private:
  int a;
  public:
```

Freebooks.pk

```
    A()
    {
      a = 0;
    }
    A(int n)
    {
      a = n;
    }
    void showA()
    {
      cout<<"a = "<<a<<endl;
    }
  };
  class B
  {
    private:
    int b;
    public:
    B()
    {
      b = 0;
    }
    B(int n)
    {
      b = n;
    }
    void showB()
    {
      cout<<"b = "<<b<<endl;
    }
  };
  class C : public A, public B
  {
    private:
    int c;
    public:
    C() : B(), A()
    {
      c = 0;
    }
    C(int x, int y, int z) : A(x), B(y)
    {
      c = z;
    }
    void showC()
    {
      A::showA();
      B:showB();
      cout<<"c = "<<c<<endl;
    }
  };
```

Freebooks.pk

```
void main()
{
  C obj(1,2,3);
  obj.showC();
  getch();
}
```

15.7.2 Ambiguity in Multiple Inheritance

An important issue in multiple inheritance is the issue of ambiguity. The ambiguity is created in multiple inheritance if the names of functions are similar in two or more parent classes. The compiler cannot determine which function to execute when the object of derived class attempts to execute such function.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  public:
  void show()
  {
    cout<<"Class A"<<endl;
  }
};
class B
{
  public:
  void show()
  {
    cout<<"Class B"<<endl;
  }
};
class C : public A, public B
{
};
void main()
{
  clrscr();
  C obj;
  obj.show();
  getch();
}
```

How above Program Works?

The above program declares three classes. The class A and class B both has the member function **show()**. The class C simply inherits both classes. The program declares an object of class C. The object has two functions with same name **show()**. One function comes from class A and the second comes from class B. The compiler will generate an error when the above program is compiled.

Freebooks.pk

15.7.2.1 Removing Ambiguity

The ambiguity can be removed from the above program as follows:

```
obj.A::show();
obj.B::show();
```

Another way to remove the ambiguity in multiple inheritance is to overload both functions in the derived class as follows:

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  public:
  void show()
  {
    cout<<"Class A"<<endl;
  }
};
class B
{
  public:
  void show()
  {
    cout<<"Class B"<<endl;
  }
};
class C : public A, public B
{
  public:
  void show()
  {
    A::show();
    B::show();
    cout<<"Class C"<<endl;
  }
};
void main()
{
  C obj;
  obj.show();
  getch();
}
```

15.8 Containership

A technique in which a class contains an object of another class as its member is called **containership**. This type of relationship between classes is also called **has-a** relationship. It means that one class **has** an object of another class as its member.

Freebooks.pk

Example

```
class A
{
};
class B
{
  A obj;
};
```

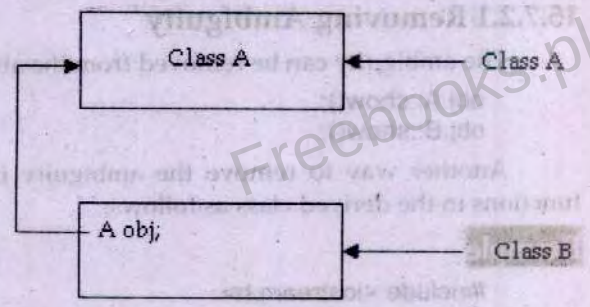


Figure 15.10: Containership

Program 15.16

Write a class Result that has an array of three integers as attribute. It has a member function to input and a member function to display average of array elements. Create another class Student that inherits Result class. It has additional attributes of roll number, name and an object of type Result. It also has member functions to input and display its data members.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
class Result
{
  private:
  int marks[3];
  public:
  void in()
  {
    for(int i=0; i<3; i++)
    {
   cout<<"Enter marks: ";
   cin>>marks[i];
    }
  }
  void show()
  {
    int t = 0;
    cout<<"\nResult Card:\n";
    for(int i=0; i<3; i++)
    {
      cout<<"Marks = "<<marks[i]<<endl;
      t = t + marks[i];
    }
    cout<<"Total Marks = "<<t<<endl;
    cout<<"Average Marks = "<<float(t)/3.0;
  }
};
class Student
{
  private:
  int rno;
  char name[50];
```

```
    Result res;
    public:
    void in()
    {
      cout<<"Enter your Roll No: ";
      cin>>rno;
      cout<<"Enter your name: ";
      gets(name);
      res.in();
    }
    void show()
    {
      cout<<"\nPersonal Information:\n";
      cout<<"Roll No = "<<rno<<endl;
      cout<<"Name = "<<name<<endl;
      res.show();
    }
  };
  void main()
  {
    clrscr();
    Student obj;
    obj.in();
    obj.show();
    getch();
  }
```

Output:

```
Enter your Roll No: 1
Enter you name: Usman Khalil
Enter marks: 70
Enter marks: 80
Enter marks: 92

Personal Information:
Roll No = 1
Name = Usman Khalil

Result Card:
Marks = 70
Marks = 80
Marks = 92
Total Marks = 242
Average Marks = 80.666667
```

Programming Exercises

1. Write a class Employee that contains attributes of employee id and his scale. The class contains member functions to input and show the attribute. Write a child class Manager that inherits Employee class. The child class has attributes of manager id and his department. It also contains the member functions to input and show its attributes.
2. Write a class LocalPhone that contains an attribute phone to store a local telephone number. The class contains member functions to input and display phone number. Write a child class NatPhone for national phone numbers that inherits LocPhone class. It additionally contains an attribute to store city code. It also contains member functions to input and show the city code. Write another class IntPhone for international phone numbers that inherits NatPhone class. It additionally contains an attribute to store country code. It also contains member functions to input and show the country code.
3. Write a class Teacher that contains the attribute teacher name, age and address. It also contains member function to input and display its attributes. Write another class Author Write another class Writer that contains the attributes writer name, address and number of books written by him. It also contains member functions to input and display its attributes. Write a third class Scholar that inherits both Teacher and Writer classes.

Freebooks.pk

4. Write a class Book that contains the attributes BookID, book name and price. It also contains member functions to input and shows its attributes. Write another class Writer that contains the attributes writer name, address and number of books written by him. It contains an array of Book objects as its member. The length of array should be 5 to store the data of five books. It also contains member functions to input and display its attributes.

Exercise Question

Q.1. What do you know about inheritance?

A programming technique that is used to reuse an existing class to build a new class is known as inheritance. The new class inherits all the behavior of the original class. The existing class that is reused to create a new class is known as super class, base class or parent class. The new class that inherits the properties and functions of an existing class is known as subclass, derived class or child class. The inheritance relationship between the classes of a program is called a class hierarchy.

Q.2. What are two Categories of Inheritance?

There two categories of inheritance are single and multiple inheritance. In single inheritance, a child class is derived from single parent class.In multiple inhertiance, a child class is derived from multiple parent classes is known as multiple inheritance.

Q.3. Does a derived class always contain all data members and functions of its base class?

Yes. The derived class always contains all data members and functions of its base class.

Q.4. How does inheritance help in reducing the amount of duplicate code in a program?

The developer does not need to write an existing class again if it is required again and again in different applications. It helps in reducing the amount of duplicate code in programs.

Q.5. Which executes first? constructor of a derived class or constructor of base class?

The base class constructor executes first.

Q.6. Can public inheritance be used to expand access to private or protected members of a base class?

Public inheritance cannot be used to expand access to private or protected members of base class.

Q.7. Can private inheritance be used to restrict access to public or protected members of a base class?

Yes. Private inheritance can be used to restrict access to public or protected members of base class.

Q.8. Define multiple inheritance.

A class may be directly derived from multiple classes

Q.9. Under multiple inheritance, what happens if the same member name appears in two different base classes?

It is important that references to the name must be qualified by the class name such as Student::SetAge and Employee::SetAge etc.

Q.10. What is scope resolution operator?

The scope resolution operator : : is used to access a data member of the parent class from the child class. It is also used to define a member function outside the class.

Q.11. Is it possible to inherit private member in derived class?

No. It is not possible to inherit private member in derived class.

Q.12. Write down two advantages of inheritance.

Inheritance allows to reuse the code. It also saves time in program development.

Freebooks.pk

Multiple Choice

Freebooks.pk

1. **The ability of a class to derive properties from a previously defined class is:**
   a. Encapsulation b. Inheritance c. Polymorphism d. Information hiding
2. **Polymorphism refers to:**
   a. The process of returning data from a function by reference
   b. The specialization of classes through inheritance
   c. The use of classes to represent objects
   d. The packaging of data defining an object as private member variables of a class
3. **Which of the following is the correct syntax for calling a base class constructor in the definition of a derived class constructor?**
   a. DerivedClass::DerivedClass() { BaseClass() ; }
   b. DerivedClass::BaseClass() { DerivedClass() ; }
   c. DerivedClass::DerivedClass() : BaseClass() { }
   d. DerivedClass::BaseClass() : DerivedClass() { }
4. **What is another name for a child class?**
   a. Derived class b. sub class c. Descendent class d. All
5. **Another name for the base class is**
   a. Parent class b. ancestor class c. super class d. All
6. **A class that inherits the members of another class.**
   a. base class b. superclass c. abstract class d. subclass
7. **What does the derived class inherit from the base class?**
   a. Public and protected class members b. public and private class members
   c. Only public data d. Everything
8. **Which of the following members of a base class are never accessible to a derived class.**
   a. Public b. Private c. Protected d. All
9. **In the following statement**
   class Car : protected Vehicle
   which is the derived class?
   a.Car b.Vehicle c. protected d. All
10. **Which of the following class declarations indicates that Ball is a derived class of Sphere?**
    a. Class Sphere: public Ball b. class Ball: public Sphere
    c. Class Ball::Sphere d. class Ball(Sphere)
11. **A subclass inherits all of the following members of its superclass except?**
    a. Constructors and destructor b. public methods
    c. Data fields d. protected methods
12. **A base class may have at most _____ child class derived from it.**
    a. 1 b. 2 c. 12 d. any number
13. **The process of declaring member of function in derived class with same name and same signature as that in parent class is known as:**
    a. Function overriding b. Overloading c. redefintion d. Overwriting
14. **When is the derived class constructor called relative to the base class constructor?**
    a. They are called together b. Depends on kind of derivation
    c. Derived class constructor first d. Base class constructor first
15. **What is normally the first step in defining a constructor for a derived class?**
    a. Instantiate an object of the base class
    b. List the members of the base class that will be inherited
    c. Initialize the member variables of the derived class
    d. None

16. **If a base class has public member functions that are not listed by a derived class, then these functions**
    a. are not available to the derived class
    b. are inherited unchanged in the derived class
    c. do not exist in the derived class
    d. are private to the derived class
17. **From the "main" function, the public functions of a base class can be seen through an object of a derived class under:**
    a. all derivations b. private derivation c. protected derivation d. public derivation
18. **When deriving a class, you should**
    a. List only base class functions that will be redefined
    b. List all the member functions of the base class
    c. Both a and b
    d. Neither a nor b
19. **Class D inherits from class B. Suppose a function f1() is defined for the base class. In what ways can D make use of f1()?**
    a. It can inherit, extend or reject f1()
    b. It can make f1() abstract, virtual or overloaded
    c. It can replace, extend or inherit f1()
    d. It can overload, replace or reject f1()
20. **Given a class A that derives from a class B that derives from a class C, when an object of class A goes out of scope, in which order are the destructors called?**
    a. C, B, then A b. A, B, then C c. unable to determine d. None
21. **If a base class has public member functions that are not listed by a derived class, then these functions**
    a. are not available to the derived class
    b. are inherited unchanged in the derived class
    c. are private to the derived class
    d. do not exist in the derived class
22. **If you have a copy constructor in the base class, but do not have a copy constructor for the derived class, then**
    a. The default constructor is used
    b. A copy constructor for the derived class is automatically created for you
    c. Syntax error will occur
23. **Which of the following are true?**
    a. Constructors of the base class are inherited in the derived class.
    b. You must define constructors in both the base and derived classes
    c. You may not call the base constructor from the derived class
    d. None
24. **When a derived class has two or more base classes, the situation is known as:**
    a. Multiple inheritances b. Polymorphism c. Access specification d. None
25. **C++ uses the member initialization list to decide:**
    a. Order of data member initialization b. Base class constructor –parameters
    c. Which members to initialize d. Nested class constructors
26. **Given the following declarations, which of the following are legal?**

```
class Base
{
  private:
        int x ;
  protected:
        int y ;
```

Freebooks.pk

```
    public:
        int z ;
};
class Derived : public Base
{
  public: Derived() ;
} ;
Derived::Derived()
{
  // What can go in here?
}
```

a. z = 1 ; b. y = 1 ; c. x = 1 ; d. A and B

Answers

1. b	2. b	3. c	4. d	5. d
6. d	7. a	8. b	9. a	10. b
11. a	12. d	13. a	14. d	15. a
16. b	17. d	18. a	19. c	20. a
21. b	22. b	23. d	24. a	25. b
26. d				

Fill in the Blanks

1. A programming technique that is used to reuse an existing class to build a new class is known as _____.
2. The existing class that is reused to create a new class is known as_____.
3. The new class that inherits the properties and functions of an existing class is known as _____.
4. The inheritance relationship between the classes of a program is called a _____.
5. A type of inheritance in which a child class is derived from single parent class is known as _____.
6. A type of inheritance in which a child class is derived from multiple parent classes is known as _____.
7. An important issue in inheritance is the accessibility of base class members by the objects of derived class known as _____.
8. The objects of derived class can access all member functions of parent class that are declared as _____.
9. The process of declaring member function in derived class with same name and same signature as that in parent class is known as _____.
10. A type of inheritance in which a class is derived from a derived class is known as _____.
11. A type of inheritance in which a derived class inherit multiple base classes is known as _____.
12. A technique in which a class contains an object of another class as its member is known as_____.
13. The _____ constructor is called before the _____ constructor.
14. _____ members of a base class are never accessible to a derived class.

Freebooks.pk

15. The base class's _____ affects the way its members are inherited by the derived class.
16. Protected members of a base class are like _____, but they may be accessed by derived classes.

Answers

1. Inheritance	2. Super class, base class or parent class
3. Subclass, derived class or child class	4. Class hierarchy
5. Single inheritance	6. Multiple inheritance
7. Accessibility	8. Protected or public
9. Function overriding	10. Multilevel inheritance
11. Multiple inheritance	12. Containership
13. base, derived	14. Private
15. access specification	16. Private members

True/ False

1. More than one class may be derived from a base class.
2. A derived class may not have any classes derived from it.
3. Base class is also called parent class.
4. The objects of derived class can access all member functions of parent class that are declared as protected or public.
5. A derived class may become a base class, if another class is derived from it.
6. A member function of a derived class may not have the same name as a member function of a base class
7. You may not pass arguments to a base class constructor In an inheritance.
8. The member functions in multilevel inheritance can pass the value to the member functions of parent classes.
9. Constructors, and destructors are not inherited in a derived class; they must be defined in the derived class.
10. In private inheritance, public members in the base class become protected in the derived class.
11. In protected inheritance, the protected members in the base class become public in the derived class.

Answers

1. T	2. F	3. T	4. T
5. T	6. F	7. F	8. T
9. T	10. F	11. F	

Freebooks.pk

CHAPTER 16

POLYMORPHISM AND VIRTUAL FUNCTIONS

Chapter Overview

16.1 Polymorphism
16.2 Pointer to Objects
    16.2.1 Array of Pointers to Objects
16.3 Pointers and Inheritance
16.4 Virtual Functions
    16.4.1 Early Binding
    16.4.2 Late Binding
16.5 Pure Virtual Functions
    16.5.1 Abstract Classes
16.6 Virtual Base Classes

Programming Exercise
Exercise Questions
Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

16.1 Polymorphism

The word **polymorphism** is a combination of two words **poly** and **morphism**. Poly means many and morphism means form. In object-oriented programming, polymorphism is the ability of objects of different types to respond to functions of the same name. The user does not have to know the exact type of the object in advance. The behavior of the object can be implemented at run time. It is called **late binding** or **dynamic binding**. Polymorphism is implemented by using **virtual functions**.

16.2 Pointer to Objects

A pointer can also refer to an object of a class. The member of an object can be accessed through pointers by using the symbol ->. The symbol is known as **member access operator**.

Syntax

The syntax of referencing an object with pointer is as follows:

```
ptr -> member
```

ptr It is the name of the pointer that references an object.

-> It is the member access operator that is used to access a member of object.

member It is the name of the class member to be accessed.

Program 16.1

Write a class with an integer data member, a function to input and a function to display it. Creates an object of the class using pointer and caiss its member functions.

```
#include <iostream.h>
#include <conio.h>
class Test
{
  private:
  int n;
  public:
  void in()
  {
    cout<<"Enter number: ";
    cin>>n;
  }
  void out()
  {
    cout<<"The value of n = "<<n;
  }
};
void main()
{
  clrscr();
  Test *ptr;
  ptr = new Test;
  ptr->in();
  ptr->out();
  getch();
}
```

Output:
Enter number: 100
The value of n = 100

Freebooks.pk

How above Example Works?

