Object Oriented Programming Using C++

SECOND YEAR DIPLOMA

(Code : 22316)

Maharashtra State Board of Technical Education (MSBTE)

Semester III – Computer Engineering Group / Information Technology Engineering Group (CO/CM/IF/CW)

Strictly as per new revised 'I' Scheme w.e.f. academic year 2018-2019

Harish G. Narula

Formerly, Assistant Professor (Senior),
Department of Computer Engineering
D. J. Sanghvi College of Engineering, Mumbai.
Maharashtra, India.

TechKnowledge™ Publications

MDO14B Price ₹ 215/-

(Book code : MDO14B)

Object Oriented Programming Using C++

Harish G. Narula

Second Year Diploma : Semester III (MSBTE)

[Computer Engineering Group, Information Technology Engineering Group (CO/CM/IF/CW)]

Copyright © by Author. All rights reserved. No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

First Edition : June 2018 **(As per 'I' Scheme)**
Second Revised Edition : June 2019 **(TechKnowledge Publications)**
Third Revised Edition : July 2020

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : 37/2, Ashtvinayak Industrial Estate, Near Pari Company,
Narhe, Pune, Maharashtra State India,
Pune – 411041

ISBN : 978-93-89233-03-2

Published by

TechKnowledge Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner,
Pune - 411 009. Maharashtra State, India

Ph : 91-20-24221234, 91-20-24225678.

Email : info@techknowledgebooks.com,

Website : www.techknowledgebooks.com

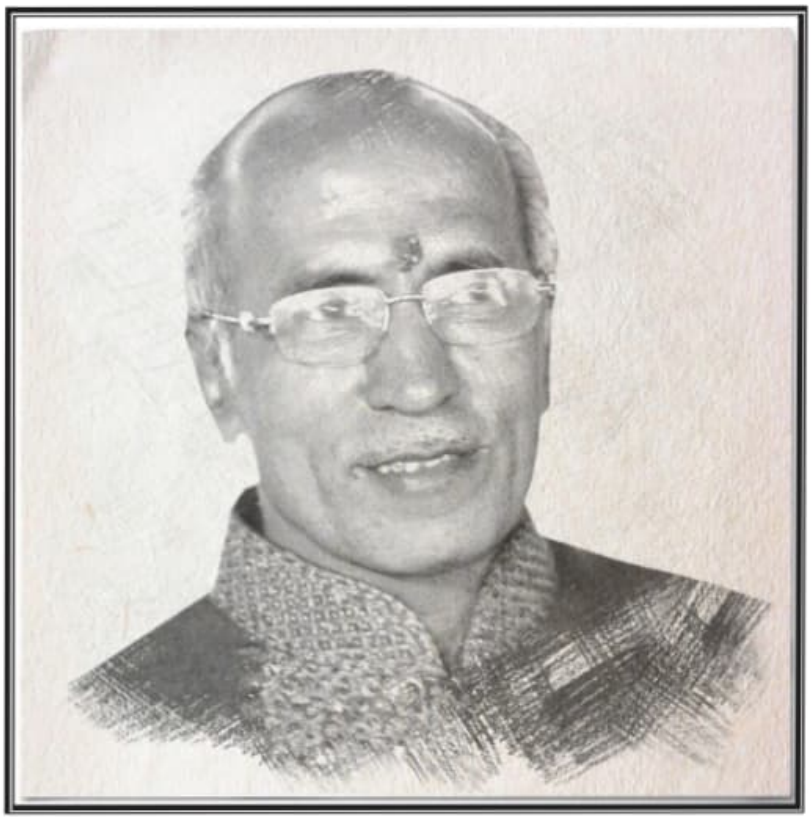
[22316] (FID : MDO14) [New Book Code : MDO14B]

We dedicate this Publication soulfully and wholeheartedly,

in loving memory of our beloved founder director,

Late Shri. Pradeepji Lalchandji Lunawat,

who will always be an inspiration, a positive force and strong support behind us.



"My work is my prayer to God"

- Lt. Shri. Pradeepji L. Lunawat

Soulful Tribute and Gratitude for all Your
Sacrifices, Hardwork and 40 years of Strong Vision...

Preface

My Dear Students,

I am extremely happy to come out with this book on "Object Oriented Programming Using C++ " for you. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

I present this book in the loving memory of **Late Shri. Pradeepji Lunawat,** my source of inspiration and a strong foundation of **"TechKnowledge Publications"**. He will always be remembered in our heart and motivate us to achieve our milestone.

I am thankful to Shri. J. S. Katre, Shri. Shital Bhandari, Shri. Arunoday Kumar and Shri. Chandroday Kumar for the encouragement and support that they have extended. I am also thankful to Seema Lunawat for technology enhanced reading, E-books support and the staff members of TechKnowledge Publications for their efforts to make this book as good as it is. I have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let me know, because that will help me to improve further.

I am also thankful to my family members and friends for patience and encouragement.

I would also like to thank Guruji Shree Swami Satyanandji Maharaj for his blessings.

- Author

❑❑❑

(Book code : MDO14B)

Syllabus

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
Unit - I : Principles of Object Oriented Programming **Refer chapters 1, 2, 3 and 4**	1a. Write simple C++ program for solving the given expression using POP approach. 1b. Write POP based C++ program using decision making and loop structure for the given situation. 1c. Write POP based C++ program using arrays to solve the given problem. 1d. Use the structure in C++ program for solving the given problem.	1.1 Procedure Oriented Programming (POP) verses Object Oriented Programming (OOP) 1.2 Basic Concepts of Object Oriented Programming, Object Oriented Languages, Applications of OOP. 1.3 C verses C++, Structure of C++ program, Simple C++ program. 1.4 Tokens, Keyword, Variables, Constants, Basic data types, User defined data types, Type casting, Operators, Expressions. 1.5 Control Structures : Decision making Statements and Loops. 1.6 Scope Resolution Operator, Memory Management Operator. 1.7 Arrays, Strings and structures in C++.
Unit - II : Classes and Objects **Refer chapter 5**	2a. Develop relevant friend functions to solve the given problem. 2b. Write C++ program to use array of given objects. 2c. Write C++ program to create the given object using constructor. 2d. Write program to delete the given object using destructor in C++ program.	2.1 Class and Object : Introduction, Specifying a Class, Access Specifiers, Defining Member Functions, Creating Objects and Memory Allocation for Objects. 2.2 Static Data Members, Static Member Function, Friend Function. 2.3 Array of Objects, Objects as Function Arguments. 2.4 Concepts of Constructors, Types of Constructors. 2.5 Multiple Constructors in a Class, Constructors with Default Arguments. 2.6 Destructors.

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
Unit - III : Extending classes using Inheritance **Refer chapter 6**	3a. Explain given type of inheritance based on its characteristic. 3b. Implement given type of inheritance in C++ program. 3c. Write C++ program using virtual base class. 3d. Use constructor in the given derived class.	3.1 Introduction to Inheritance, defining a derived class, Visibility Modes and Effects. 3.2 Types of Inheritance – Single, multilevel, multiple, hierarchical hybrid. 3.3 Virtual base class, abstract class, constructors in derived class.
Unit - IV : Pointers and Polymorphism in C++ **Refer chapter 7**	4a. Create C++ programs to perform the given arithmetic operations using pointers. 4b. Use function overloading to solve the given problem. 4c. Use operator overloading to solve the given problem. 4d. Implement run time polymorphism using virtual functions in the given C++ program.	4.1 Concepts of Pointer : Pointer declaration, Pointer operator, address operator, Pointer arithmetic. 4.2 Pointer to Object : Pointer to Object, this pointer, Pointer to derived class. 4.3 Introduction of Polymorphism, Types of Polymorphism, 4.4 Compile time Polymorphism : Function overloading, operator overloading, overloading of unary and binary operators, Rules for operator overloading. 4.5 Run time polymorphism : Virtual functions, rules for virtual functions, pure virtual function.
Unit - V : File operations **Refer chapter 8**	5a. Identify relevant class for performing the given file operation. 5b. Write statement to open and close the given file in C++. 5c. Develop C++ program to perform read/write operation from/to the given file.	5.1 C++ Stream Classes, Classes for File stream operations. 5.2 Opening Files, Closing Files, Reading from and Writing to Files. 5.3 Detection of End of File, File Modes

Note : *To attain the COs and competency, above listed UOs need to be undertaken to achieve the 'Application Level' and above of Bloom 's' Cognitive Domain Taxonomy'*

Practicals (Lab Manual)

Sr. No.	Practical Outcomes (PrOs)	Unit No.	Approx. Hrs. Required
1.	Develop minimum 2 programs using constants, variables, arithmetic expression, operators, exhibiting data type conversion.	I	02*
2.	Develop a program to implement decision making statements (If-else, switch).	I	02
3.	Develop a program to demonstrate control structures (for, while, do-while)	I	02
4.	Develop a program to implement 1-dimension array.	I	02*
5.	Develop a program to perform matrix operations using multi-dimensional array.	I	02
6.	Develop programs that implements a class and use it with objects.	II	02*
7.	Develop programs that implements a class and create array of objects.	II	02*
8.	Write a program to implement friend function.	II	02*
9.	Write a program to implement inline function.	II	02*
10.	Write a program to implement all types of constructors (constructor overloading) with destructor.	II	02*
11.	Write a program for implementing single inheritance	III	02*
12.	Write a program for implementing multi level inheritance.	III	02*
13.	Write a program for implementing multiple inheritance.	III	02*
14.	Develop minimum 1 program to demonstrate Pointer to object.	IV	01*
15.	Develop minimum 1 program to demonstrate Pointer to derived class.	IV	01*
16.	Write a program to demonstrate operator overloading for Unary operator.	IV	02
17.	Write a program to demonstrate operator overloading for Binary operator.	IV	02
18.	Write a program to demonstrate function overloading.	IV	02*
19.	Write a program to read and write data to and from a file.	V	02
	TOTAL		**38**

❑❑❑

INDEX

UNIT I

Chapter 1 : Structured Programming with C++ 1-1 to 1-29

Syllabus : Principles of Object Oriented Programming

Unit Outcomes : Write simple C++ program for solving the given expression using POP approach.

Topics and Sub-Topics : Procedure Oriented Programming (POP) Verses Object Oriented Programming (OOP), Basic Concepts of Object Oriented Programming and Object Oriented Languages, Applications of OOP, C verses C++ , Structure of C++program, Simple C++ program, Tokens, Keyword, Variables, Constants, Basic data types, User defined data types, Type casting, Operators, Expressions, Memory Management Operator

1.1 Computer 1-1
1.2 Programming and Programming Languages 1-1
1.3 Binary Number System 1-2
1.4 History of C / C++ Programming Languages 1-5
1.5 Procedure Oriented Programming 1-5
1.6 Basics of Object Oriented Programming (OOP) and OOP Languages 1-6
1.6.1 Procedure Oriented Versus Object Oriented Programming Language or 'C' Language versus 'C++' Language 1-7
1.6.2 Applications of Object Oriented Programming Language 1-7
1.7 Features of C Programming Language 1-7
1.8 C++ as a Superset of C Programming Language 1-7
1.9 Tokens of C / C++ 1-8
1.9.1 Character Set of C / C++ 1-8
1.9.2 Keywords 1-8
1.9.3 Identifiers 1-9
1.9.4 Data Types 1-9
1.9.5 Constants and Variables 1-11
1.9.6 Escape Sequences 1-11
1.9.7 Operators 1-11
1.10 Structure of C++ program 1-11
1.10.1 Simple C++ Program 1-11
1.10.2 Dynamic Initialization of Variables 1-12
1.10.3 User Defined Data type 1-13
1.10.3.1 Enumerated Data type 1-13
1.11 Integrated Development Environment (IDE) 1-13
1.12 Operators 1-14
1.12.1 Unary Operators 1-14
1.12.2 Binary Operators 1-16
1.12.3 Ternary Operator 1-18
1.12.4 Assignment Operators 1-18
1.12.5 Selection Operators 1-19
1.12.6 Memory Management Operators 1-19
1.13 Precedence and Associativity of Operators 1-20
1.14 Expressions 1-20
1.15 Formatted and Unformatted IO Functions 1-22
1.15.1 Formatted IO Functions 1-22
1.15.2 Unformatted IO Functions 1-23
1.16 C++ IO Techniques 1-23
1.17 Manipulators 1-24
1.18 Basic Program Examples 1-24
1.18.1 Type Casting 1-26
1.19 Exam Pack (University and Review Questions) 1-27

Chapter 2 : Control Statements 2-1 to 2-26

Syllabus : Principles of object Oriented Programming

Unit Outcomes : Write POP based C++ program using decision making and loop structure for the given situation.

Topics and Sub-Topics : Control Structures : Decision making Statements and Loops

2.1 Control Structures : Decision making Statements and Loops 2-1
2.2 for Loop 2-1
2.2.1 Programs Based on for Loop 2-2
2.2.2 Nested for Loop 2-7
2.3 while and do-while Loops 2-10
2.3.1 Programs Based on while and do-while Loop 2-12
2.4 if-else Selective Statement 2-13
2.4.1 Programs using if-else Statement 2-14
2.4.2 if-else Ladder or if-else if 2-16
2.5 Switch-Case Selective Statement 2-18
2.6 Branching Statements (break, continue and goto) 2-20
2.7 Some More Practice Programs 2-22
2.8 Exam Pack (MSBTE and Review Questions) 2-25

Chapter 3 : Function, Arrays and Strings 3-1 to 3-27

Syllabus : Principles of object Oriented Programming

Unit Outcomes

Write POP based C++ program using arrays to solve the given problem.

Topics and Sub-Topics

Arrays in C++, Strings in C++, Scope Resolution Operator.

3.1 Functions 3-1

3.2 Programs Based on Function 3-3

3.3 Recursive Functions 3-4

3.4 Arrays 3-6

3.5 Multi-dimensional Arrays 3-13

3.6 Strings 3-19

3.7 The Header File string.h 3-19

3.7.1 strlen() function 3-20

3.7.2 strcpy() function 3-20

3.7.3 strcmp() function 3-21

3.7.4 strcat() function 3-21

3.8 Programs Based on Strings 3-22

3.9 Storage Classes and use of Scope Resolution Operator 3-24

3.10 Exam Pack (MSBTE and Review Questions) 3-27

Chapter 4 : Structures and Unions 4-1 to 4-12

Syllabus : Principles of Object Oriented Programming

Unit Outcomes : Use the structure in C++ program for solving the given problem

Topics and Sub-Topics : Structures in C++

4.1 Structures 4-1

4.2 Arrays of Structure Variable 4-4

4.3 Nested Structures 4-9

4.4 Unions 4-11

4.5 Exam Pack (MSBTE and Review Questions) 4-12

UNIT II

Chapter 5 : Classes and Objects 5-1 to 5-30

Syllabus : Classes and Objects

Unit Outcomes

Develop relevant friend functions to solve the given problem.

Write C++ program to use array of given objects.

Write C++ program to create the given object using constructor.

Write program to delete the given object using destructor in C++ program.

Topics and Sub-Topics

Class and Object : Introduction, Specifying a Class, Access Specifiers, Defining Member Functions, Creating Objects and Memory Allocation for Objects.

Static Data Members, Static Member Function, Friend Function.

Array of Objects, Objects as Function Arguments.

Concepts of Constructors, Types of Constructors.

Multiple Constructors in a Class, Constructors with Default Arguments.

Destructors.

5.1 Comparison of Procedure Oriented Programming and Object Oriented Programming 5-1

5.1.1 Class and Object : Introduction 5-2

5.1.2 Access Specifiers 5-3

5.1.3 Specifying a Class 5-4

5.1.4 Creating Objects and Memory Allocation of Objects 5-4

5.2 Defining Member Functions of a Class 5-4

5.2.1 Internally Defined Functions 5-5

5.2.1.1 Nesting of Member Function 5-7

5.2.2 Externally Defined Functions 5-8

5.2.3 Inline Member Functions 5-10

5.3 Constructor 5-12

5.3.1 Default Constructor 5-12

5.3.2 Parameterized Constructor 5-14

5.3.3 Copy Constructor (Multiple Constructors/Constructor Overloading) 5-16

5.3.4 Constructors with Default Arguments 5-18

5.4 Destructor 5-19

5.5 Friend Function 5-20

5.6 Static Members 5-22

5.7 Array of Objects 5-22

5.8 Objects as Function Arguments 5-26

5.9 Exam Pack (MSBTE and Review Questions) 5-27

UNIT III

Chapter 6 : Inheritance 6-1 to 6-22

Syllabus : Extending classes using Inheritance

Unit Outcomes

Explain given type of inheritance based on its characteristic.

Implement given type of inheritance in C++ program.

Write C++ program using virtual base class.

Use constructor in the given derived class.

Topics and Sub-Topics

Introduction to Inheritance, defining a derived class, Visibility Modes and Effects, Types of Inheritance – Single, multilevel, multiple, hierarchical, hybrid, Constructors in derived class.

6.1 Inheritance 6-1

6.1.1 Visibility Modes and Effects 6-1

6.1.2 Single Inheritance 6-2

6.1.2.1 Constructor in Derived Class 6-6

6.1.3 Multi Level Inheritance 6-6

6.1.4 Multiple Inheritance 6-11

6.1.5 Hybrid Inheritance 6-13

6.1.6 Problem in Multiple and Hybrid Inheritance 6-15

6.1.7 Hierarchical Inheritance 6-16

6.2 Exam Pack (MSBTE and Review Questions) 6-20

UNIT IV

Chapter 7 : Pointers & Polymorphism in C++ **7-1 to 7-37**

Syllabus : Pointers and Polymorphism in C++

Unit Outcomes

Create C++ programs to perform the given arithmetic operations using pointers.

Use function overloading to solve the given problem.

Use operator overloading to solve the given problem.

Implement run time polymorphism using virtual functions in the given C++ program.

Topics and Sub-topics

Concepts of Pointer : Pointer declaration, Pointer operator, address operator, Pointer arithmetic.

Pointer to Object : Pointer to Object, this pointer, Pointer to derived class. Virtual Base Class and Abstract Class

Introduction of Polymorphism, Types of Polymorphism,

Compile time Polymorphism : Function overloading, operator overloading, overloading of unary and binary operators, Rules for operator overloading.

Run time polymorphism : Virtual functions, rules for virtual functions, pure virtual function.

7.1 Pointers 7-1

7.1.1 Referencing and De-referencing (Operators in Pointers) 7-2

7.2 Programs Related to Pointers 7-2

7.2.1 Simple Referencing and De-referencing using Pointers 7-2

7.2.2 Pointer to Pointer 7-3

7.2.3 Pointer Arithmetic: Increment and Decrement Operators Operation on Pointer Variables 7-4

7.2.4 Pointer to Array 7-4

7.2.5 Passing Pointers to Functions 7-5

7.2.6 Call by Value and Call by Reference 7-5

7.3 Function Overloading or Function Polymorphism 7-10

7.3.1 Constructor Overloading 7-13

7.4 Operator Overloading (Overloading of Unary and Binary Operators) 7-14

7.5 Polymorphism 7-23

7.5.1 Dynamic Binding using Virtual Function or Compile/Run time Polymorphism 7-24

7.5.2 Rules for Virtual Functions 7-24

7.5.2.1 Pointer to Derived Class 7-26

7.5.3 Virtual Base Class and Abstract Class 7-29

7.6 this Pointer 7-30

7.7 Pointer to Object 7-31

7.8 Exam Pack (MSBTE and Review Questions) 7-34

UNIT V

Chapter 8 : File Operations **8-1 to 8-13**

Syllabus : File Operation

Unit Outcomes : Identify relevant class for performing the given file operation, Write statement to open and close the given file in C++, Develop C++ program to perform read/write operation from/to the given file.

Topics and Sub-Topics : C++ Stream Classes , Classes for File Stream Operations, Opening Files, Closing Files, Reading from and Writing to Files, Detection of End of File , File Modes

8.1 Concept of Streams and C++ stream Classes 8-1

8.1.1 File Stream Classes 8-1

8.1.2 Advantages of Stream Classes 8-2

8.2 File Management Functions and File Modes (Opening and Closing Files) 8-2

8.2.1 Reading from and Writing to Files 8-4

8.2.2 Detecting End of File 8-5

8.3 Random and Binary File Handling 8-7

8.4 Exceptions and Error Handling During File Operations 8-11

8.5 Solved Example 8-11

8.6 Exam Pack (MSBTE and Review Questions) 8-12

- **Appendix A :** Solved MSBTE Question Papers of Winter 2018 and Summer 2019 ....A-1 to A-14
- **Appendix B :** Solved MSBTE Question Paper of Winter 2019 ....B-1 to B-3

Lab Index

Statement	Name of the Program	Page Nos.
Statement 1	**Develop minimum 2 programs using constants, variables, arithmetic expression, operators, exhibiting data type conversion.**	L-1
	• **Program 1 :** Write a C / C++ program to accept a number and display its square.	L-1
	• **Program 2 :** Write a program to accept basic salary from the keyboard. Calculate the gross salary that includes basic salary, 50% DA and 40% HRA.	L-1
	• **Program 3 :** Write a program to accept a number and display its equivalent ASCII using type casting.	L-2
Statement 2	**Develop a program to implement decision making statements (If-else, switch).**	L-2
	• **Program 4 :** Write a program to check if the entered number is prime number or not.	L-2
	• **Program 5 :** Write a program to check if the entered number is Armstrong or not.	L-2
	• **Program 6 :** Write a program to display the class according to the marks scored by a student. The marks scored is taken as input and the class is displayed according to the following range :	L-3
	• **Program 7 :** Write a program to display a user entered number in words.	L-4
Statement 3	**Develop a program to demonstrate control structures (for, while, do-while).**	L-4
	• **Program 8 :** Write a program to display first n elements of Fibonacci series.	L-4
	• **Program 9 :** Write a program to calculate the value of the following series. $1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}$	L-5
	• **Program 10 :** Write a program to count the number of digits in a user entered number.	L-6
	• **Program 11 :** Write a program to find the sum and product of all the digits of a user entered number using the do-while loop.	L-6
Statement 4	**Develop a program to implement 1-dimension array.**	L-6
	• **Program 12 :** Write a program to sort numbers in ascending order. **OR** Write a program to implement bubble sorting algorithm for sorting numbers in ascending order.	L-6
Statement 5	**Develop a program to perform matrix operations using multi-dimensional array.**	L-8
	• **Program 13 :** Write a program to multiply two matrices using a function.	L-8
Statement 6	**Develop programs that implements a class and use it with objects.**	L-10
	• **Program 14 :** Write a program to find area of circle using Object Oriented Programming such that the class circle must have three member functions namely : (a) read() to accept the radius from the user. (b) compute() for calculating the area. (c) display() for displaying the result.	L-10

Marks	Class
70-100	Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

Statement	Name of the Program	Page Nos.
Statement 7	**Develop programs that implements a class and create array of objects.**	L-11
	• **Program 15 :** Write a program to declare a class 'staff' having data members as name and department. Accept this data for 10 staffs and display names of staff that are in cm department.	L-11
Statement 8	**Write a program to implement friend function.**	L-12
	• **Program 16 :** Write a program to add two complex numbers using a friend function.	L-12
Statement 9	**Write a program to implement inline function.**	L-13
	• **Program 17** : Write a program to find area of circle using Object Oriented Programming such that the class circle must have three inline functions namely : (a) read() to accept the radius from the user. (b) compute() for calculating the area. (c) display() for displaying the result.	L-13
Statement 10	**Write a program to implement all types of constructors (constructor overloading) with destructor.**	L-13
	• **Program 18 :** Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the parameterized constructor and the class circle must have two inline functions namely : (a) compute() for calculating the area. (b) display() for displaying the result.	L-13
	• **Program 19 :** Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the copy constructor and the class circle must have two inline functions namely : (a) compute() for calculating the area. (b) display() for displaying the result.	L-14
	• **Program 20 :** Write a program to demonstrate the destructor.	L-15
Statement 11	**Write a program for implementing single inheritance.**	L-15
	• **Program 21 :** Write a program to add two numbers using single inheritance such that the base class function must accept the two numbers from the user and the derived class function must add these numbers and display the sum.	L-15
Statement 12	**Write a program for implementing multi level inheritance.**	L-17
	• **Program 22 :** Write a program to calculate percentage of a student using multi level inheritance. The base class function will accept the marks in three subjects from user. A class will be derived from the above mentioned class that will have a function to find the total marks obtained and another class derived from this will have functions to calculate and display the percentage scored.	L-17
Statement 13	**Write a program for implementing multiple inheritance.**	L-18
	• **Program 23 :** Write a program to define the following relationship using multiple inheritance.	L-18
Statement 14	**Develop minimum I program to demonstrate Pointer to object.**	L-19
	• **Program 24 :** Write a program to declare a class birthday having data members day, month and year. Accept this information for 5 objects using pointer to the array of object	L-19
Statement 15	**Develop minimum I program to demonstrate Pointer to derived class.**	L-20
	• **Program 25 :** Write a program to demonstrate use of pointer to object.	L-20
Statement 16	**Write a program to demonstrate operator overloading for Unary operator.**	L-21
	• **Program 26 :** Write a program to negate the values of two variables contained in an object.	L-21

Statement	Name of the Program	Page Nos.
	• **Program 27 :** Write a program to negate the values of two variables contained in an object by overloading the operator unary negate i.e. '-'.	L-21
Statement 17	**Write a program to demonstrate operator overloading for Binary operator.** • **Program 28 :** Write a program to overload binary operator "+=" to add two complex numbers.	L-22 L-22
Statement 18	**Write a program to demonstrate function overloading.** • **Program 29 :** Write a program to add two numbers using function overloading such that one function adds two integers, second function adds two float numbers and the third function adds a float number with an integer.	L-23 L-23
Statement 19	**Write a program to read and write data to and from a file.** • **Program 30 :** Write a program to write and read string, integer and float from/to a file.	L-24 L-24

❑❑❑

UNIT I

CHAPTER 1

Structured Programming with C++

Syllabus

Principles of Object Oriented Programming

- **Unit Outcomes**
  - Write simple C++ program for solving the given expression using POP approach.
- **Topics and Sub-Topics**
  - Procedure Oriented Programming (POP) Verses Object Oriented Programming (OOP)
  - Basic Concepts of Object Oriented Programming and Object Oriented Languages, Applications of OOP
  - C verses C++ , Structure of C++program, Simple C++ program
  - Tokens, Keyword, Variables, Constants, Basic data types, User defined data types, Type casting, Operators, Expressions
  - Memory Management Operator.