The above program declares a class **Test** and a pointer **ptr** of type **Test**. The pointer can refer to an object of the class. The **new** operator creates an object in the memory and stores the address in **ptr**. The program then uses the pointer to call the member functions of the object being referenced by the pointer. The symbol -> is used to access the member of any object through pointer. The use of dot operator with pointer to access any member is invalid.

16.2.1 Array of Pointers to Objects

An array can store same type of data. An array of pointers can store memory addresses of the objects of same class. It allows the user to create a large number of objects in memory using **new** operator and process them easily using loops. Each element of the array will refer to a different object in the memory.

Program 16.2

Write a class that contains an attribute **name**, a function to input and a function to display name. Creates array of pointers in wich each element referes to an object of the class.

```
#include <iostream.h>
#include <conio.h>
class Person
{
  private:
  char name[50];
  public:
  void get()
  {
    cout<<"Enter your name: ";
    cin>>name;
  }
  void show()
  {
    cout<<"Your name = "<<name<<endl;
  }
};
void main()
{
  Person *ptr[5];
  int i;
  for(i=0; i<5; i++)
  {
    ptr[i] = new Person;
    ptr[i]->get();
  }
  for(i=0; i<5; i++)
    ptr[i]->show();
  getch();
}
```

Output:

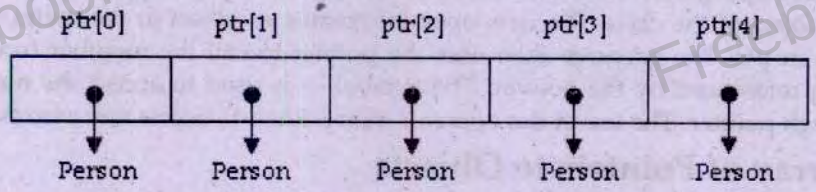
```
Enter your name: Abdullah
Enter your name: Usman
Enter your name: Ejaz
Enter your name: Adnan
Enter your name: Nadeem
Your name = Abdullah
Your name = Usman
Your name = Ejaz
Your name = Adnan
Your name = Nadeem
```

How above Program Works?

The above program declares a class **Person**. It also declares an array **ptr** of five elements to store references of five objects of the class. The first **for** loop uses **new** operator to

Freebooks.pk

create five objects in memory and store their address in array. It also calls the member function **get()** to input names. The second loop calls the **show()** function to display names.



16.3 Pointers and Inheritance

Pointers have very important capability of storing the addresses of different objects. A pointer can store the address of object whose type is same as the type of pointer. It can also store the address of any object that belongs to any child class of the class of pointer. Suppose there are three classes A, B and C. The class A is a parent class whereas B and C are child classes. A pointer **ptr** of class A can store the addresses all objects of A as well as B and C.

The type of pointer does not change when a pointer of parent class refers to an object of child class. That is why, the pointer always executes the member function of parent class even if refers to a child object in the memory.

Example

```
class A
{
};
class B : public A
{
};
class C : public A
{
};
void main()
{
  A ptr;
  ptr = new A;          // Valid statement
  ptr = new B;          // Valid statement
  ptr = new C;          // Valid statement
};
```

The above example shows that a pointer of parent class can also refer to the object of any child class.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  public:
  void show()
  {
    cout<<"Parent class A..."<<endl;
  }
};
```

Freebooks.pk

```
class B : public A
{
  public:
  void show()
  {
    cout<<"Child class B..."<<endl;
  }
};
class C : public A
{
  public:
  void show()
  {
    cout<<"Child class C..."<<endl;
  }
};
void main()
{
  A obj1;
  B obj2;
  C obj3;
  A *ptr;
  ptr = &obj1;
  ptr->show();
  ptr = &obj2;
  ptr->show();
  ptr = &obj3;
  ptr->show();
  getch();
}
```

Output:
Parent class A...
Parent class A...
Parent class A...

How above Program Works?

The program declares three classes A, B and C. The class A is the parent class whereas the classes B and C are child classes. The program declares one object of each class. It also declares a pointer **ptr** of class A. The address of each object is stored in pointer and **show()** function is called. Each time the **show()** function declared in parent class is executed.

The reason is that when a member function is called, the compiler checks the type of pointer. It executes the member function according to the type of pointer. The type of **ptr** is A so the compiler executes **show()** function of class A regardless of the type of object is refers.

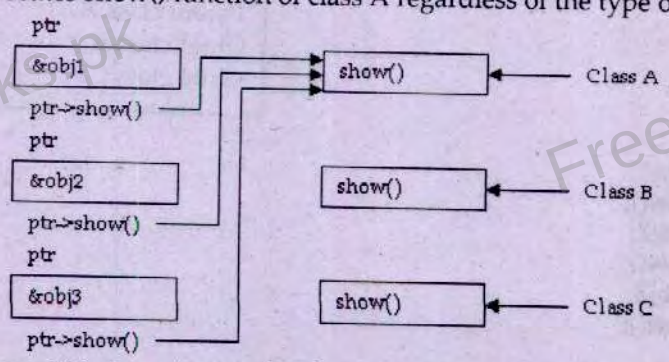


Figure 16.1: Accessing member function with pointers in inheritance

Freebooks.pk

16.4 Virtual Functions

Virtual means existing in effect but not in reality. A type of function that appears to exist in some part of a program but does not exist really is called **virtual function**. Virtual functions are used to implement polymorphism. They enable the user to execute completely different functions by the same function call.

A virtual function is defined in the parent class and can be overridden in child classes. It is defined by using the keyword **virtual**.

Example

```
#include <iostream.h>
#include <conio.h>
class A
{
  public:
  virtual void show()
  {
    cout<<"Parent class A..."<<endl;
  }
};
class B : public A
{
  public:
  void show()
  {
    cout<<"Child class B..."<<endl;
  }
};
class C : public A
{
  public:
  void show()
  {
    cout<<"Child class C..."<<endl;
  }
};
void main()
{
  clrscr();
  A obj1;
  B obj2;
  C obj3;
  A *ptr;
  ptr = &obj1;
  ptr->show();
  ptr = &obj2;
  ptr->show();
  ptr = &obj3;
  ptr->show();
  getch();
}
```

Output:
Parent class A...
Child class B...
Child class C...

Freebooks.pk

How above Program Works?

The program declares three classes A, B and C. The class A is the parent class whereas the classes B and C are child classes. Each class has a member function **show()**. The **show()** function in parent class A is declared as virtual function. The child classes B and C override that member function.

The program declares one object of each class. It also declares a pointer **ptr** of class A. The address of each object is stored in pointer and **show()** function is called. Each time the **show()** function is called, the corresponding function is executed.

The reason is that when a member function is declared as virtual in parent class and is called with pointer, the compiler checks the type of object referred by the pointer. It executes the member function according to the type of object not type of pointer. The type of **ptr** in each call is different so the compiler executes different function each time.

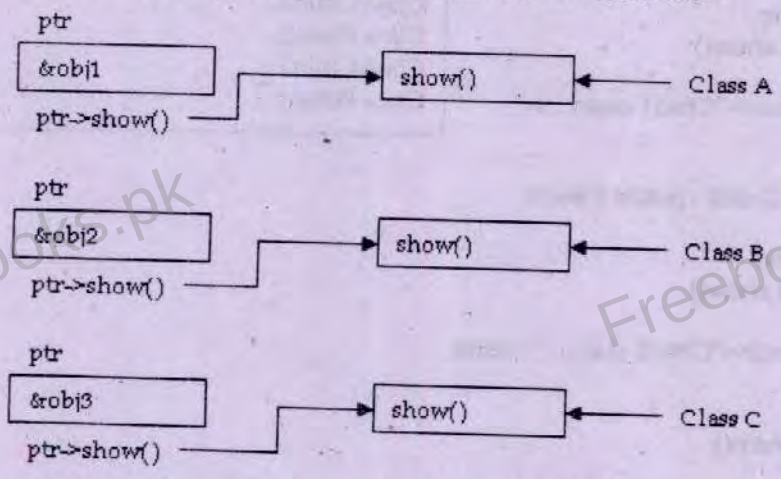


Figure 16.2: Accessing member function with pointers in inheritance with virtual functions

16.4.1 Early Binding

The assignment of types to variables and expressions at compilation time is known as **early binding**. It is also called **static binding**. The early binding occurs when everything required to call a function is known at compile time. Early binding enables the compiler to know exactly which function will be called when a certain statement is executed.

When a program is compiled, the compiler checks the function calls and decides which function is to be executed. This process takes place during compilation process in normal programs with functions. It is an example early binding.

16.4.2 Late Binding

The assignment of types to variables and expressions at execution time is known as **late binding**. It is also called **dynamic binding**. The late binding occurs when some information to call a function is decided at execution time. The compiler does not know at compile time which function will be executed. It provides more flexibility.

The use of virtual function to implement polymorphism is an example of late binding. In virtual functions, the compiler does not know at compile time which object is referred by the pointer. The compiler executes the function depending on the contents of the pointer rather than the type of compiler.

Freebooks.pk

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  public:
  virtual void show()
  {
    cout<<"Parent class...\n";
  }
};
class Child1 : public Parent
{
  public:
  void show()
  {
    cout<<"Child1 class...\n";
  }
};
class Child2 : public Parent
{
  public:
  void show()
  {
    cout<<"Child2 class..."<<endl;
  }
};
void main()
{
  clrscr();
  Parent *ptr[5];
  int op, i;
  cout<<"Enter 1 for Parent, 2 for Child1 and 3 for Child3."<<endl;
  for(i=0; i<5; i++)
  {
    cout<<"Which object to create? ";
    cin>>op;
    if(op==1)
   ptr[i] = new Parent;
    else if(op==2)
   ptr[i] = new Child1;
    else
   ptr[i] = new Child2;
  }
  for(i=0; i<5; i++)
    ptr[i]->show();
  getch();
}
```

Output:

```
Enter 1 for Parent, 2 for Child and 3 for Child2.
Which class to create? 1
Which class to create? 3
Which class to create? 1
Which class to create? 2
Which class to create? 1
Class Parent...
Class Child2...
Class Parent...
Class Child1...
Class Parent...
```

How above Example Works?

The above program declares three classes Parent, Child1 and Child2. It declares an array of five pointers of type Parent. Each element of the array can store the address of an

object of any of three classes. The program inputs number from the user and then creates an object according to the user input and stores the address of newly created object in array. The second loop displays member function **show()** of the objects being referred by the pointers.

```
for(i=0; i<5; i++)
    ptr[i]->show();
```

The compiler cannot decide at compile-time which object each element of the array will be referring. It is decided at run time.

16.5 Pure Virtual Functions

A type of virtual function that has no body is known as **pure virtual function**. A function can be declared as pure virtual function by adding two things:

- The keyword **virtual** at the start of the function declarator
- The **= 0** at the end of function declarator

The pure virtual functions are used in the classes which are not used directly. The user can inherit the class and then override the pure virtual function in the child class.

16.5.1 Abstract Classes

A type of class that contains any pure virtual function is called **abstract class**. An abstract class cannot be used directly. It means that no object of an abstract class can be created. However, a child class can inherit an abstract class and use it by overriding its pure virtual function.

The abstract classes are used to create a model class. Any user who creates a new class by inheriting an abstract class has to override the pure virtual functions. The child class also becomes an abstract class if any of the pure virtual functions is not overridden in child class.

Syntax

The syntax of declaring pure virtual function is as follows:

```
virtual return-type function-name() = 0;
```

virtual	It is the keyword used to declare a virtual function.
Return-type	It indicates the type of value return by the function.
f-name	It indicates the name of the function.
= 0	It indicates that the function is a pure virtual function.

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  public:
  virtual void show()=0;
};
class Child1 : public Parent
{
  public:
  void show()
  {
    cout<<"Child1 class..."<<endl;
  }
```

Output:
Child1 class…
Child2 class…

Freebooks.pk

```
};
class Child2 : public Parent
{
  public:
  void show()
  {
    cout<<"Child2 class..."<<endl;
  }
};
void main()
{
  clrscr();
  Parent *ptr[2];
  ptr[0] = new Child1;
  ptr[1] = new Child2;
  ptr[0]->show();
  ptr[1]->show();
  getch();
}
```

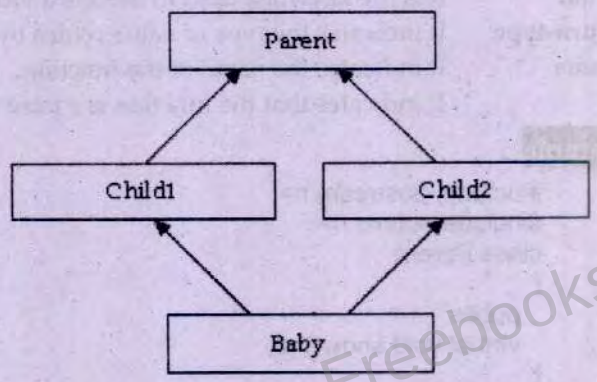
How above Example Works?

The above program declares an abstract class Parent and two child classes. The child classes override the pure virtual function declared in parent class. It is important to declare a pure virtual function in parent class and override it in all child classes to implement polymorphism. A pointer of type Parent will refer to the objects Child1 and Child2 only if these classes are declared as child classes of the Parent class.

16.6 Virtual Base Classes

A class that is inherited by child classes using **virtual** keyword is called **virtual base class**. The virtual base classes are necessary in a situation where the member of base class is duplicated in multilevel inheritance.

Example




```
#include <iostream.h>
#include <conio.h>
class Parent
{
  protected:
  int n;
};
class Child1 : public Parent
{
};
class Child2 : public Parent
{
};
class Baby : public Child1, public Child2
{
  public:
  void set()
  {
    n = 10;
```

```
    cout<<"n = "<<n<<endl;
  }
};
void main()
{
  Baby obj;
  obj.set();
  getch();
}
```

How above Program Works?

The above program declares four classes. The class Parent is the parent classes whereas Child1 and Child2 are its child classes. The class Baby inherits both Child1 and Child2. The only data member **n** declared in Parent is inherited by Child1 and Child2. The class Baby inherits both Child1 and Child2. It receives two copies of same data member **n**. When the program attempts to execute **obj.set()** function, the compiler generates an error. The reason is that the object **obj** contains two copies of data member n that creates ambiguity.

Example

```
#include <iostream.h>
#include <conio.h>
class Parent
{
  protected:
  int n;
};
class Child1 : virtual public Parent
{
};
class Child2 : virtual public Parent
{
};
class Baby : public Child1, public Child2
{
  public:
  void set()
  {
    n = 10;
    cout<<"n = "<<n<<endl;
  }
};
void main()
{
  Baby obj;
  obj.set();
  getch();
}
```

Output:
n = 10

How above Program Works?

The only difference in the above program is that **Child1** and **Child2** classes inherits **Parent** class using **virtual** keyword. It allows these classes to inherit only one copy of the data member **n** to eliminate ambiguity.

Freebooks.pk

Exercise Questions

Q.1. Define polymorphism.

The word polymorphism is a combination of two words poly and morphism. Poly means many and morphism means form. In object-oriented programming, polymorphism is the ability of objects of different types to respond to functions of the same name. The user does not have to know the exact type of the object in advance. Polymorphism is implemented by using virtual functions.

Q.2. What do you know about virtual function?

Virtual means existing in effect but not in reality. A function that appears to exist in some part of a program but does not exist really is called virtual function. Virtual functions are used to implement polymorphism. They enable user to execute completely different functions by same function call.

Q.3. What is an abstract class?

An abstract class is a class that can only be a base class for other classes. You cannot create an instance of an abstract class. Instead, it contains operations that are common to all of its derived classes.

Q.4. When do early binding and late binding occur?

The early binding occurs when everything required to call a function is known at compile time. The late binding occurs when some information to call a function is decided at execution time.

Q.6. What does it mean to override a pure virtual function?

It is a process of creating a function with the same signature as in derived class, thereby replacing the virtual function.

Q.7. What is difference between overloading and overriding?

Function overloading is a process of writing multiple functions with same name but different signature. Function overriding is a process of writing a member function in child class with same name as written in the base class.

Q.8. Given the following class definitions:

```
class B
{
        public:
        int data;
        B()
        {
          data = 0;
        }
        virtual void print()
        {
          cout << "B "
        };
};
class C : public B
{
        int num;
        public:
        C()
        {
          num = 3;
        }
        virtual void print()
        {
          cout << "C ";
        };
}
```

```
class A : public B
{
}
class D : public C
{
        public:
        virtual void print()
        {
            C::print();
            cout << "D ";
        }
};
```

a- Which class is the parent of class A?
class B.

b- Which are the children of class C?
class D.

c- In the following program, write next to each line the value that the line prints, or ERROR if it leads to an error. Do not write anything if the line does not print anything.