1.1 Computer

- The computer as seen by you can be as illustrated in the Fig. 1.1.1.

Fig. 1.1.1 : Computer

- Besides other units, it is made of a monitor (standard display unit), CPU (Central Processing Unit) and keyboard (standard input device).
- A **computer** can be visualized as a system with Input/Output (I/O) devices, Central Processing Unit (CPU) and the memory interconnected as shown in the Fig. 1.1.2. This is the hardware view of a computer.

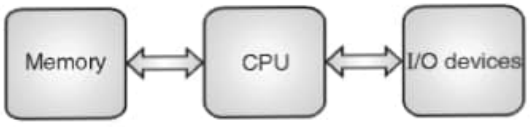


Fig. 1.1.2 : Hardware structure of a computer

- As a programmer, the **computer** can be seen as a programmable system. It is a system that can be programmed by the programmer.

1.2 Programming and Programming Languages

- **Programming** is to give the machine a list of steps to perform a particular task.

- If the system to which the programming is done is a computer than it is called as **Computer Programming**.
- The programming of any system has to be done in the language understood by that system.
- A digital system like computer understands only binary language (which consists only of 0s and 1s), also called as **machine language**. But, programming in machine language is almost impossible for a human being. Hence, the manufacturers of the processor develop a language called as **assembly language**.
- Assembly language is simpler than the machine language, but making huge software using this is again very difficult. It includes the following :
  - An English language word that specifies the operation to be performed and
  - The operands, which are the data on which the operation is to be performed.
- Machine language and assembly language are called as low level languages.
- To remove the complexity of programming, which was there because of low level languages, **High Level Languages (HLL)** were developed. HLL are simplest for programming the processors. Some of these languages are FORTRAN, BASIC, COBOL, C, C++, JAVA, .net, etc.
- HLL programming languages like C / C++ are structured programming languages and hence are quite easy for the programmers. C / C++ are sometimes also referred to as middle level languages as they are not fully high level languages and neither are they low level languages.
- In this subject we have to learn these two programming languages viz. C and C++. Also we have to make some small programs using these languages.

1.3 Binary Number System

- As seen above that a computer can understand only binary language, let us first see what is this binary language ?
- Binary number system can have only two representations namely '0' and '1'.
- All numbers and characters are to be represented using '0' or '1' only.
- The conversion from binary to decimal and decimal to binary can be done as shown below with examples.
- Decimal to binary : The steps to be followed for converting a decimal number to a binary number are :
  - Divide the decimal number by 2.
  - Note down the quotient and remainder as shown in the example below.
  - Repeat the above procedure till the quotient is zero.
  - The last remainder is the MSB (Most Significant Bit i.e. the bit with highest weight) and the first remainder is the LSB (Least Significant Bit).

    E.g. $(5)_{10} = (\ ?\ )_2$

$(5)_{10}$ means the number 5 in decimal (base 10) number system. Base 10 means 10 different representations 0 to 9.

$(?)_2$ means the number in binary (base 2) number system. Base 2 means 2 different representations 0 and 1.

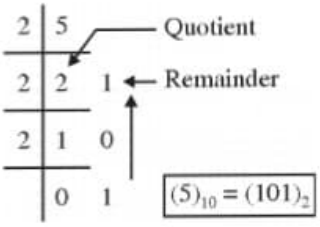


Fig. 1.3.1

- **Binary to decimal :** The steps to be followed for converting a binary number to a decimal number are :
  - Multiply the binary digits (bits) with powers of 2 according to their positional weight. The position of the first bit (going from right to left) is 0 and it keeps on incrementing as you go towards left for every bit.

E.g. $(11100)_2 = (\ ?\ )_{10}$

$$= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

$$= 16 + 8 + 4 + 0 + 0 = (28)_{10}$$

- This is how your decimal data is stored in the computer. You store 5, but it stores 101 in binary form.
- Let us see some bitwise logical operations that are performed on binary data.
- There are various operations that are performed on binary data like AND, OR, EXOR and NOT. These operations are explained below.

1. **AND :** The output is 1 if and only **if A AND B** (A & B being the inputs) are 1.

A(INPUT)	B(INPUT)	Y(OUTPUT)
0	0	0
0	1	0
1	0	0
1	1	1

2. **OR :** The output is 1 if and only **if A OR B** (A & B being the inputs) is 1.

A(INPUT)	B(INPUT)	Y(OUTPUT)
0	0	0
0	1	1
1	0	1
1	1	1

3. **NOT :** The output **is NOT A** (A being the input) i.e. complement of A.

A(INPUT)	Y(OUTPUT)
0	1
1	0

4. **EXOR :** The output is 1 if and only **if EXCLUSIVELY A OR B** (A & B being the inputs) are 1.

A(INPUT)	B(INPUT)	Y(OUTPUT)
0	0	0
0	1	1
1	0	1
1	1	0

Some examples of performing these operations on the binary data are given below :

1. 5 AND 3 :

In 'C' programming this is written as 5 & 3.

$$\begin{array}{c|cc} 2 & 5 & \\ 2 & 2 & 1 \\ 2 & 1 & 0 \\ & 0 & 1 \end{array} \quad (5)_{10} = (101)_2$$

$$\begin{array}{c|cc} 2 & 3 & \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (3)_{10} = (011)_2$$

Note : We can prefix 0's to a number in binary as in decimal. Here, $(011)_2$ is written to make it 3 digit data, as $(5)_{10} = (101)_2$ is 3 digit.

Now, 5 & 3

$$\begin{array}{r} (1\,0\,1)_2 \\ \&\ (0\,1\,1)_2 \\ \hline (0\,0\,1)_2 \\ \hline \end{array}$$

Note : & is bitwise AND operator i.e. it works on each bit as shown in the above example.

$$(001)_2 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= (1)_{10}$$

$$(5)_{10} \,\&\, (3)_{10} = (1)_{10}$$

2. 13 AND 10 :

$$\begin{array}{c|cc} 2 & 13 & \\ 2 & 6 & 1 \\ 2 & 3 & 0 \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (13)_{10} = (1101)_2$$

$$\begin{array}{c|cc} 2 & 10 & \\ 2 & 5 & 0 \\ 2 & 2 & 1 \\ 2 & 1 & 0 \\ & 0 & 1 \end{array} \quad (10)_{10} = (1010)_2$$

Now; 13 and 10

$$\begin{array}{r} (1\,1\,0\,1)_2 \\ \&\ (1\,0\,1\,0)_2 \\ \hline (1\,0\,0\,0)_2 \\ \hline \end{array}$$

$$(1000)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = (8)_{10}$$

$$(13)_{10} \,\&\, (10)_{10} = (8)_{10}$$

3. 7 OR 6 :

In 'C' programming, this is written as 7|6.

$$\begin{array}{c|cc} 2 & 7 & \\ 2 & 3 & 1 \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (7)_{10} = (111)_2$$

$$\begin{array}{c|cc} 2 & 6 & \\ 2 & 3 & 0 \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (6)_{10} = (110)_2$$

Now; 7 OR 6

$$\begin{array}{r} (1\,1\,1)_2 \\ |\ (1\,1\,0)_2 \\ \hline (1\,1\,1)_2 \\ \hline \end{array}$$

$$(111)_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$= (7)_{10}$$

$$\therefore \quad (7)_{10} | (6)_{10} = (7)_{10}$$

4. 5 OR 2 :

$$\begin{array}{c|cc} 2 & 5 & \\ 2 & 2 & 1 \\ 2 & 1 & 0 \\ & 0 & 1 \end{array} \quad (5)_{10} = (101)_2$$

$$\begin{array}{c|cc} 2 & 2 & \\ 2 & 1 & 0 \\ & 0 & 1 \end{array} \quad (2)_{10} = (010)_2$$

Now, 5 OR 2

$$\begin{array}{r} (1\,0\,1)_2 \\ |\ (0\,1\,0)_2 \\ \hline (1\,1\,1)_2 \\ \hline \end{array}$$

$$(111)_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$= (7)_{10}$$

$$\therefore \quad (5)_{10} | (2)_{10} = (7)_{10}$$

5. 6 EXOR 3 :

In C, this is written as 6 ^ 3

$$\begin{array}{c|cc} 2 & 6 & \\ 2 & 3 & 0 \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (6)_{10} = (110)_2$$

$$\begin{array}{c|cc} 2 & 3 & \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (3)_{10} = (011)_2$$

Now; 6 EXOR 3

$$\begin{array}{r} (1\,0\,1)_2 \\ \wedge\ (0\,1\,1)_2 \\ \hline (1\,0\,1)_2 \\ \hline \end{array}$$

$$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= (5)_{10}$$

$$\therefore \quad (6)_{10} \wedge (3)_{10} = (5)_{10}$$

6) NOT 3 :

In C, this is written as ~ 3

$$\begin{array}{c|cc} 2 & 3 & \\ 2 & 1 & 1 \\ & 0 & 1 \end{array} \quad (3)_{10} = (011)_2$$

Now; ~ 3 is 0 0 0 1 1

1 1 1 0 0

Note : 1) The data may be of more than 2 bits and hence you will get the output depending on the size of data.

2) In computer binary language you will get this value as $(-4)_{10}$.

3) Hence if you say $\sim(10)_{10}$, you will get $-(11)_{10}$ and so on.

1.4 History of C / C++ Programming Languages

- C programming language was developed by Dennis Ritchie at the Bell Laboratories of AT & T. He developed a complier that can convert the HLL namely C to the machine language that can be understood by the processor or the computer. But his compiler could understand only if certain rules were followed while writing the program.
- We need to understand and know these rules to program our computers using this programming language. C was the most popular programming language.
- Some other languages had become popular after some time of evolution of C. This was because these other programming languages were object oriented i.e. more significance was given to object.
- While 'C' was a procedure oriented language i.e. more significance was given to the procedure to perform a task.
- C++ which is an object oriented programming language was then developed again at Bell Laboratories by Bjarne Stroustrup. Earlier, this language was called as C with classes, but was later renamed as C++ in 1983.
- Object oriented and procedure oriented concepts of a programming language will be gradually seen in this book. Some of these concepts will be cleared in this chapter itself, while some will come as you reach towards the end of this book.

1.5 Procedure Oriented Programming

➔ **(W-14, W-15,W-16, S-17)**

Q. 1.5.1 Write any two characteristics of procedure oriented programming. (Ref. Sec. 1.5) **(W-14, 2 Marks)**

Q. 1.5.2 List any four features of POP. (Ref. Sec. 1.5) **(W-15, 4 Marks)**

Q. 1.5.3 In procedure oriented programming all data are shared by all functions. Is this statement TRUE? Justify your answer. (Ref. Sec. 1.5) **(W-16, 4 Marks)**

Q. 1.5.4 What are the features of procedure oriented programming? (Ref. Sec. 1.5) **(S-17, 4 Marks)**

- A procedure oriented programming language is one in which procedure i.e. method or functions are given more significance. Here, one function calls another.
- There can be global variables and local variables to each function.
- Fig 1.5.1 shows how a procedure oriented programming operation.
- In this type of languages the task is divided into smaller tasks or sub tasks called as procedures or functions or methods. Any function can be called at any instant.
- There are global and local variables as seen in the Fig. 1.5.1. The global variables can be accessed by all the functions, while the local variables are local to the corresponding function.
- A procedure oriented programming language follows top bottom approach.
- Top bottom approach refers to approaching to a problem as a big task or as a whole. Then dividing this task into small instructions or operations. Hence, initially the task is broken into small tasks and then these smaller tasks are written in detail.

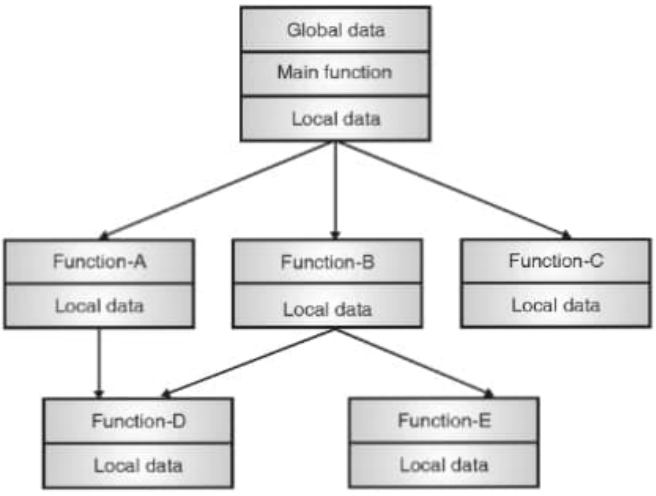


Fig. 1.5.1 : Structure of a procedure oriented programming language

- The disadvantage of such type of programming is that it is difficult to trace which functions are using a data and also error correction is difficult.
- C programming language is an example of procedure oriented language.

Syllabus Topic : Basic Concepts of Object Oriented Programming and Object Oriented Languages

1.6 Basics of Object Oriented Programming (OOP) and OOP Languages

➔ **(S-14, W-14, W-16)**

Q. 1.6.1 Describe any four basic concepts of OOP. (Ref. Sec. 1.6) **(S-14, 4 Marks)**

Q. 1.6.2 List any four object oriented languages. (Ref. Sec. 1.6) **(W-14, W-16, 2 Marks)**

- Object oriented programming as the name says gives more significance to the objects which has data and functions built around it.
- The data of the object can be accessed by the functions associated with it. The functions of one object can access the data of another object through the functions of that object.
- Object oriented programming uses bottom up approach wherein the smaller tasks are first dealt in detail and gradually creating the entire huge system.

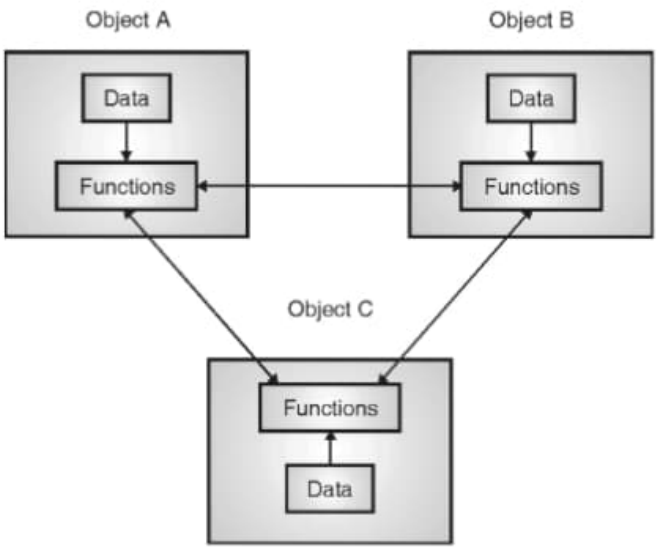


Fig. 1.6.1 : Structure of an Object Oriented Programming

- Fig. 1.6.1 shows the structure of a object oriented programming language.
- C++, Python, Ruby, C# and Java are examples of object oriented programming languages.
- There are few important terms related to Object Oriented Programming that we need to understand. First the concept of objects and classes and then the specialties of OOPs viz. Data Abstraction, Encapsulation, Inheritance, Polymorphism, etc. Let us understand them one by one.

1. Class	2. Object
3. Data abstraction	4. Data encapsulation
5. Inheritance	6. Polymorphism

➔ **1. Class**

It is a type or a category of things. It is similar to a structure with the difference that it can also have functions besides data items. A structure, we have seen, can have only data variables but a class can have data members as well as function members.

➔ **2. Object**

It is an instance or example of a class. You can imagine it to be similar to a variable of class like we have a variable of a structure.

➔ **3. Data abstraction**

Data abstraction is like defining or abstracting the object according to the required parameters. For example; if there is a class for circle we need to just define the radius of the object of this class. We need not bother about anything else of that object.

➔ **4. Data encapsulation**

The data of an object is hidden from other objects. This is called as encapsulation. Encapsulation is achieved by putting data and functions associated with it into a class.

➔ **5. Inheritance**

The mechanism of deriving the properties of one class into another class is known as inheritance. We will see in detail about this concept in a special chapter dedicated on Inheritance.

➔ **6. Polymorphism**

Poly refers to multiple and morph refers to different forms. Hence, polymorphism means multiple forms of the same thing. This topic will also be covered in detail in the later chapters

Syllabus Topic : Procedure Oriented Programming (POP) Verses Object Oriented Programming (OOP), C verses C++

1.6.1 Procedure Oriented Versus Object Oriented Programming Language or 'C' Language versus 'C++' Language

➔ **(S-14, W-14, S-15, S-16, S-17, W-17)**

Q. 1.6.3 Differentiate between POP and OOP (any four points) (Ref.sec.1.6.1)
(S-14, W-14, S-15, S-16, S-17, W-17, 4 Marks)

Sr. No	Procedure Oriented Programming (POP) like 'C'	Object Oriented Programming (OOP) like 'C++'
1.	In this case emphasis is given on method or procedure.	In this case emphasis is on data or objects rather than procedure.
2.	Large programs are divided into smaller parts using functions.	Programs have classes and objects. Only the functions in a class can access the data in a given object.
3.	Data can be accessed by any function, and hence there is a scope of unexpected change of data	Data can be accessed only by functions of that class and hence it cannot be changed unexpectedly
4.	Global data is allowed. Hence data is not hidden	No global data. Data is encapsulated
5.	If changes are done in some data, the corresponding changes are to be implemented in all the functions using the data.	Adding of new data or changes in the data requires to change only the functions of the class
6.	It follows top-down approach in program design.	It follows bottom-up approach in program design.
7.	It cannot model the real world problems precisely.	It is used to model the real world problems precisely

Syllabus Topic : Applications of OOP

1.6.2 Applications of Object Oriented Programming Language

➔ **(S-14 , S-15, S-16, W-16)**

Q. 1.6.4 Enlist the applications of OOP. (Ref. Sec.1.6.2)
(S-14, S-15, S-16, W-16, 4 Marks)

OOP finds wide applications in the IT industry. OOP models the real world problems quite preciecely and hence it is more used today than the procedure oriented programming languages. The areas for application of OOP include :

- Real-time systems
- Simulation and modelling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

1.7 Features of C Programming Language

- C programming language as already discussed has one major feature of being a structured programming language and hence it is very easy for the programmer.
- It is a procedure oriented language and hence has all features discussed in section 1.5.
- C programming language has become popular because of its major features like reliable, simple and easy to use.

1.8 C++ as a Superset of C Programming Language

- C++ was earlier called as C with classes, since it is a superset of C with addition of classes.
- C++ supports all features of C with few exceptions that will be discussed in the further chapters.
- C++ is an object oriented programming language with most of the programming rules same as that of C.
- The object oriented features of C++ will be deal within the chapters 5 and 6.

Syllabus Topic : Tokens

1.9 Tokens of C / C++

Q. 1.9.1 Explain the tokens of C/C++ (Ref. Sec.1.9) **(5 Marks)**

- Before going for programming, let us see a very important thing required for programming i.e. the tokens of a programming language.
- The C / C++ programs are made up of different things termed as tokens.
- The different tokens of C/C++ are its
  - character set
  - keywords
  - identifiers
  - constants and variables
  - data types
  - operators
- We will see these tokens in the subsequent sub-sections.

1.9.1 Character Set of C / C++

- The character set of any programming language indicates the different characters the program can contain. Character set includes all the alphabets, digits and special symbols supported by the processor.
- When we will be writing C / C++ programs, all these will be found in our program. Table 1.9.1 gives a list of C/C++ character set.

Table 1.9.1 : Character Set of C /C++

Sr. No.	Characters	List included
1.	Alphabets (Upper case and lower case)	A, B, C .....Z a, b, c, ...z
2.	Digits (numbers)	0,1,2,....9
3.	Special symbols (all those seen on a keyboard, nothing besides that)	< > { } ( ) [ ] , . ; : ! ? ' " / + * = % & # @ | \ ~ ` $ ^ _ - (The names used for these symbols are given in the table 1.9.2)
4.	Other special characters	Blank Space, Tab, Carriage Return (Enter Key)

Table 1.9.2 : Special Symbols and their names

Symbol	Name of Symbol	Symbol	Name of Symbol
,	Comma	&	Ampersand
.	Period or dot	*	Asterisk
;	Semicolon	-	Minus Sign
:	Colon	+	Plus Sign
?	Question Mark	<	Opening Angle (Less than sign)
!	Exclamation Mark	>	Closing Angle (Greater than sign)
|	Pipe	(	Left Parenthesis
/	Forward Slash	)	Right Parenthesis
\	Back Slash	[	Left Square Bracket
'	Single Quotes	]	Right Square Bracket
"	Double Quotes	{	Left Brace
~	Tilde	}	Right Brace
_	Underscore	#	Hash
^	Caret	$	Dollar
%	Percentage		

- Each of these special symbols may have some significance which will be discussed in the forthcoming chapters.

Syllabus Topic : Keywords

1.9.2 Keywords

Q. 1.9.2 What are keywords? List any 5 keywords are used in C/C++. (Ref. Sec.1.9.2) **(4 Marks)**

- These are some special words that have a predefined meaning for the C / C++ compiler. Hence, these words cannot be used as identifiers (identifiers are discussed in section 1.9.3).
- These are a set of words which are reserved for the certain operations and hence are also sometimes referred as reserved words.
- All keywords are in lower case.

– The keywords used in C / C++ are as given below :

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

– These keywords will be used in different places in programming. Their significance and use will be studied wherever required in the forthcoming chapters.

– These keywords include all those keywords also as declared by ANSI (American National Standards Institute) C. This institute has published the standards to be used in C programming language.

1.9.3 Identifiers

– Identifiers are names given to different user defined things like variables, constants, functions, classes, objects, structures, unions, etc. While making these identifiers we need to follow some rules. These rules are stated below :

1. The identifier can consist of alphabets, digits and a special symbol i.e. '_' (underscore).
2. An identifier cannot start with a digit. It can start either with an alphabet or underscore.
3. It cannot contain any special symbol except underscore. Blank spaces are also not allowed.
4. It cannot be a keyword.
5. It is case sensitive i.e. an alphabet capital in one identifier with same name in another identifier with that alphabet small case will be considered different (for more details see examples in this section).
6. Earlier; there was a limit of the length of the identifier to be 32 characters, but now this limit is removed. Hence, an identifier can be as long as required and minimum of one character.

Example 1.9.1 :

A list of valid and invalid identifiers is given below with reasons wherever required.

1. simple_interest: Valid
2. char : Invalid, because it is a keyword
3. 3friends: Invalid, because starts with a digit
4. _3friends: Valid
5. Simple interest: Invalid, because blank spaces are not allowed
6. #3friends: Invalid, because no special symbol except underscore is allowed.
7. void: Invalid, because keyword not allowed.
8. Void: Valid, case sensitive.

Syllabus Topic : Basic Data Types

1.9.4 Data Types

Q. 1.9.3 List data types in C/C++ (Ref. Sec.1.9.4) **(4 Marks)**

– The data type decides the type of data and the memory locations required for storing that type of data in the memory.

– The data types of C / C++ can be divided into three types: Primitive, Derived and User defined data types.

– The different primitive types of data that can be used in C / C++ are integer, character, fraction type numbers, etc.

– Table 1.9.3 shows the different primitive data types and the memory space required for storing them.

Table 1.9.3 : Data types

Sr. No.	Data type	Type of data to be stored	Range	Space required in memory (bytes)
1.	char	1 character in ASCII form	– 127 to 128	1
2.	signed char	1 character in ASCII form	– 127 to 128	1
3.	unsigned char	1 character in ASCII form	0 to 255	1
4.	int	Integer nos.	– 32768 to 32767	2
5.	signed int	Integer nos.	– 32768 to 32767	2
6.	unsigned int	Integer nos.	0 to 65535	2
7.	short int	Integer nos.	– 32768 to 32767	2
8.	long int	Integer nos.	– 2147483648 to 2147483647	4
9.	signed short int	Integer nos.	– 32768 to 32767	2
10.	signed long int	Integer nos.	– 2147483648 to 2147483647	4
11.	unsigned short int	Integer nos.	0 to 65535	2
12.	unsigned long int	Integer nos.	0 to 4294967296	4
13.	float	Fraction nos.	$3.4e^{-38}$ to $3.4e^{38}$	4
14.	double	Fraction nos.	$1.7e^{-308}$ to $1.7e^{308}$	8
15.	long double	Fraction nos.	$3.4e^{-4932}$ to $1.1e^{4932}$	10

Example 1.9.2 :

Some examples of data to be stored are listed below with the data types that will be best suited for them

1. Age in years: int type data must be used, as years is an integer value.
2. Rate of interest: float data must be used, as rate of interest is a fraction type value.
3. Alphabet : char type of data is to be used, as a character is to be stored.
4. Principal amount: float type data must be used, as principal is in rupee and paisa.
5. Runs made by a batsman: int type data must be used, as no. of runs will always be integer.
6. Factorial of a number: int type data must be used, as the factorial is always an integer.
7. Radius of a circle: float type data must be used, as radius will mostly be a fraction number.
8. Area of a circle: float type data must be used, as area is radius^2 X 3.14, which has to be fraction number.
9. User input as 'Y' or 'N': char type data must be used, as there is only a character to be taken from the user as yes or no. Boolean type of data may also be used for this.

– Note : the arithmetic operations can be done even on char type of data. This is because the processor stores ASCII (American Standard Code for Information Interchange) to store the characters.

– The ASCII value for capital 'A' is 65, 'B' is 66, 'C' is 67 and so on. While the ASCII values for small alphabets are 97 for 'a', 98 for 'b', 99 for 'c' and so on.

– The user defined data types are structure, union, class and enumeration. The derived data types are array, function, pointer and reference.

– The derived and user defined data types will be studied in the subsequent chapters wherever it is needed.

Syllabus Topic : Constants and Variables

1.9.5 Constants and Variables

Q. 1.9.4 Define terms constants and variables (Ref. Sec. 1.9.5) **(2 Marks)**

- Constants are values given to the identifiers that do not change their values throughout the execution of the program.
- Constants can be defined either by writing the keyword const before the data type or by using #define.
- Constants are used to declare the values that remain constant, for e.g. value of pi.
- The use of constants in program will be discussed later wherever required.
- Variables are values given to identifiers that can change their values during the execution of the program.

Example 1.9.3 :

A variable can be defined with the data type as shown in the examples below :

1. For declaring age as an integer type data the syntax (grammar or method of writing) is: int age;
2. For declaring rate of interest as float data type the syntax is: float rate;
3. For declaring a character data type element the syntax is: char x;

1.9.6 Escape Sequences

- Escape sequence is a character followed by a backslash (\).
- They are used especially to perform some special operations like going to new line, providing a horizontal tab, vertical tab etc.
- The following is a list of escape sequences.
  1. \n Newline
  2. \t Horizontal Tab
  3. \v Vertical Tab
  4. \b Backspace
  5. \r Carriage Return
  6. \f Form feed
  7. \a Audible Alert (bell)
  8. \\ Backslash
  9. \? Question mark
  10. \' Single quote
  11. \" Double quote

1.9.7 Operators

- Operators are different special symbols used for indicating an operation to be performed.
- The operators will be dealt in detail in section 1.12.

Syllabus Topic : Structure of C++ Program , Simple C++ Program

1.10 Structure of C++ program

➔ **(W-15, S-16,S-17, W-17)**

Q. 1.10.1 With suitable diagram describes structure of C++ program. (Ref. Sec. 1.10 and Prog.1.10.1) **(W-15, 4 Marks)**

Q. 1.10.2 Write structure of C++ program. (Ref. Sec. 1.10 and Prog. 1.10.1) **(S-16, 4 Marks)**

Q. 1.10.3 Explain the structure of C++ program with suitable example. (Ref. Sec. 1.10 and Prog.1.10.1) **(S-17, 4 Marks)**

Q. 1.10.4 Explain the structure of C++ program. (Ref. Sec. 1.10 and Prog. 1.10.1) **(W-17, 4 Marks)**

- To begin with a very simple program in C, to display the string "Hello Friend" on the standard output device i.e. the monitor of a computer.
- We will then understand the meaning of each word written in the program and also how to execute this program.

1.10.1 Simple C++ Program

➢ **Program 1.10.1 : Write is a simple program to display a statement "Hello Friend".** **W-15, S-16, S-17, W-17**

```
//This is a simple program to display a statement
// "Hello Friend"
#include<iostream.h>
#include<conio.h>
```

```
void main ()
{
    cout<<"Hello Friend";
    getch();
}
```

Output :

```
Hello Friend
```

Explanation of the above program :

- The first line that starts with "//", indicates that this is a comment line. The comments are written in a program for future reference of a programmer.
- The comments are neglected by the compiler (compiler is a software tool that converts the C / C++ program to the machine language).
- In comments you can write anything, it may be helpful to understand your program by somebody else or by yourself after you see the program after a long period.
- Any statement that begins with the '#' sign is meant for the pre-processor (pre-processor is a tool that works on the program before the compiler, discussed in section 1.11).
- The next statement i.e. "#include<iostream.h>, indicates the preprocessor to include a header file name iostream.h with the program written by the user.
- Our programs will definitely have some data to be taken from the user and to display some data on the monitor. The 'cin' and 'cout' are the statements to handle the input and output operations respectively.
- The meaning of these words and the process required to be performed for taking the input and displaying the output is mentioned in the header file namely iostream.h.
- In this program we have used the statement cout, hence we have included this header file in our program.
- Any program of C / C++ should have a main function (function is a set of statements to perform a given task). In the statement void main (), the word "void" indicates that this function is not going to return any data to its caller function. The word "main" is identifier which identifies the function.
- The brackets at the end of this statement, is a place where parameters can be passed to this function. We will see more about the return type of a function and its parameters in chapter 7.
- The curly braces ({ and }) indicate the start and end of the function. The "{" indicates beginning of function, while the "}" indicates end of the function.
- "cout" statement is used to display a data on the standard output device i.e. the monitor. The "<<" associated with the cout statement is called as the insertion operator.
- The statements given on the right of the insertion operator are displayed on the monitor.
- If the data written on the right of the insertion operator is in double quotes, then the data will be displayed as it is. If it is not in double quotes then it must be name of a variable whose value is to be displayed.
- The getch() function is an inbuilt function used to accept an input from the user. This makes the computer to wait for the user and hence, makes to output screen available for the user until he presses a key.

1.10.2 Dynamic Initialization of Variables

Q. 1.10.5 What do you mean by dynamic initialization of variable ? How is it achieved ? (Ref. Sec. 1.10.2) **(5 Marks)**

- The initialization of variables is said to be dynamic in C++ i.e. the value of a variable can be initialized when it is declared.

  For e.g. int i=5;

- A program example for the same can be as given below :

```
//This is a program to multiply two numbers and display the
//result
#include<iostream.h>
#include<conio.h>
void main ()
{
    int i=5,j=4,k;
   k=i*j;
   cout<<"Product is:"<<j;
}
```

Output :

```
Product is: 20
```

Syllabus Topic : User defined Data Types

1.10.3 User Defined Data type

There are many user defined data types like pointers, structures, etc. We will see these in later chapters. One of these user defined data type is Enumerated data type

1.10.3.1 Enumerated Data type

Q. 1.10.6 Write a short note on: Enumerated Data type (Ref. Sec. 1.10.3.1) **(5 Marks)**

- These are used to contain values that are not limited to the values that are possible for fundamental data types.
- For example if there could be a data type months, its values can be January, February, March, April, May, June, July, August, September, October, November or December. Similarly, if there could be a data type called as colors; the values could be red, blue, green, cyan, yellow, pink, while, black etc.
- Thus, such special things can be created using the "enum".
- The syntax of creating a enum is as given below:
- enum enum_name { <value1>, <value2>...};
- For example an enum of the week days can be as shown below:

  enum WeekDays {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
- The syntax of declaring a variable of enum is as shown below:

  enum_name enum_variable;
- For example if we want to create a variable of above declared enum of weekdays,

  WeekDays today;
- To asssign a value to a variable of enum, the syntax is as shown below:

  enum_variable = value;
- For example if the value "Sunday" is to be assigned to the above created enum_variable we need to write,

  today = Sunday;
- A program example for the same can be as given below :

```
#include <iostream.h>
void main()
{
  clrscer();
  enum WeekDays { Sunday, Monday, Tuesday,
Wednesday, Thursday, Friday, Saturday };
  WeekDays today;
  today = Sunday;
  if (today == Sunday || today == Saturday)
  cout << "Holiday";
  else
  cout << "Go to work.";
 getch();
}
```

Output :

```
Holiday
```

1.11 Integrated Development Environment (IDE)

- IDE is a toolkit to write and execute a program. The most widely used IDE with the C / C++ programs is the Turbo C IDE.
- This IDE contains of five tools namely : **Editor, Preprocessor, Compiler, Linker and Loader.**
- **Editor :** It is used to type and edit the program.
- **Preprocessor :** It is responsible for performing certain operations before the compiling, like including header files, declaring constants, etc. It always begins with the '#' sign.

  For e.g. #include<iostream.h>

  #define pi 3.14;
- **Compiler :** It is used to convert the C / C++ program to machine language. As the processor cannot understand the language of C / C++, we need to convert it into a machine understandable language i.e. machine language.
- **Linker :** It is responsible for linking (connecting) the different functions in a program. We may write multiple functions in the program or conditional/loop operations in our program. These kinds of operations are to be handled by the linker.
- **Loader :** It loads the binary file from the memory and gets is executed by the processor. The binary file that the loader loads is the machine language program understood by the processor.
- The above program 1.10.1 in section 1.10 can be executed using the following steps :

  1. Double click TC.exe to get the editor window.

2. Type the program in the editor window.
3. Save the program as .cpp for e.g. prog101.cpp.
4. Compile the program by pressing Alt + F9 i.e. press the Alt and F9 key together.
5. Execute the program by pressing Ctrl + F9
6. See the output on the screen and press enter to come back to editor window.

Syllabus Topic : Operators

1.12 Operators

➔ **(W-17)**

Q. 1.12.1 Explain different operator used in C++. (Ref. Sec. 1.12) **(W-17, 4 Marks)**

- The operators are one of the tokens of C/C++. There are different types of operators classified based on their functions.
- Many of these operators although listed here, but will be understood in the subsequent chapters as and when we use them. So you may just go through the operators and understand them later while programming.
- The data on which the operation is performed are called as operands.
- If an operator requires one operand, it is called as Unary operator. If an operator requires two operands, then it is called as Binary operator and if it requires three operands, it is called as Ternary operator.

1.12.1 Unary Operators

➔ **(W-15, S-16, S-17, W-17)**

Q. 1.12.2 How address of (&) operator is used in pointers, explain with example. (Ref. Sec. 1.12.1(4)) **(W-15, 2 Marks)**

Q. 1.12.3 Explain pointer operator and address operator with example. (Ref. Sec. 1.12.1) **(S-16, 2 Marks)**

Q. 1.12.4 Give significance of '&' and '*' operators. (Ref. Sec. 1.12.1(4 & 5)) **(S-17, 2 Marks)**

Q. 1.12.5 What is use of scope resolution operator ? (Ref. Sec. 1.12.1(6)) **(W-17, 2 Marks)**

Q. 1.12.6 State scope resolution operator and memory management operator in C++. (Ref. Sec. 1.12.1 and 1.12.6) **(S-16, S-17, 4 Marks)**

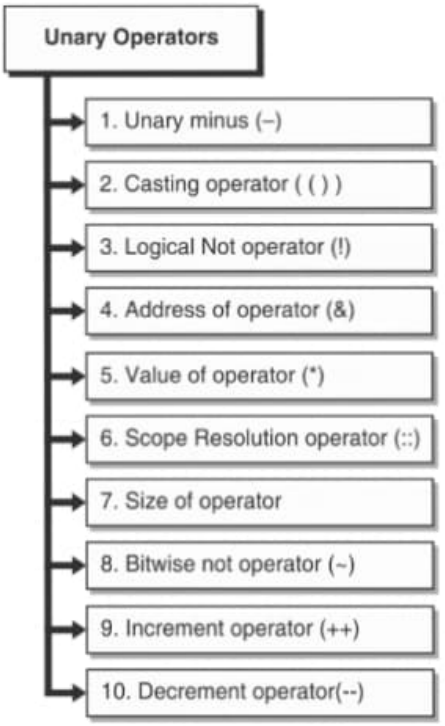


Fig. C.1.1 : Unary operators

➔ **1. Unary minus (–) :**

- The symbol shown in the bracket is used as unary minus operator.
- It returns the negative value of the variable to which it is preceded.
- For e.g. if x = 3 and y = 6 then

  y = – x;

- will make the value of y as – 3.

➔ **2. Casting Operator (( )) :**

1. It is many a times required to convert one data type to another.
   - The casting operator is used for type conversion i.e. it can be used to convert a data of one type to another data type.
   - For e.g. if we have int x=3;

     float y=5.6;

then the statement, x=(int) y;

will result in the value of x as 5.

- The above method of casting is used in C. In C++, the casting is to be done in a slightly different manner i.e. we need to mention x = int (y).
- We will see the use of this operator in the programming

➜ 3. Logical Not operator (!)

- The symbol shown in the brackets i.e. the exclamation mark is used as a logical not operator. It is used to check certain conditions in a condition statement. It performs the logical not of the result obtained from the expression on its right.
- For e.g. if x=1, then y=!x;

  will result with y having the value 0.
- We will see some more logical not operator based expressions and program followed by this section.

➜ 4. Address of operator (&) :

- This operator returns the address of the variable associated with it. It will be studied more deeply in the chapter on Pointers.
- For e.g. if we write y=&x;

  then the memory address allocated to the variable x will be copied into the variable y. For this the variable y must be a pointer variable.

➜ 5. Indirection operator or value of operator (*) :

- This operator returns the value of the data at the given address. This operator will also be studied in more details in the chapter on pointers.
- For e.g. z = *y;

  will give the value of the data stored at the address given by y.

➜ 6. Scope Resolution operator (::) :

- It resolves the scope of a variable to be inside the function or outside the function. This operator will also be studied in a later chapter.
- For e.g. if an external variable x is to be accessed then the same can be accessed using the scope resolution operator by the statement, y=::x;

➜ 7. Size of operator :

- This operator is used to know the size of a variable as required to store its value in memory. It can also be used to find the size of a data type.
- As seen in the section 1.9.4, the space required to store different data type is different ranging from 1 byte to 10 bytes.
- For e.g. the statement sizeof(int);

  will return the value 2, as int requires 2 bytes.
- Another e.g.:

```
int x,y;
char a;
x=sizeof(a);
y=sizeof(float);
```

  then x will have 1, and y will have 4

➜ 8. Bitwise not operator (~)

- This operator is used to perform bitwise NOT operation as discussed in the section 1.3. We have also seen some examples of NOT operations.
- The bitwise NOT operator can be used to perform binary NOT operation. This operation can be performed by using the operator given in brackets i.e. ~.
- For e.g. If x=3,

  then y=~x;

  will result in y having a value of -4

➜ 9. Increment Operator (++)

- It returns the value of the variable added with one and stores the result in the variable itself.
- For e.g. if x=5,

  then x++;

  will make the value of x equal to 6.
- This "x++;" (also called as post increment operator) can also be written as "++x;" (also called as pre increment operator).
- It can also be used to store the result in another variable. But in this case the post increment and pre increment statements will have different behavior as explained below with examples.
- In post increment case, for e.g. if x=5,

  then y=x++;

  will make the value of y equal to 5 and x equal to 6.

As the name says post increment, it first gives the previous value and then increments.

- In pre increment case, for e.g. if x=5,

  then y=++x;

  will make the value of y equal to 6 and x equal to 6.

  As the name says pre increment, it first increments and then gives the incremented value.

- More such examples will be seen with programs.

➜ **10. Decrement Operator(--)**

- It returns the value of the variable subtracted with one and stores the result in the variable itself.
- For e.g. if x=5,

  then x--;

  will make the value of x equal to 4.

- This "x--;" (also called as post decrement operator) can also be written as "--x;" (also called as pre decrement operator).
- It can also be used to store the result in another variable. But in this case the post decrement and pre decrement statements will have different behavior as explained below with examples.
- In post decrement case, for e.g. if x=5,

  then y=x--;

  will make the value of y equal to 5 and x equal to 4.

- As the name says post decrement, it first gives the previous value and then decrements.
- In pre decrement case, for e.g. if x=5,

  then y=--x;

  will make the value of y equal to 4 and x equal to 4.

  As the name says pre decrement, it first decrements and then gives the decremented value.

- More such examples will be seen with programs.

1.12.2 Binary Operators

Q. 1.12.7 Explain bitwise operators. (Ref. Sec. 1.12.2(II)) **(5 Marks)**

Q. 1.12.8 Explain logical operators (Ref. Sec. 1.12.2(III)) **(5 Marks)**

- Operators that require two operands are called as binary operators.
- These operators are further classified into various types namely the Arithmetic operators, Logical operators, Bitwise operators and Relational operators.
- We will see these operators one by one in this section.

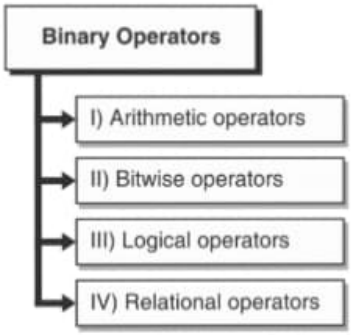


Fig. C.1.2 : Binary operators

➜ **(I) Arithmetic operators**

- This set includes the basic arithmetic operators to perform basic arithmetic operations like addition, subtraction, multiplication and division. There are five operators in this set. They are:

  1. * to find the product
  2. / to find the quotient after division
  3. % to find the remainder after division
  4. + to find the sum
  5. – to find the difference

- One important thing to be noted here is that the '/ ' operator returns only the quotient, while the '%' operator (also called as MOD operator) returns the remainder after division.
- The sign of the remainder is always the same as that of the dividend.
- MOD operator is possible only for int or char type of data. It doesn't work on float and double type of data.
- For example of each of these operators

  1. 2 * 2 = 4;
  2. For int type of data, 5 / 3 =1;

     For float type of data, 5 / 3 = 1.67
  3. 5 % 3 =2;
  4. 2 + 2 = 4;
  5. 3 – 2 = 1;

➜ (II) Bitwise operators

- These operators work bitwise on a data.
- They perform different operations on bits of a data like AND, OR, EXOR and NOT. These operations work in the manner as explained in section 1.3.
- The operators are listed below :
  1. ~ to perform bitwise NOT operation.
  2. & to perform bitwise AND operation.
  3. | to perform bitwise OR operation.
  4. ^ to perform bitwise EXOR operation.
  5. << to perform bitwise left shift operation.
  6. >> to perform bitwise right shift operation.
- These operators are use to perform bitwise binary operations on the data.
- As we have addition, subtraction operations in decimal data for performing arithmetic operations; similarly AND, OR, NOT are basic bitwise operations on the binary data.
- The result of these operations can be understood from the following examples and referring the section 1.3 of binary operations :

1. 5 & 3 = 1

$$\begin{array}{rcl} (5)_{10} & = & (0\ \ 1\ \ 0\ \ 1)_2 \\ (3)_{10} & = & (0\ \ 0\ \ 1\ \ 1)_2 \\ \hline & & (0\ \ 0\ \ 0\ \ 1)_2 = (1)_{10} \end{array}$$

2. 12 | 9 = 13

$$\begin{array}{rcl} (12)_{10} & = & (1\ \ 1\ \ 0\ \ 0)_2 \\ (9)_{10} & = & (1\ \ 0\ \ 0\ \ 1)_2 \\ \hline & & (1\ \ 1\ \ 0\ \ 1)_2 = (13)_{10} \end{array}$$

3. 8 ^ 10 = 2

$$\begin{array}{rcl} (8)_{10} & = & (1\ \ 0\ \ 0\ \ 0)_2 \\ (10)_{10} & = & (1\ \ 0\ \ 1\ \ 0)_2 \\ \hline & & (0\ \ 0\ \ 1\ \ 0)_2 = (2)_{10} \end{array}$$

4. ~7 = – 8

$$\begin{array}{rcl} (7)_{10} & = & (0\ \ 1\ \ 1\ \ 1)_2 \\ \hline & & (1\ \ 0\ \ 0\ \ 0)_2 = (-8)_{10} \end{array}$$

(According to C /C++/Java, wherein it will take more no. of bits and hence -8 will be the result as also discussed in section 1.3). Hence; in general the ~x is always equal to – (x + 1).

5. 10 << 2 = 40

Assuming the data to be char i.e. 8 bit data

$$(10)_{10} = (0\,0\,0\,0\,1\,0\,1\,0)_2$$

After shifting left once $(0\,0\,0\,1\,0\,1\,0\,0)_2$

After shifting left for the second time $(0\,0\,1\,0\,1\,0\,0\,0)_2 = (40)_{10}$

> **Note :** When shifting to left, each of the bit is shifted left. The first bit is lost and the last bit is inserted as 0.

6. 13 >> 3 = 1

Assuming the data to be char i.e. 8 bit data

$$(13)_{10} = (0\,0\,0\,0\,1\,0\,1\,1)_2$$

After shifting left once $(0\,0\,0\,0\,0\,1\,0\,1)_2$

After shifting left for the second time $(0\,0\,0\,0\,0\,0\,1\,0)_2$

After shifting for the third time $(0\,0\,0\,0\,0\,0\,0\,1)_2 = (1)_{10}$

> **Note :** When shifting to right, each of the bit is shifted right. The first bit is inserted as 0 and the last bit is lost.

➜ (III) Logical operators

- Logical operators follow the same truth table as for the bitwise operators as seen in section 1.3; but they are used to check conditions instead of performing operations on a data.
- The logical operators are AND and OR. The symbols used for these operators in C are && and || respectively.
- For example a statement

  y > 5 && y < 10;

  will result in "true" i.e. 1 if the value of y is greater than 5 AND less than 10, else it will result in false i.e. 0.
- Another example a statement

  y > 5 || y==2;

  will result in "true" i.e. 1 if the value of y is greater than 5 OR equal to 2, else it will result in false i.e. 0.
- Logical operators will be understood in more details with the expressions and program examples followed by this section.

➜ (IV) Relational operators

- The relational operators are used to test the relation between two variables or a variable and constant.
- The operators like '<' (less than) and '>' (greater than) used in the above examples are relational operators. We have seen the example also of the "==" (equality operator) in the above logical operators.

➜ (II) Bitwise operators

- These operators work bitwise on a data.
- They perform different operations on bits of a data like AND, OR, EXOR and NOT. These operations work in the manner as explained in section 1.3.
- The operators are listed below :
  1. ~ to perform bitwise NOT operation.
  2. & to perform bitwise AND operation.
  3. |to perform bitwise OR operation.
  4. ^ to perform bitwise EXOR operation.
  5. << to perform bitwise left shift operation.
  6. >> to perform bitwise right shift operation.
- These operators are use to perform bitwise binary operations on the data.
- As we have addition, subtraction operations in decimal data for performing arithmetic operations; similarly AND, OR, NOT are basic bitwise operations on the binary data.
- The result of these operations can be understood from the following examples and referring the section 1.3 of binary operations :

1. 5 & 3 = 1

$$\begin{array}{rcl} (5)_{10} & = & (0\ \ 1\ \ 0\ \ 1)_2 \\ (3)_{10} & = & (0\ \ 0\ \ 1\ \ 1)_2 \\ \hline & & (0\ \ 0\ \ 0\ \ 1)_2 = (1)_{10} \end{array}$$

2. 12 | 9 = 13

$$\begin{array}{rcl} (12)_{10} & = & (1\ \ 1\ \ 0\ \ 0)_2 \\ (9)_{10} & = & (1\ \ 0\ \ 0\ \ 1)_2 \\ \hline & & (1\ \ 1\ \ 0\ \ 1)_2 = (13)_{10} \end{array}$$

3. 8 ^ 10 = 2

$$\begin{array}{rcl} (8)_{10} & = & (1\ \ 0\ \ 0\ \ 0)_2 \\ (10)_{10} & = & (1\ \ 0\ \ 1\ \ 0)_2 \\ \hline & & (0\ \ 0\ \ 1\ \ 0)_2 = (2)_{10} \end{array}$$

4. ~7 = – 8

$$\begin{array}{rcl} (7)_{10} & = & (0\ \ 1\ \ 1\ \ 1)_2 \\ \hline & & (1\ \ 0\ \ 0\ \ 0)_2 = (-8)_{10} \end{array}$$

(According to C /C++/Java, wherein it will take more no. of bits and hence -8 will be the result as also discussed in section 1.3). Hence; in general the ~x is always equal to – (x + 1).

5. 10 << 2 = 40

Assuming the data to be char i.e. 8 bit data

$(10)_{10} = (0\,0\,0\,0\,1\,0\,1\,0)_2$

After shifting left once $(0\,0\,0\,1\,0\,1\,0\,0)_2$

After shifting left for the second time $(0\,0\,1\,0\,1\,0\,0\,0)_2 = (40)_{10}$

> **Note :** When shifting to left, each of the bit is shifted left. The first bit is lost and the last bit is inserted as 0.

6. 13 >> 3 = 1

Assuming the data to be char i.e. 8 bit data

$(13)_{10}$ =	$(0\,0\,0\,0\,1\,0\,1\,1)_2$
After shifting left once	$(0\,0\,0\,0\,0\,1\,0\,1)_2$
After shifting left for the second time	$(0\,0\,0\,0\,0\,0\,1\,0)_2$
After shifting for the third time	$(0\,0\,0\,0\,0\,0\,0\,1)_2 = (1)_{10}$

> **Note :** When shifting to right, each of the bit is shifted right. The first bit is inserted as 0 and the last bit is lost.

➜ (III) Logical operators

- Logical operators follow the same truth table as for the bitwise operators as seen in section 1.3; but they are used to check conditions instead of performing operations on a data.
- The logical operators are AND and OR. The symbols used for these operators in C are && and || respectively.
- For example a statement

  y > 5 && y < 10;

  will result in "true" i.e. 1 if the value of y is greater than 5 AND less than 10, else it will result in false i.e. 0.
- Another example a statement

  y > 5 || y==2;

  will result in "true" i.e. 1 if the value of y is greater than 5 OR equal to 2, else it will result in false i.e. 0.
- Logical operators will be understood in more details with the expressions and program examples followed by this section.

➜ (IV) Relational operators

- The relational operators are used to test the relation between two variables or a variable and constant.
- The operators like '<' (less than) and '>' (greater than) used in the above examples are relational operators. We have seen the example also of the "==" (equality operator) in the above logical operators.