```
A var1;  B var2;
var1.print();                    B
C var3;
var3.print();                    C
D var4;
var4.print();                    C D
cout << var3.num << endl;        ERROR accessing private
cout << var2.data << endl;       0
```

Q.9. Answer the questions related to the following code:

```
int total = 0;
class Vehicle
{
        private:
         int id;
        public:
         int get_id ()
         {
            return id;
         }
         void set_id (int newid)
         {
            id = newid;
         }
         Vehicle (int initid)
         {
            id = initid;  total++;
         }
         ~Vehicle ()
         {
            total- -;
         }
};
void main ()
{
   Vehicle a(22), b(33), c(99), x(44), y(33);
```

```
    cout << "Count of vehicles: " << total << endl ;          // LINE 1
    Vehicle *xp;
    {
      Vehicle y(55);
      xp = new Vehicle(77);
      cout << "Count of vehicles: " << total  << endl ;        // LINE 2
    }
    delete xp;
    cout << "Count of vehicles: " << total << endl ;          // LINE 3
}
```

a. Briefly explain why it is not legal for the following statement to appear in the main program?

int foo = xp->id;

The id is a private member of the class Vehicle. If you need to access the private member outside the class, you need to use public function of the class.

b. What is the output produced by LINE 1 of the main program above?

Count of vehicle : 5

c. What is the output produced by LINE 2 of the main program above?

Count of vehicle : 7

d. What is the output produced by LINE 3 of the main program above?

Count of vehicle : 5

Q.10. Answer the questions related to the following code:

```
class X
{
    public:
            void x1();
    protected:
            void x2();
};
class Y: public X
{
    public:
            void y1();
    protected:
            void y2();
};
class Z: public Y
{
    public:
            void z1();
};
void main()
{
    X temp1;        Y temp2;        Z temp3;
}
```

a. Name ALL member functions of all classes visible through temp1 in the main function.

void x1()

b. Name ALL member functions of all classes visible through temp2 in the main function.

void x1() , void y1()

c. Name ALL member functions of all classes visible through temp3 in the main function.

void x1(), void y1(), void z1()

Multiple Choice

1. **Polymorphism refers to:**
   a. Ability to assign multiple meanings to one function name.
   b. Overriding base class functions.
   c. Overloading functions
   d. None
2. **You should make a function a virtual function if:**
   a. Every class that is derived from this class use all the member functions from this class.
   b. Every class that is derived from this class needs to re-define this function.
   c. Only in the derived classes
   d. None
3. **Dynamic binding in C++ occurs at:**
   a. Compile time b. link time c. run time d. None
4. **Dynamic binding is useful for:**
   a. Functions that are overridden- b. functions that are defined once
   c. Functions that are undefined d. none of the above
5. **The compiler performs _____ on virtual functions:**
   a. static binding b. dynamic binding
   c. additional error checking d. None of these
6. **Functions that are dynamically bound by the compiler are:**
   a. Destructor b. Virtual c. Constructor d. None
7. **A _____ of a base class expects to be overridden in a derived class.**
   a. Constructor function b. virtual function c. destructor function d. None

Answers

1. a	2. b	3. c	4. a
5. b	6. b	7. b	

Fill in the Blanks

1. The word polymorphism is a combination of two words _____ and _____.
2. Poly means _____and morphism means _____.
3. _____ is the ability of objects of different types to respond to functions of the same name
4. A type of function that appears to exist in some part of a program but does not exist really is called _____.
5. A virtual function is defined in the _____class and can be overridden in child classes
6. Virtual function is defined by using the keyword _____.
7. The assignment of types to variables and expressions at compilation time is known as _____.
8. Early binding is also called _____.
9. The assignment of types to variables and expressions at execution time is known as _____.
10. Late binding is also called _____.
11. A type of virtual function that has no body is known as _____.

Freebooks.pk

12. A type of class that contains any pure virtual function is called _____.
13. A class that is inherited by child classes using virtual keyword is called _____.

Answers

1. Poly, morphism	2. many, form
3. Polymorphism	4. virtual function
5. Parent	6. Virtual
7. Early binding	8. Static binding
9. Late binding	10. Dynamic binding
11. Pure virtual function	12. Abstract class
13. Virtual base class	

True / False

1. Dynamic binding in C++ occurs at compile time.
2. The compiler performs static binding on virtual functions.
3. Functions that are dynamically bound by the compiler are Virtual function.
4. Polymorphism refers to the ability to assign multiple meanings to one function name.
5. Dynamic binding is also called late binding.
6. Early binding is also called static binding.

Answers

1. F	2. F	3. T	4. T	5. T
6. T				

Freebooks.pk

CHAPTER 17

TEMPLATES

Chapter Overview

17.1 Templates
17.2 Function Templates
17.2.1 Declaring Function Template
17.2.2 Using Function Templates
17.3 Class Templates
17.3.1 Using Class Templates

Programming Exercise
Multiple Choices
True/False

Freebooks.pk

17.1 Templates

Templates are used to define **generic** definition of functions or classes. A template is not related to specific data types and can work with different data types.

There are two types of templates:

- Function templates
- Class templates

17.2 Function Templates

Function templates are used to define **generic** definition of functions. These functions are not related to specific data types and can work with different data types. The technique of function overloading is used to declare different functions with same name. These functions execute similar type of tasks. Function overloading is very useful technique but it increases the code. The user has to declare many functions for same type of tasks.

17.2.1 Declaring Function Template

A function template consists of statements which are independent of particular data types. It is capable of processing different types of data passed to it by the user. For example, a function template can calculate the average of two integers as well as two floating point numbers. It enables the user to write less code.

Syntax

The syntax of declaring a function template is as follows:

```
template <class T>
T FunctionName (T Parameter(s));
```

template	It is the keyword that is used to define a function template.
class	The keyword **class** is used to define the name of general data type.
T	It is the name of general data type. Any letter can be used as the name.
FunctionName	It is the name of the function.
Parameter(s)	It is the list of parameters that will be passed to the function. The type of parameters is the general data type T that is defined in the template.

Example

The following example declares a function template with two parameters. The parameters can be of any data tyep such as integers or floating point numbers:

```
template <class T>
T Max (T val1, T val2)
{
   if(val1 > val2)
      return val1;
   else
      return val2;
}
```

The above example declares a function template **Max** to find and return the maximum of two numbers. **T** denotes an unspecified or generic data type. The function **Max** will compare two numbers of the same type and return the larger number. Both parameters and the return types are of same type **T**.

A type parameter is an identifier whose scope is limited to the function itself. The type parameters always appear inside <>. Each type parameter consists of keyword **class** followed by parameter name. When multiple type parameters are used, they are separated by commas. Each type parameter must be referred in the function prototype.

The following declaration is valid:

```
template <class T1, class T2, class T3>
T3 Relation(T1, T2);
```

The following declaration is not valide because T2 is not used in function prototyp:

```
template <class T1, class T2>
int Compare (T1, T1);
```

The following declaration is also invalid because keyword **class** is missing with T2:

```
template <class T1, T2>                    // illegal! class missing for T2
int Compare (T1, T2);
```

For static and inline functions, the respective keyword must appear after the template clause. The keywords cannot appear before template clause.

The following declaration is valid:

```
template <class T>
inline T Max (T val1, T val2);
```

The following declaration is invalid because keyword **inline** is used before template clause:

```
inline template <class T>
T Max (T val1, T val2);
```

17.2.2 Using Function Templates

A function template represents an algorithm from which executable implementations of the function can be generated by binding its type parameters to particular built-in or user-defined data types.

Examples

The following examples are based on the function template declared above.

```
cout<<Max(19, 5);
```

The above statement will display 19.

```
cout<<Max(10.5, 20.3);
```

The above statement will display 20.3.

```
cout<<Max('a','b');
```

The above statement will display b.

- In the first call to **Max**, both parameters are integers. Therefore, **T** is bound to **int**.
- In the second call, both parameters arguments floating point values. Therefore, **T** is bound to **double**.
- In the third call, both parameters are characters. Therefore, **T** is bound to **char**.

Three functions are generated by compiler to handle above function calls as follows:

```
int Max (int, int);
double  Max (double, double);
char    Max (char, char);
```

Freebooks.pk

When the compiler finds a call to a template function, it attempts to substitute type parameter by examining the type of parameters in the call. It does not attempt any implicit type conversions. Therefore, it cannot resolve the binding of same type parameter to different types. For example:

```
Max(10, 12.6);
```

The above statement will generate and an error because data types of both parameters are different.

Program 17.1

Write a template that accepts two numeric values and displays the maximum number.

```
#include <iostream.h>
#include <conio.h>
template <class Type>
Type Max(Type a, Type b)
{
  if(a>b)
    return a;
  else
    return b;
}
void main()
{
  clrscr();
  int n;
  float m;
  n = Max(10,50);
  cout<<"Maximum of two integers: "<<n<<endl;
  m = Max(3.5,2.2);
  cout<<"Maximum of two floats: "<<m<<endl;
  getch();
}
```

Output:
Maximum of two integers: 50
Maximum of two floats: 3.5

How above Program Works?

The above program declares a function template that accepts two values and returns the maximum of the two. The program calls the function twice. The first call passes two integer values and second call passes two floating point values. The templates works with both function calls.

Program 17.2

Write a function template that accepts a value of any type and then dispalys it.

```
#include <iostream.h>
#include <conio.h>
template <class Type>
void show(Type a)
{
  cout<<a<<endl;
}
void main()
{
  clrscr();
```

Output:
Hello
100
50.75
*

Freebooks.pk

```
    show("Hello");
    show(100);
    show(50.75);
    show('*');
    getch();
}
```

Program 17.3

Write a template for finding minimum value in an array.

```
#include <iostream.h>
#include <conio.h>
template <class T>
T findMin(T arr[],int n)
{
  int i;
  T min;
  min=arr[0];
  for(i=0;i<n;i++)
  {
     if(min > arr[i])
        min=arr[i];
  }
  return(min);
}
void main()
{
  clrscr();
  int iarr[]={5,4,3,2,1};
  char carr[]={'z','y','c','b','a'};
  double darr[]={3.3,5.5,2.2,1.1,4.4};
  cout<<"Generic Function to find Minimum from Array"<<endl;
  cout<<"Integer Minimum is: "<<findMin(iarr,5)<<"\n";
  cout<<"Character Minimum is: "<<findMin(carr,5)<<"\n";
  cout<<"Double Minimum is: "<<findMin(darr,5)<<"\n";
  getch();
}
```

17.3 Class Templates

Class temperates are used to define **generic** definition of classes. These classes are not related to specific data types and can work with different data types. A class template consists of statements which are independent of particular data types. It is capable of processing different types of data passed to it by the user.

Syntax

The syntax of declaring a class template is as follows:

```
template <class T>
class ClassName
{
  class body;
};
```

Freebooks.pk

template	It is the keyword that is used to define a function template.
class	The keyword **class** is used to define the name of general data type.
T	It is the name of general data type. Any letter can be used as the name.
ClassName	It is the name of the class.

17.3.1 Using Class Templates

A class template represents a generic class from which executable implementations of the class can be generated by binding its type parameters to built-in or user-defined types.

Example

```
template <class Type>
class Test
{
  private:
  Type arr[3];
}
```

The above example declares a class template. The declaration can be used easily to create different object that contain arrays of different data types as follows:

```
Test<int>       s1(10);        // Array of integers
Test<double>    s2(10);        // Array of doubles
```

The first statement creates an object of class Test that contains an array of integers. The second statement creates an object of class Test that contains an array of doubles.

Program 17.4

Write a class template that stores five values of any type in an array.

```
#include <iostream.h>
#include <conio.h>
template <class Type>
class Test
{
  private:
  Type arr[3];
  public:
  void input()
  {
    for(int i=0; i<3; i++)
   cin>>arr[i];
  }
  void output()
  {
    cout<<"The values in the array are as follows:\n";
    for(int i=0; i<3; i++)
      cout<<arr[i]<<"\t";
    cout<<endl;
  }
};
void main()
{
  clrscr();
  Test <int> x;
```

Output:

```
Enter three integers:
1
2
3
Enter three characters:
a
b
c
The values in the array are as follows:
1  2  3
The values in the array are as follows:
a  b  c
```

Freebooks.pk

```
    Test <char>y;
    cout<<"Enter three integers: "<<endl;
    x.input();
    cout<<"Enter three characters: "<<endl;
    y.input();
    x.output();
    y.output();
    getch();
}
```

How above Program Works?

The above program declares a class template. It then declares two objects of the class. The object **x** works with integers and the object **y** works with characters. The data type in bracket in declaration statement indicates the type of data used by the class template.

Programming Exercises

1. Write a function template that finds the minimum value in the array and returns it.
2. Write a function template that accepts three parameters and displays them in reverse order.
3. Write a class template that that inputs the index of the array and displays the value in the specified index.

Multiple Choice

1. **Which of the following are valid template prefixes?**
   a. template<class T> b. template<class Me>
   c. template <class T, class Me> d. All
2. **Which of the following is preceded with a function template?**
   a. template b. template <class int> c. template <class T> d. template <void>
3. **Which of the following is a correct template prefix?**
   a. template <class T> b. template <Class > c. <template> class T d. All
4. **Writing a template class:**
   a. Allows the user to skip the implementation of that template class
   b. Allows the user to write one class definition that can hold different data types
   c. Both a and b
   d. None
5. **When is a function template required?**
   a. when implementation details of function are independent of parameters data types
   b. when all functions should be function templates
   c. when two different functions have different implementation details
   d. when two functions have the same type of parameters

Freebooks.pk

Answers

1. d	2. c	3. a	4. b	5. d

True / False

1. In a class template implementation, every use of the class name as the name of the class should be followed by <T>.
2. All classes should be converted to templates.
3. A class template may not use dynamic memory allocation.
4. In a template, all members must be private.
5. Classes can be defined as templates.
6. In a template function definition, all parameters must be of the template class (T).
7. If you define a function template, then the compiler will create a separate function definition for every data type that exists.
8. If your program defines a class template, then the compiler will generate a class for each different data type for which it is instantiated.

Answers

1. T	2. F	3. F	4. T
5. T	6. F	7. F	8. T

Freebooks.pk

CHAPTER 18

FILE HANDLING

Chapter Overview

18.1 Files
- 18.1.1 Advantages of Files
- 18.1.2 Types of Files

18.2 File Access Methods
- 18.2.1 Sequential Access Method
- 18.2.2 Random Access Method

18.3 Stream
- 18.3.1 Types of Streams
- 18.3.2 Predefined Stream objects
- 18.3.3 Stream Class Hierarchy

18.4 Opening Files
- 18.4.1 Default Opening Modes
- 18.4.2 Verifying File Open

18.5 Closing Files

18.6 Formatted Files I/O
- 18.6.1 Writing Data to Formatted Files I/O
- 18.6.2 Reading Data from Formatted Files I/O
- 18.6.3 Detecting End-of-File
- 18.6.4 Reading Lines from Files

18.7 Character I/O
- 18.7.1 Writing Single Character
- 18.7.2 Reading Single Character

18.8 Binary I/O
- 18.8.1 Writing Data in Binary I/O
- 18.8.2 Reading Data in Binary I/O

18.9 Object I/O
- 18.9.1 Writing an Object to Disk
- 18.9.2 Reading an Object from Disk

18.10 Accessing Records Randomly
- 18.10.1 File Pointers
- 18.10.2 The seekg() Function
- 18.10.3 The seekp() Function

18.11 Printing the Files through Streams

Programming Exercise
Exercise Questions
Multiple Choices
Fill in the Blanks
True/False

Freebooks.pk

Freebooks.pk

18.1 Files

A **file** is a collection of related **records**. A record contains data about an entity. A file can be used to save any type of data. Files are stored on secondary storage. The data is stored in files permanently.

Normally, a program inputs data from user and stores it in variables. The data stored in the variables is temporary. When the program ends, all data stored in variables is also destroyed. The user has to input data again from keyboard when the program is executed next time. The data has to be entered each time program is executed that wastes time. The user may also type wrong data at different times.

A data file can be used to provide input to a program. It can also be used to store the output of a program permanently. If a program gets input from a file instead of keyboard, it will get the same data each time it is executed. There will be far less chance of errors.

18.1.1 Advantages of Files

Some important advantages of files are as follows:

- Files can store large amount of data permanently.
- Files can be updated easily.
- One file can be used by many programs for same input.
- Files save time and effort to input data via keyboard.

18.1.2 Types of Files

C++ provides two types of files. This categorization is based on how data is stored in files. Following are two types of files:

1. Text Files

A type of file that stores data as readable and printable characters is called **text file**. A source program of C++ is an example of text file. The user can easily view and read the contents of a text file. It can also be printed to get a hard copy.

2. Binary Files

A type of file that stores data as non-readable binary code is called **binary file**. An object file of a C++ program is an example of binary file. It consists of binary code that is converted by the compiler from source file. The user cannot read the contents of this file. It can only be read and processed by computer.

18.2 File Access Methods

The way in which a file can be accessed is called **file access method**. It depends on the manner in which data is stored in the files. Different file access methods are as follows:

18.2.1 Sequential Access Method

Sequential access method is used to access data in the same sequence in which it is stored in a file. This method reads and writes data in a sequence. In order to read the last record, it has to read all records stored before the last one.

Sequential access files are the simplest way of organizing a file. In sequential access files, the contents are written one after the other. The length of each record is not fixed. It means that each record occupies different amount of memory

The advantage of this method is that the memory is not wasted. The disadvantage is that it take more time to search a record. For example, if the user needs to find the third record, he has to read the first two records also. It is not possible to directly jump to the third record. The reason is that the record length is not fixed. It is not possible to determine the position of the required record in the file. Word processing program usually store files in a sequential format. Similary, the audio tapes are accessed sequentially.

18.2.2 Random Access Method

Random access method is used to access any data item directly without accessing the preceding data. This method does not read or write data in sequence. It is very fast access method as compared to sequential access method. A searching operation with random access method takes far less time to find a data item.

Random access files store data in fixed length records. The main disadvantage of this method is that fixed size of the record is used even if the user wants to store a small sized record. It wastes memory space. Suppose a record of 10 bytes is used to store a complete sentence in a file. If a sentence consists of ony single letter like 'a', it will occupy 10 bytes to store this letter. However, this method speeds up access time because the position of each record can be determined easily as each record takes fixed length. For example, if a record length is fixed as 10 bytes, the fifth record will start at byte number 50. It is easier to jump directly to that location instead of reading the first four records before accessing the fifth.

Database management systems store files in a random access format. Similarly, audio CDs are accessed randomly.

18.3 Stream

A **stream** is a series of bytes associated with a file. It consists of data that is transferred from on location to another. Each **stream** is associated with a specific file by using the **open** operation. The information can be exchanged between a file and the program once a file is opened. This exchange is performed with the help of streams.

18.3.1 Types of Streams

There are two main types of streams i.e. input stream and output stream. The streams are automatically established when a program starts execution. The user can use them to input and output data at any time.

Following are two main streams in C++ language:

1. Standard Input Stream

Standard input stream is used to input data. It establishes a connection between the standard input device and a program. The act of reading data from an input stream is called **extraction**. It is performed using **extraction operator >>** with **cin** object. The **cin** is an object of **istream** and handles input.

2. Standard Output Stream

Standard input stream is used to output data. It establishes a connection between the standard output device and a program. The act of writing data to an output stream is called **insertion**. It is performed using **insertion operator <<** with **cout** object. The **cout** is an object of **ostream** and handles output.

Freebooks.pk

18.3.2 Predefined Stream objects

C++ provides four predefined streams. These streams are opened when C++ program is executed. The predefined streams are as follows:

Stream Name	Used for	Linked to
cin	Standard input	Keyboard
cout	Standard ouput	Monitor
cerr	Standard error	Monitor
clog	Buffered error display	Monitor

Table 18.1: Predefined Stream Objects

- **cin:** It is an object of **istream** class. It is connected to the standard input device such as keyboard.
- **cout:** It is an object of **ostream** class. It is connected to the standard output device such as monitor.
- **cerr:** It is an object of **ostream** class. It is connected to the standard error device. The output through **cerr** is unbuffered. It means that the output will appear immediately on the screen. It is used to inform the user about some error that has occurred.
- **clog:** It is similar to **cerr** object but **clog** is buffered. It means that the output will be held in the buffer till the buffer becomes full or it is flushed.

The standard streams described above can be redirected to other devices or files. The **cerr** object is similar to **cout** object. The difference is that even if **cerr** is redirected by the user to some other device, the error message will be displayed on the screen.

18.3.3 Stream Class Hierarchy

The stream classes are arranged in a hierarchy. C++ provides the following classes to perform output and input of characters with files:

- **ofstream:** This stream class is used to write on files.
- **ifstream:** This stream class is used to read from files
- **fstream:** This stream class is used to both read and write from/to files.

These classes are derived directly or indirectly from the classes **istream** and **ostream**. The **cin** object used for input is an object of class **istream**. The **cout** object used for output is an object of class **ostream**. The stream class hierarchy is as follows:

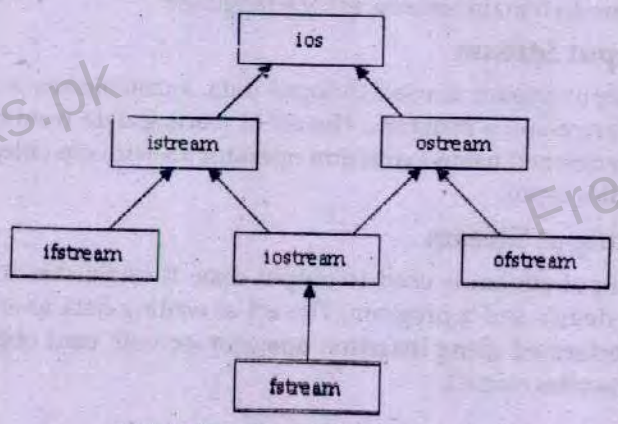


Figure 18.1: Stream Class Hierarchy

18.4 Opening Files

A file should be opened before it can be processed. A file pointer is declared and associated with the file to be opened. A file can be opened by first creating an object of **ifstream**, **ofstream** or **fstream**. The object is then associated with a real file. An open file is represented by a stream object in a program. Any input or output operation performed on this stream object is applied to the physical file associated to it.

Syntax

The member function **open()** of stream object is used to open a file as follows:

```
open(filename, mode);
```

filename It is the name of the file to be opened.

mode It is the mode in which the file is to be opened. It is optional parameter.

Different types of modes for opening a file are as follows:

Mode	Description
ios::in	It is used to open a file for input operations.
ios::out	It is used to open a file for output operations.
ios::binary	It is used to open a file in binary mode.
ios::ate	It is used to set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
ios::app	It is used to perform all output operations at the end of file, appending the content to current content of file. It can only be used in the streams that are opened for output-only operations.
ios::trunc	It is used to delete the previous contents of a file if the file opened for output operations already exists.
ios::nocreate	It is used to open file only if it exists. The file is not created if it does not exist.
ios::noreplace	It is used to open a file only if the file does not exist otherwise open fails.

All of the above flags can be combined using pipe sign (|). For example, the following statements will open the file **example.txt** in binary mode to write data at the end of file.

```
ofstream Test;
Test.open ("test.txt", ios::out | ios::app | ios::binary);
```

The three files **ifstream**, **ofstream** and **fstream** have constructors that automatically call the **open()** member function. The constructors have same parameters as **open()** function. The following format can also be used to open a file:

```
ofstream Test ("test.txt", ios::out | ios::app | ios::binary);
```

The above statement combines object construction and stream opening in a single statement. Both forms of opening a file are valid and equivalent.

18.4.1 Default Opening Modes

The member function **open()** in the classes **ofstream**, **ifstream** and **fstream** uses a default mode to open a file if mode parameter is not used by the user. The default modes of these classes are as follows:

Class	Default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

Freebooks.pk

The default value is used only if the **open()** function is called without any value for the **mode** parameter.

18.4.2 Verifying File Open

The function **is_open()** is used to check if a stream object has opened a file successfully. The function has no parameters and returns a value of **true** if the file is open. It retuns false if the file is not opened.

```
if (!Test)
  cout<<"Error in opening the file.";
```

18.5 Closing Files

The opened file should be closed when the input or output operations on the file are finished. The file should be closed so that its resources become available again. The member function **close()** of stream object is used to close a file. It takes no parameters. The function is used as follows:

```
Test.close();
```

The stream object associated with a file becomes available to open another file when the first file is closed. The file also becomes available again to be opened by other processes. If an object is destroyed while it was associated with an open file, the destructor automatically calls the member function **close()** function.

18.6 Formatted Files I/O

The **formatted I/O** stores data on disk as a series of characters. Each digit or character takes one byte on the disk. For example, the number 38.125 is stored as the characters '3', '8', '.', '1', '2' and '5'. It means that it will occupy six bytes. It is not an efficient way of storing data as it may take a large amount of memory to store simple values. However, it is very easy to implement and is appropriate in some situations.

18.6.1 Writing Data to Formatted Files I/O

The **insertion operator** << is used with **cout** object. It is frequently used in programs to write the output on the screen. The same operator can also be used to write the data in files. The operator is used with a stream object of **ofstream** or **fstream** to write data to files.

Program 18.1

Write a program that writes three character in afile using formatted I/O.

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
void main()
{
  clrscr();
  int n = 10;
  char ch = '*';
  double d = 38.125;
  ofstream file("c:\\test.txt");
  if(!file)
  {
    cout<<"File opening error.";
```

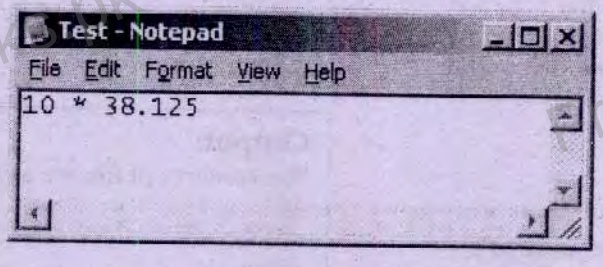
```
        exit(1);
    }
    file<<n<<' '<<ch<<' '<<d;
    file.close();
    getch();
}
```

How above Program Works?

The above program declares a stream object **file** of **ofstream** class. It also initializes the stream object with **test.txt** file on C: drive. The file will be created if it does not exist. If the file exists, the data in the file will be truncated. The program verifies if the file is opened successfully or not. The following statement stores the data in the file:

```
file<<n<<' '<<ch<<' '<<d;
```

The **insertion operator** << is used with stream object **file** to store data in a file. It is used in the same way as it is used with **cout** object to display output on screen. The single blank space is used to separate the one value from the other. It helps extraction operation while reading a file. The blank space tells the extraction operator that one value has ended and the next value has started. Finally, the program closes the file so that it may be used by some other process. The program will store the data in the file as follows:



Program 18.2

Write a program that inputs the names of five cities and stores them in a file **city.txt**.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
void main()
{
  clrscr();
  char city[50];
  ofstream file("c:\\city.txt");
  for(int i=0; i<5; i++)
  {
    cout<<"Enter the name of any city: ";
    cin>>city;
    file<<city<<'\n';
  }
  file.close();
  getch();
}
```

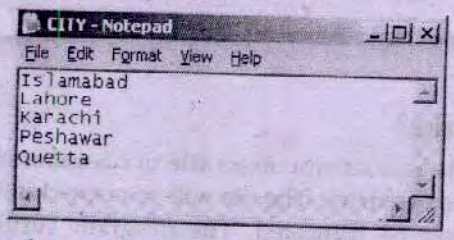
Output:

```
Enter the name of any city: Islamabad
Enter the name of any city: Lahore
Enter the name of any city: Karachi
Enter the name of any city: Peshawar
Enter the name of any city: Quetta
```

Freebooks.pk

How above Program Works?

The above program declares a stream object **file** of **ofstream** class. It inputs the names of five cities and stores them in a file **city.txt** as follows:



18.6.2 Reading Data from Formatted Files I/O

The **extraction operator >>** is used with **cin** object. It is frequently used in programs to read input from the keyboard. It can also be used to read the data from files. It is used with a stream object of **ifstream** or **fstream** to read data from files.

Program 18.3

Write a program that reads the contents of the file **test.txt** and displays on screen.

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
void main()
{
  int n;
  char ch;
  double d;
  ifstream file("c:\\test.txt");
  if(!file)
  {
    cout<<"File opening error.";
    exit(1);
  }
  file>>n>>ch>>d;
  cout<<"The contents of file are as follows: "<<endl;
  cout<<n<<endl<<ch<<endl<<d<<endl;
  file.close();
  getch();
}
```

Output:
The contents of file are as follows:
10
*
38.125

How above Program Works?

The above program declares a stream object **file** of **ifstream** class. It checks if the file **test.txt** exists. It then reads the contents of file in variables **n**, **ch** and **d**. Finally, the program displays the contents on the screen.

18.6.3 Detecting End-of-File

The word **eof** stands for **end of file**. The **eof()** function is used to find if the control has reached the end of file or not. It returns **true (1)** if the control has reached the end of file and returns **false (0)** otherwise. This member function is very useful in displaying all records in the file where the number of records are unknown.

Freebooks.pk

Program 18.4

Write a program that displays all records from **city.txt** prepared in previous example.

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
void main()
{
  char city[50];
  ifstream file("c:\\city.txt");
  if(!file)
  {
    cout<<"Error in opening file.";
    exit(1);
  }
  cout<<"The list of cities is as follows:"<<endl;
  while(!file.eof())
  {
    file>>city;
    cout<<city<<endl;
  }
  file.close();
  getch();
}
```

Output:
Islamabad
Lahore
Karachi
Peshawar
Quetta

How above Program Works?

– The above program uses **eof()** function to detect the end of file. It stores each record in **city** variable and then displays it on the screen.

18.6.4 Reading Lines from Files

The string stored in the files with **insertion operator <<** should not contain any blank space. If the string contains blank space, the string before the space will be stored. The problem can be solved by using **'\n'** after each record and then reading it as complete line. The function **getline()** is used to read a complete line from the file.

Program 18.5

Write a program that stored five lines of strings in a file and then displays them on the screen by reading these lines.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <fstream.h>
void main()
{
  char str[100];
  ofstream out("c:\\strings.txt");
  ifstream in("c:\\strings.txt");
  for(int i=0; i<5; i++)
  {
    cout<<"Enter a string: ";
```

Output:
Enter a string: Usman Khalil
Enter a string: Abdullah Ashfaq
Enter a string: Ali Ahmad
Enter a string: Jamal Nasir
Enter a string: Mahmood Khan
The list of strings is as follows:
Usman Khalil
Abdullah Ashfaq
Ali Ahmad
Jamal Nasir
Mahmood Khan

Freebooks.pk

```
        gets(str);
        out<<str<<'\n';
      }
      out.close();
      cout<<"The list of strings is as follows:"<<endl;
      while(!in.eof())
      {
        in.getline(str,100);
        cout<<str<<endl;
      }
      in.close();
      getch();
    }
```

How above Program Works?

The above program inputs five strings from the user and stores it in the file followed by '\n' character. Each string is stored in a separate file. The program then uses **getline()** function to read the entire line and then displays the line on the screen.

18.7 Character I/O

The data can also be written to files character by character. Similarly, the contents of a file can be read character by character. The **put()** function is used to write single character in a file. The **get()** function is used to read single character from a file.

18.7.1 Writing Single Character

The **put()** function is used to write single character in a file. The syntax of using this function is as follows:

```
obj.put(ch);
```

obj It is the name of stream object that is associated with a file.
put It is the name of a function used to write a character in file.
ch It is the character to be written in the file. It can be a constant or variable.

Program 18.6

Write a program that inputs five characters from the user and stores them in a file.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <fstream.h>
void main()
{
  char ch;
  ofstream out("c:\\chars.txt");
  for(int i=0; i<5; i++)
  {
    cout<<"Enter a character: ";
    cin>>ch;
    out.put(ch);
  }
  out.close();
  getch();
}
```

18.7.2 Reading Single Character

The **get()** function is used to read single character from a file. The syntax of using this function is as follows:

```
obj.get(ch);
```

obj It is the name of stream object that is associated with a file.
get It is the name of a function used to read a character in file.
ch It is the character variable used to store the character read from the file.

Program 18.7

Write a program that reads the characters from a text file and display them on screen.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <fstream.h>
void main()
{
  clrscr();
  char ch;
  ifstream in("c:\\chars.txt");
  while(!in.eof())
  {
    in.get(ch);
    cout<<ch<<endl;
  }
  in.close();
  getch();
}
```

Output:
a
b
c
d
e

Program 18.8

Write a program that reads the characters from a text file. It counts total number of characters and total number of vowels in the file.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <fstream.h>
#include <ctype.h>
void main()
{
  clrscr();
  char ch;
  int t, v;
  t = v = 0;
  ifstream in("c:\\chars.txt");
  while(!in.eof())
  {
    in.get(ch);
    ch = tolower(ch);
    if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
   v++;
    t++;
```

Freebooks.pk

```
        cout<<ch<<endl;
      }
      cout<<"Total characters: "<<t<<endl;
      cout<<"Total vowels: "<<v<<endl;
      in.close();
      getch();
    }
```

18.8 Binary I/O

Binary I/O is used to store data in files in binary format. It is an efficient method to store a large amount data as compared to formatted I/O. For example, the formatted I/O takes 4 bytes to store 1234. On the other hand, binary I/O takes only 2 bytes to store this amount. The reason is that binary I/O stores it as integer value whereas formatted I/O stores it as four characters.

18.8.1 Writing Data in Binary I/O

The data can be stored in binary I/O using **write()** function of any object of **ofstream** or **fstream**. The syntax of this function is as follows:

```
obj.write((char*)&r, sizeof(r));
```

obj It is the name of stream object that is associated with a file.
write It is the name of a function used to write data with binary I/O.
char* It is the address of character data in fixed length bytes.
r It is the name of variable that contains data to be stored in the file. The ampersand is used only if it is a structure variable or an object.
sizeof(r) It calculates the size of r to determine the number of bytes required to store data.

Program 18.9

Write a program that inputs five integers and stores it in a file using binary I/O.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
void main()
{
  clrscr();
  int n;
  ofstream out("c:\\data.txt", ios::binary);
  for(int i=1; i<=5; i++)
  {
    cout<<"Enter an integer: ";
    cin>>n;
    out.write((char*)n, sizeof(n));
  }
  out.close();
  getch();
}
```