- A list of all the relational operators is given below :
  1. == used to check if the two things are equal.
  2. != used to check if the two things are not equal.
  3. < used to check if the first data is less than the second one.
  4. > used to check if the first data is greater than the second one.
  5. <= used to check if the first data is less than or equal to the second one.
  6. >= used to check if the first data is greater than or equal to the second one.
- The examples of these relational operators are as shown with the logical operators above.

1.12.3 Ternary Operator

- An operator that requires three operands is called as a ternary operator.
- There is only one ternary operator in C / C++. This operator is used to check a condition and accordingly do one of the two things based on the condition being true or false.
- The syntax (way of writing) of this operator is as given below:

  (condition) ? <value if condition is true> : <value if condition is false>;
- Hence; as shown in the syntax, first the condition is to be written in brackets followed by a question mark (?). Then the operation that is to be performed if condition is true and then a colon (:) followed by the operation to be performed if the condition is false.
- For example:

  z = (x>y)?x:y;

  This statement will put the value of x into z if the given condition i.e. x > y is true. Else; the value of y will be put into z. Hence, the value of the greater variable will be put into z.
- A slightly complicated use of this operator is to find the greatest of three numbers as shown in the example

  g=(x>y)?((x>z)?x:z)):((y>z)?y:z);

  This statement will give the largest of x, y and z into the variable g.

1.12.4 Assignment Operators

- These operators are used to assign the value of the expression or variable on the right of the assignment operator to the variable on its left.
- The simple assignment operator is '='. But there are some more assignment operators called as composite assignment operators.

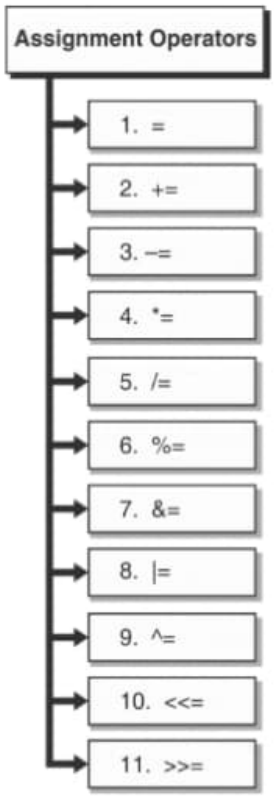


Fig. C.1.3 : Assignment Operators

The different assignment operators are as listed below :

➜ **1. = :** This operator assigns the value of the expression or variable on its right to the variable on its left. For e.g. y=x+2;

➜ **2. +=:** This operator adds the variable on its left and right and the result is put into the variable on its left. For e.g. y+=x; is same as y=y+x;

➜ **3. –= :** This operator subtracts the variable on its right from the variable on its left and the result is put into the variable on its left. For e.g. y-=x; is same as y=y-x;

➜ **4. *= :** This operator multiplies the variable on its left and right and the result is put into the variable on its left. For e.g. y*=x; is same as y=y*x;

➜ **5. /= :** This operator divides the variable on its right from the variable on its left and the result is put into the variable on its left. For e.g. y/=x; is same as y=y/x;

➜ **6. %= :** This operator finds the remainder by dividing the variable on its right from the variable on its left and the result is put into the variable on its left. For e.g. y%=x; is same as y=y%x;

➜ **7. &= :** This operator ANDs the variables on its left and right and the result is put into the variable on its left. For e.g. y&=x; is same as y=y&x;

➜ **8. |= :** This operator ORs the variables on its left and right and the result is put into the variable on its left. For e.g. y|=x; is same as y=y|x;

➜ **9. ^= :** This operator EXORs the variables on its left and right and the result is put into the variable on its left. For e.g. y^=x; is same as y=y^x;

➜ **10. <<= :** This operator shifts in left direction the variable on its left for the number of times indicated by the variable or value on right and the result is put into the variable on its left. For e.g. y<<=x; is same as y=y<<x;

➜ **11. >>= :** This operator shifts in right direction the variable on its left for the number of times indicated by the variable or value on right and the result is put into the variable on its left. For e.g. y>>=x; is same as y=y>>x;

1.12.5 Selection Operators

– These operators are used to select certain element of a set of elements. Here, we will make a list of these operators and see the detailed use of these operators when we study the corresponding topic where these operators are required.

– The different operators in this set are listed below:

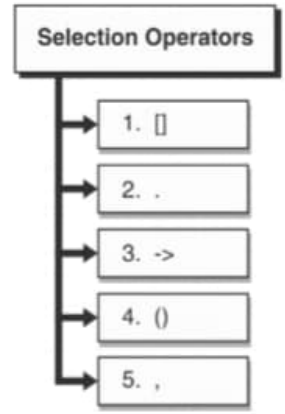


Fig. C.1.4 : Selection operators

➜ **1. [] :** This operator is used to select an element of Array. We will understand more about this in the chapter on Array.

➜ **2. . :** This is called as period operator and is used to select an element of a structure, union or class. This will be studied in detail in the chapter on structure and union.

➜ **3. -> :** This is used to select an element of a structure, union or class pointed by a pointer. This will be studied in detail in the chapter on structure and union.

➜ **4. () :** This is called as a function call operator and is used to call or select a function. This will be studied in detail in the chapter on function.

➜ **5. , :** The comma (,) operator is used to separate the different values etc.

Syllabus Topic : Memory Management Operator

1.12.6 Memory Management Operators

➜ **(W-15,S-16,S-17)**

Q. 1.12.9 List two memory management operators available in C++ and state its use in one line. (Ref. Sec. 1.12.6) **(W-15, 2 Marks)**

Q. 1.12.10 State scope resolution operator and memory management operator in C++. (Ref. Sec.1.12.1 and 1.12.6) **(S-16, S-17, 4 Marks)**

- The **new** operator is used in C++ for allocating memory dynamically. In the C programming language we used malloc() function for the same. The differences between the new operator and the malloc() function are listed in the Table 1.12.1 :

Table 1.12.1

Sr. No.	new operator	malloc() function
1.	It is an operator	It is function
2.	It returns the exact data type	It returns the void type data which is to be type casted to the required data type.
3.	It calls the constructor and can also be used to define the primitive data types.	It does not call the constructor

- The **delete** operator is used in C++ for deallocating memory dynamically allocated by new operator.
- The programs based on the new and delete operators will be seen in chapters 5 and 7.

1.13 Precedence and Associativity of Operators

- The precedence of the operators means the sequence in which the operators will be operated on, in case of multiple operators in a statement i.e. which operator will be executed first and which operator will be executed later.
- The associativity of operators refers to the direction in which the operation will be performed in case of equal precedence operators i.e. if multiple additions are there in a statement then it will be performed from left to right. We will see more about this in the examples followed by this section.
- The precedence and associativity table is as given in Table 1.13.1

Table 1.13.1 : Precedence and Associativity of operators

Precedence	Operators	Operation	Associativity
1.	()	Function call	Left to right
	[]	Array element select	
	->	Structure element select	
	.	Structure element select	
2.	+	Unary plus	Right to left
	-	Unary minus	
	++	Increment operator	
	--	Decrement operator	
	!	Logical NOT	
	~	Bitwise NOT	
	*	Pointer value of operator	
	&	Pointer address of operator	
	sizeof	Size of a variable or data type	
	()	Type cast operator	
3.	*	Multiply	Left to right
	/	Divide and get quotient	
	%	Divide and get remainder	
4.	+	Add	Left to right
	-	Subtract	
5.	<<	Left shift operator	Left to right
	>>	Right shift operator	
6.	<	Less than	Left to right
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
7.	==	Equality	Left to right
	!=	Inequality	
8.	&	Bitwise AND	Left to right
9.	^	Bitwise EXOR	Left to right
10.	|	Bitwise OR	Left to right

Syllabus Topic : Expressions

1.14 Expressions

Let us see some examples of how the processor performs operations when some statements are given to us in the following examples. We will also see some examples wherein an expression is to be written in C / C++, so how are they written.

(I) Determine the value of the following expressions if a=7, b=3 and c=4;

1. a % b

   = 7 % 3

   = 1 (since the remainder after dividing 7 by 3 is 1)

2. a / c

   = 7 / 4

   = 1 (since the quotient after dividing 7 by 4 is 1)

3. a * b / c

   = 7 * 3 / 4

   = 21 / 4 (since; the associativity of arithmetic operations is left to right first operation is *)

   = 5 (since; quotient is 5)

4. a * (c % b)

   = 7 * (4 % 3)

   = 7 * 1 (since; bracket opening is to be done first, and remainder of 4 divided by 3 is 1)

   = 7

5. 2 * b+ 3* (a-c)

   = 2 * 3 + 3 * (7 – 4)

   = 6 + 3 * (3)

   = 6 + 9 (since precedence of multiply is more than add, first multiplication is done)

   = 15

6. a * c % b

   = 7 * 4 % 3

   = 28 % 3

   = 1

7. a + b – c

   = 7 + 3 – 4

   = 6

(II) Suppose the following statements are written:

int i=9, j=6;

float x=0.5, y=0.1;

char a='a', b='b';

Find the values of the following expressions

1. (3 * i – 2 * j) % (2 * a – b)

   = (27 – 12) % (2 * 97 – 98) ....(since; the ASCII values stored in char type variables is 97 for a & 98 for b)

   = (15) % (96)

   = 15

2. 2 * (j/5) + (4 * (j-3)) % (i + j – 2)

   = 2 * (6 / 5) + (4 * (6 – 3)) % (9 + 6 -2)

   = 2 * (1) + (4 * (3)) % (13)

   = 2 + (12) % 13

   = 14

3. (x > y) && (i > 0) && (j > 5)

   = 1 && 1 && 1 .....(if condition is true its result is 1 else it is 0)

   = 1

4. ((x < y) && (i > 0)) || (j > 3)

   = (0 && 1) || (1)

   = (0) || (1)

   = 1

5. a==99

   = 0 .....(since; condition given is false, the value of a is 97 as discussed in section 1.13)

6. ++i

   = 10 .....(since; it is pre increment it will return the value after incrementing i.e. 10)

7. i++

   = 9 .....(since; this is post increment it will return the value before incrementing i.e. 9)

8. !(b==98)

   = ! (1).....(since; the condition given is true i.e. b=98)

   = 0.....(since; after performing NOT operation the value becomes 0)

(III) Write the C assignment expressions for the following :

1. A is greater than B and greater than C

   (A > B) && (A > C)

2. A is either less than B or greater than C

   (A < B) || (A > C)

3. Side is equal to square root of $(a^2 + b^2 + 2ab)$

   Side = pow ((a * a + b * b + 2 * a * b), 0.5)

Note : 1. All the operations including that of multiplication is to be specified; no operation is implied. For e.g. we cannot write x = ab, instead we need to write x= a* b.

2. Only the round brackets i.e. () are allowed in expressions. Other brackets have different meaning in the C / C++ programming languages.

3. pow(x,y) is an available function that can be used to perform x^y operation. x and y are the variables to be given in the brackets as shown in above example.

4. $$a = \frac{xy + z\left(\frac{x}{y}\right) + zy}{x + y + z}$$

$$a = x(* y + z * (x / y) + z * y) / (x + y + z);$$

5. $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x_1 = (- b + pow ((b * b - 4 * a * c) , 0.5))/(2 * a);$$

$$x_2 = (- b - pow ((b * b - 4 * a * c) , 0.5))/(2 * a);$$

6. $$z = a + \frac{1}{1 + \frac{k + a}{2}}$$

$$z = a + 1 (1 + (k + a) / 2);$$

1.15 Formatted and Unformatted IO Functions

- Inbuilt IO functions are available in C for displaying data on the standard output device i.e. monitor and accepting the input from the standard input device i.e. keyboard.
- These IO functions are classified as formatted and unformatted functions based on the formatting permitted or not. These are discussed in more details in the following sub sections.

1.15.1 Formatted IO Functions

- There are two formatted IO functions in C namely printf() to display a data on monitor and scanf() to accept data from keyboard.

1. printf()

Syntax : printf("Format string", list of variables or expressions)

The format string can contain the following :

(a) Character set (A-Z, a-z, 0-9, all special symbols on the keyboard)

(b) Blank spaces

(c) Escape sequences (as discussed in section 1.9.6)

(d) Field width

The width of a value can be fixed by specifying the field width. Examples of such case are given below.

(e) Format specifiers

Format specifiers specify the format of the variable data whose value is to be displayed. The format specifiers for different types of data is given in the table 1.15.1.

Table 1.15.1 : Format specifiers

Sr. No.	Format specifier	Data type
1.	%d or % i	For integer type of data
2.	%f	For float type of data
3.	%c	For char type of data
4.	%lf	For double type of data
5.	%Lf	For long double type of data

☞ **Examples :**

1. printf("The number of buildings is %d",b);

This statement will print the output with the format specifier replaced with the value of b.

2. printf("The simple interest is %f",si);

This statement will print the output where the format specifier %f will be replaced with the float type value of the variable si.

3. printf("The simple interest is %5.2f",si);

The format specifier in this statement is accompanied with a field width i.e. 5.2 (%5.2f), which indicates five digits before the fraction point and 2 digits after the fraction point. This statement will display the output wherein the value of si will be displayed with five digits before the fraction point and only two digits after the fraction point.

2. scanf()

Syntax : scanf("format String", address of variables);

Here, format string can contain format specifier, field width and assignment suppression character. The assignment suppression character (*) is used to discard one of the user entered value.

The address of the variable is obtained with the help of the address of operator (&).

☞ **Examples :**

1. scanf("%d",&x);

   This statement is used to accept a int type value from user in the variable x

2. scanf("%d %d %f",&x,&y,&z);

   This statement is used to accept two int type data into variables x and y. It also accepts a float type data into the variable z.

1.15.2 Unformatted IO Functions

– The unformatted IO functions do not have any format specifier. They are mostly used to accept and display only one character or a string (string is a set of characters to make a word or sentence).

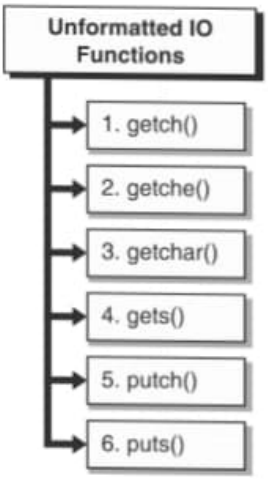


Fig. C.1.5 : Unformatted IO Functions

– The different unformatted IO functions are listed below;

➜ **1. getch() :** This function is used accept one character from the user.

➜ **2. getche() :** This function is used accept one character from the user and echo it (display it on the screen).

➜ **3. getchar() :** This function is used to accept one character from the user and echo it (display it on the screen) and also wait after that for the user to press the enter key.

➜ **4. gets() :** This function is used to accept a string from the user. We will see in details of this when studying the chapter on strings.

➜ **5. putch() :** These functions are used to display a character on the monitor. putchar().

➜ **6. puts() :** This function is used to display a string on the monitor.

1.16 C++ IO Techniques

➔ **(W-15, S-16, S-17)**

Q. 1.16.1 Distinguish between ios member functions and manipulators.
(Ref. Sec. 1.16 and 1.17) **(5 Marks)**

Q. 1.16.2 Describe syntax of 'cin' and 'cout' with example. (Ref. Sec. 1.16 and Prog. 1.18.1) **(W-15, 4 Marks)**

Q. 1.16.3 Explain insertion and extraction operators.
(Ref. Sec. 1.16) **(S-16, 4 Marks)**

Q. 1.16.4 Which are the input-output operator in C++ ? Give suitable example.
(Ref. Sec. 1.16 and Prog. 1.18.1) **(S-17, 2 Marks)**

– The cin and cout are available in iostream.h. The different IO functions seen in section 1.15 are of C programming language. The C++ techniques of accepting input from keyboard is with the help of cin and displaying output is with the help of cout.

– The cin is always associated with an extraction operator (>>). This operator extracts the input from the standard input device i.e. the keyboard and gives it to the variable on the right of it.

– For e.g. cin>>x; will extract the input from the keyboard and the user entered value will be put in to the variable x. The user has to press the enter key after the user has entered the input.

– Multiple extraction operators can be concatenated in a single statement and hence, multiple values taken from the user and given to different variables. For e.g. cin>>x>>y;

– The cout statement is used for displaying an output on the monitor. It is always associated with the insertion operator. We have already seen the use of this statement in the program 1.10.1.

– The concatenation of insertion operators make it possible to combine multiple things and display on the monitor. If a particular statement is written in double quotes then it indicates that is to be displayed as it is.

- While if it is not written in double quotes then it must be the name of a variable whose value is to be displayed.

1.17 Manipulators

Q. 1.17.1 Write short note on : Manipulators.
(Ref. Sec. 1.17) **(8 Marks)**

Q. 1.17.2 Distinguish between ios member functions and manipulators.
(Ref. Sec. 1.16 and 1.17) **(5 Marks)**

- Manipulators are used to perform certain special operations like moving the cursor to new line, fixing the width of a data value, etc. These are used to specify certain formats of the output to be displayed when using the cout statement.
- The different manipulators used in C++ are endl, setw(), setprecision() and setfill(). These manipulators are available in the header file "iomanip.h".
- The manipulator **endl** is similar to the escape sequence "\n". It is used to bring the cursor to the next line.
- Example :

  cout<<"Hello"<<endl;

  This statement displays the string "Hello" and brings the cursor in the next line.
- Another manipulator i.e. **setw**() is used to set a width of the value of a variable. It works similar to the field of the format string in the printf() statement of C.
- It is used when a data is to be displayed in proper aligned format.
- For example:

  cout<<setw(6)<<temp;

  This statement displays the temp with six character spaces i.e. if temp is 2 digit value, 4 blank spaces will be displayed and then the 2 digit value will be displayed.

  The manipulator **setprecision**() is used to set the precision for a data after the decimal point i.e. number of digits after the decimal point.
- It is used to decide the precision required after the decimal point.

  For e.g. cout<<setprecision(2)<<x;

  will display the value of x with only two digits after decimal point.
- Another manipulator that may be used is **setfill**(). This manipulator is used to fill the extra spaces with the said character when the setw() manipulator is used. the extra spaces that are allocated to the data to be displayed can be filled with the said character in such case.
- For e.g.

  cout<<setfill('0')<<setw(10)<<x;

  will display the value of x in 10 spaces. The extra spaces allocated to display the data will be filled with '0', preceding the data value.

1.18 Basic Program Examples

➔ **(W-15, S-17)**

➢ **Program 1.18.1 : Write a C / C++ program to accept a number and display its square.** **W-15, S-17**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int x, y;
    cout<<"Enter a number:";
    cin>>x;
    y=x*x;
    cout<<"The square of "<<x<<" is "<<y;
    getch();
}
```

Output :

```
Enter a number:6
The square of 6 is 36
```

Explanation :

- In this program another header file is included i.e. conio.h for the two functions used in the program namely clrscr() and getch().
- The function clrscr() is used to clear the user screen and the function getch() as already discussed accepts a character from user.

- Here, we use this function, so that the processor waits for us to enter the character before closing the output window after displaying the result.
- As the processor operation is very fast it displays the result on the output screen and closes the output screen, before we can see the output. Hence, to make the processor to wait for us to see the result we give a getch() function.
- The variables x and y are declared as integer type data.
- Using the first cout statement the user is asked to provide a number whose square is to be found.
- The cin operator is used to accept the input from the user. The statement cin is associated with the extraction operator (>>), which indicates that the cin statement extracts the input from the standard input device and provides it in the variable x. The standard input device is keyboard.
- The next statement assigns the value of x multiplied by itself to provide the square in the variable y.
- The next cout statement is said to have concatenated insertion operators. The insertion operator is used along with the cout statement to display a data on the monitor.

➢ **Program 1.18.2 : Write a C / C++ program to accept two numbers and display its product.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int x, y, z;
    cout<<"Enter two number:";
    cin>>x>>y;
    z=x*y;
    cout<<"The product is "<<z;
    getch();
}
```

Output :

```
Enter two number:3
4
The product is 12
```

Explanation :

- The two numbers are accepted in the variables x and y by concatenating the extraction operator in a single statement using cin.
- The product is calculated using the multiply operator (*).

➢ **Program 1.18.3 : Write a program to calculate the simple interest taking principal, rate of interest and number of years as inputs from user.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    float rateOfInterest, years, principal, simpleInterest;
    cout<<"Enter the principal amount, rate of interest and
no. of years:";
    cin>>principal>>rateOfInterest>>years;
    simpleInterest = principal * rateOfInterest*years / 100;
    cout<<"The simple interest is "<<simpleInterest;
    getch();
}
```

Output :

```
Enter the principal amount, rate of interest and no. of
years:1000
9.5
10
The simple interest is 950
```

Explanation :

All the variables are taken as float as these parameters are normally with fraction part included like principal has rupees and paisa; rate of interest is fraction etc.

➢ **Program 1.18.4 : Write a program to accept basic salary from the keyboard. Calculate the gross salary that includes basic salary, 50% DA and 40% HRA.**

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

```
    float basic,hra,da,gross;
    cout<<"Enter the basic salary:";
    cin>>basic;
    hra=40*basic/100;
    da = 50*basic/100;
    gross=basic+da+hra;
    cout<<"The total salary is "<<gross;
    getch();
}
```

Output :

```
Enter the basic salary:10000
The total salary is 19000
```

Explanation :

- To calculate HRA we cannot write hra=40/100 * basic; as the division 40/100 will return the quotient of this division which is 0 and hence the hra will become 0.
- Once 40 is multiplied with basic the result is automatically type casted (upgraded) to the type of higher size i.e. float as one of the data was float (the variable basic is float type).

➢ **Program 1.18.5 : Write a program to swap two integers.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int a, b, temp;
    cout<<"Enter two numbers:";
    cin>>a>>b;
    temp=a;
    a=b;
    b=temp;
    cout<<"After swapping the values are a="<<a<<"
and b="<<b;
    getch();
}
```

Output :

```
Enter two numbers:3
4
After swapping the values are a=4 and b=3
```

Explanation :

For swapping two numbers we need to use a temp variable and store one of the values in that temp variable. The other value is to be then copied as shown in the program above.

➢ **Program 1.18.6 : Write a program to shift a number three bits left and display the result.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int a, b;
    cout<<"Enter a number to be shifted left:";
    cin>>a;
    b=a<<3;
    cout<<"After shifting three times left the value is "<<b;
    getch();
}
```

Output :

```
Enter a number to be shifted left:5
    After shifting three times left the value is 40
```

Syllabus Topic : Type Casting

1.18.1 Type Casting

➢ **Program 1.18.7 : Write a program to accept one int type data and one float type data. Multiply the two numbers and display the result.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int n1;
    float n2, result;
    cout<<"Enter one integer and one float type number
each:";
    cin>>n1>>n2;
    result=n1*n2;
```

```
    cout<<"The product is :"<<result;
    getch();
}
```

Output :

```
Enter one integer and one float type number each:2
3.9
The product is :7.8
```

Explanation :

- The two number being multiplied are of int type and float type respectively.
- Hence the result is automatically type casted (upgraded) to the type of higher size i.e. float. Hence the result variable is taken as float.

➢ **Program 1.18.8 : Write a program to accept a float number and display the integer part using type casting operator.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int a;
    float b;
    cout<<"Enter a float number:";
    cin>>b;
    a=(int)b;
    cout<<"The integer part of the number is:"<<a;
    getch();
}
```

Output :

```
Enter a float number:3.14
The integer part of the number is:3
```

Explanation :

This program uses type casting operator to go to a lower data type size.

➢ **Program 1.18.9 : Write a program to accept a number and display its equivalent ASCII using type casting.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int a;
    cout<<"Enter a number:";
    cin>>a;
    cout<<"This is ASCII value of: "<<(char)a;
    getch();
}
```

Output :

```
Enter a number:65
This is ASCII value of: A
```

Explanation :

This program also uses type casting operator to go to a lower data type size.

1.19 Exam Pack (MSBTE and Review Questions)

Q. 1 List any four features of POP.
(Refer Section 1.5) **(W-15, 4 Marks)**

Q. 2 Write any two characteristics of procedure oriented programming. **(Refer Section 1.5) (W-14, 2 Marks)**

Q. 3 In procedure oriented programming all data are shared by all functions. Is this statement TRUE? Justify your answer.
(Refer Section 1.5) **(W-16, 4 Marks)**

Ans. : No. The global data is only shared with all the functions

Q. 4 What are the features of procedure oriented programming ? **(Refer Section 1.5) (S-17, 4 Marks)**

☞ **Syllabus Topic : Basic Concepts of Object Oriented Programming and Object Oriented Languages**

Q. 5 Describe any four basic concepts of OOP.
(Refer Section 1.6) **(S-14, 4 Marks)**

Q. 6 List any four object oriented languages.

(Refer Section 1.6) (W-14, W-16, 2 Marks)

☞ **Syllabus Topic : Procedure Oriented Programming (POP) Verses Object Oriented Programming (OOP),C verses C++**

Q. 7 Differentiate between POP and OOP (any four points) **(Refer Section 1.16)**

(S-14, W-14, S-15, S-16, S-17, W-17, 4 Marks)

☞ **Syllabus Topic : Applications of OOP**

Q. 8 Enlist the applications of OOP.

(Refer Section 1.6.2) (S-14, S-15, W-16, 4 Marks)

☞ **Syllabus Topic : Tokens**

Q. 9 Explain the tokens of C/C++

(Refer Section 1.9) (5 Marks)

☞ **Syllabus Topic : Keywords**

Q. 10 What are keywords. List any 5 keywords are used in C/C++ **(Refer Section 1.9.2) (4 Marks)**

☞ **Syllabus Topic : Basic Data Types**

Q. 11 List data types in C/C++

(Refer Section 1.9.4) (4 Marks)

☞ **Syllabus Topic : Constants and Variables**

Q. 12 Define term constants and variables

(Refer Section 1.9.5) (2 Marks)

☞ **Syllabus Topic : Structure of C++ Program, Simple C++ Program**

Q. 13 Explain the structure of C++ program. **(Refer Section 1.10) (W-17, 4 Marks)**

Q. 14 Explain the structure of C++ program with suitable example.

(Refer Section 1.10 and Refer Program 1.10.1)

(S-17, 4 Marks)

Q. 15 With suitable diagram describe structure of C++ program.

(Refer Section 1.10 and Refer Program 1.10.1)

(W-15, 4 Marks)

Q. 16 Write structure of C++ program.

(Refer Program 1.10.1) (S-16, 2 Marks)

Q. 17 What do you mean by dynamic initialization of variable ? How is it achieved ?

(Refer Section 1.10.2) (5 Marks)

☞ **Syllabus Topic : User defined Data Types**

Q. 18 Write a short note on: Enumerated Data type.

(Refer Section 1.10.3.1) (5 Marks)

☞ **Syllabus Topic : Operators**

Q. 19 Explain different operator used in C++.

(Refer Section 1.12) (W-17, 4 Marks)

Q. 20 Give significance of '&' and '*' operators.

(Refer Section 1.12.1(4 & 5)) (S-17, 2 Marks)

Q. 21 Explain pointer operator and address operator with example.

(Refer Section 1.12.1) (S-16, 2 Marks)

Q. 22 How address of (&) operator is used in pointers, explain with example.

(Refer Section 1.12.1) (W-15, 2 Marks)

Q. 23 What is use of scope resolution operator ?

(Refer Section 1.12.1(6)) (W-17, 2 Marks)

Q. 24 State scope resolution operator and memory management operator in C++.

(Refer Sections 1.12.1 and 1.12.6)

(S-16 , S-17,4 Marks)

Q. 25 Explain bitwise operators.

(Refer Section 1.12.2(II)) (5 Marks)

Q. 26 Explain logical operators

(Refer Section 1.12.2(III)) (5 Marks)

Q. 27 Select the correct answer :

(Refer Section 1.12.1) (2 Marks)

int a = 29, b = 10; float c;

C = (float) (a/b);

The correct value of c is

(i) 2.9 (ii) 2

(iii) 3 (iv) compiler dependent.

Ans. : (ii) 2

Since; the operation is performed with a/b, which are both integers, the result is integer only. The result when converted to float remains float.

☞ **Syllabus Topic : Memory Management Operator**

Q. 28 State scope resolution operator and memory management operator in C++.

(Refer Sections 1.12.1 and 1.12.6)

(S-16 , S-17, 4 Marks)

Q. 29 List two memory management operators available in C++ and state its use in one line.

(Refer Section 1.12.6) **(W-15, 2 Marks)**

☞ **Syllabus Topic : Expressions**

Q. 30 Explain insertion and extraction operators.

(Refer Section 1.16) **(S-16, 4 Marks)**

Q. 31 Distinguish between ios member functions and manipulators.

(Refer Sections 1.16 and 1.17) **(5 Marks)**

Q. 32 Which are the input-output operator in C++ ? Give suitable example.

(Refer Section 1.16 and Refer Program 1.18.1)

(S-17, 2 Marks)

Q. 33 Describe syntax of 'cin' and 'cout' with example.

(Refer Section 1.16 and Refer Program 1.18.1)

(W-15, 4 Marks)

Q. 34 Write short note on : Manipulators.

(Refer Section 1.17) **(8 Marks)**

❑❑❑

UNIT I

CHAPTER 2

Control Statements

Syllabus

Principles of object Oriented Programming

- **Unit Outcomes**
  - Write POP based C++ program using decision making and loop structure for the given situation.
- **Topics and Sub-Topics**
  - Control Structures : Decision making Statements and Loops.

Syllabus Topic : Control Structures : Decision making Statements and Loops

2.1 Control Structures: Decision making Statements and Loops

- In programming we need to sometimes control the flow of operation other than just the sequential statements. In this case we need the control statements.
- Control statements are classified into two types viz. the **iterative statements (or loops)** and **conditional (or decision making) statements**.
- **Iterative statements** are use to perform certain operation repetitively for certain number of times. In C++, we have three iterative statements viz. for loop, while loop and do-while loop. Iterative statements are also called as repetitive statements as they repeat a set of statements for a given number of times.
- **Conditional statements** are use to perform the operations based on a particular condition i.e. if a condition is true perform one task else perform another task.
- In C++ we have two conditional statements namely if-else statements and switch-case statements. Conditional statements are also called as selective statements i.e. these statement select a particular statement to be executed based on the condition.
- In the subsequent sections we will see all these statements one by one in detail.

2.2 for Loop

- for is a iterative statement. It is used to repeat a set of statements number of times. The syntax (method of writing) **for** statement is given below :

```
for(initializations; condition; increment /
decrement / updating)
{
    -
    -
    -
    statements;
    -
    -
    -
}
```

- The sequence of execution of the for loop is such that the **initialization statements are executed first**. These initialization statements are executed only once. They are used to initialize the values of the variables.
- The **second step** is **checking the condition** specified. There can be only one condition. If more than one conditions are required they can be combined using the logical AND, OR

operators. If the condition is false the execution of the loop will be terminated i.e. the execution will go outside the braces of for loop, if the condition is not true.

- The **third step** is to **execute all the statements** inside the curly braces. These statements will be executed sequentially. The number of statements can be of any count. There can be another control statement if required inside one control statement.
- The **fourth step** is the **increment / decrement** or updating operations. These operations are not necessarily increment or decrement operations, but mostly these are increment decrement and hence called so. We can update the variables over here before starting the next iteration of the iterative statements.
- **Finally the control goes back to the second step**. As said the first step is executed only once, the steps that are repeated continuously are the second, third and fourth steps. After the fourth step the condition is checked again. If the condition is true the execution continues, else the control goes outside the for loop i.e. the curly braces.
- In the following sub section we will see some programs using the for loop.

2.2.1 Programs Based on for Loop

➢ **Program 2.2.1 : Write a program to find the factorial of a number.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n,fact;
    cout<<"Enter a number: ";
    cin>>n;
    for(i=1,fact=1;i<=n;i++)
    {
        fact=fact * i;
    }
    cout<<"Factorial of this number is:"<<fact<<endl;
    getch();
}
```

Output :

```
Enter a number: 4
Factorial of this number is: 24
```

Explanation :

- In this program we accept a number from user in the variable n. We initialize the value of i to 1 and also the value of the variable fact to 1.
- Remember, initialization part of the for loop can have multiple initializations. Update or increment / decrement operations can be multiple. But the condition statement has to be only one, if required to be combined with AND / OR operator.
- Now we keep on incrementing the value of i and multiplying this to the variable fact. Hence first time the value of fact is multiplied by 1, then by 2, then by 3 and so on. Thus we implement the logic of fact = $1 \times 2 \times 3 \times 4 \times \ldots\ldots \times n$.
- Note, the value of fact is initialized to 1, because it is to be multiplied by different numbers. Hence if we don't initialize it, some random number will be there in the variable fact which will be multiplied by these numbers i.e. 1,2,3... Also we cannot initialize the variable fact to 0, else i will always be multiplied with 0 and result will always be 0.

➢ **Program 2.2.2 : Write a program to display the multiplication table of a user entered number. The table must be up to 10.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n;
    cout<<"Enter a number: ";
    cin>>n;
    for(i=1;i<=10;i++)
    {
        cout<<n<<" X "<<i<<" = "<<n*i<<endl;
    }
    getch();
}
```

Output :

```
Enter a number: 5
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50
```

Explanation :

- In this program we accept a number from user in the variable n. We initialize the value of i to 1 and also then keep on calculating the value of n*I, until the value of i is equal to 10. These values are displayed in the multiplication table format.

➢ **Program 2.2.3 : Write a program to display odd numbers upto n.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n;
    cout<<"Enter a number: ";
    cin>>n;
    for(i=1;i<=n;i+=2)
    {
        cout<<i<<endl;
    }
    getch();
}
```

Output :

```
Enter a number: 10
1
3
5
7
9
```

Explanation :

- In this program we increment the value of i by 2. As already discussed it is not necessary that only increment decrement statements can be written in the updating statements; we can also put some other operations like the addition statement written in this program.
- The condition checked over here is that i must be less than or equal to n. As it is specified that the odd numbers are to be printed upto n, we want the value of I to reach maximum upto n.

➢ **Program 2.2.4 : Write a program to display first n elements of Fibonacci series.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a=0,b=1,c,i,n;
    cout<<"Enter a number: ";
    cin>>n;
    cout<<"Fibonacci Series\n0\n1\n";
    for(i=1;i<=n-2;i++)
    {
        c=a+b;
        cout<<c<<endl;
        a=b;
        b=c;
    }
    getch();
}
```

Output :

```
Enter a number: 10
Fibonacci Series
0
1
1
2
3
```

```
5
8
13
21
34
```

Note : Fibonacci series is a series wherein the first two elements are 0 and 1. Thereafter the next values are the sum of previous two elements. Hence it can be said that the series is 0,1,1,2,3,5,8,13,21,34,55 . . .

Explanation :

- We first display the values 0 and 1 i.e. the first two elements of Fibonacci series. Also these two values are initialized in the variables a and b.
- In every iteration, the next value is calculated and stored in the variable c and is printed. Also the values of a and b are updated accordingly i.e. the latest two values calculated must always be in a and b.
- The condition statement of for loop is important in this case. Since, the first two elements are already printed the counting of printing the elements of Fibonacci series in the loop must go only up to n-2 i.e. two values less which are printed outside the loop.

➢ **Program 2.2.5 : Write a program to calculate the value of the following series.**

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$$

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n;
    float sum=0.0;
    cout<<"Enter a number: ";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        sum=sum+1.0/i;
    }
    cout<<"The value of the series is:"<<sum;
    getch();
}
```

Output :

```
Enter a number: 4
The value of the series is:2.083333
```

Explanation :

- In this program we have taken the sum variable as float and initialized it to 0. This is because the result of this expression has to be a fraction value. And the terms are to be calculated and added to this sum, hence initially 0.
- Another important thing is we have written sum=sum+1.0/i. Instead of using 1, we have used 1.0. If we would have written 1 over there, the result would always be 0. Because 1/i, for i as integer and even 1 as integer, the result is always 0 i.e. the quotient.

➢ **Program 2.2.6 : Write a program to calculate the value of the following series.**

$$1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}$$

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n,fact=1;
    float sum=0.0;
    cout<<"Enter a number: ";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        fact=fact*i;
        sum=sum+1.0/fact;
    }
    cout<<"The value of the series is:"<<sum;
    getch();
}
```

Output :

```
Enter a number: 5
The value of the series is:1.716667
```

Explanation :

- The factorial is calculated for every term in the variable fact. The previous value of fact is just updated by multiplying the next number to the previous factorial. For example if the variable fact has the factorial of 4, when multiplied by 5 (i), the resultant is the factorial of 5.
- Remaining logic is same as the previous program.

➢ **Program 2.2.7 : Write a program to calculate the sine of an angle using the following series for x as the angle in radians.**

$$\text{Sin } x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \frac{x^n}{n!}$$

Accept the angle from user in degrees and display the result correct upto 6 decimal places

```
#include<iostream.h>
#include<math.h>
#include<conio.h>
void main()
{
    clrscr();
    int i, fact=1,sign=-1;
    float x,numerator,sum,term;
    cout<<"Enter an angle in degrees: ";
    cin>>x;
    x=x*3.14/180;
    term=x;
    sum=term;
    for(i=3;term>=0.0000001;i=i+2)
    {
        fact=fact*i*(i-1);
        numerator=pow(x,i);
        term=numerator/fact;
        sum=sum + sign * term;
        sign=sign*-1;
    }
    cout<<"The value of the series is:"<<sum;
    getch();
}
```

Output :

```
Enter an angle in degrees: 45
The value of the series is:0.706821
```

Explanation :

- The formula given in the question is for an angle in radians. Hence, we first take the angle in degrees from user and then convert it in radians.
- The first term is x, hence the value of x is first put into the variable term and hence into the variable sum.
- The variable term is used to calculate the value of each term and then to be added in the variable sum, which is used to calculate the value of the entire series.
- In the for loop, the value of i is started from 3, as seen in the series. The condition is that term should be less than 0.0000001 i.e. term's weight must be less than the sixth decimal place and hence allowing the change in the sum upto sixth decimal place.
- The factorial is calculated for every term in the variable fact. The previous value of fact is just updated by multiplying the next two numbers to the previous factorial.
- For example if the variable fact has the factorial of 3, when multiplied by 5 and 4 (i and i-1), the resultant is the factorial of 5.
- The numerator is simply calculated by using the pow() function in the header file math.h.
- The term is calculated by dividing the numerator by factorial.
- Then this term is added with the sum with proper sign. The variable sign is used for this. The sign is initialized to -1, and in every iteration its sign is changed by multiplying it with -1.
- The value of i, is incremented by 2; according to the series the terms require the factorial and exponent incremented by two w.r.t. its previous value.

➢ **Program 2.2.8 : Write a program to calculate the sine of an angle using the following series for x as the angle in radians.**

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \frac{x^n}{n!}$$

Accept the angle in degrees and value of n from user.

```
#include<iostream.h>
#include<math.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n,fact=1,sign=-1;
    float x,numerator,sum,term;
    cout<<"Enter an angle in degrees and the value of n: ";
    cin>>x>>n;
    x=x*3.14/180;
    term=x;
    sum=term;
    for(i=3;i<=n;i=i+2)
    {
        fact=fact*i*(i-1);
        numerator=pow(x,i);
        term=numerator/fact;
        sum=sum + sign * term;
        sign=sign*-1;
    }
    cout<<"The value of the series is:"<<sum;
    getch();
}
```

Output :

```
Enter an angle in degrees and the value of n: 45
10
The value of the series is:0.706821
```

Explanation :

- The only change in this program compared to the previous one is that the value of n is taken from the user and the value of i is varied up to n.

➢ **Program 2.2.9 : Write a program to display the square, cube, fourth power, ...nth power of x, where x is an integer taken from user.**

```
#include<iostream.h>
#include<math.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n;
    float x;
    cout<<"Enter the value of x and the value of n: ";
    cin>>x>>n;
    for(i=2;i<=n;i++)
    {
        cout<<x<<" raised to "<<i<<" is
"<<pow(x,i)<<endl;
    }
    getch();
}
```

Output :

```
Enter the value of x and the value of n: 3
5
3 raised to 2 is 9
3 raised to 3 is 27
3 raised to 4 is 81
3 raised to 5 is 243
```

➢ **Program 2.2.10 : Write a program to display the value of s for t = 1, 5, 10, 15, 20, 0...100.**

$$s = s_0 + v_0 + \frac{1}{2} a t^2$$

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int t;
    float s,s0,v0,a;
    cout<<"Enter the values of s0, v0 and a: ";
    cin>>s0>>v0>>a;
cout<<"t\tS"<<endl<<"1\t"<<s0+v0+0.5*a<<endl;
    for(t=5;t<=100;t+=5)
    {
        cout<<t<<"\t"<<s0+v0+0.5*a*t*t<<endl;
```

```
    }
    getch();
}
```

Output :

```
Enter the values of s0, v0 and a: 2 3 9.8
t    S
1     9.9
5     127.500002
10    495.00001
15    1107.500021
20    1965.000038
25    3067.50006
30    4415.000086
35    6007.500117
40    7845.000153
45    9927.500193
50    12255.000238
55    14827.500288
60    17645.000343
65    20707.500403
70    24015.000467
75    27567.500536
80    31365.00061
85    35407.500689
90    39695.000772
95    44227.500861
100   49005.000954
```

Explanation :

- The values of s0, v0 and a are taken from the user.
- The output is displayed in a tabular form with the values of t and S separated by a horizontal tab (using the escape sequence "\t").
- The first value i.e. with t=1, is displayed outside the loop, as it doesn't match the series of 5, 10, 15, 20…100.
- The remaining values are calculated and displayed using a for loop wherein the value of t is incremented by 5 after every iteration.
- The new value of S is calculated and displayed again separated with a horizontal tab.
- This loop is repeated until the value of t reaches 100.

2.2.2 Nested for Loop

- A for loop inside another for loop is called as nested for loop.
- When a particular operation has two references, we require nested for loop. For example if we want to keep a reference of row number and column number, then we can use a nested for loop.
- Many programs based on nested for loop is given below. In this case the variable i keeps a track of row number and j keeps a track of column number.

➢ **Program 2.2.11 : Write a program to display the following**

1
11
111
1111
11111

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,j;
    for(i=1;i<=5;i++)
    {
        for(j=1;j<=i;j++)
        {
            cout<<"1";
        }
    cout<<endl;
    }
    getch();
}
```

Output :

```
1
11
111
1111
11111
```

Explanation :

- The inner loop should display the number of 1s equal to the line number. Hence, the value of j is taken from 1 to I, which keeps the record of line number.
- Hence, you will notice in the last four programs we have written, I has always kept a track of line number. This is just to make it convenient for understanding.

➢ **Program 2.2.12 : Write a program to display the following for the user specified number of lines.**

```
        *
       **
      ***
     ****
    *****
   ******
  *******
 ********
|
|
n lines
```

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,j,n;
    cout<<"Enter the number of lines:";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n-i;j++)
        {
            cout<<" ";
        }
        for(j=1;j<=i;j++)
        {
            cout<<"*";
        }
        cout<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of lines:8
        *
       **
      ***
     ****
    *****
   ******
  *******
 ********
```

Explanation :

- In this program there are two operations to be performed in each line viz. print blank spaces and some stars based on the line number.
- Hence, we have two inner loops. Finally the third thing to be done is endl, to go to the next line.
- The first inner loop is used to display the number of blank spaces. The blank spaces in first line will be five, if the number of lines is six.
- The next line number of blanks spaces will reduce by one and keep on reducing in every line.
- Hence in general the number of blank spaces in each line is n-I, where n is the number of lines and i is the current line number.
- The same variable i.e. j can be used in the next for loop also as it is done in the above program.
- The next for loop i.e. the inner second one is used to display the number of stars equal to the line number.

– Finally once everything is done for the line we use endl to go to the next line for the next iteration. This process will continue for n lines, where the value of n is taken from user.

➢ **Program 2.2.13 : Write a program to display the following asking the user for the number of lines.**

1
12
123
1234
12345
123456

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,j,n;
    cout<<"Enter the number of lines:";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=i;j++)
        {
            cout<<j;
        }
        cout<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of lines:6
1
12
123
1234
12345
123456
```

Explanation :

In this program, the operation is almost similar to the pervious programs. Only the difference being that instead of stars, here we print the value of j itself. Hence the numbers are printed instead of stars.

➢ **Program 2.2.14 : Write a program to display the following asking the user for the number of lines.**

1
121
12321
1234321
123454321
12345654321

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,j,n;
    cout<<"Enter the number of lines:";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n-i;j++)
        {
            cout<<" ";
        }
        for(j=1;j<=i;j++)
        {
            cout<<j;
        }
        for(j=i-1;j>=1;j--)
        {
            cout<<j;
        }
        cout<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of lines:6
                1
               121
              12321
             1234321
            123454321
           12345654321
```

Explanation :

- Here, we have to perform 3 operations in each line and then next line i.e. endl.
- The first operation is blank spaces. The second operation is printing j i.e. numbers from 1 to line number.
- The third operation is to print the numbers from i-1 (where i is line number) to 1. For each of these three operations there is an inner for loop.

➢ **Program 2.2.15 : Write a program to display the following asking the user for the number of lines.**

A
ABA
ABCBA
ABCDCBA
ABCDEDCBA
ABCDEFEDCBA

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,j,n;
    cout<<"Enter the number of lines:";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n-i;j++)
        {
            cout<<" ";
        }
        for(j=1;j<=i;j++)
        {
            cout<<(char)(j+64);
        }
        for(j=i-1;j>=1;j--)
        {
            cout<<(char)(j+64);
        }
        cout<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of lines:6
                A
               ABA
              ABCBA
             ABCDCBA
            ABCDEDCBA
           ABCDEFEDCBA
```

Explanation :

- This program is exactly same as the previous one. The only difference being we have to type cast the number to char and add 64 to it to get the ASCII value of 'A'.
- Hence, in the above program we have
  cout<<(char)(j+64);
  instead of the statement of previous program
  cout<<j;

2.3 while and do-while Loops

- while and do-while loops are also for repetitive operations.
- The operations are slightly different than the for loop, but the same operations can be implemented by for, while or do-while loops.

- Although there is one major difference in a do-while loop wherein one particular operation cannot be implemented. This will be discussed later in this section.
- The syntax of while loop is given below :

```
Syntax of while loop:
while(condition)
{
    -
    -
    statements;
    -
    -
    -
}
```

- The operation of the while loop is such that, first the condition is checked. If the condition is true, then the statements are executed.
- Once the statements are executed, the condition is again checked and this keeps on repeating, until the condition is false. If the condition is false, the statements inside the loop are not executed, instead the control directly comes out of the loop.
- The syntax of do-while loop is given below :

```
Syntax of do-while loop:
do
{
    -
    -
    statements;
    -
    -
    -
} while(condition);
```

- In this case the operation is slightly different i.e. first the statements are executed and then the condition is checked. If the condition is true the statements are executed again. If the condition is false, the statements are not executed again.
- One major point to be noted is that in case of do-while loop, the statements are executed at least once even if the condition is not true for the first statement. On the other hand, in case of for loop and while loop, even for the first time the statements are executed only if the condition is true.
- The implementation of a for loop converted to while and similarly do-while loop is shown in Fig. 2.3.1(a) and 2.3.1(b) respectively.

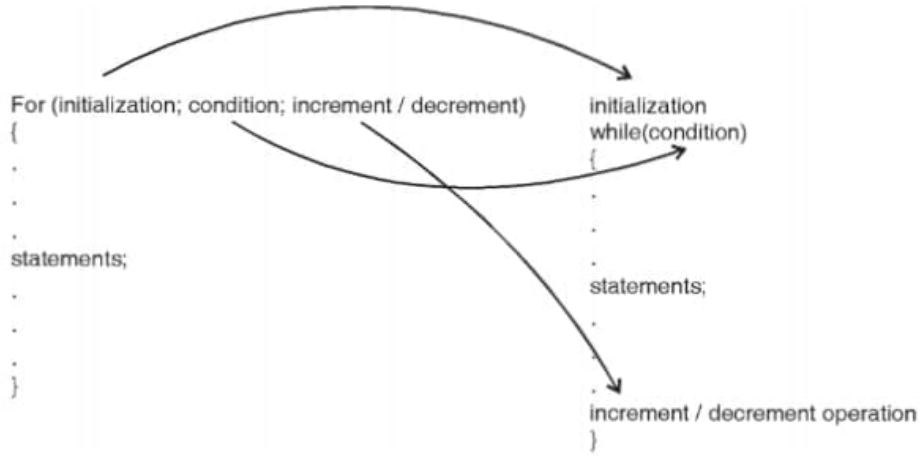


Fig. 2.3.1(a) : Implementation of a while loop related to for loop

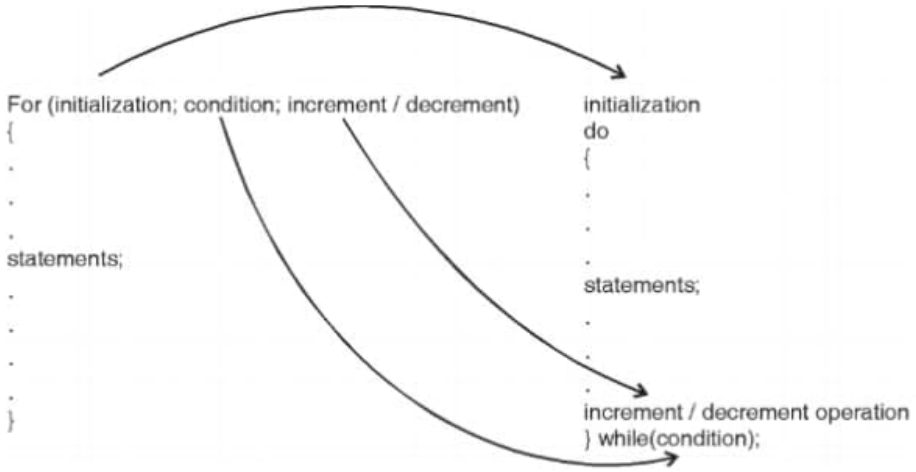


Fig. 2.3.1(b) : Implementation of a do-while loop related to for loop

- You will note (in Fig. 2.3.1) that the initialization statements are removed outside the loop, the condition is in the brackets associated with the while part and increment decrement are to be inside the loop at the end so as to implement a similar operation as that in the for loop.
- Differences between the while and do-while loop is given in Table 2.3.1.

Table 2.3.1 : Differences between while and do-while loop

	while loop		do-while loop

1	Syntax of while loop:
	while(condition)
	{
	-
	-
	statements;
	-
	-
	-
		1	do
			{
			-
			-
			statements;
			-
			-
			-
			} while(condition);

2	This is called as a entry controlled loop, as the entry inside the loop is possible only if the condition is true.	2	This is called as exit controlled loop, as the entry inside this loop is sure i.e. no condition is checked to enter inside the loop. Nut the exit is possible only if the condition is false.
3	If the condition is not true for the first time, the control will never enter into the loop. Hence there is a possibility that the control never enters into the loop and the statements inside the loop are never executed.	3	Even if the condition is not true for the first time the control will enter into the loop. Hence the statements inside the loop will be executed at least once even if the condition is not true for the first time.
4	There is no semicolon (;) after the condition in the syntax of the while loop.	4	There is a semicolon (;) after the condition in the syntax of the do-while loop.