Program 18.10

Write a program that uses a structure variable to input records of three students and stores it in **students.txt** in binary I/O. Each record consists of Roll No, Name and Marks.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
struct Student
{
  int rno;
  char name[50];
  int marks;
};
void main()
{
  clrscr();
  Student s;
  ofstream out("c:\\students.txt", ios::binary);
  for(int i=1; i<=3; i++)
  {
    cout<<"Enter your Roll No: ";
    cin>>s.rno;
    cout<<"Enter your name: ";
    cin>>s.name;
    cout<<"Enter your marks: ";
    cin>>s.marks;
    out.write((char*)&s, sizeof(s));
  }
  out.close();
  getch();
}
```

18.8.2 Reading Data in Binary I/O

The data can be read in binary I/O using **read()** function of any object of **ifstream** or **fstream**. The syntax of this function is as follows:

```
obj.read((char*)&r, sizeof(r));
```

obj It is the name of stream object that is associated with a file.

read It is the name of a function used to read data with binary I/O.

char* It is the address of character data in fixed length bytes.

r It is the name of variable that stores data read from the file. The ampersand is used only if it is a structure variable or an object.

sizeof(r) It calculates the size of **r** to determine the number of bytes required to store data.

Program 18.11

Write a program that reads the records stored in **students.txt** file and displays them.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
struct Student
{
  int rno;
  char name[50];
  int marks;
};
```

```
void main()
{
  Student s;
  ifstream in("c:\\students.txt", ios::binary);
  while(!in.eof())
  {
    in.read((char*)&s, sizeof(s));
    cout<<"Roll No: "<<s.rno<<endl;
    cout<<"Name: "<<s.name<<endl;
    cout<<"Marks: "<<s.marks<<endl;
  }
  in.close();
  getch();
}
```

Program 18.12

Write a program that creates a file to store name and email of the user using structure.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
struct email
{
  char name[20];
  char id[30];
};
void main ( )
{
  clrscr();
  email user;
  email check;
  cout<<"Enter a name: ";
  cin>>user.name;
  cout<<"Enter the email address : ";
  cin>>user.id;
  ofstream out("c:/email.txt", ios::out | ios::binary);
  out.write( (char *) &user, sizeof (struct email) );
  out.close( );
  cout<<endl<<"Contents of file are : ";
  ifstream in ("c:/email.txt", ios::in | ios::binary);
  in.read((char *) &check, sizeof(struct email));
  cout<<endl<<check.name;
  cout<<endl<<check.id;
  in.close( );
  getch();
}
```

Output:
Enter a name: Abdullah
Enter the email address: abdullah@itseries.com.pk
Contents of file are:
Abdullah
abdullah@itseries.com.pk

18.9 Object I/O

The object can also be written in and read from the files. The objects are normally written and read in binary mode. It helps the user to use as much memory as is needed to store the contents of an object.

Freebooks.pk

18.9.1 Writing an Object to Disk

The method of writing an object to disk is same as writing a structure with binary I/O. The **write()** method is used for writing an object to a disk.

Program 18.13

Write a program that stores an object to a file **country.txt** using binary I/O.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
class Country
{
  private:
  int id;
  char name[50];
  public:
  void get()
  {
    cout<<"Enter country id: ";
    cin>>id;
    cout<<"Enter country name: ";
    cin>>name;
  }
  void show()
  {
    cout<<"Country ID: "<<id<<endl;
    cout<<"Country Name: "<<name<<endl;
  }
};
void main()
{
  clrscr();
  Country cn;
  ofstream out("c:\\country.txt", ios::binary);
  cn.get();
  out.write((char*)&cn, sizeof(cn));
  out.close();
  getch();
}
```

18.9.2 Reading an Object from Disk

The method of reading an object from disk is same as reading a structure with binary I/O. The **read()** method is used for reading an object from a disk.

Program 18.14

Write a program that reads the contents of **country.txt** and display on the screen.

```
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
class Country
{
```

```
  private:
  int id;
  char name[50];
  public:
  void get()
  {
    cout<<"Enter country id: ";
    cin>>id;
    cout<<"Enter country name: ";
    cin>>name;
  }
  void show()
  {
    cout<<"Country ID: "<<id<<endl;
    cout<<"Country Name: "<<name<<endl;
  }
};
void main()
{
  clrscr();
  Country cn;
  ifstream in("c:\\country.txt", ios::binary);
  in.read((char*)&cn, sizeof(cn));
  cn.show();
  in.close();
  getch();
}
```

18.10 Accessing Records Randomly

The method of accessing a record directly is called **random access**. In this method, the required record is accessed directly without having to access the preceding records. It is an efficient and quick method of accessing records in files.

The records in binary I/O are stored with fixed length. Each record takes the same number of bytes in the memory. It helps the user to move to a particular location in the file to access a particular record.

18.10.1 File Pointers

A **file pointer** is used to access the record in files. The file streams can be created for input (ifstream) or output (ofstream). Each file object has two file pointers known as **get pointer** and **put pointer**. The get pointer is used with **ifstream**. The put pointer is used with **ofstream**. The **fstream** can perform both input and output operations and it has one 'get' pointer and one 'put' pointer. The 'get' pointer indicates the byte number in the file from where the next input will occur. The 'put' pointer indicates the byte number in the file where the next output will be made. There are two functions used to move these pointers in a file:

- **seekg ( ):** It belongs to **ifstream** class
- seekp ( ): It belongs to ofstream class

18.10.2 The seekg() Function

The word **seekg** stands for **seek get**. The **seekg()** function is used to specify the position of the file pointer for reading data from file.

Syntax

The syntax of **seekg()** function is as follows:

```
obj.seekg(position, offset);
```

obj It is the name of stream object that is associated with a file.
seekg It is the name of function.
position It indicates the byte number from where the data is to be read.
offset It indicates the position of a character from the start of file. The offset starts from 0. The offset of first character is 0 and so on. The positive offset means to move forward in the file and negative offset means to move backward. It is an optional parameter. The possible values are as follows:

- **ios::beg:** It means that compiler will count the position from the beginning of file. It is the default value.
- **ios::cur:** It means that compiler will count the position from current position.
- **ios::end:** It means that compiler will count the position from the end of file.

Example

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <fstream.h>
void main()
 {
  ofstream out("c:\\mydoc.txt",ios::binary);
  char text[80], ch;
  strcpy(text,"It is a test file");
  out<<text;
  out.close( );
  ifstream in("c:\\mydoc.txt",ios::binary);
  in.seekg(8);
  cout<<endl<<"The contents from position 8 are:"<<endl;
  while(!in.eof())
  {
    in.get(ch);
    if(!in.eof())
      cout<<ch;
  }
  in.close( );
  getch();
 }
```

Output:
The contents from position 8 are:
test file

How above Example Works?

The above program creates a file **mydoc.txt** and stores the text "It is a test file" in it. The program uses the statement "in.seekg(8);" to move the get pointer to position 8:

The **while** loop displays characters in file one by one untile **eof** occurs. The characters that come after position 8 are displayed because the get pointer is already set to position 8.

18.10.3 The seekp() Function

The word **seekp** stands for **seek put**. The **seekp()** function is used to specify the position of file pointer for writing data to file.

Freebooks.pk

Syntax

The syntax of **seekp()** function is as follows:

```
obj.seekp(position, offset);
```

obj It is the name of stream object that is associated with a file.

seekp It is the name of function.

position It indicates the byte number from where the data is to be written.

offset It indicates the position of a character from the start of file. The offset starts from 0. The offset of first character is 0 and so on. The positive offset means to move forward in the file and negative offset means to move backward. It is an optional parameter. The possible values are as follows:

- **ios::beg:** It means that compiler will count the position from the beginning of file. It is the default value.
- **ios::cur:** It means that compiler will count the position from current position.
- **ios::end:** It means that compiler will count the position from the end of file.

Program 18.15

Write a program that manages a telephone directory.

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <iomanip.h>
#include <conio.h>
class phonebook
{
  char name[20],phno[6];
  public:
  void getdata();
  void showdata();
  char *getname()
  {
    return name;
  }
  char *getphno()
  {
    return phno;
  }
  void update(char *nm, char *telno)
  {
    strcpy(name,nm);
    strcpy(phno,telno);
  }
};
void phoneBook :: getdata()
{
  cout<<"\nEnter Name : ";
  cin>>name;
  cout<<"Enter Phone No. : ";
  cin>>phno;
}
```

```
void phoneBook :: showdata()
{
  cout<<"\n"<<setw(15)<<name;
  cout<<setw(8)<<phno;
}
void main()
{
  phoneBook rec;
  fstream file;
  file.open("c:\\phone.dat", ios::ate | ios::in | ios::out | ios::binary);
  char ch,nm[20],telno[6];
  int choice,found=0;
  while(1)
  {
        cout<<"\n*****Phone Book*****\n";
        cout<<"1) Add New Record\n";
        cout<<"2) Display All Records\n";
        cout<<"3) Search Telephone No.\n";
        cout<<"4) Search Person Name\n";
        cout<<"5) Update Telephone No.\n";
        cout<<"6) Exit\n";
        cout<<"Choose your choice : ";
        cin>>choice;
        switch(choice)
        {
          case 1:
                rec.getdata();
                cin.get(ch);
                file.write((char *) &rec, sizeof(rec));
                break;
          case 2:
                file.seekg(0,ios::beg);
                cout<<"\n\nRecords in Phone Book\n";
                while(file)
                {
                  file.read((char *) &rec, sizeof(rec));
                  if(!file.eof())
                        rec.showdata();
                }
                file.clear();
                getch();
                break;
          case 3:
                cout<<"\n\nEnter Name : ";
                cin>>nm;
                file.seekg(0,ios::beg);
                found=0;
                while(file.read((char *) &rec, sizeof(rec)))
                {
                  if(strcmp(nm,rec.getname())==0)
                  {
                        found=1;
```

```
                rec.showdata();
            }
        }
        file.clear();
        if(found==0)
          cout<<"\n\n---Record Not found---\n";
          getch();
          break;
case 4:
        cout<<"\n\nEnter Telephone No : ";
        cin>>telno;
        file.seekg(0,ios::beg);
        found=0;
        while(file.read((char *) &rec, sizeof(rec)))
        {
          if(strcmp(telno,rec.getphno())==0)
          {
                found=1;
                rec.showdata();
          }
        }
        file.clear();
        if(found==0)
          cout<<"\n\n---Record Not found---\n";
        getch();
        break;
        case 5:
                cout<<"\n\nEnter Name : ";
                cin>>nm;
                file.seekg(0,ios::beg);
                found=0;
                int cnt=0;

                while(file.read((char *) &rec, sizeof(rec)))
                {
                  cnt++;
                  if(strcmp(nm,rec.getname())==0)
                  {
                        found=1;
                        break;
                  }
                }
                file.clear();
                if(found==0)
                  cout<<"\n\n---Record Not found---\n";
                else
                {
                  int location = (cnt-1) * sizeof(rec);
                  cin.get(ch);
                  if(file.eof())
                        file.clear();
                  cout<<"Enter New Telephone No : ";
```

```
                    cin>>telno;
                    file.seekp(location);
                    rec.update(nm,telno);
                    file.write((char *) &rec, sizeof(rec));
                    file.flush();
                  }
                  break;
            case 6:
                goto out;
        }
    }
out:
file.close();
}
```

18.11 Printing the Files through Streams

All streams work similarly even if they are connected to different devices like files, printer or monitor. Every stream is associated to a device or a file using the open function.

Program 18.16

Write a program that prints a file on the printer as well as on the screen.

```
# include <iostream.h>
#include <fstream.h>
void main( )
{
  ofstream print;
  print.open("LPT1");
  print<<"Hello printer...";
  print.close( );
}
```

How above Program Works?

The above program creates two streams **monitor** and **print**. The open function links these streams to **CON** (console) and **LPT1** (the port that connects printer). The following statement sends the text to the printer:

```
print<<"Hello printer...";
```

Programming Exercises

1. Write a program that inputs up to 10 integer values from a data file and displays them on the screen. If there are not 10 integers in the file, the message "The file is finished." should be displayed after the last number.
2. Write a program that counts the number of words in a file.
3. Write a program that copies the contents of one file to another file as a string.
4. Write a program that inputs character from the user and appends it in an existing file. The input ends if the user enters a full stop ".".
5. Write a program that copies the contents of a file to other file character by character.
6. Suppose a file "test.dat" contains three integers separated by spaces. Write a program to open file, read the integers and display them in reverse order separated by spaces.
7. Write a program that inputs three strings from user and appends to an existing file.

Freebooks.pk

Exercise Questions

Q.1. How is a file closed in which the data has been written?

The file is closed by closing the stream object connected with the file. The close method of the stream object is used for this purpose as out.close();

Q.2. Why the cout<< cannot be used to write data to a file?

The cout object is used with << operator write data to standard output. It cannot be used to write to a text file. An object of ofstream is required for this purpose.

Q.3. List the advantages of using files instead of standard input/output.

The data is stored permanently in a file even after the program has been executed. The data written by one program can be used by other programs also. Programs can accept inputs from multiple external sources not just a single source.

Q.4. Which three things are required before writing data to a text file?

- Include the fstream library — #include <fstream.h>
- Define a file variable — ofstream out;
- Open the file for output — out.open("file.dat");

Multiple Choice

1. **A file is stored in?**
   a. RAM b. Hard disk c. ROM d. Cache
2. **Which of the following can be used to create files and write information to them?**
   a. Ofstream b. outstream c.ifstream d. None
3. **ostream is:**
   a. Class b. library c. Command d. None
4. **Which can be used to create files, read data from them and write data to them?**
   a. Ofstream b. fstream c.ifstream d. None
5. **You can open a file using the following modes:**
   a. ios::in b. ios::out c. ios::app d. All
6. **Which of the following can be used to create files and read information from file?**
   a. Ofstream b. fstream c. ifstream d. None
7. **Which of the following member function writes a single character to a file?**
   a. Write b. get c.put d. None
8. **Which of the following member function can be used to store binary data to a file?**
   a. write b. put>> c. both a and b d. None
9. **What do the following statements do?**
   ofstream stream;
   stream.open("test.txt");
   a. Open a file for input
   b. Open a file for output, the statement fails if the file already exists
   c. Open a file for output, the contents of the file is destroyed if the file already exists
   d. None
10. **What do the following statements do?**
    ifstream stream;
    stream.open("test.txt");
    a. Open a file for input
    b. Open a file for output, the statement fails if the file already exists
    c. Open a file for input, the statement fails if the file does not exist

Freebooks.pk

d. both b and c

11. Which of the following directive is used to access files from a C++ program?

a.#include <fstream> b. #include <iostream>

c. #include <filestream> d. None

12. Which of the following can be used to open a file in c:\example\scores.txt on Windows?

a. stream.open("c:\program\test.txt");

b. stream.open("test.txt");

c. stream.open("c:\\program\\test.txt");

d. stream.open("\program\test.txt");

13. Which of the following member function reads a single character from a file?

a. put b. read c. get d. None

14. Which of the following function is used to detect the end of file?

a. EOF b. null c. end d. None

15. Which of the following mode is used to open a file for binary input?

a. ios::in | ios::binary b. ios::binary | ios::in

c. ios::out | ios::binary d. Both a and b

16. Which of the following function is used to move the file pointer for input?

a. stream.seekg(length); b. stream.seekp(length);

c. stream.tellp(); d. None

17. The functions seekg and seekp may have two arguments. The first argument is the offset and second argument may be _____ to indicate the base for the offset.

a. ios::beg b. ios::end c. ios::cur d. All

18. Which of the following function is used to move the file pointer for output?

a. stream.seekg(length); b. stream.seekp(length);

c. stream.tellg(); d. None

19. What is the purpose of invoking the close function?

a. If this function is not invoked, the data may not be saved properly in the file

b. If this function is not invoked, the file may be deleted

c. Both a and b

d. None

Answers

1. b	2. a	3. a	4. b	5. d	6. c
7. c	8. a	9. c	10. a	11. a	12. c
13. a	14. a	15. d	16. a	17. d	18. b
19. a	20.	21.	22.	23.	24.

Fill in the Blanks

1. A _____ is a collection of related records.
2. A type of file that stores data as readable and printable characters is called_____.
3. A source program of C++ is an example of _____.
4. A type of file that stores data as non-readable binary code is called _____.
5. An object file of a C++ program is an example of _____.
6. The way in which a file can be accessed is called _____.
7. A _____ is a series of bytes associated with a file
8. The act of reading data from an input stream is called _____.
9. The act of writing data to an output stream is called _____.
10. A file can be opened by first creating an object of _____.

Freebooks.pk

11. The function _____is used to check if a file stream has open a file successfully
12. The member function _____of stream object is used to close a file.
13. The function _____is used to read a complete line from the file.
14. The _____function is used to write single character in a file.
15. The _____function is used to read single character from a file.
16. Data can be stored in binary I/O using _____function of any object of ofstream or fstream.
17. Data can be read in binary I/O using _____function of any object of ifstream or fstream
18. Each file object has two file pointers knonw as get pointer and _____pointer .
19. The word seekg stands for _____.
20. _____function is used to specify the position of the file pointer for reading data from file
21. The word seekp stands for _____.
22. The _____function is used to specify the position of file pointer for writing data to file.