- We will do some basic programs using these loops in the subsequent section. We will first see some programs already implemented using for loop now implemented using while and do-while loops.
- There after we will see some programs wherein we will notice that the while or do-while loop give better implementation.

2.3.1 Programs Based on while and do-while Loop

➢ **Program 2.3.1 : Write a program to count the number of digits in a user entered number.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,count=0;
    cout<<"Enter a number:";
    cin>>n;
```

```
    while(n!=0)
    {
        n=n/10;
        count++;
    }
    cout<<"Number of digits="<<count;
    getch();
}
```

Output :

```
Enter a number:7567
Number of digits=4
```

➢ **Program 2.3.2 : Write a program to reverse the digits of a user entered number and store in another variable.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,reverse=0,digit;
    cout<<"Enter a number:";
    cin>>n;
    while(n!=0)
    {
        digit=n%10;
        n=n/10;
        reverse=reverse*10+digit;
    }
    cout<<"The reverse number is:"<<reverse;
    getch();
}
```

Output :

```
Enter a number:1234
The reverse number is:4321
```

Explanation :

- The digits are separated in the same way as seen in the previous programs.
- The variable reverse is initialized to 0. The digit is added to the previous reverse number multiplied by 10. This is done because we first get the lower digit and we have to put it to the highest digit.

2.4 if-else Selective Statement

- This is a very important statement used to check the condition and accordingly execute a set of statements, based on whether the condition is true or false. Hence it is called as selective statement. It selectively executes some statements and doesn't execute some.
- The syntax of the if-else condition is as given below

```
Syntax of the if-else statement
if(condition)
{
    -
    -
    statements1;
    -
    -
}
else
{
    -
    -
    statements2;
    -
    -
}
```

- The set of statements named as statements1 in the above syntax are executed if the condition given with the if statement is true. The set of statements statements2 are not executed in this case.
- If the condition specified in the if statement is false then the statements named as statements2 in the above syntax are executed. The set of statements statements1 are not executed in this case.

Note: The else part is optional i.e. we can have a if statement without the else statement. As seen in the above given syntax, only the first half i.e.

```
if (condition)
{
    -
    -
    Statements
    -
    -
}
will be there
```

– The subsequent section deals with the programs using if-else statement.

2.4.1 Programs using if-else Statement

➢ **Program 2.4.1 : Write a program to check if the entered number is prime number or not.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i=2,n;
    cout<<"Enter a number:";
    cin>>n;
    while(n%i!=0)
    {
        i++;
    }
    if(n==i)
    {
        cout<<"Prime Number";
    }
    else
    {
        cout<<"Not a prime number";
    }
    getch();
}
```

Output :

```
Enter a number:17
Prime Number
```

Explanation :

– A simple logic is used to find out whether the entered number is prime or not. The logic is similar to the one used by us to check the number being prime or not.

– The number entered by user is checked for divisibility by 2, 3 and so on up to the entered number. If there is a divisibility then the further checking stops and the control comes out of the while loop. In worst case the control will come out of the while loop when the value of i is equal to n, because a number is definitely divisible by itself.

– Finally when the control comes out of the while loop, if n is equal to i, then the number is a prime number else it is not. This is checked using the if-else statement

➢ **Program 2.4.2 : Write a program to display first n prime numbers where the value of n is taken from the user.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n,x=1;
    cout<<"Enter a number:";
    cin>>n;
    cout<<"The following are "<<n<<" prime
    numbers"<<endl;
    while (n!=0)
    {
        x++;
        i=2;
        while(x%i!=0)
        {
            i++;
        }
        if(x==i)
        {
            cout<<x<<endl;
```

```
                n--;
            }
        }
    getch();
}
```

Output :

```
Enter a number:10
The following are 10 prime numbers
2
3
5
7
11
13
17
19
23
29
```

Explanation :

- Another while loop is made outside the earlier while loop (as in earlier program) to keep a track of the number of prime numbers generated. In this case n is taken as a count to keep the track of these numbers.
- A variable x is continuously incremented after being checked for prime number.
- The value of i is initialized to 2 after every outer loop iteration so that the checking should again start with the divisor being 2.

➢ **Program 2.4.3 : Write a program to check if the entered number is Armstrong or not.**

> **Note :** A number is said to be Armstrong number if the sum of the cube of its digits is equal to the number itself. For e.g. 153, the sum of the cubes is $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$ i.e. the original number itself.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int sum=0,digit,x,copy;
    cout<<"Enter a number:";
    cin>>x;
    copy=x;
    while(x!=0)
    {
        digit=x%10;
        sum=sum+digit*digit*digit;
        x=x/10;
    }
    if(sum==copy)
    {
        cout<<"Armstrong Number";
    }
    else
    {
        cout<<"Not an Armstrong Number";
    }
    getch();
}
```

Output :

```
Enter a number:789
Not an Armstrong Number
```

Explanation :

- The variable sum is initialized to 0, and then the cubes of the user entered number are added to this variable sum.
- The method of separating the digits is same as seen in quite a lot previous programs. But in this case the cube of the digit is taken and then added to the variable sum
- Another variable called as copy is declared in this program to keep the copy of the user entered number as the number entered by the user will become 0 after dividing it by 10 in every iteration. Hence, we would not have any copy of the original number without the variable copy. To avoid this problem the variable copy is declared and keeps the value of the user entered number.

➢ **Program 2.4.4 : Write a program to display first n Armstrong numbers where the value of n is taken from the user.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int sum,digit,x=99,copy,n;
    cout<<"Enter a number:";
    cin>>n;
    cout<<"The list of "<<n<<" Armstrong number is given below"<<endl;
    while(n!=0)
    {
        x++;
        copy=x;
        sum=0;
        while(x!=0)
        {
            digit=x%10;
            sum=sum+digit*digit*digit;
            x=x/10;
        }
        if(sum==copy)
        {
            cout<<copy<<endl;
            n--;
        }
        x=copy;
    }
    getch();
}
```

Output :

```
Enter a number: 4
The list of 4 Armstrong number is given below
153
370
371
407
```

Explanation :

Similar to the previous program only that the external loop keeps a track of number of values found to be Armstrong number.

Note :

1. The Armstrong numbers are only three digit numbers, hence the value of x is initialized to 99 i.e. checking of the numbers to be Armstrong or not begins from 100, the first 3 digit number
2. There are only 4 Armstrong numbers, hence do not give a very huge number as the value of n else the system may hang.

2.4.2 if-else Ladder or if-else if

Q. 2.4.1 Compare switch() control statement with if-else if. (Ref. Secs. 2.4.2 and 2.5) **(5 Marks)**

– In some cases we have to check multiple cases of a particular condition. In such cases we have to use an if-else ladder. A set of if-else statements as shown below is called as if-else ladder.

```
Syntax:
    if(condition)
    {
        statements;
    }
    else
    {
        if (condition)
        {
            statements;
        }
        else
        {
            if (condition)
            {
                statements;
            }
            else
            {
```

```
                if(condition)
                {
                        statements;
                }
                |
                |
                |
                else
                {
                        statements;
                }
            }
        }
    }
}
```

- In this case the first condition is checked, if it is true the statements inside the if statement are executed. But if the condition is false it goes to the else statement.
- Again there is a condition with if; the statements are executed if this second condition is true. Else it again goes to the else and again checks the condition associated with this if statement. Thus if one condition satisfies no other else is checked thereafter.
- A use of such a if-else statement is shown in the program below.

➢ **Program 2.4.5 : Write a program to display the class according to the marks scored by a student. The marks scored is taken as input and the class is displayed according to the following range:**

Marks	Class
70-100	Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int marks;
    cout<<"Enter marks scored (0-100):";
    cin>>marks;
    if(marks>=70)
    {
        cout<<"Distinction";
    }
    else
    {
        if (marks>=60)
        {
            cout<<"First Class";
        }
        else
        {
            if (marks>=50)
            {
                cout<<"Second Class";
            }
            else
            {
                if(marks>=40)
                {
                    cout<<"Pass Class";
                }
                else
                {
                    cout<<"Fail";
                }
            }
        }
    }
    getch();
}
```

Output :

```
Enter marks scored (0-100):89
Distinction
```

Explanation :

The above program is said to implement if else ladder with the operation as explained just before the program.

2.5 Switch-Case Selective Statement

Q. 2.5.1 Compare switch() control statement with if-else if. (Ref. Secs. 2.4.2 and 2.5) **(5 Marks)**

- In the previous section we have seen the if-else ladder. A better solution of this is switch-case. Using switch-case the if-else ladder can be implemented in a much better way.
- The syntax of switch-case is given below :

```
switch(expression / variable)
{
case label1: statements;
            break;
case label2:statements;
            break;
case label3:statements;
            break;
     |
     |
case labeln:statements;
            break;
default : statements;
}
```

Note : Break statement transfers the control outside the current loop. We will see some more cases of break statement along with the for / while /do-while statements in section 2.6.

- The expression or variable can be given in the brackets associated with the switch statement. The values of this expression / variable are the labels associated with the cases inside the switch statement.
- The value of the expression / variable is first compared with the label1. If they are equal, then the statements followed by the corresponding case are executed. The break statement followed by these statements, transfers the control after the switch statement.
- If the expression / variable is not equal to label1, then it is directly compared to label2 without executing the statements followed by the label1.
- Hence the statements of only that case are executed with the correct label value of the expression / variable.

Note :

1. Break statement is not compulsory. But if the break statement is nut written everything will be executed after the case statements.
2. Default is not necessary. But in case if default is not written and some case occurs which is not considered then there will be no operation performed and the user will not be able to realize the problem in the program.
3. The label can only be value, it cannot be a condition or an expression.
4. The brackets associated with the switch statement can have either the variable or expression and no condition.

- We will see some program examples of this in the subsequent section.

➢ **Program 2.5.1: Write a program to display a user entered number in words.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,digit,rev=0;
    cout<<"Enter a number:";
    cin>>n;
    while(n!=0)
    {
        digit=n%10;
        rev=rev*10+digit;
        n=n/10;
    }
    while(rev!=0)
    {
        digit=rev%10;
        rev=rev/10;
        switch(digit)
        {
            case 0:cout<<"Zero ";
```

```
            break;
            case 1:cout<<"One ";
            break;
            case 2:cout<<"Two ";
            break;
            case 3:cout<<"Three ";
            break;
            case 4:cout<<"Four ";
            break;
            case 5:cout<<"Five ";
            break;
            case 6:cout<<"Six ";
            break;
            case 7:cout<<"Seven ";
            break;
            case 8:cout<<"Eight ";
            break;
            case 9:cout<<"Nine ";
            break;
        }
    }
    getch();
}
```

Output :

```
Enter a number:513
Five One Three
```

Explanation :

The number is first reversed using the logic we have already done in one of the previous program. This is done because the last digit is separated first using the technique we have already studied. Using the switch case statement the corresponding words of the digits are selected and printed.

- **Program 2.5.2 : Write a menu driven program to perform add / subtract / multiply / divide / modulus based on the users choice.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int no1,no2,result,choice;
    cout<<"Enter two numbers:";
    cin>>no1>>no2;
    cout<<"1.Add\n2.Subtract\n3.Multiply\n4.Divide\n5.Modulus\nEnter your choice:";
    cin>>choice;
    switch(choice)
    {
        case 1:result=no1+no2;
        cout<<"Sum= "<<result;
        break;
        case 2:result=no1-no2;
        cout<<"Difference= "<<result;
        break;
        case 3:result=no1*no2;
        cout<<"Product="<<result;
        break;
        case 4:result=no1/no2;
        cout<<"Quotient="<<result;
        break;
        case 5:result=no1%no2;
        cout<<"Remainder="<<result;
        break;
        default:cout<<"Invalid choice";
    }
    getch();
}
```

Output :

```
Enter two numbers: 12
6
1. Add
2. Subtract
3. Multiply
4. Divide
```

```
5. Modulos
Enter your choice: 3
Product=72
```

Explanation :

- Such a program wherein the user is given a choice to perform one of the many listed operations is called as a menu driven program.
- Here, we have first taken from the user the two numbers on which the operation is to be performed. Then we have listed the menu, based on which the user has to enter his choice and the switch case performs the operation indicated by the users choice.

2.6 Branching Statements (break, continue and goto)

Q. 2.6.1	Explain the following with example:	
	(i) break (Ref. Sec. 2.6)	
	(ii) continue (Ref. Sec. 2.6)	**(4 Marks)**
Q. 2.6.2	Write a short note on continue statement (Ref. Sec. 2.6)	**(5 Marks)**

- These statements transfer the control to a different place in the program. We will see the operation of each of these.
- We have already seen the use of break statement in switch-case. Let us see the use of continue and break in other loop statements.
- The break statement neglects the statements after it in the loop and transfers the control outside the loop as shown in Fig. 2.6.1.
- The Fig. 2.6.1 shows the operation of break statement in each of the loop statements i.e. for, while and do-while statements.

```
for(intitialization; condition; inc / dec)
{
-
-
-
break;
-
-
-
}
```

(a) Operation of break statement in a for loop

Fig. 2.6.1 Contd…

```
while(condition)
{
-
-
-
break;
-
-
-
}
```

(b) Operation of break statement in a while loop

```
do
{
-
-
-
break;
-
-
-
} while(condition);
```

(c) Operation of break statement in a do-while loop

Fig. 2.6.1

- The continue statement also neglects the statements after it in the loop and transfers the control back to the starting of the loop for next iteration.
- The Fig. 2.6.2 shows the operation of continue statement in each of the loop statements i.e. for, while and do-while statements.

```
for(intitialization; condition; inc / dec)
{
-
-
-
continue;
-
-
-
}
```

(a) Operation of continue statement in a for loop

```
while(condition)
{
-
-
-
continue;
-
-
-
}
```

(b) Operation of continue statement in a while loop

Fig 2.6.2 Contd…

```
do
{
-
-
-
continue;
-
-
-
}while(condition);
```

(c) Operation of continue statement in a do-while loop
Fig. 2.6.2

- The goto statement transfers the control to the label specified with the goto statement. A label is any name given to a particular part in the program. The label is to be followed with a colon (:).
- The syntax of a goto statement is as shown below :

```
-                          -
goto label1                label2:
-                          -
-          or              -
label1:                    -
-                          goto label2
-                          -
```

- Thus, it shows that the label can be before or after the goto statement and hence the control can be transferred to earlier or next statements using a goto statement.
- We will be using these statements (especially break) in our further chapters but still let us see the use of these statements with one program each.

➢ **Program 2.6.1: Write a program to demonstrate the use of goto statement.**

or

Write a program to accept numbers from user and add them. Stop accepting the numbers if the sum is greater than or equal to 100 and display the sum.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,total=0;
    again:
    cout<<"Enter a number:";
    cin>>n;
    total=total+n;
    if(total<100)
    goto again;
    cout<<"Total="<<total;
    getch();
}
```

Output :

```
Enter a number:56
Enter a number:12
Enter a number:33
Total=101
```

Explanation :

- In this program, the label again is used besides the cout statement. The condition checks if the value of the total is not yet equal to 100, then it transfers the control back to the cout statement and accept another number from the user.
- The statement goto again; transfers the control to this cout statement which asks for another input from the user.

➢ **Program 2.6.2: Write a program to demonstrate the use of break statement**

Or

Write a program to accept 10, 2-digit numbers from user and add them. If the user enters a three digit number stop accepting the numbers and display the sum.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,total=0,i;
    for(i=1;i<=10;i++)
    {
        cout<<"Enter a number:";
        cin>>n;
        if(n>99)
```

```
            break;
            total=total+n;
    }
    cout<<"Total="<<total;
    getch();
}
```

Output :

```
Enter a number:45
Enter a number:54
Enter a number:22
Enter a number:59
Enter a number:121
Total=180
```

Explanation :

- In this program, the break statement is executed if a number greater than 99 (i.e. largest two digit number) is given. The sum is directly displayed without accepting the other numbers in this case.
- The break statement when executed the control is directly transferred outside the loop and the cout statement is directly executed.

➢ **Program 2.6.3 : Write a program to demonstrate the use of continue statement.**

Or

Write a program to accept 5, 2-digit numbers from user and add them. If the user enters a three digit number, this number should be discarded and again accepted. Also an indication must be given to the user that he has entered a number greater than 99 and hence it is not accepted. The sum must be displayed at the end.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,total=0,i;
    for(i=1;i<=5;i++)
    {
            cout<<"Enter a number:";
            cin>>n;
            if(n>99)
            {
                cout<<"Number is greater than 99\n";
                i--;
                continue;
            }
            total=total+n;
    }
    cout<<"Total="<<total;
    getch();
}
```

Output :

```
Enter a number:12
Enter a number:121
Number is greater than 99
Enter a number:22
Enter a number:11
Enter a number:89
Enter a number:99
Total=233
```

Explanation :

- Here if the number entered by user is greater than 99, then this number is not considered and the user is asked to enter the number again.
- Also the value of i is decremented as this number is not to be considered as one of the five numbers to be added.

2.7 Some More Practice Programs

➢ **Program 2.7.1 : Write a program to find GCD of two numbers.**

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

```
    int n1,n2,gcd;
    cout<<"Enter two numbers:";
    cin>>n1>>n2;
    if (n1<n2)
    gcd=n1;
    else gcd=n2;
    while(n1%gcd!=0 || n2%gcd!=0)
    {
        gcd--;
    }
    if(gcd==1)
    cout<<"GCD doesnt exits";
    else
    cout<<"GCD="<<gcd;
    getch();
}
```

Output :

```
Enter two numbers:125
20
GCD=5
```

Explanation :

- We start with the lower number of the two, since the highest GCD possible of two numbers is the smaller number.
- We find the number by which both the user entered numbers are divisible. We repeat it in the while loop until we get such a number.
- If a GCD doesn't exist for two numbers the value of gcd reaches to 1, with which every number is divisible. Hence the control comes out of the while loop.
- Now we check if the value of gcd is equal to 1 then display gcd doesn't exist else display the calculated value of GCD.

➢ **Program 2.7.2 : Write a program to find LCM of two numbers.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n1,n2,lcm;
    cout<<"Enter two numbers:";
    cin>>n1>>n2;
    if (n1>n2)
    lcm=n1;
    else lcm=n2;
    while(lcm%n1!=0 || lcm%n2!=0)
    {
        lcm++;
    }
    cout<<"LCM="<<lcm;
    getch();
}
```

Output :

```
Enter two numbers:25
10
LCM=50
```

Explanation :

- We start with the larger number in this case as the smallest LCM possible is the larger of two numbers.
- Then we keep on incrementing the value of the variable lcm until we find that both the user entered numbers are divisible by this variable lcm.
- Finally we print the value of the calculated LCM.

➢ **Program 2.7.3 : Find the output of the following program.**

```
#include<iostream.h>
void main()
{
    int i=1,j=1;
    for(;;)
    {
        if(i>3) break;
        else j+=i;
        cout<<j<<endl;
        i+=j;
    }
}
```

Output :

```
2
5
```

➢ **Program 2.7.4 : Write a program to convert a decimal number to binary format.**

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    int n,i;
    cout<<"Enter a number";
    cin>>n;
    cout<<"Binary form is:";
    for(i=15;i>=0;i--)
    {
        cout<<n/(int)pow(2,i);
        n=n%(int)(pow(2,i));
    }
    getch();
}
```

Output 1 :

```
Enter a number12
Binary form is:0000000000001100
```

Output 2 :

```
Enter a number8
Binary form is:0000000000001000
```

Explanation :

- We have already seen how a number is converted into binary form in chapter 1. But that technique gives reverse answer as seen there.
- Here, to get the answer in proper order, we are dividing the number first by 2^{15}, the 2^{14}, then 2^{13}, and so on up to 2^0. This logic is implemented in the for loop. 2^{15} is selected because the size of int data type is 16-bits i.e. from bit 0 to bit15.
- In this for loop we start the value of i from 15 and reach to 0. We have taken the quotient to get the required bit and we repeat the same thing with the remainder of the division.

➢ **Program 2.7.5 : Find the output of the following program.**

```
#include<iostream.h>
#include<conio.h>
int main()
{
    for(int i=0;i<8;i++)
    {
        if(i%2==0)
        cout<<i+1<<endl;
        else if(i%3==0)
        continue;
        else if (i%5==0)
        break;
        cout<<"\n End of Program \n";
    }
    cout<<"\nEnd of program \n";
}
```

Output :

```
1

 End of Program

 End of Program
3

 End of Program
5

 End of Program

End of Program
```

2.8 Exam Pack (MSBTE and Review Questions)

☞ **Syllabus Topic : Control Structures : Decision making Statements and Loops**

Q. 1 Write a program to display a series of 15 terms of Fibonacci series.
(Refer Program 2.2.4) **(10 Marks)**

Q. 2 Write a program to generate Fibonacci series.
(Refer Program 2.2.4) **(5 Marks)**

Q. 3 Write a program to display following pattern using nested loops :

A

ABA

ABCBA

ABCDCBA **(Refer Program 2.2.15)** **(10 Marks)**

Q. 4 Select the correct answer :

(a)
```
I = 10 ;
do
{
    ..... do something .....
} while (I < 10) ;
```

(i) "do something" will be executed at least once

(ii) "do something" will not be executed at all

(iii) do-while loop is not a valid loop

(iv) None of the above. **(2 Marks)**

Ans. : (i) is the correct answer.

The do while loop is executed once, even if the condition is false as already discussed in **Section 2.3.**

Q. 5 Write a program to reverse the digits of a positive integer number for example, if the number 8375 is entered, the number displayed should be 5738.
(Refer Program 2.3.2) **(10 Marks)**

Q. 6 Write a program to check the entered number is prime or not. **(Refer Program 2.4.1)** **(8 Marks)**

Q. 7 Compare switch() control statement with if-else if.
(Refer Sections 2.4.2 and 2.5) **(5 Marks)**

Q. 8 Write a C++ program to check if the given number is "Armstrong Number". An Armstrong number of 3 digits is an integer such that the sum of the cubes of its digits is equal to the no. itself.
(Refer Program 2.4.3) **(10 Marks)**

Q. 9 Write a program that reads any positive integer n and prints the digits in words. (For example, if input is 1265 then output is one two six five).
(Refer Program 2.5.1) **(10 Marks)**

Q. 10 Explain the following with example:-

(i) break **(Refer Section 2.6)**

(ii) continue **(Refer Section 2.6)** **(4 Marks)**

Q. 11 Write a short note on continue statement.
(Refer Section 2.6) **(5 Marks)**

Q. 12 Write a C++ program to find the Greatest Common Divisor of two numbers.
(Refer Program 2.7.1) **(10 Marks)**

Q. 13 What is the output of the following program :

```
#include<iostream.h>
void main()
{
    int i=1,j=1;
    for(;;)
    {
        if(i>3) break;
        else j+=i;
        cout<<j<<endl;
        i+=j;
    }
}
```

(Refer Program 2.7.3) **(5 Marks)**

Q. 14 Write a program in C++ to convert a given integer to binary format. (Use of class 10 is optional) Write output of your program for two different type of inputs. **(Refer Program 2.7.4)** **(10 Marks)**

Q. 15 Find the output of the following:

```
#include<iostream.h>
#include<conio.h>
int main()
```

```
{
        for(int i=0;i<8;i++)
        {
            if(i%2==0)
            cout<<i+1<<endl;
            else if(i%3==0)
            continue;
            else if (i%5==0)
            break;
            cout<<"\n End of Program \n";
        }
        cout<<"\nEnd of program \n";
}
```

(Refer Program 2.7.5) **(5 Marks)**

❑❑❑

UNIT I

CHAPTER 3

Function, Arrays and Strings

Syllabus

Principles of object Oriented Programming

- **Unit Outcomes**
  - Write POP based C++ program using arrays to solve the given problem.
- **Topics and Sub-Topics**
  - Arrays in C++
  - Strings in C++
  - Scope Resolution Operator.

3.1 Functions

Q. 3.1.1 What is the difference between declaration and definition ? (Ref. Sec. 3.1) **(5 Marks)**

- In chapter 2, we had seen statements that are written inside a function. We were using these statements inside the main function.
- Function is a self-contained block of statements that performs a given task. The programs we were writing until now had only one function i.e. the main function. In this chapter, most of our programs will have two or more functions.
- The syntax of a function is given below :

```
return_type function_name (argument_list)
{
  -
  -
  statements;
  -
  -
}
```

Let us see the meaning of each of these terms used in the syntax

1. "return_type" is the data type of the data to be returned by the function. The main function we have always been writing has return type of void i.e. no data is returned. The return type of the data can be any of the data types like int, char, float etc.
2. The "function_name" can be anything as far as the rules of identifiers seen in chapter one are considered. For example "main" is the name of function we have been writing until now.
3. In the brackets we write the "argument_list". Argument list is a set of data types along with the identifiers to accept a set of data passed to the function. Until now we were not passing any data to the main function, and hence the brackets have always been empty. In the programs we will be writing in this chapter we will be having functions that accept the data from another functions.
4. Statements inside the function are the statements to be executed when that function is called. We can call a function using the function call operator seen in chapter one. We will see about all these in the subsequent programs.

- Some important points to be noted for using functions:
  1. A return_type is a data type of the value that the function returns to the caller function.
  2. A function that does not return any data is called as a void function i.e. its return type is void.
  3. Argument_list is a set of value that the function wants from the caller function.
  4. The parameters passed by the caller function are called as actual parameters.
  5. The parameters received by the called function are called as formal parameters.
  6. The number of actual and formal parameters must always be same.

7. The data types of the actual and formal parameters must also be same.
8. If a function is called before its definition, then a prototype declaration of that function must be written before calling the function as shown in program 3.1.1 and the figure associated with it i.e. Fig. 3.1.1.

➢ **Program 3.1.1 : Write a program to add two numbers using function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n1,n2,sum;
    cout<<"Enter two numbers:";
    cin>>n1>>n2;
    int add (int a, int b);
    sum=add(n1,n2);
    cout<<"Sum="<<sum;
    getch();
}
int add (int a, int b)
{
    int c;
    c=a+b;
    return c;
}
```

Output :

```
Enter two numbers:4
6
Sum=10
```

Explanation :

– In the above program we have formal parameters, actual parameters, prototype declaration, caller function and called function. First of all let us identify these elements as shown in the Fig. P. 3.1.1.

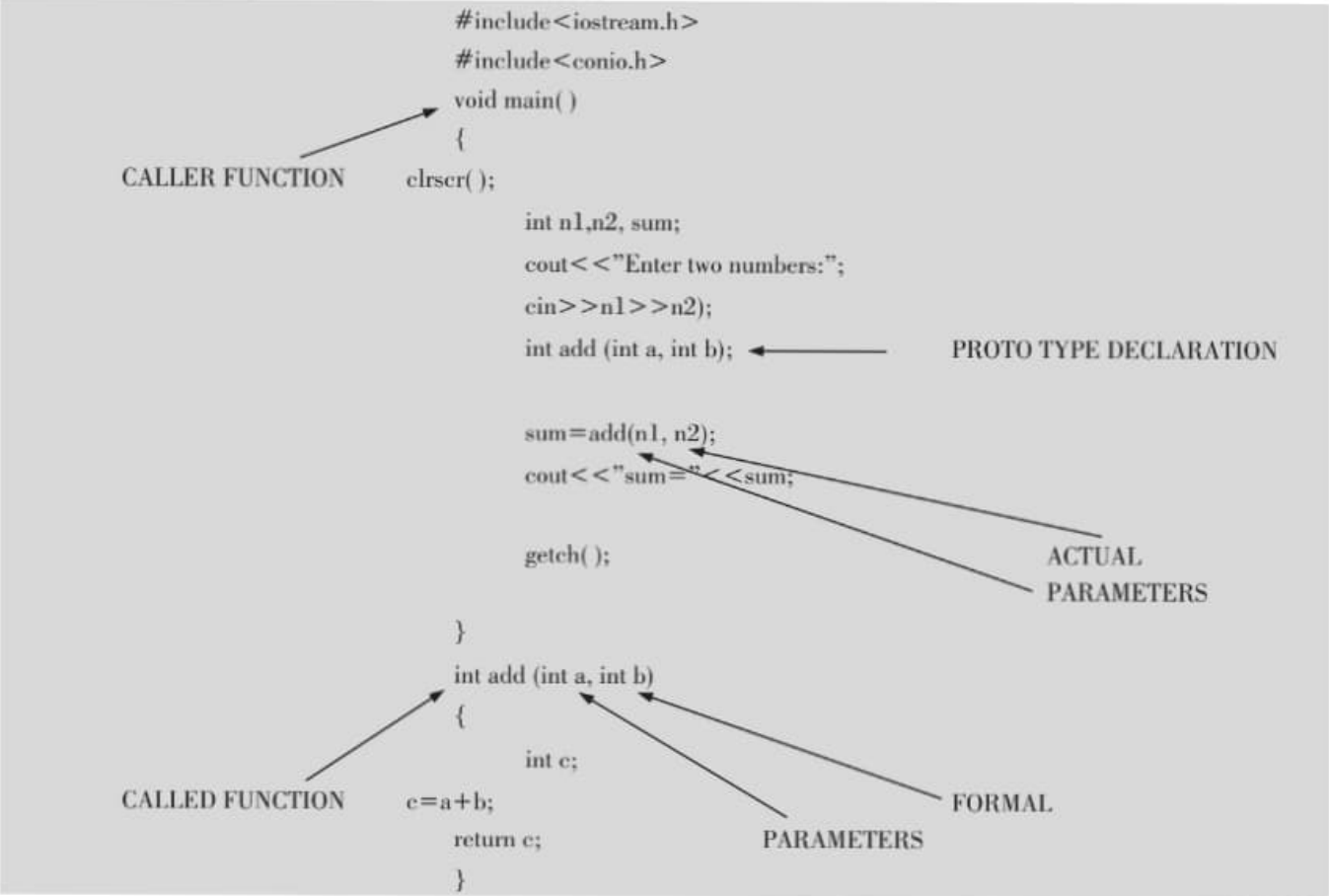


Fig. P. 3.1.1 : Different terminologies used in a function writing

- In this case the main function calls the add function, hence the main function is called as caller function, while the add function is called as the called function.
- The prototype declaration is just the header statement of the function definition. The function definition is the place where the entire function is written. For example in the above program the following is the add function definition.
- int add (int a, int b)

```
{
    int c;
    c=a+b;
    return c;
}
```

- If the function is called before it is defined then we need to declare the prototype of the function as shown in the above example. We first have a prototype declaration and then the function is called. The prototype declaration can be written without writing the identifiers of the arguments, for e.g. int add(int , int) in the above program.
- To call the function we need to simply write the name of the function. In this program the add function has an argument list of two integers i.e. 'a' and 'b'. Hence we need to pass two integer parameters while calling this function.
- The parameters are to be passed while calling the function. The parameters passed are called as actual parameters, while the ones that are received are called as the formal parameters. In this case the actual parameters i.e. "no1" and "no2" are passed to the "add" function.
- The formal parameters that are used to accept these actual parameters are 'a' and 'b'. As you can see and we have already discussed in the rules of writing functions that the number and data types to the actual and formal parameters must be same.
- Hence the data types of 'a' is same as that of 'no1' and 'b' is same as that of 'no2'. Also the number of actual parameters is two and that of the formal parameters is also two.
- Finally the return type of the "add" function is "int". Hence it must be returning an "int" type of data. The main function is a void function and hence doesn't return any parameter. The parameter to be returned is to be returned using the return statement as shown in the function i.e. return c.
- The parameter returned by a called function must be accepted by the caller function using the same type of data as the one returned by called function. Hence the variable "sum" in the main function is used to accept the value returned by the "add" function.
- The variable used to accept the returned value from the called function must be written on the left of the function call statement separated by an equal to (=) sign, for example, sum=add(no1,no2); in our program.

3.2 Programs Based on Function

➢ **Program 3.2.1 : Write a program to find the value of ^nC_r (Combination), using a function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,r,ncr;
    cout<<"Enter the values of n and r:";
    cin>>n>>r;
    int fact (int no);
    ncr=fact(n)/(fact(r)*fact(n-r));
    cout<<"nCr="<<ncr;
    getch();
}
int fact (int no)
{
    int i,ans;
    for(i=1,ans=1;i<=no;i++)
    {
        ans=ans*i;
    }
    return ans;
}
```

Output :

```
Enter the values of n and r:5
3
nCr=10
```

Explanation :

- Here is the real use of the function. in this program we have to calculate the factorial of many numbers viz n, r and (n-r). Without using functions we would have to write the same code for three times. But functions facilitate the concept of code reusability. We have written this function only once and we have called it thrice, hence saving the programmers work as well as memory space to store the program.
- Using the function to find factorial, we have found the value of ${}^{n}C_{r} = \frac{n!}{r!(n-r)!}$.

➢ **Program 3.2.2 : Write a program to find the value of ${}^{n}P_{r}$ (Permutation), using a function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,r,npr;
    cout<<"Enter the values of n and r:";
    cin>>n>>r;
    int fact (int no);
    npr=fact(n)/fact(n-r);
    cout<<"nPr="<<npr;
    getch();
}
int fact (int no)
{
    int i,ans;
    for(i=1,ans=1;i<=no;i++)
    {
        ans=ans*i;
    }
    return ans;
}
```

Output :

```
Enter the values of n and r:5
3
nPr=60
```

Explanation :

- Using the function to find factorial, we have found the value of ${}^{n}P_{r} = \frac{n!}{(n-r)!}$.

3.3 Recursive Functions

Q. 3.3.1 Explain recursion and write a recursive program for calculating factorial of a given number N. (Ref. Sec. 3.3 and Prog. 3.3.1) **(10 Marks)**

- A function that calls itself is called as a recursive function.
- A recursive function must definitely have a condition that exits from calling the function again.
- Hence there must be a condition that calls the function itself if that condition is true. If the condition is false then it will exit from the loop of calling itself again. The condition could also be implemented vice versa i.e. if the condition is false then the function calls itself else if the condition is true, it will exit from the loop calling itself again.
- The following programs are based on recursive functions.

➢ **Program 3.3.1 : Write a program to find the factorial of a number, using a recursive function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int no,factorial;
    cout<<"Enter a number:";
    cin>>no;
    int fact (int no);
    factorial=fact(no);
    cout<<"Factorial="<<factorial;
    getch();
}
int fact (int no)
{
    if(no==1)
```

```
    return 1;
    else
    return (no * fact (no-1));
}
```

Output :

```
Enter a number:5
Factorial=120
```

Explanation :

- The number "no" is passed to the function fact to find the factorial. If the value of the variable "no" received by the function is equal to 1, then the function returns 1, else it calls the same function itself again by passing the value of the variable "no" as "no"-1 and multiplying it by the variable "no". Let us understand this with an example. This is demonstrated in Fig. P. 3.3.1. The figure assumes that the number entered by the user is 5.
- As said, we have assumed that the number entered by the user is 5. Now the function "main" calls the function "fact" for the first time with passing the value of the variable "no" as 5.
- This is shown in the left part of the figure. The function checks the value of "no" to be equal to 1.
- Since the condition is not true, the control goes to the else part of the if-else statement. Here the value to be returned is the value of "no" multiplied with the value to be obtained by calling the same function again to find the factorial of 4 (as the factorial of 5 can be written as 5 multiplied with the factorial of 4).
- Hence, the same function is called again (in the Fig. P. 3.3.1, another function is shown just for understanding purpose), with the value being "no"-1 i.e. 5-1=4.
- This time the function again checks the value of "no" to be equal to 1. Since the condition is false, it again goes to the else part and calculates the value to be returned as "no" multiplied with the factorial of "no"-1 i.e. 3. Hence again calls the same function again.
- The third time when the function is called, the value received in the variable "no" is 3. Again in this case it is not equal to 1, hence it goes to the else part. The same is repeated i.e. calling the function again with the value being passed as 2. This time again since the value of the variable "no" is not equal to 1, it goes to the else part calling itself again with the value of "no" as 1.
- This time the received value of the variable "no" is equal to 1, hence the condition is true and the statements under "if" are executed. The statement says to return the value 1.

#include<<iostream.h>
#include<conio.h>
void main( )
{
clrscr( );
int no,factorial;
cout<<"Enter a number:";
getch( );
}
 	
120
	5
	
	int fact (int no)
	{
	if (no==1)
	return 1;
	else
	return (no*fact (no-1));
	}
	24
		4
		int fact (int no)
		{
		if (no==1)
		return 1;
		else
		return (no*fact (no-1));
		{
		6
			2
			int fact (int no)
			{
			if (no==1)
			return 1
			else
			return (no*fact (no-1));
			}
			2
				1
				int fact (int no)
				{
				if (no==1)
				return 1;
				else
				return (no + fact (no-1));
				}
				
				1

	called first time	called second time	called third time	called fifth time

Fig. P. 3.3.1 : Operation of a recursive function

- This value is received by the statement "return (no * fact (no-1))" in the case where the function was called in the fourth time of Fig. P. 3.3.1. This function now multiplies this received value i.e. 1 with the value of "no" in this case i.e. 2. Hence, the value 2 is returned back to the case where this function was called in the third time of the Fig. P. 3.3.1. In this case the value to be returned is again calculated as the value received multiplied by the value of the variable "no" in this case i.e. 2 multiplied by 3, returning the result i.e. 6.
- The value 6 is received by the previous case called function and multiplied by the value of the variable "no" in this case i.e. 6 multiplied by 4, returning the result i.e. 24. This value is received by the first called function.
- This function again calculates the multiplication of the received value with the value of the variable "no" in this case i.e. 24 multiplied by 5, returning the final result i.e. 120, to the "main" function. The "main" function displays this result.

Note :

1. The value returned and received in each case is shown in the Fig. P. 3.3.1.
2. For understanding the concept, in this figure we have shown the function named "fact" for five times. Actually, the function is only once. It is written only once in the program and also it is stored only once in the memory. But it is executed five times in this case, hence shown for five times.
3. The variables "no" and "ans" are different for the function every time the same function is called again i.e. different memory spaces are allocated to these variables every time the function is called.

Syllabus Topic : Arrays in C++

3.4 Arrays

➔ **(S-17)**

- It is a collection of multiple data of same data type. For example we can have an array of int type data or float type data etc.
- Remember, array can have data of same type only i.e. all elements of an array have to be of same type only. We cannot have an array of combination of different data types.
- We need to note two very important points about array
  1. The starting index i.e. index of the first element of an array is always zero.
  2. The index of last element is n-1, where n is the size of the array.
- An array has static memory allocation i.e. memory size once allocated for an array cannot be changed
- Syntax of declaring an array

  data_type array_name [array_size];

  where,

  data_type is the data type like int, float, double etc.

  array_name is the identifier i.e. the name of the variable

  array_size is an integer value which determines size of the array or memory locations to be allocated to an array.

  For e.g. int a[10];

  is an array of int type data named 'a' and can store upto 10 integers indexed from 0 to 9 i.e. 0 to n-1, where n is the size.
- The memory allocated to an array also depends on the data type i.e. the space required for each element.

  For e.g.: int a[10]; will require 20 byte memory locations, as each integer type data requires 2 byte memory locations; while float x[20] will require 80 byte memory locations, as each float type data requires 4 bytes of memory.
- To access an element of an array we have to use the array selection operator i.e.[].

  For e.g. if we want to access the 3rd element of an array named 'a', then we need to access it as a[2]. Remember 3rd element will be indexed 2, as the index of first element is 0. Hence to access an element we need to use the array name or identifier and write the index number in square brackets.

  Another e.g. to access the 9th element of an array 'x', we need to write x[8].
- If we need to initialize the elements of an array in the program itself they can be done in curly braces as shown in the example below :

  int a[ ] = {1,2,3,4,5,6,7,8,9,10};

  In this case, we need not specify the size in the square brackets.

  The values will be stored in the array as shown in the Fig. 3.4.1 each of these locations shown in the Fig. 3.4.1 are of two bytes i.e. to store one integer type of data.

a[0]	1
a[1]	2
a[2]	3
a[3]	4
a[4]	5
a[5]	6
a[6]	7
a[7]	8
a[8]	9
a[9]	10

Fig. 3.4.1 : Memory allocation for an array 'a' of 10 integer elements

- We will see some very basic programs of accepting and displaying the elements of an array and then move on to some programs that use these operations. Later we will also see some programs of arrays using functions.

➢ **Program 3.4.1 : Write a program to accept 'n' integers from user into an array and display them one in each line.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,i,a[100];
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    cout<<"The numbers entered are"<<endl;
    for(i=0;i<=n-1;i++)
    {
        cout<<a[i]<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of elements:5
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
The numbers entered are
1
2
3
4
5
```

Explanation :

- The array named 'a' is declared with a size of 100 elements. The size is taken to be 100, just to be on a higher side, so that whatever number user enters as the size of array, it will fit in this array. But for this program, the value of 'n' must not be entered greater than 100.
- The first "for" loop is to accept 'n' integers from the user in different index locations numbered from 0 to n-1. In the loop the value of 'i' is incremented after every iteration and hence the value entered by the user is stored into the next index element.
- The second "for" loop is to display these integers one after the other in different lines. Again in this loop the value of 'i' varies from 0 to n-1. Here also the value of 'i' is incremented after every iteration and hence the element displayed in every iteration is the consecutive next one in the array.

➢ **Program 3.4.2 : Write a program to evaluate the value of the standard deviation (s.d.) and display the result.**

$$\text{s.d.} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

where, $\bar{x}$ is the average of all the numbers.

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

```
    clrscr();
    int n,i,a[100],sum=0;
    float avg,sum1=0,sd;
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
        sum=sum+a[i];
    }
    avg=sum;
    avg=avg/n;
    for(i=0;i<=n-1;i++)
    {
        sum1 = sum1 + pow(a[i]-avg,2);
    }
    sd = sum1/n;
    sd = pow(sd,0.5);
    cout<<"The standard deviation is "<<sd;
    getch();
}
```

Output :

```
Enter the number of elements:5
Enter a value:3
Enter a value:4
Enter a value:4
Enter a value:5
Enter a value:4
The standard deviation is 0.632456
```

Explanation :

- The method to initialize and accept the elements from user is same as in previous programs.
- The average is calculated in the same manner so as to get the average in "float" type data format.
- Then the new sum is calculated to find the value of the term $(x_i - x)^2$ in the variable sum1.
- Finally the above calculated term is divided by "n" i.e. the total number of elements and its square root is calculated. This is the value of standard deviation which is then displayed.

➢ **Program 3.4.3 : Write a program to find the largest of 'n' numbers taken from user, using a function. The array of numbers must be passed to the function. The largest number must be found in the function and returned to the main function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,i,a[100],large;
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    int largest (int a[], int n);
    large=largest(a,n);
    cout<<"The largest number is "<<large;
    getch();
}
int largest (int a[], int n)
{
    int i,large;
    large=a[0];
    for(i=1;i<=n-1;i++)
    {
        if(large<a[i])
        large=a[i];
    }
    return large;
}
```

Output :

```
Enter the number of elements:5
Enter a value:2
Enter a value:33
Enter a value:12
Enter a value:24
Enter a value:11
The largest number is 33
```

Explanation :

- The method to initialize and accept the elements from user is same as in previous programs. Also the logic to find the largest number is same; the only difference was that this is done in a function. For this function, an array is passed.
- The method of passing the array is almost similar to that of any other parameter. Only that in the proto type declaration and in the definition, the square brackets are used to indicate the variable is an array.

➢ **Program 3.4.4 : Write a program to find and display the reverse of an array using a function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,i,a[100];
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    void reverse (int a[], int n);
    reverse(a,n);
    getch();
}
void reverse (int a[], int n)
{
    int i,rev[100];
    for(i=0;i<=n-1;i++)
    {
        rev[n-i-1]=a[i];
    }
    cout<<"The reverse of this array is:\n";
    for(i=0;i<=n-1;i++)
    {
        cout<<rev[i]<<endl;
    }
}
```

Output :

```
Enter the number of elements:5
Enter a value:1
Enter a value:23
Enter a value:43
Enter a value:12
Enter a value:22
The reverse of this array is:
22
12
43
23
1
```

Explanation :

- We have passed the array in the same manner as in the previous program. Here the array is reversed and put into another array. The logic to reverse is very simple in this case, the first element of the array "a" is copied into the last element of array "rev". Initially, the value of "i" is 0, hence the value of the expression n-i-1, will be n-1 i.e. the last element. Hence the element number 0 of the array "a" is copied into the n-1 element of the array "rev".
- When the value of "i" is incremented, the element number 1 of the array "a" is copied into the n-i-1 i.e. second last element of the array "rev". This continues until the value of I, reaches "n".

➢ **Program 3.4.5 : Write a program to sort numbers in ascending order. OR Write a program to implement bubble sorting algorithm for sorting numbers in ascending order.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,i,a[100],x,index;
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    void ascend (int a[], int n);
    ascend(a,n);
    getch();
}
void ascend (int a[], int n)
{
    int i,j,temp;
    for(i=0;i<=n-2;i++)
    {
        for(j=0;j<=n-2;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    cout<<"After sorting\n";
    for(i=0;i<=n-1;i++)
    {
        cout<<a[i]<<endl;
    }
}
```

Output :

```
Enter the number of elements:5
Enter a value:4
Enter a value:6
Enter a value:1
Enter a value:85
Enter a value:9
After sorting
1
4
6
9
85
```

Explanation :

- The method of accepting and passing the elements of the array is similar to the previous programs. Now let us see the bubble sorting algorithm in detail.
- Bubble sorting algorithm says that compare the consecutive elements from the beginning of array till the end. Wherever the proper sorting is not found, swap the numbers. And this entire set of comparisons is to be repeated for n-1 times. Let us take the example of numbers as given in the output of the program to understand this logic.
- The first two elements are compared i.e. 4 and 6, since they are already sorted i.e. 4<6, no swapping takes place. The next comparison is the next two elements i.e. 6 and 1; now these are not sorted i.e. 6 > 1, hence these are swapped. This continues until the last two elements are compared as shown in Fig. P. 3.4.2.
- You will notice in Fig. P. 3.4.2, after all the comparisons done in the first iteration; the largest number has reached to the last position. But the numbers are still not sorted properly. To sort these numbers we need to repeat the above process for n-1 times.

Output :

Note : Since the "j" and the "j+1" elements are compared the value of "j" must go maximum upto n-2. When the value of "j" is n-2, we will be comparing the n-2 and n-1 i.e. the last two elements. Also since the value of "i" is starting from 0, we have to go upto n-2, as we have to perform n-1 comparisons

	Initial values	After first comparision	After second comparision	After third comparision	After fourth comparision
Iteration 1		4>6, No, hence no swap	6>1, Yes, hence swap	6>85, No, hence no swap	85>9, Yes, hence swap
	4	4	4	4	4
	6	6	1	1	1
	1	1	6	6	6
	85	85	85	85	9
	9	9	9	9	85
Iteration 2		4>1, Yes, hence swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85
Iteration 3		1>4, No, hence no swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85
Iteration 4		1>4, No, hence no swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85

Fig. P. 3.4.2 : Bubble sorting example

- You may also notice in the Fig. P. 3.4.2, that the sorting is already completed after the second iteration. The remaining two iterations do no swapping. But, we have to consider the worst case condition i.e. if all the numbers entered by the user were in exactly reverse order. In this case we would require all the four iterations. This is explained in the Fig. P. 3.4.3.
- Here, you will notice that till the last iteration the swapping is required. The worst case condition is one in which the numbers are exactly in the reverse order as expected to be. You will notice in this case the numbers are in descending order and they are to be sorted in ascending order.

	Initial values	**After first comparision**	**After second comparision**	**After third comparision**	**After fourth comparision**
Iteration 1		5>4, Yes, hence swap	5>3, Yes, hence swap	5>2, Yes, hence swap	5>1, Yes, hence swap
	5	4	4	4	4
	4	5	3	3	3
	3	3	5	2	2
	2	2	2	5	1
	1	1	1	1	5
Iteration 2		4>3, Yes, hence swap	4>2, Yes, hence swap	4>1, Yes, hence swap	4>5, No, hence no swap
	4	3	3	3	3
	3	4	2	2	2
	2	2	4	1	1
	1	1	1	4	4
	5	5	5	5	5
Iteration 3		3>2, Yes, hence swap	3>1, Yes, hence swap	3>4, No, hence no swap	4>5, No, hence no swap
	3	2	2	2	2
	2	3	1	1	1
	1	1	3	3	3
	4	4	4	4	4
	5	5	5	5	5
Iteration 4		2>1, Yes, hence swap	2>3, No, hence no swap	3>4, No, hence no swap	4>5, No, hence no swap
	2	1	1	1	1
	1	2	2	2	2
	3	3	3	3	3
	4	4	4	4	4
	5	5	5	5	5

Fig. P. 3.4.3 : Worst case condition of sorting numbers using Buble sorting

➢ **Program 3.4.6 : Write a program to insert an element at location of array.** **S-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,i,a[100],x,insert,index;
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    cout<<"Enter the element to be inserted and the index:";
    cin>>insert>>index;
    for(i=n;i>=index+1;i--)
    {
        a[i]=a[i-1];
    }
    a[index]=insert;
    n++;
    cout<<"After inserting:\n";
    for(i=0;i<=n-1;i++)
    {
        cout<<a[i]<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of elements:4
Enter a value:1
Enter a value:2
Enter a value:4
Enter a value:5
Enter the element to be inserted and the index:3
2
After inserting:
1
2
3
4
```

3.5 Multi-dimensional Arrays

Q. 3.5.1 Explain multi-dimensional arrays. Write a program to add two matrices of size m × n (Ref. Sec. 3.5 and Prog. 3.5.2) **(7 Marks)**

- Multi dimensional arrays are used to store data that requires multiple references for e.g. a matrix requires two references namely row number and column number. Hence, "matrix" is a best example of two dimensional arrays.
- We can also have an array of more than two dimensions. But, we will not require an array of more than two dimensions as per our syllabus.
- If an two dimensional array i.e. matrix is to be declared of a size 3 × 3, then the declaration statement will be as below:

  int a[3][3];

  The size of the array will be 3 × 3, but the indices will be from 0 to 2 in both the rows and columns. The representation of the same can be as shown in the Fig. 3.5.1.

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]

Fig. 3.5.1 : Arrangement of the elements in a two dimensional array

- As seen in the Fig. 3.5.1, the first brackets indicate the row number while the second brackets indicate the column number. To understand how a matrix element can be accessed Fig. 3.5.1 explains it all, but the elements of this matrix are stored in a different manner in the memory as shown in Fig. 3.5.2. This figure assumes the values in matrix as 1, 2, 3 ... 9.

	Memory
a[0][0]	1
a[1][0]	2
a[2][0]	3
a[0][1]	4
a[1][1]	5
a[2][1]	6
a[0][2]	7
a[1][2]	8
a[2][2]	9

Fig. 3.5.2 : Arrangement of the elements in a two dimensional array

- As shown in Fig. 3.5.2, the elements are stored in a sequential manner with all the elements of a row together one below the other, and then the next row and so on.
- The method of accepting elements of an m × n matrix and displaying it in natural form is shown in program 3.5.1 (Note: m is the number of rows and n is the number of columns). The values of m and n must be taken from user.
- Natural form display of a 2 × 4 matrix is as shown below.

  1 2 3 4

  5 6 7 8

➢ **Program 3.5.1 : Write a program to accept an m × n matrix and display it in natural form.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int m,n,i,j,a[10][10];
    cout<<"Enter the number of rows and columns:";
    cin>>m>>n;
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<"Enter a value:";
            cin>>a[i][j];
        }
    }
    cout<<"The entered matrix is:\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<a[i][j]<<"\t";
        }
        cout<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of rows and columns:3
3
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
Enter a value:7
Enter a value:8
Enter a value:9
The entered matrix is:
1    2    3
4    5    6
7    8    9
```

Explanation :

- First, the user is asked for number of rows and columns. To accept the elements of the matrix, we need a nested "for" loop as shown. The outer loop has "i" as a counter for the row number and hence is counted from 0 to m-1; while the inner loop has "j" as a counter for the column number and hence is counted from 0 to n-1.