Answers

1. file	2. text file	3. text file
4. binary file	5. binary file	6. file access method
7. stream	8. extraction	9. insertion
10. ifstream, ofstream or fstream	11. is_open()	12. close()
13. **getline()**	14. put()	15. get()
16. write()	17. read()	18. put
19. seek get	20. seekg()	21. seek put
22. seekp()		

True / False

1. << is the stream insertion operator .
2. If the open function fails to open a file successfully you will always get an error message.
3. cout is an object of the istream class.
4. stream is a sequence of characters for input or output.
5. Only one file stream object can be declared per C++ program.
6. By default, files are opened in binary mode.
7. The function is_open() is used to check if a file stream has open a file successfully.
8. A file pointer is used to access the record in files.
9. Each file object has two file pointers known as get pointer and put pointer.
10. The seekp() function is used to specify the position of file pointer for reading data to file.
11. The seekg() function is used to specify the position of the file pointer for writing data from file.
12. The put() function is used to read single character in a file.
13. Random access method is used to access any data item directly without accessing the preceding data.
14. An object file of a C++ program is an example of text file.
15. Files are stored on secondary storage.
16. A program that manipulates files directly must include the header file <istream>.
17. Each line of a text file is terminated by the string "ofstream."

Answers

1. T	2. F	3. F	4. T	5. F	6. F
7. T	8. T	9. T	10. F	11. F	12. F
13. T	14. F	15. T	16. F	17. F	

Freebooks.pk

CHAPTER 19

DATA STRUCTURES

Chapter Overview

19.1 Data Structures
- 19.1.1 Types of Data Structures
- 19.1.2 Data Structure Operations

19.2 Linked List
- 19.2.1 Types of Linked Lists
  - 19.2.1.1 Singly-linked List
  - 19.2.1.2 Doubly-Linked List
  - 19.2.1.3 Circularly-Linked List

19.3 Stack
- 19.3.1 Representation of Stack
  - 19.3.1.1 Stack using Arrays
  - 19.3.1.2 Stack using Linked List

19.4 Queue
- 19.4.1 Representation of Queue
  - 19.4.1.1 Queue using Arrays
  - 19.4.1.2 Queue using Linked List

19.5 Trees
- 19.5.1 Tree Terminology
- 19.5.2 Binary Trees
- 19.5.3 Traversing Binary Trees
- 19.5.4 Insertion in Binary Tree
- 19.5.5 Binary Tree Deletion
- 19.5.6 Advantages of Binary Tree
- 19.5.7 Complete Binary Trees
- 19.5.8 Binary Search Trees

19.6 Graphs
- 19.6.1 Types of Graphs
  - 19.6.1.1 Directed Graph
  - 19.6.1.2 Undirected Graphs

Exercise Questions
Multiple Choices
True/False

Freebooks.pk

19.1 Data Structures

A **data structure** is a specialized format for organizing and storing data. Different types of data structures include array, file, record, table and tree etc. Any data structure is designed to organize data for a specific purpose. A data structure helps the user to store data in such a way that it can be used and processed efficiently.

19.1.1 Types of Data Structures

Two types of data structures are as follows:

1. Linear Data Structures

A type of data structure in which the data elements are stored in a linear sequence is known as **linear data structures**. Array, stack, queue and linked list are examples of linear data structures.

2. Non-linear Data Structures

A type of data structure in which the data elements are stored in a non-linear sequence is known as **non-linear data structures**. Trees and graphs are examples of non-linear data structures.

19.1.2 Data Structure Operations

The data appearing in data structures are processed by means of certain operations. Some important operations on data structures are as follows:

1. Traversing

The process of accessing each record exactly once so that certain items in the record may be processed is called **traversing**. It also known as **visiting**.

2. Searching

The process of finding the location of record with a given key value is called searching. It may also inlcude finding the locations of all records which satisfy one or more conditions.

3. Inserting

The process of adding new record to the data structure is called inserting.

4. Deleting

The process of removing a record from the data structure is called deleting.

Example

An organization contains a membership file in which each record contains the following data for a given member:

- Name
- Address
- Phone
- Ages
- Gender

1. Suppose the organization wants to announce a meeting through a mailing. The file will be traversed to obtain Name and Address for each member.
2. Suppose one wants to find the names of all members living in a certain area. The file will be traversed to obtain the required data.
3. Suppose one wants to obtain Address for a given Name. The file will be searched for the record containing Name.
4. Suppose a new person joins the organization. The record of the new person will be inserted in the file.

Freebooks.pk

5. Suppose a member leaves. The record of the leaving person will be deleted form file.
6. Suppose a member has moved and has a new address and phone. The record of that person will be searched and his address and phone will be updated.
7. Suppose one wants to find the number of 65 or older. The file will be traversed to count such members.

19.2 Linked List

A **linked list** is one of the fundamental data structures used in programming. It consists of a sequence of nodes. Each node contains data fields and a reference field that points to the next nodes. A linked list allows the insertion and removal of nodes at any point in the list but it does not allow random access. Any node in the list can be accessed only by starting from the first node sequentially.

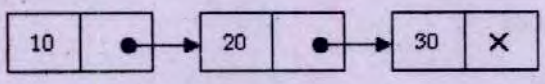


Figure 19.1: A linked list with three nodes

The above figure shows a linked list with three nodes. Each node consists of two parts. One part contains the data and second part is a pointer that refers to the next node. The last node contains NULL value in its second part because there is no node after it.

19.2.1 Types of Linked Lists

Different types of linked lists are as follows:

- Singly-linked lists
- Doubly-linked lists
- Circularly-linked lists

19.2.1.1 Singly-linked List

A **singly-linked list** is a type of linked list in which each node is linked to the next node in the list. Each node contains data fields and a reference field that points to the next node. A singly-linked list allows the insertion and removal of nodes at any point in the list but it does not allow random access. Any node in the list can be accessed only by starting from the first node sequentially.

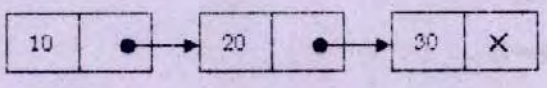


Figure 19.2: A singly-linked list with three nodes

Program 19.1

Write a program that implements singly-linked list

```
#include <iostream.h>
#include <conio.h>
struct link
  {
    int data;
    link *next;
  };
```

```
class List
{
  private:
  link *first;
  public:
  List();
  void add(int d);
  void del(int v);
  void show();
};
List::List()
{
  first = NULL;
}
void List::add(int d)
{
  link *ptr, *temp;
  if(first==NULL)
  {
    first = new link;
    first->data = d;
    first->next = NULL;
  }
  else
  {
    ptr = first;
    while(ptr->next!=NULL) ptr = ptr->next;
    temp = new link;
    temp->data = d;
    temp->next = NULL;
    ptr->next = temp;
  }
}
void List::del(int v)
{
  link *temp, *pre;
  temp = first;
  if(temp->data==v)
  {
    first = temp->next;
    delete temp;
    cout<<endl<<v<<" has been deleted."<<endl;
    return;
  }
  pre = temp;
  while(temp!=NULL)
  {
    if(temp->data == v)
    {
  pre->next = temp->next;
  delete temp;
  cout<<"\nValue deleted."<<endl;
  return;
```

```
        }
        pre = temp;
        temp = temp->next;
      }
      cout<<endl<<v<<" not found!"<<endl;
    }
    void List::show()
    {
      link *temp;
      temp = first;
      cout<<"The list is as follows:\n";
      while (temp!=NULL)
      {
        cout<<temp->data<<" ";
        temp = temp->next;
      }
    }
    void main()
    {
      List l;
      clrscr();
      l.add(10);
      l.add(20);
      l.add(30);
      l.add(40);
      l.add(50);
      l.show();
      cout<<endl;
      l.del(35);
      l.del(10);
      l.show();
      getch();
    }
```

Output:
The list is as follows:
10 20 30 40 50

35 not found.

10 has been deleted.
The list is as follows:
20 30 40 50

19.2.1.2 Doubly-Linked List

A **doubly-linked list** is a more sophisticated type of linked list. It is also known as **two-way linked list**. Each node in this type of linked list has two links. The first link points to the previous node or to a NULL value if it is the first node. The second link points to the next node or to a NULL value if it is the last node.

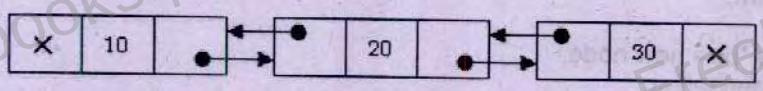


Figure 19.3: A doubly-linked list with three nodes

The above figure shows a doubly-linked list with three nodes. Each node consists of three parts. The first part is a pointer that refers to the previous node. The second part contains the data and third part is a also a pointer that refers to the next node. The first part of the first node contains NULL value because there is no node before it. The last part of the last node also contains NULL value because there is no node after it.

Freebooks.pk

Program 19.2

Write a program that implements doubly-linked list.

```
#include<iostream.h>
#include<conio.h>
#include<iostream.h>
class node
{
 public:
   int data;
   node *next;
   node *previous;
   node():data(0),next(NULL),previous(NULL) {}
};
class doubly
{
 private:
    node *first;
    node *last;
 public:
    doubly():first(NULL),last(NULL) { }
    void insert_s(int);
    void insert_end(int);
    void insert_loc(int,int);
    void remove_s();
    void remove_end();
    void remove_loc(int);
    void show_forword();
void show_reverse();
~doubly();
};
void doubly::insert_s(int value)
{
 if(first==NULL)
  {
   first=new node;
   first->next=NULL;
   first->previous=NULL;
   first-> data=value;
   last=first;
   return;
  }
 node *ptr= new node;
 ptr->next=first;
first->previous=ptr;
first=ptr;
first->data=value;
first->previous=NULL;
 }
void doubly::insert_end(int value)
{
 if(first==NULL)
  {
```

```
        first=new node;
        first->next=NULL;
        first->previous=NULL;
        first->data=value;
        last=first;
        return;
       }
    node *ptr= new node;
    last->next=ptr;
    ptr->previous=last;
    last=ptr;
    last->data=value;
    last->next=NULL;
    }
    void doubly::insert_loc(int value,int loc)
    {
     node *ptr=first;
    int i=1;
    while(i<loc)
    {
     ptr=ptr->next;
      i++;
     }
    node *ptr2=new node;
    ptr2->previous=ptr->next->previous;
    ptr->next->previous=ptr2;
    ptr2->next=ptr->next;
    ptr->next=ptr2;
    ptr2->data=value;
    }
    void doubly::remove_s()
    {
     if( first->next==NULL)
      {
       delete first;
       last=NULL;
       return;
      }
       node *ptr=first;
       first=first->next;
       first->previous=NULL;
       delete ptr;
    }
    void doubly::remove_end()
    {
      if(first->next==NULL)
       {
        delete first;
        last=NULL;
        return;
       }
    node *ptr=last;
```

```
last=last->previous;
last->next=NULL;
delete ptr;
}
void doubly::remove_loc(int loc)
{
  node *ptr=first;
  int i=1;
  while (i<loc)
  {
    ptr=ptr->next;
    i++;
  }
node *ptr2=ptr->next;
ptr->next=ptr2->next;
ptr2->next->previous=ptr;
delete ptr2;
}
void doubly::show_forword()
{
 cout<<"\nValue displayed in forwarding order\n";
node *ptr=first;
while( ptr->next!=NULL)
{
  cout<<ptr->data<<" ";
  ptr=ptr->next;
}
cout<<ptr->data;
}
void doubly::show_reverse()
{
 cout<<"\nvlaue displayed in backword order\n";
node *ptr=last;
  while( ptr->previous!=NULL);
  {
    cout<<ptr->data<<" ";
    ptr=ptr->previous;
  }
cout<<ptr->data;
}
doubly::~doubly()
{
  node *ptr=first;
  while(first->next!=NULL)
  {
   first=first->next;
   delete ptr;
   ptr=first;
  }
delete ptr;
}
```

```
void main()
{
  clrscr( );
  doubly d;
  d.insert_s(5);
  d.insert_s(4);
  d.insert_s(3);
  d.insert_s(2);
  d.insert_s(1);
  d.show_forword();
  cout<<endl;
  d.show_reverse();
  d.insert_end(50);
  d.insert_end(51);
  d.insert_loc(3,33);
  cout<<endl;
  d.show_forword();
  cout<<endl;
  d.show_reverse();
  cout<<endl;
  d.remove_s();
  d.remove_end();
  d.remove_loc(4);
  d.show_forword();
  cout<<endl;
  d.show_reverse();
  getch();
}
```

19.2.1.3 Circularly-Linked List

A type of linked list in which the last node points to the first node instead of pointing to NULL is known as **circularly-linked list**.

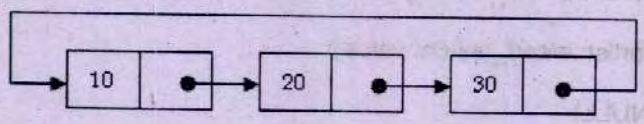


Figure 19.4: A circularly-linked list with three nodes

The above figure shows a circularly-linked list with three nodes. Each node consists of two parts. The first part contains the data and second part is a pointer that refers to the next node. However, the last part of the last node does not contain NULL value. It refers to the first node in the list.

Program 19.3

Write a program that implements circularly-linked list.

```
#include<iostream.h>
#include<conio.h>
class node
{
  public:
```

```
    int data;
    node *link;
    node():data(0),link(NULL){}
};
class circularlist
{
 private:
    int i;
    node *first;
    node *last;
 public:
    circularlist():i(0),first(NULL),last(NULL) { }
    void insert_start(int);
    void insert_last(int);
    void remove_start();
    void remove_last();
    void show();
    ~ circularlist();
};
void circularlist::insert_start(int value)
{
  if(first==NULL)
   {
    first=new node;
    first-> data=value;
    first-> link=NULL;
    last=first;
    return;
   }
  node *ptr= new node;
  ptr-> data=value;
  ptr-> link=first;
  first=ptr;
  last ->link=first;
}
void circularlist::insert_last(int value)
{
  if(first==NULL)
   {
    first=new node;
    first->data=value;
    first->link= first;
    last=first;
    return;
   }
node *ptr= new node;
last->link=ptr;
last=ptr;
last->data=value;
}
void circularlist::remove_start()
{
  if(first==NULL)
```

```
  {
    cout<<"No value to display";
    return;
  }
 if( first->link==first)
  {
   delete first;
   last=NULL;
   return;
  }
 node *ptr=first;
 first=first->link;
 last->link=first;
  delete ptr;
}
void circularlist::remove_last()
{
 if( first==NULL)
  {
   cout<<"No vlaue to display";
   return;
  }
  if( first->link==first)
   {
    delete first;
    last = NULL;
    return;
   }
   node *ptr=first;
  while (ptr->link!=last)
   ptr=ptr->link;
   ptr->link=first;
   delete last;
   last=ptr;
}
void circularlist::show()
{
  if(first==NULL)
   {
    cout<<"No value to display";
    return;
   }
    node *ptr=first;
    while( ptr->link!=first)
     {
     cout<<ptr->data<<" ";
     ptr=ptr->link;
     }
cout<<ptr->data<<" ";
   }
circularlist::~circularlist()
{
```

```
    node *ptr=first;
    while (first->link!=last)
    {
      first=first->link;
      cout<<ptr->data<<"deleted"; // press alt+f5 to see the deleted values
      delete ptr;
      ptr=first;
    }
    cout<<ptr->data<<"deleted";
    delete ptr;
    ptr=first->link;
    cout<<ptr->data<<"deleted";
}
void main()
{
    clrscr();
    circularlist c;
    c.insert_start(5);
    c.insert_start(6);
    c.insert_start(7);
    c.insert_start(8);
    c.insert_start(9);
    c.show();
    cout<<"\n";
    c.insert_last(94);
    c.insert_last(95);
    c.insert_last(96);
    c.insert_last(97);
    c.insert_last(98);
    c.insert_last(99);
    c.show();
    cout<<"\n";
    c.remove_start();
    c.show();
    cout<<"\n";
    c.remove_last();
    c.show();
    getch();
}
```

19.3 Stack

A stack is a type of data structure based on the technique of **Last In First Out (LIFO)**. LIFO is a technique for storing data in a data structure and retrieving it. It stores the newly entered object on the top of all previously entered objects. It is known as **push** operation. Similarly, the object that is stored at the top of the stack is removed first. It is known as **pop** operation. It means that the object that was entered last is removed first. That is why this technique is known as **Last In First Out**.

There are two restrictions while processing a stack:

- A push operation is not allowed when the stack is full.
- A pop operation is not allowed when the stack is empty.

Example

An example of a stack in real world is a pile of plates. The first plate begins the pile. The second plate is placed on top of first plate. The third plate is placed on the top of second plate so on. A plate may be removed from the pile at any time but only from the top. There is always a certain number of plates on the pile. Pushing a new plate on the pile increases the number by 1 and popping a plate from pile decreases the number by 1.

The user cannot pop a plate if the pile is empty. Similary, the user cannot push a plate on the pile if it already contains the maximum possible number of plates. It means that these two conditions must be monitored while processing a pile.

Figure 19.5: Empty stack. Pop operation not possible

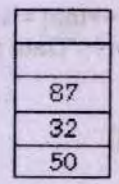


Figure 19.6: Stack after 3 push operations

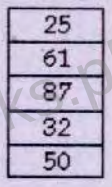


Figure 19.7: Stack full. Push operation not possible

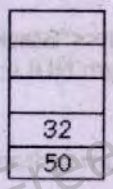


Figure 19.8: Stack after 3 pop operations

19.3.1 Representation of Stack

The stack can be represented in computer in two ways:

19.3.1.1 Stack using Arrays

A stack can be represented by using arrays. The length of an array represents the maximum number of data elements that can be pushed on the stack. The user cannot push unlimited number of items on it. The representation of stack with array is simple and easier to implement. However, it provides less flexibility.

19.3.1.2 Stack using Linked List

A stack can be represented by allocating memory dynamically using linked list. The length of stack is not specified in program code. It is maintained during execution according to the exact requirement of the user. The user can push unlimited number of items on the stack. The dynamic representation of stack with pointers is more complex to implement. However, it provides more flexibility.

Program 19.4

Write a program that implements a stack using arrays.

```
#include <iostream.h>
#include <conio.h>
class Stack
{
  private:
  int arr[5];
  int top;
```

Freebooks.pk

```
    public:
    Stack()
    {
      top = -1;
    }
    void push(int v)
    {
      if(top==4)
        cout<<"Stack is full."<<endl;
      else
      {
        arr[++top] = v;
        cout<<"Data pushed successfully."<<endl;
      }
    }
    int pop()
    {
      if(top==-1)
      {
        cout<<"Stack empty.";
        return NULL;
      }
      else
        return arr[top--];
    }
};
void main()
{
  Stack s;
  clrscr();
  s.push(10);
  s.push(20);
  s.push(30);
  s.push(40);
  s.push(50);
  s.push(60);
  cout<<s.pop()<<endl;
  cout<<s.pop()<<endl;
  cout<<s.pop()<<endl;
  cout<<s.pop()<<endl;
  cout<<s.pop()<<endl;
  cout<<s.pop()<<endl;
  cout<<s.pop()<<endl;
  getch();
}
```

Output:

```
Data pushed successfully.
Data pushed successfully.
Data pushed successfully.
Data pushed successfully.
Data pushed successfully.
Stack is full.
50
40
30
20
10
Stack empty.
Stack empty.
```

How above Program Works?

The above program implements a stack with five elements using arrays. It declares an object of stack s. The program attemps to push six elements on the stack. Five elements are pushed successfully but sixth elements is not pushed as stack can only store five elements. The program then pops for seven times. Five elements are popped and dipslayed on the screen but sixth and seventh pop operations dispalys error message as the stack is empty.

Freebooks.pk

Program 19.5

Write a program that implements a stack dynamically using pointers.

```
#include <iostream.h>
#include <conio.h>
struct link
{
  int data;
  link *next;
};
class Stack
{
  private:
  link *top;
  public:
  Stack();
  void push(int d);
  int pop();
  void show();
};
Stack::Stack()
{
  top = NULL;
}
void Stack::push(int d)
{
  link *ptr, *temp;
  if(top==NULL)
  {
    top = new link;
    top->data = d;
    top->next = NULL;
  }
  else
  {
    ptr = top;
    while(ptr->next!=NULL)
      ptr = ptr->next;
    temp = new link;
    temp->data = d;
    temp->next = NULL;
    ptr->next = temp;
  }
}
int Stack::pop()
{
  link *temp, *pre;
  int n;
  temp = top;
  if(top==NULL)
  {
    cout<<"Stack empty."<<endl;
```

```
        return NULL;
      }
      pre = temp;
      while(temp->next!=NULL)
      {
        pre = temp;
        temp = temp->next;
      }
      pre->next = NULL;
      n = temp->data;
      delete temp;
      return n;
    }
    void Stack::show()
    {
      link *temp;
      temp = top;
      while (temp!=NULL)
      {
        cout<<temp->data<<" ";
        temp = temp->next;
      }
    }
    void main()
    {
      Stack s;
      s.push(10);
      s.push(20);
      s.push(30);
      s.push(40);
      s.pop();
      s.pop();
      s.show();
      s.push(50);
      s.show();
      getch();
    }
```

19.4 Queue

A **queue** is a type of data structure based on the technique of **First In First Out (FIFO)**. FIFO is a technique for storing data in a data structure and retrieving it. It stores the newly entered object at the end of all previously entered objects. Similarly, the object that is stored at first position is removed first. It means that the object that was entered first in the queue is removed first. That is why this technique is known as First In First Out.

Example

An example of a queue in real world is a queue of people waiting to buy tickets. The person who came first in the queue will be given the ticket first. The person who joined the queue at last time is entertained last. The position of the first person is not changed if a new person joins the queue becuase he is added at the end. However, the positions of all persons in the queue are changed when the first person buys the ticket and leaves the queue. In this situation, each person in the queue moves one step forward.

Freebooks.pk

Figure 19.9: An empty queue

25				

Figure 19.10: A queue with after one element inserted

25	84	56		

Figure 19.11: A queue with three elements inserted. 25 at front and 56 at rear

84	56			

Figure 19.12: The queue after deleting an element. 84 at front and 56 at rear

19.4.1 Representation of Queue

The queue can be represented in computer in two ways:

19.4.1.1 Queue using Arrays

A queue can be represented by using arrays. The length of an array represents the maximum number of data elements that can be inserted in the queue. The user cannot insert unlimited number of items in it. The representation of queue with array is simple and easier to implement. However, it provides less flexibility.

19.4.1.2 Queue using Linked List

A queue can be represented by allocating memory dynamically using linked list. The length of queue is not specified in program code. It is maintained during execution according to the exact requirement of the user. The user can insert unlimited number of items in the queue. The dynamic representation of queue with pointers is more complex to implement. However, it also provides more flexibility.

Program 19.6

Write a program that implements a queue using array.

```
#include<iostream.h>
#include<conio.h>
const int size=10;
class queue
{
  private:
  int first,last,count;
  int array[size];
  public:
  queue();
  void insert(int);
  int remove();
};
queue::queue():first(0),last(-1),count(0)
{
}
void queue::insert(int value)
{
```

```
    if(count>=size)
    {
     cout<<"Queue is full\n";
     return;
    }
    if(last>=size-1)
    {
    last=-1;
    }
  array[++last]=value;
  count++;
  }

  int queue::remove()
  {
   if(count<=0)
   {
    cout<<"queue is empty\n";
    return NULL;
   }
   if(first>=size)
   {
   first=0;
   }
   count--;
   return array[first++];
  }
  void main()
  {
   clrscr();
   queue q;
   q.insert(10);
   q.insert(20);
   q.insert(30);
   cout<<q.remove();
   cout<<q.remove();
   cout<<q.remove();
   q.insert(40);
   q.insert(50);
   q.insert(60);
   q.insert(70);
   q.insert(80);
   q.insert(90);
   q.insert(100);
   q.insert(110);
   q.insert(120);
   q.insert(130);
   cout<<q.remove();
   cout<<q.remove();
   cout<<q.remove();
   cout<<q.remove();
   cout<<q.remove();
   cout<<q.remove();
```

```
    cout<<q.remove();
    cout<<q.remove();
    cout<<q.remove();
    cout<<q.remove();
    cout<<q.remove();
    cout<<q.remove();
    getch();
  }
```

Program 19.7

Write a program that implements a queue using linked list.

```
#include<iostream.h>
#include<conio.h>
class node
 {
   public:
   int data;
   node *link;
   node(){data=0;link=NULL;}
 };
class queue
 {
   private:
   node *first,*last;
   public:
   queue ():first(NULL),last (NULL) { }
   void insert(int);
   int remove();
  ~queue();
 };
void queue::insert(int value)
{
 node *ptr=NULL;
 ptr= new node;
   if(ptr==NULL)
   {
     cout<<"Queue is full\n";
     return;
   }
   if(first==NULL)
   {
     first=ptr;
     first->data=value;
     first->link=NULL;
     last=first;
     return;
   }
 last->link=ptr;
 last=ptr;
 last->data=value;
 last->link=NULL;
 }
```

```
int queue::remove()
{
  if(first==NULL)
  {
  cout<<"Queue empty!\n";
  return NULL;
  }
node *ptr=first;
first=first->link;
int value=ptr->data;
delete ptr;
return value;
}
queue::~queue()
{
 node *ptr=first;
    while(first!=NULL)
    {
  first=first->link;
      delete ptr;
      ptr=first;
    }
}
void main()
{
  clrscr( );
  queue q;
  q.remove();
  for(int i=1;i<100;i++)
  q.insert(i);
  for(i=1;i<100;i++)
        cout<<q.remove()<<" ";
  getch();
}
```

19.5 Trees

Tree is a non-linear data structures. Trees implement a hierarchical structure that consists of a set of **nodes**. Each node is called a **member** of the tree. This type of data structure is used to represent data containing hierarchical relationship between elements.

19.5.1 Tree Terminology

The basic terminologies related to tree are as follows:

- **Root:** The node at the top of the tree is called the **root**. There is only one root in a tree. The root node is the starting point of a tree. It does not have a parent node. However, it may have as many child nodes as desired.
- **Edge:** The line drawn from a parent node to its child node is called an **edge**.
- **Path:** The line from node to node along the edges that connect them is called a **path**.
- **Branch:** A path ending in a leaf is called a **branch**.
- **Parent:** Any node (except the root) has exactly one edge running upward to another node. The node above a node is called the **parent** of that node.

Freebooks.pk

- **Child:** The nodes that are directly linked to a node one level above them are called **child nodes** or **children** to the **parent node**.
- **Sibling:** The nodes that share a parent node are a set of **sibling nodes**.
- **Degree:** The **degree** of a node is equal to the number of its child nodes.
- **Leaf:** A node that has no children is called **leaf node** or a **leaf**. There can be only one root in a tree but there can be many leaves. These are also called **terminal nodes**.
- **Subtree:** A set of all nodes that consists of any node (except the root) and all nodes below that node is called **subtree**.
- **Levels:** The level of a particular node refers to its position in the tree. The root node is the top level node and can be referred as level 0. The children are at level 1 and grandchildren at level 2 and so on.
- **Depth / Height:** The depth or height of a tree is the maximum number of nodes in a branch of the tree. It is always one more than the larges level number of a tree.
- **Empty Tree:** A tree that contains no node is called an **empty tree**.

Example

Following is an example of a tree:

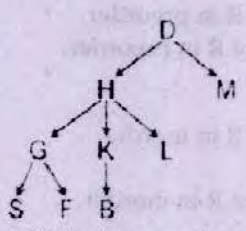


Figure 19.13: An example of tree

In the above figure, **D** is the root node. Each of the letters in the above tree represents a node. **H** is a parent node of **G**, **K** and **L**. These three siblings to each other. A set of nodes **G**, **S** and **F** is the left sub-tree extending from **H**. The **S**, **F**, **B**, **L** and **M** are leaf nodes.

19.5.2 Binary Trees

A type of **tree** in which every node has at most two children is called a **binary tree**. It can be defined as a finite set of elements. A binary tree is either empty or is partitioned into three disjoint subsets.

- The first subset contains a single element called **root** of the tree.
- The other two subsets are themselves binary trees called left and right sub trees.

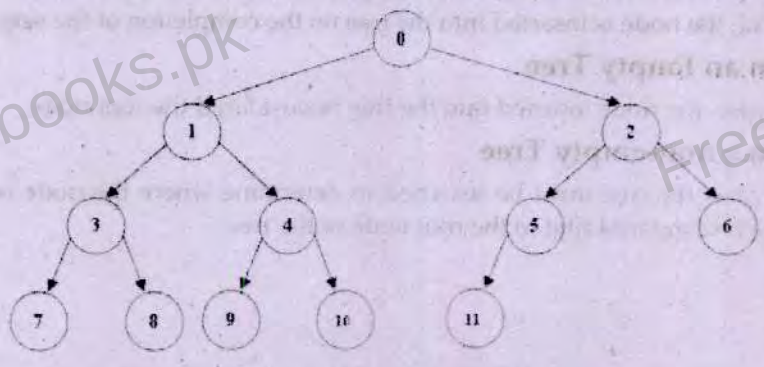


Figure 19.14: A binary tree

Freebooks.pk

The array indices for the relatives of each node are shown below. A dash indicates that the relative node does not exist.

Node/Index	0	1	2	3	4	5	6	7	8	9	10	11
Parent	-	0	0	1	1	2	2	3	3	4	4	5
Left Child	1	3	5	7	9	11	-	-	-	-	-	-
Right Child	2	4	6	8	10	-	-	-	-	-	-	-
Left Sibling	-	-	1	-	3	-	5	-	7	-	9	-
Right Sibling	-	2	-	4	-	6	-	8	-	10	-	-

19.5.3 Traversing Binary Trees

There are three standard ways of traversing a binary tree. These are as follows:

1. Preorder

1. Process the root R.
2. Traverse the left subtree of R in preorder.
3. Traverse the right subtree of R in preorder.

2. Inorder

1. Traverse the left subtree of R in inorder.
2. Process the root R.
3. Traverse the right subtree of R in inorder.

3. Postorder

1. Traverse the left subtree of R in postorder.
2. Traverse the right subtree of R in postorder.
3. Process the root R.

19.5.4 Insertion in Binary Tree

A binary tree is constructed by the repeated insertion of new nodes in it. The insertion must maintain the order of tree. It means that the values to the left of a given node must be less than that node and the values to the right must be greater.

Inserting a node into a tree consists of two separate operations:

- First, the tree must be searched to determine where the node is to be inserted.
- Second, the node is inserted into the tree on the completion of the search.

Insertion in an Empty Tree

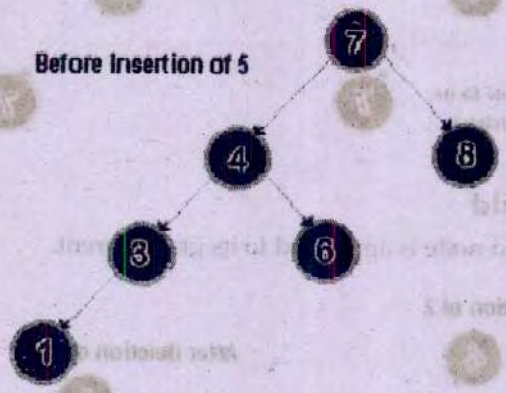
In this case, the node inserted into the tree isconsidered the root node.

Inserting in a non-empty Tree

In this case, the tree must be searched to determine where the node is to be inserted. The new node is compared first to the root node of the tree.

Freebooks.pk

Suppose the following tree exists:



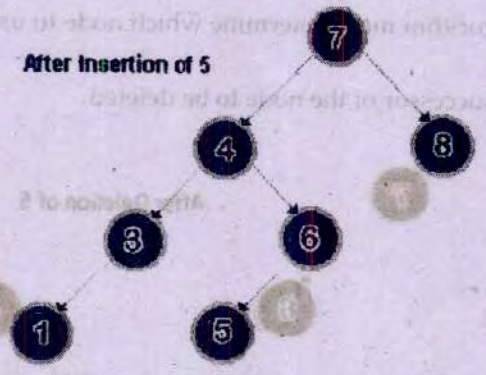
If the value of the new node is less than the value of the root node, the new node is:

- appended as the left leaf of the root node if the left subtree is empty
- else the search continues down the left subtree.

If the value of the new node is greater than the value of the root node, the new node is:

- appended as the right leaf of root node if the right subtree is empty
- else, the search process continues down the right subtree

The value five will be inserted in the above tree as follows:



19.5.5 Binary Tree Deletion

The algorithm to delete a node from binary tree is complex. The algorithm consists of two separate operations of searching and deletion. A node can be deleted from the tree once it has been determined by searching algorithm. The order of the binary tree must be kept intact when the node is deleted from the tree.

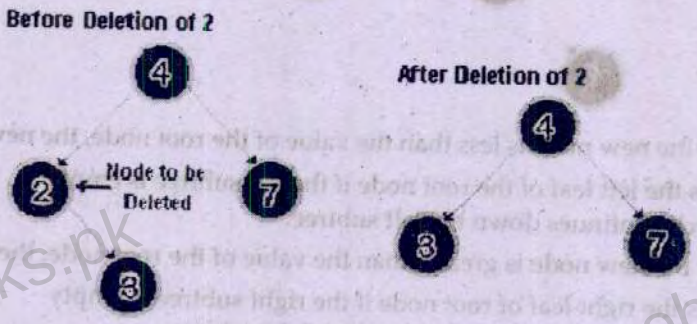
Special Cases that have to be considered are as follows:

The node to be deleted has no children

In this case the node may simply be deleted from the tree:

The node has one child

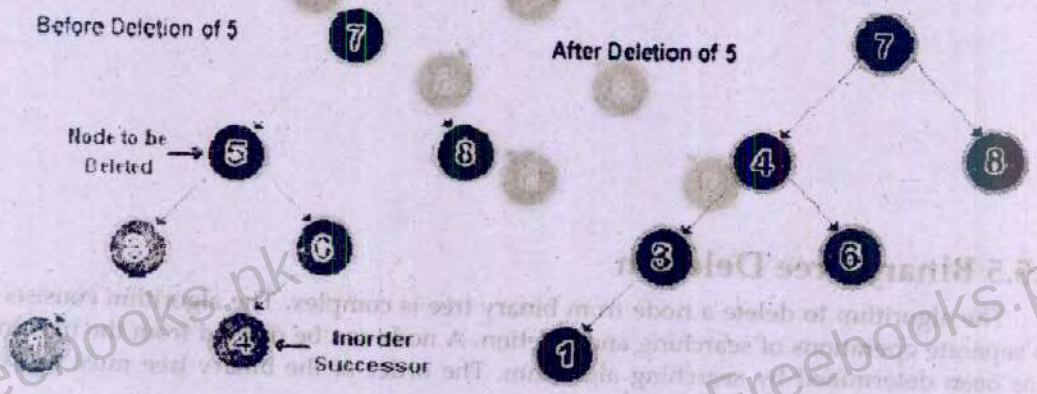
In this case, the child node is appended to its grandparent.



The node to be deleted has two children

In this case, the algorithm must determine which node to use in place of the node to be deleted:

- Use the inorder successor of the node to be deleted.



- Else if no right subtree exists replace the node to be deleted with the it's left child

19.5.6 Advantages of Binary Tree

The process of searching through a large amount of data is generally time consuming. A binary tree is better than a linked list because it allows for faster inserts, searches and deletes. It maintains the order of the tree as well. Its advantage over an array is that it does not require the maximum size of structure to be determined before the user starts adding values to it.

19.5.7 Complete Binary Trees

A tree is called a **complete binary tree** if all its levels, except the last, have maximum number of possible nodes and if all the nodes at last level appear as far left as possible.

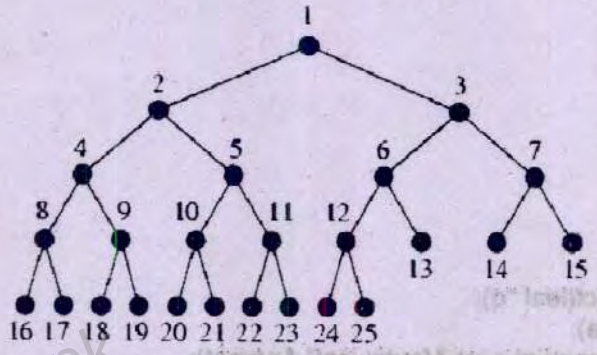


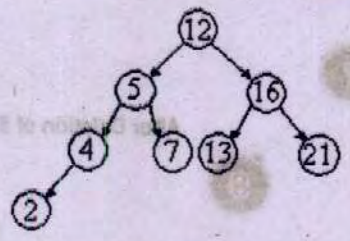
Figure 19.15: A binary tree

19.5.8 Binary Search Trees

A tree is called binary search tree if each node N of the tree has the following property:

- The value of N is greater than every value in the left subtree of N.
- The value of N is less than every value in the right subtree of N.

The sorted values can be obtained by traversing the binary search tree using inorder traversal:



The inorder traversal on the above tree will display the following values:

2, 4, 5, 7, 12, 13, 16, 17

Implementing a Tree

The tree can be implemented by using linked list. Each node of a tree will contain:

1. Value stored at the node
2. A pointer to the next node on the right
3. A pointer to the next node at the left

Program 19.8

Write a program that implements a binary tree data structure.

```
#include <iostream.h>
#include <conio.h>
#define YES 1
#define NO 0
struct leaf
{
  int data;
  leaf *l;
  leaf *r;
};
class tree
{
   private:
   leaf *p;
   public:
   tree();
   ~tree();
   void destruct(leaf *q);
   tree(tree& a);
   void findparent(int n,int &found,leaf* &parent);
   void findfordel(int n,int &found,leaf *&parent,leaf* &x);
   void add(int n);
   void transverse();
   void in(leaf *q);
   void pre(leaf *q);
   void post(leaf *q);
   void del(int n);
};
tree::tree()
{
   p=NULL;
}
```

```
tree::~tree()
{
  destruct(p);
}
void tree::destruct(leaf *q)
{
   if(q!=NULL)
   {
     destruct(q->l);
     del(q->data);
     destruct(q->r);
   }
}
void tree::findparent(int n,int &found,leaf *&parent)
{
   leaf *q;
   found=NO;
   parent=NULL;
   if(p==NULL)
     return;
   q = p;
   while(q!=NULL)
   {
     if(q->data==n)
     {
    found=YES;
    return;
     }
     if(q->data>n)
     {
    parent=q;
    q=q->l;
     }
     else
     {
    parent=q;
    q=q->r;
     }
   }
}
void tree::add(int n)
{
   int found;
   leaf *t,*parent;
   findparent(n,found,parent);
   if(found==YES)
     cout<<"\nSuch a Node Exists";
   else
   {
     t=new leaf;
     t->data=n;
     t->l=NULL;
```

```
        t->r=NULL;
        if(parent==NULL)
      p=t;
        else
      parent->data > n ? parent->l=t : parent->r=t;
      }
    }
    void tree::transverse()
    {
      int c;
      cout<<"\n1.InOrder\n2.Preorder\n3.Postorder\nChoice: ";
      cin>>c;
      switch(c)
      {
        case 1:
      in(p);
      break;
        case 2:
      pre(p);
      break;
        case 3:
      post(p);
      break;
      }
    }
    void tree::in(leaf *q)
    {
      if(q!=NULL)
      {
        in(q->l);
        cout<<"\t"<<q->data<<endl;
        in(q->r);
      }
    }
    void tree::pre(leaf *q)
    {
      if(q!=NULL)
      {
        cout<<"\t"<<q->data<<endl;
        pre(q->l);
        pre(q->r);
      }
    }
    void tree::post(leaf *q)
    {
      if(q!=NULL)
      {
        post(q->l);
        post(q->r);
        cout<<"\t"<<q->data<<endl;
      }
    }
```

```
void tree::findfordel(int n,int &found,leaf *&parent,leaf *&x)
{
   leaf *q;
   found=0;
   parent=NULL;
   if(p==NULL)
     return;
   q=p;
   while(q!=NULL)
   {
     if(q->data==n)
     {
    found=1;
    x=q;
    return;
     }
     if(q->data>n)
     {
    parent=q;
    q=q->l;
     }
     else
     {
    parent=q;
    q=q->r;
     }
   }
}
void tree::del(int num)
{
   leaf *parent,*x,*xsucc;
   int found;
   // If EMPTY TREE
   if(p==NULL)
   {
     cout<<"\nTree is Empty";
     return;
   }
   parent=x=NULL;
   findfordel(num,found,parent,x);
   if(found==0)
   {
     cout<<"\nNode to be deleted NOT FOUND";
     return;
   }
   // If the node to be deleted has 2 leaves
   if(x->l != NULL && x->r != NULL)
   {
     parent=x;
     xsucc=x->r;
     while(xsucc->l != NULL)
     {
```

```
            parent=xsucc;
            xsucc=xsucc->l;
            }
            x->data=xsucc->data;
            x=xsucc;
          }
          // if the node to be deleted has no child
          if(x->l == NULL && x->r == NULL)
          {
            if(parent->r == x)
          parent->r=NULL;
            else
          parent->l=NULL;
            delete x;
            return;
          }
          // if node has only right leaf
          if(x->l == NULL && x->r != NULL )
          {
            if(parent->l == x)
          parent->l=x->r;
            else
          parent->r=x->r;
            delete x;
            return;
          }
          // if node to be deleted has only left child
          if(x->l != NULL && x->r == NULL)
          {
            if(parent->l == x)
          parent->l=x->l;
            else
          parent->r=x->l;
            delete x;
            return;
          }
        }
        void main()
        {
          clrscr();
          tree t;
          int data[]={32,16,34,1,87,13,7,18,14,19,23,24,41,5,53};
          for(int i=0;i<15;i++)
            t.add(data[i]);
          t.transverse();
          t.del(16);
          t.transverse();
          t.del(41);
          t.transverse();
          getch();
        }
```

19.6 Graphs

A set of items connected by edges is known as **graph**. Each item is called a **vertex** or **node**. A connection between two vertices of a graph is known as **edge**. An edge is sometimes called **arc**. Graphs are a generalization of trees. In a graph, a node can have any number of incoming edges. The following figure shows a graph that contains six vertices.

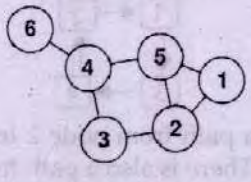


Figure 19.16: A graph with six vertices

19.6.1 Types of Graphs

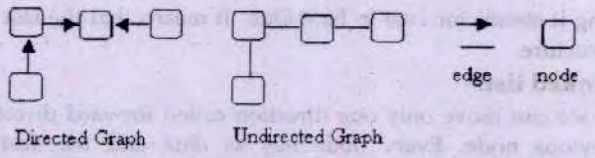
There are two kinds of graphs:

19.6.1.1 Directed Graph

A type of graph whose edges are ordered as pairs of vertices is called **directed graph**. It means that each edge can be followed from one vertex to another vertex. It is represented by arrows which specify a certain direction of a path. The nodes are not equally adjacent to one another. A node is only adjacent to the one it is pointing to. We will represent a directed edge by lines with arrows on the end of them.

19.6.1.2 Undirected Graphs

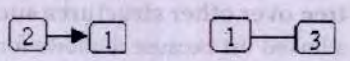
A type of graph whose edges are unordered pairs of vertices is called **undirected graph**. It means that each edge connects two vertices.



The edges in a directed graph are directed from one node to another. On the other hand, the edges in undirected graph are plain lines without any direction. In a directed graph, you can only go from node to node following the direction of the arrows. In an undirected graph, you can go either way along an edge.

Example

The following figure shows two graphs (one directed and one undirected):



In the directed graph, there is an edge from node 2 to node 1. Therefore:

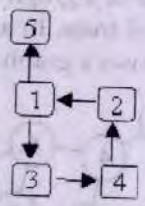
- The two nodes are **adjacent** (they are **neighbors**).
- Node 2 is a **predecessor** of node 1.
- Node 1 is a **successor** of node 2.
- The **source** of the edge is node 2, and the **target** of the edge is node 1.

Freebooks.pk

In the undirected graph, there is an edge between node 1 and node 3. Therefore:

- Nodes 1 and 3 are adjacent (they are neighbors).

Now consider the following directed graph:



In the above graph, there is a path from node 2 to node 5: 2 -> 1 -> 5. There is a path from node 1 to node 2: 1->3->4->2. There is also a path from node 1 back to itself: 1->3->4->2->1. The first two paths are acyclic paths as no node is repeated. The last path is a cyclic path because node 1 occurs twice.

Exercise Questions

Q.1. What do you know about data structure?

Data structure refers to the way we arrange data or the way in which data is stored in memory. There are different types of data structures. Array is a simple data structure.

Q.2. What is linked list? List different types of Link-List.

A linklist is a type of data structure that is built from structures and pointers. It is a linear and dynamic data structure. It provides flexiblity as the number of nodes in it is not specified at compile time. There three types of linklist are Single Linklist , double Linklist and circular Linklist.

Q.3. What does LIFO mean?

Litterally speaking it stands for Last in First Out. It means that the last item entered into a first item removed from a structure.

Q.4. What is single linked list?

In single linklist we can move only one direction called forward direction and could not back from the node to previous node. Every node has its data and the address of the next node.

Q.5. What is double linked list?

In double linklist, we can move in both directions. It is a two-way linklist. Every node has previous node address, its data and next node address. The first node is called head of the list and has null value in previous nodes address and the last node has also null value in the next node address.

Q.6. What is circular linked list?

The circular linklist is similar to double link list but it has no head node and it has no null value in any node. Actually in a circular linklist the lat node of the link connects to the first node of the list so the double link list becomes the circular linklist.

Q.7. List advantages of binary tree over other structures such as a linked list or an array?

A binary tree is better than a linked list because it allows for faster inserts, searches, and deletes. It maintains the order of the tree as well. It's advantage over an array is that it does not require the maximum size of the struture to be determined before we start adding values to it.

Q.8. What is a tree?

A tree is a structure starting with a root node. It may have as many child nodes as desired. The child nodes can contain any values. The children can have children themselves.

Q.10. Can a child node also be a parent node?

A child node can be parent node. A root can have children and children can have more children.

Freebooks.pk

Q.11. How does a leaf node differ from a child node?

A child node is a decendant of a node. Similarly, leaf is also a descendant of a node. However, a child can have children of its own whereas a leaf can not have children.

Q.12. What is the maximum number of children that a node can have in a binary tree?

The maximum number of children that a node can have in binary tree is two (2).

Q.13. What is the minimum number of nodes that a tree can have? What is the maximum?

The minimum number of nodes in a tree is zero that indicates an empty tree. There is no limit on the maximum number of nodes. The only the limit is the computer's memory.

Multiple Choice

1. **Which of the following is used to create a linked list?**
   a. header file b. Exception c. Both a and b d. Struct
2. **Which of the following is a basic linked list operation?**
   a. Appending a node b. traversing the list c. Inserting or deleting a node d. All
3. **The last node in a linked list points to:**
   a. First node b. previous node c. NULL d. All
4. **List contains pointers to the nodes before it and after it:**
   a. Singly-linked b. doubly-linked c. circular-linked d. b and c
5. **Which of following type of list does not contain a NULL pointer at the end of the list?**
   a. Circular-linked b. doubly-linked c. NULL-linked d. None
6. **Which can be used to facilitate adding nodes to the end of linear linked list?**
   a. head pointer b. Zero head node c. tail pointer d. precede pointer
7. **Which of the following is used to stores and retrieves items in a last-in-first-out manner:**
   a. Array b. Queue c. Stack d. None
8. **The item that is removed first from a stack is called:**
   a. Front b. Top c. Base d. None
9. **The last in, first-out (LIFO) property is found in the ADT:**
   a. List b. Stack c. Queue d. Tree
10. **Which of the following operation is used to add an item to the top of the stack?**
    a. createStack b. push c. pop d. None
11. **Which of the following operation is used to retrieves and then removes the top of the stack?**
    a. createStack b. push c. pop d. None
12. **Which of the following is used to store and retrieves items in a last-in-first-out manner?**
    a. Array b. Queue c. Stack d. None
13. **A stack has two primary operations:**
    a. Push and pull b. Push and pop c. Insert and delete d. Append and delete
14. **A queue is a data structure that stores and retrieves items in this manner:**
    a. last in, first out b. first in, last out c. first in, first out d. None
15. **When an element is added to a queue, it is added to the:**
    a. rear b. middle c. Front d. All
16. **When an element is removed, it is removed from the:**
    a. rear b. middle c. Front d. All
17. **Binary trees can be divided into:**
    a. Leaves b. Branch c. Subtrees d. None
18. **An operation that can be performed on a binary search tree is:**
    a. Insertion b. finding c. deleting d. All
19. **Each node in a binary tree has:**
    a. Exactly one child b. Exactly two children c. At most two children d. None
20. **The first node in a binary tree list is called the:**
    a. head pointer b. Binary node c. Root node d. None

Freebooks.pk

Answers

1. d	2. d	3. c	4. d	5. a
6. c	7. c	8. b	9. b	10. b
11. c	12. c	13. b	14. c	15. a
16. c	17. c	18. d	19. c	20. c

True/ False

1. Linked list consists of a sequence of nodes.
2. A linked list allows the insertion and removal of nodes at any point in the list but it does not allow random access.
3. A doubly linked list is also known as two-way linked list.
4. A stack is a type of data structure based on the technique of Last In First Out (LIFO).
5. A stack is a specialized type of list.
6. A push operation is now allowed when the stack is full.
7. The NULL can be assigned only to pointers that point to numbers.
8. A linked list is not fixed in size.
9. A linked list is called "linked" because each node in the series has a pointer that point to the next node in the list.
10. A pop operation is not allowed when the stack is empty.
11. If there are no nodes in a linked list, you cannot append a node to the list.
12. A stack can be represented by using arrays.
13. A new node must always be made the last node in the list.
14. A new node cannot become the first node in the list.
15. Each node must have a minimum of two children In a binary tree.
16. Linked lists are less complex to code and manage than arrays.
17. Nodes in a linked list are stored in contiguous memory.
18. The first item placed onto a stack is always the last item removed from the stack.

Answers

1. T	2. T	3. T	4. T	5. T	6. T
7. F	8. T	9. T	10. T	11. F	12. T
13. F	14. F	15. F	16. F	17. F	18. T

Freebooks.pk