- The displaying of the elements of the matrix also requires a nested for loop as required for accepting the matrix elements from the user. The only additional statement is the "endl" manipulator, for every row to be displayed in a new line.

➢ **Program 3.5.2 : Write a program to add two matrices of size m × n.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int m,n,i,j,a[10][10],b[10][10],c[10][10];
    cout<<"Enter the number of rows and columns:";
    cin>>m>>n;
    cout<<"Enter the elements of Matrix 1\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<"Enter a value:";
            cin>>a[i][j];
        }
    }
    cout<<"Enter the elements of Matrix 2\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<"Enter a value:";
            cin>>b[i][j];
        }
    }
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            c[i][j]=a[i][j]+b[i][j];
        }
    }
    cout<<"The sum of two matrices is:\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<c[i][j]<<"\t";
        }
        cout<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of rows and columns:2
3
Enter the elements of Matrix 1
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
Enter the elements of Matrix 2
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
The sum of two matrices is:
2    4    6
8    10   12
```

Explanation :

- The method to accept the elements of the matrix is same as the previous program.
- The corresponding elements of the matrices "a" and "b" are added together and the result is put into the matrix "c". This "c" matrix is then displayed in natural form.

➢ **Program 3.5.3 : Write a program to find the transpose of a matrix of size m × n. Use a function transpose to do this operation.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int m,n,i,j,a[10][10];
    cout<<"Enter the number of rows and columns:";
    cin>>m>>n;
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<"Enter a value:";
            cin>>a[i][j];
        }
    }
    cout<<"The original Matrix is:\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<a[i][j]<<"\t";
        }
        cout<<endl;
    }
    void transpose (int a[10][10],int m, int n);
    transpose(a,m,n);
    getch();
}
void transpose (int a[10][10], int m, int n)
{
    int b[10][10],i,j;
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            b[j][i]=a[i][j];
        }
    }
    cout<<"The transpose of this matrix is:\n";
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=m-1;j++)
        {
            cout<<b[i][j]<<"\t";
        }
        cout<<endl;
    }
}
```

Output :

```
Enter the number of rows and columns:3
2
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
The original Matrix is:
1    2
3    4
5    6
The transpose of this matrix is:
1    3    5
2    4    6
```

Explanation :

- The method to accept the elements of the matrix is same as the previous program.
- The transpose of a matrix is shown below.

  If matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$, the transpose of this matrix will be $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

  i.e. the column 1 of matrix A, will be the row 1 of its transpose; similarly column 2 will become row 2 and so on.
- The transpose is found by just placing the a[i][j] term in b[j][i], using the same type of nested "for" loop.
- Also while displaying the transpose of the matrix i.e. the matrix "b", we need to take care that the number of rows and columns are reversed. Hence, the value of outer "for" loop i.e. "i" varies from 0 to n-1, while that of the inner loop i.e. "j" varies from 0 to m-1.

➢ **Program 3.5.4 : Write a program to multiply two matrices using a function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int m,n,p,i,j,a[10][10],b[10][10];
 cout<<"Enter the number of rows and columns of matrix 1:";
    cin>>m>>n;
    cout<<"Enter the values of matrix 1\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<"Enter a value:";
            cin>>a[i][j];
        }
    }
    cout<<"The number of rows for matrix 2 will be
"<<n<<endl;
    cout<<"Enter the columns of matrix 2:";
    cin>>p;
    cout<<"Enter the elements of matrix 2:\n";
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=p-1;j++)
        {
            cout<<"Enter a value:";
            cin>>b[i][j];
        }
    }
  void MatMul (int a[10][10],int b[10][10], int m, int n, int p);
    MatMul(a,b,m,n,p);
    getch();
}
void MatMul (int a[10][10], int b[10][10], int m, int n, int p)
{
    int i,j,k,c[10][10];
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=p-1;j++)
        {
            c[i][j]=0;
            for(k=0;k<=n-1;k++)
            {
                c[i][j]=c[i][j]+a[i][k]*b[k][j];
            }
        }
    }
    cout<<"The resultant matrix is:\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=p-1;j++)
        {
            cout<<c[i][j]<<"\t";
        }
```

```
        cout<<endl;
    }
}
```

Output :

```
Enter the number of rows and columns of matrix 1:3
2
Enter the values of matrix 1
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
The number of rows for matrix 2 will be 2
Enter the columns of matrix 2:3
Enter the elements of matrix 2:
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
The resultant matrix is:
9    12    15
19   26    33
29   40    51
```

Explanation :

- The method to accept the elements of the matrices is same as the previous program.
- To multiply two matrices their sizes must be m × n and n × p respectively. Hence the number of rows of the matrix 2 is kept same as the number of columns of matrix 1. The size of the resultant matrix will be m × p.
- To calculate the resultant matrix the two matrices and their sizes are passed to the function.
- The function has three level of nested "for" loops. The outer two loops are used to select an element of the resultant matrix whose value is to be calculated. Initially the value is made zero. Then, the corresponding elements of matrix 1 and 2 are multiplied and added to get the terms in an element of the resultant matrix. The terms required to get an element of the resultant matrix are shown in the Fig. P. 3.5.4 considering the example of the matrices to be multiplied to be of sizes m × n and n × p respectively.

$$\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{bmatrix} \times \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{bmatrix} = \begin{bmatrix} (a_{0,0}\times b_{0,0}) + (a_{0,1}\times b_{1,0}) & (a_{0,0}\times b_{0,1}) + (a_{0,1}\times b_{1,1}) & (a_{0,0}\times b_{0,2}) + (a_{0,1}\times b_{1,2}) \\ (a_{1,0}\times b_{0,0}) + (a_{1,1}\times b_{1,0}) & (a_{1,0}\times b_{0,1}) + (a_{1,1}\times b_{1,1}) & (a_{1,0}\times b_{0,2}) + (a_{1,1}\times b_{1,2}) \\ (a_{2,0}\times b_{0,0}) + (a_{2,1}\times b_{1,0}) & (a_{2,0}\times b_{0,1}) + (a_{2,1}\times b_{1,1}) & (a_{2,0}\times b_{0,2}) + (a_{2,1}\times b_{1,2}) \end{bmatrix}$$

Fig. P. 3.5.4 : Multiplication of matrices

- Hence, in the Fig. P. 3.5.4 you will notice that to calculate each term of the resultant matrix, we need to add many product terms. This is what is implemented by the three level nested loops. Initially the element of the resultant matrix is made zero, then each of the products are added to this element and hence obtain the entire expression value.
- To find each element, you will notice that the column number of matrix 1 is changed while the row number for matrix 2. For example in the first term, $(a_{0,0}\times b_{0,0}) + (a_{0,1}\times b_{1,0})$, as seen the column number of matrix 1 is 0 for the first term and 1 for the second term, while the row number of matrix 2 is 0 for the first term and 1 for the second term. Hence, the third nested loop varies the value of k from 0 to n-1 i.e. the number of columns for matrix 1 or number of rows for matrix 2.

Syllabus Topic : Strings in C++

3.6 Strings

Q. 3.6.1 What is a string? Explain with a program example.
(Ref. Sec. 3.6 and Prog. 3.6.1) **(4Marks)**

- We have seen array of integers. Similarly we can also have array of float numbers. But when we make an array of characters i.e. "char" type data, it is termed as a string.
- A string i.e. an array of characters is normally terminated with a null character to indicate the termination.
- The null character is just used as a reference to indicate the termination of a string. The ASCII value of the null character is 0 (zero). It is written as '\0'.
- Hence we can say "an array of characters terminated with a null character is called as string".
- An example declaration of a string can be char a[100];
- If a string "INDIA" is to be stored in this array then it will consume 6 spaces or 6 memory bytes (as we know each character requires 1 byte space in memory). This string will be stored as shown below;

a[0]	'I'
a[1]	'N'
a[2]	'D'
a[3]	'I'
a[4]	'A'
a[5]	'\0'

Note : Whenever the compiler reads a string it inserts a null character at the end. If a string is read using the "cin" command, the first blank space is treated as a null character. In this case we will not be able to use "cin" to read a sentence, we can read only a word. Hence if we want to read a sentence, we cannot use "cin", instead we have to use gets() function studied in chapter 1. To store a string of 5 characters we need a array size of atleast 6 characters i.e. 5 characters and 1 null character

- In the following sections we will see some programs related to strings. Also we will study some of the functions available in the header file named "string.h"

➢ **Program 3.6.1 : Write a program to accept and display a string.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    clrscr();
    char a[100];
    cout<<"Enter a string\n";
    gets(a);
    cout<<"The entered string is: "<<a;
    getch();
}
```

Output :

```
Enter a string
How are you?
The entered string is: How are you?
```

Explanation :

- To accept the string from user, we have used the function "gets()". This function as seen in chapter 1, is in the header file "stdio.h". Hence we have included this file. When using the function "gets()", we need to write the name of the string i.e. character array in the brackets as seen in the program.
- The method to display a string is the same as any other data type i.e. "cout".

3.7 The Header File string.h

Q. 3.7.1 Explain the functions supported by string.h file.
(Ref. Secs. 3.7, 3.7.1, 3.7.2, 3.7.3, 3.7.4)
(8 Marks)

- This header file has many functions that perform certain operations on strings.

- These functions make our programming work much simpler. We will see some of the functions supported by this header file in the following sub- sections.

3.7.1 strlen() function

- This function returns an integer value that is the length of the string passed to the function.
- When returning the length of the string it does not consider the space required for null character. Hence it returns the exact length of the string neglecting the space required for null character.
- Let us see a program example of this function to find the length of a string entered by the user.

➢ **Program 3.7.1 : Write a program to accept a string and display its length.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr();
    int l;
    char a[100];
    cout<<"Enter a string\n";
    gets(a);
    l=strlen(a);
    cout<<"The length of the entered string is: "<<l;
    getch();
}
```

Output :

```
Enter a string
Hello
The length of the entered string is: 5
```

Explanation :

- To accept the string from user, we have used the function "gets()". This function is in the header file "stdio.h".
- The length of the string is found using the function strlen(), which is in the header file string.h.

3.7.2 strcpy() function

- This function copies the second string into the first string passed to the function.
- The second string remains unchanged. Only the first string is changed and gets a copy of the second string. For e.g. strcpy(str1,str2), will copy the string "str2" into the string "str1".
- Let us see a simple program example of this function to copy a user entered string into another string.

➢ **Program 3.7.2 : Write a program to accept a string, copy it into another string and display this new string.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr();
    char a[100],b[100];
    cout<<"Enter a string\n";
    gets(a);
    strcpy(b,a);
    cout<<"The new string is "<<b;
    getch();
}
```

Output:

```
Enter a string
Hello, how are you?
The new string is Hello, how are you?
```

Explanation :

- In this program, we accept the string from user in the variable "a" and then it is copied in the string variable "b". This is then displayed on the screen.
- This operation of coping the string "a" in the string "b" is done by the function strcpy(), which is available in the header file "string.h".

3.7.3 strcmp() function

- This function compares the two string variables passed to it. It returns an integer value equal to
  - 0 (zero), if the two strings are equal.
  - Negative value, if the first string is smaller than the second string.
  - Positive value, if the first string is greater than the second string.
- Both the strings remain unchanged.
- The string is smaller means its alphabetical sequence is smaller. For example, "Hello" is lesser than "Hi"; because the first character "H" is same, but the second character "e" is smaller than "i". "e" is smaller than "i" because "e" comes before "i" in the alphabets i.e. A, B, C, ….Z. The function compares the ASCII value of the characters.
- Let us see a simple program example of this function to compare two a user entered strings.

➢ **Program 3.7.3 : Write a program to accept two strings, compare them and display if they are equal or not. If they are not equal display the one which is greater.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr();
    char a[100],b[100];
    cout<<"Enter two strings:\n";
    gets(a);
    gets(b);
    if(strcmp(a,b)==0)
    cout<<"The strings are equal ";
    else if(strcmp(a,b)>0)
    cout<<a<<" string is greater";
    else
    cout<<b<<" string is greater";
    getch();
}
```

Output :

```
Enter two strings:
Hello
Hi
Hi string is greater
```

Explanation :

- In this program, we accept two strings from user in the variables "a" and "b". These are compared using the strcmp() function as already discussed.
- The result of their equality is then displayed on the screen.

3.7.4 strcat() function

- This function concatenates (joins) the two string variables passed to it. It returns a string of te combination of the two in the first string variable.
- Let us see a simple program example of this function to compare two a user entered strings.

➢ **Program 3.7.4 : Write a program to accept two strings, join them and display the result.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr();
    char a[100],b[100];
    cout<<"Enter two strings:\n";
    gets(a);
    gets(b);
    strcat(a,b);
    cout<<"The concatenated string is "<<a;
    getch();
}
```

Output :

```
Enter two strings:
Harish
Narula
The concatenated string is HarishNarula
```

Explanation :

- In this program, we accept two strings from user in the variables "a" and "b". These are concatenated using the strcat() function as already discussed.
- The result of their concatenation is then displayed on the screen.

3.8 Programs Based on Strings

➢ **Program 3.8.1 : Write a program to accept a string and find its length without using the string header file.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    clrscr();
    char a[100];
    int len=0;
    cout<<"Enter a string:\n";
    gets(a);
    while(a[len]!='\0')
    {
        len++;
    }
    cout<<"The length of this string is "<<len<<"
characters.";
    getch();
}
```

Output :

```
Enter a string:
Hello
The length of this string is 5 characters.
```

Explanation :

- In this program, we accept a string from user in the variable "a".
- The length of this string is now calculated using a "for" loop, wherein a value "len" is incremented until we come across a null character i.e. "\0".
- This is a method to find the length of a string without using the functions of the header file "string.h".

➢ **Program 3.8.2 : Write a program to reverse a user entered string.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr();
    int l;
    char a[100];
    cout<<"Enter a string\n";
    gets(a);
    strrev(a);
    cout<<"The reversed string is: "<<a;
    getch();
}
```

Output :

```
Enter a string
Hello
The reversed string is: olleH
```

Explanation :

- Another function in the header file "string.h" is strrev(). This function reverses the string passed to it, as seen in the above program.

➢ **Program 3.8.3 : Write a program to reverse a user entered string (Do not use the string header file). Write a function to do this work of reversing and displaying the string**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n=0;
    char a[100];
    cout<<"Enter a string:";
    gets(a);
    while(a[n]!='\0')
    {
        n++;
    }
    void reverse (char a[100], int n);
    reverse(a,n);
    getch();
}
void reverse (char a[100], int n)
{
    int i;
    char temp;
    for(i=0;i<=(n-1)/2;i++)
    {
        temp=a[n-i-1];
        a[n-i-1]=a[i];
        a[i]=temp;
    }
    cout<<"The reverse of this string is: ";
    cout<<a;
}
```

Output :

```
Enter a string:Hello
The reverse of this string is: olleH
```

Explanation :

- The reversing of a string is done in the same way as it was done for an array of integers in program 3.4.10.

➢ **Program 3.8.4 : Write a program to check whether the entered string is palindrome or not (Do not use the string header file).**

Note : Palindrome is a string which is same when read from either of the sides. For e.g. nitin, madam, malayalam etc.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n=0,i;
    char a[100],rev[100];
    cout<<"Enter a string:";
    gets(a);
    while(a[n]!='\0')
    {
        n++;
    }
    for(i=0;i<=(n-1);i++)
    {
        rev[n-i-1]=a[i];
    }
    for(i=0;i<=n-1;i++)
    {
        if(a[i]!=rev[i])
        break;
    }
    if(i==n)
    cout<<"The string is palindrome.";
    else
    cout<<"The string is not palindrome.";
    getch();
}
```

Output :

```
Enter a string:nitin
The string is palindrome.
```

Explanation :

- The method of accepting and reversing the string is as seen in the previous program i.e. program 3.8.5. The only difference is that the reverse of the string is stored in another array of characters i.e. "rev".
- Now the reversed string is compared with the original string. If any element is not found same then the break statement transfers the control outside the "for" loop. And in this case the value of "i" and "n" will not be same, hence it will display that the string is not palindrome.
- If all the elements of the original and the reverse string are same then the control will be transferred outside the "for" loop when the condition is not true i.e. i<=n-1. In this case the string read from both sides is same i.e. it is a palindrome. Hence it displays that the string entered is palindrome if the value of "i" is equal to "n" after the control is transferred outside the "for" loop.

Syllabus Topic : Scope Resolution Operator

3.9 Storage Classes and use of Scope Resolution Operator

Q. 3.9.1 Explain storage class with one programming example. (Ref. Sec. 3.9) **(10 Marks)**

- The different locations in the computer where we can store data and their accessibility, initial values etc. vary based on the way they are declared. These different ways are termed as different storage classes.
- In C / C++, we have four storage classes, namely

1. Automatic
2. Register
3. Static
4. External or Global

- Let us see these storage classes one by one

➜ **1. Automatic storage class**

- In this case data is stored in memory
- The initial value of such a variable is garbage.
- The scope of the variable is local i.e. limited to the function in which it is defined.
- The life of such variables is till the control remains in the particular function where it is defined.
- For e.g.:

  int i; or auto int i;

- In all our programs till now we have been using the automatic storage class for our variables.

➜ **2. Register storage class**

- In this case data is stored in CPU register
- The initial value of such a variable is garbage.
- The scope of the variable is local i.e. limited to the function in which it is defined.
- The life of such variables is till the control remains in the particular function where it is defined.

  For e.g.:

  register int i;

- In this case the data is stored in a small memory inside the processor called as its registers.
- The advantage of such storage class is that since the data is in the processor itself, its access and operations on such data is faster.
- There is a limitation on the size of the data that can be declared to be register storage class. The data should be such that it doesn't require more than 4 bytes. Hence double and long double data types cannot be declared as register.
- Also there is a limitation on the maximum number of variables in a function that can be of register class. The limitation is that a maximum of 3 register class variable can be declared in a function.

➜ **3. Static storage class**

- In this case data is stored in memory.
- The initial value of such a variable is zero.
- The scope of the variable is local i.e. limited to the function in which it is defined.
- The life of such variables is till the program is alive.

  For e.g.:

  static int i;

- If a variable is declared static, its value remains unchanged even if the function execution is completed.
- When the execution to that function returns, the previous value is retained.
- Thus it says the initialization is only once. If you have an initialization statement of a static member, it will be executed only once i.e. for the first time when this function is called.
- Let us see program examples to differentiate between the static and auto member variables.

➢ **Program 3.9.1 : Write a program to initialize an automatic variable and increment it in the function. Call this function thrice and print the value of the variable every time after incrementing.**

```
#include<iostream.h>
#include<conio.h>
void incr()
{
    int i=0;
    i++;
    cout<<i<<endl;
}
int main()
{
    incr();
    incr();
    incr();
    getch();
}
```

Output :

```
1
1
1
```

Explanation :

- Every time the function incr() is called, the value of the variable 'i' is initialized to zero, then incremented and displayed on the screen. This is because, 'i' is a automatic variable.

➢ **Program 3.9.2 : Write a program to initialize a static variable and increment it in the function. Call this function thrice and print the value of the variable every time after incrementing.**

```
#include<iostream.h>
#include<conio.h>
void incr()
{
    static int i=0;
    i++;
    cout<<i<<endl;
}
int main()
{
    incr();
    incr();
    incr();
    getch();
}
```

Output :

```
1
2
3
```

Explanation :

- In this case the value of 'i' is initialized to zero only once i.e. for the first time when the function incr() is called. This is because, the variable 'i' is a static variable.
- Every time the function incr() is called, the value of the variable 'i' is incremented and displayed on the screen.
- Hence the value increments with respect to the previous value.

➜ **4. External or Global storage class**

- In this case data is stored in memory
- The initial value of such a variable is zero.
- The scope of the variable is global i.e. it is accessible from anywhere in the program.
- The life of such variables is till the program is alive.
- Let us see some program examples to understand the global variables.

➢ **Program 3.9.3 : Write a program to demonstrate the access of global variable.**

```
#include<iostream.h>
#include<conio.h>
int a=5;
void main()
{
    int a=10;
    cout<<a<<endl;
    cout<<::a<<endl;
    a=::a+a;
    cout<<a<<endl;
    cout<<::a<<endl;
    ::a=a;
    cout<<a<<endl;
    cout<<::a<<endl;
    getch();
}
```

Output :

```
10
5
15
5
15
15
```

Explanation :

- The global variable as seen is accessed with the help of scope resolution operator "::". Although the name of the global and local variable is same i.e. 'a', we can access them separately.
- Whenever the scope resolution operator is associated with the variable 'a', it indicates that the global variable is to be accessed; else the local variable is considered.

➢ **Program 3.9.4 : Find the output of the following program.**

```
#include<iostream.h>
#include<conio.h>
int x;
void f1()
{
    ++x;
}
void main()
{
    int x = 10;
    f1();
    x = ::x + 10;
    cout<<x<<" "<<::x<<endl;
    getch();
}
```

Output :

```
11 1
```

➢ **Program 3.9.5 : Find the output of the following program.**

```
#include<iostream.h>
#include<conio.h>
void f1()
{
    extern int n3;
    static int n1;
    int n2 = 20;
    n1 = n1 + 10;
    n2 = n1 + n2;
    n3 = n1 + n2;
    cout<<n1<<" "<<n2<<" "<<n3<<endl;
}
int n3;
void main()
{
    register int i;
    for (i = 1; i <=3; i++) f1();
    getch();
}
```

Output :

```
10 30 40
20 40 60
30 50 80
```

3.10 Exam Pack (MSBTE and Review Questions)

Q. 1 What is the difference between declaration and definition ? **(Refer Section 3.1)** **(5 Marks)**

Q. 2 Explain recursion and write a recursive program for calculating factorial of a given number N.
(Refer Section 3.3 and Program 3.3.1) (10 Marks)

Q. 3 Write a program to find factorial value of given integer number using recursion.
(Refer Program 3.3.1) **(5 Marks)**

☞ **Syllabus Topic : Arrays in C++**

Q. 4 Explain multi- dimensional arrays. Write a program to add two matrices of size m × n
(Refer Section 3.5 and Program 3.5.2) **(7 Marks)**

Q. 5 Write a program to insert an element at location of array. **(Refer Program 3.4.6)** **(4 Marks)**

☞ **Syllabus Topic : Strings in C++**

Q. 6 Write a program in C++ to transpose a matrix. **(Refer Program 3.5.3)** **(10 Marks)**

Q. 7 What is a string? Explain with a program example.
(Refer Section 3.6 and Program 3.6.1) **(4 Marks)**

Q. 8 Explain the functions supported by string.h file. **(Refer Section 3.7)** **(8 Marks)**

Q. 9 Write a program to find whether a given string is a palindrome or not.
(Refer Program 3.8.4) **(10 Marks)**

☞ **Syllabus Topic : Scope Resolution Operator**

Q. 10 Select the correct answer **(4 Marks)**

1. # define square (A) A * A
   a = square (2 + 3)
   Value of a will be
   (i) 25 (ii) 13
   (iii) 11 (iv) 10

Solution :

(ii) This function square() can accept only one variable, so it finds the square of 2 and 3 separately and then adds the result. Hence $2^2 + 3^2$ gives 13.

2. Two static variables are declared in one file with the same name. One of them is global, another is local to a function. Then
   (i) Compiler will give syntax error
   (ii) Local static variable will be ignored
   (iii) Global variable will be ignored
   (iv) None of the above.

Solution :

(iv) No error will be given. We have seen we can access the global and local variable separately i.e. using scope resolution operator for the global variable as shown in program 3.9.3.

Q. 11 Explain storage class with one programming example. **(Refer Section 3.9)** **(10 Marks)**

Q. 12 Find the output of the following program:

```
void f()
{
        extern int n3;
        static int n1;
        int n2 = 20;
        n1 = n1 + 10;
        n2 = n1 + n2;
        n3 = n1 + n2;
        cout<<n1<<" "<<n2<<" "<<n3<<endl;
}
int n3;
void main()
{
        register int i;
        for (i = 1; i <=3; i++) f1();
}
```

(Refer Program 3.9.5) **(5 Marks)**

❑❑❑

UNIT I

CHAPTER 4

Structures and Unions

Syllabus

Principles of Object Oriented Programming

- **Unit Outcomes**
  - Use the structure in C++ program for solving the given problem
- **Topics and Sub-Topics**
  - Structures in C++

Syllabus Topic : Structures in C++

4.1 Structures

(W-14,S-14, W-15, S-17, W-17)

Q. 4.1.1	Define a structure with it's syntax. (Ref. Sec. 4.1)	**(W-14, 2 Marks)**
Q. 4.1.2	Define structure. Give syntax of defining structure. (Ref. Sec. 4.1)	**(W-15, 4 Marks)**
Q. 4.1.3	Give syntax and example of defining structure and declaring structure variables. (Ref. Sec. 4.1)	**(S-17, 4 Marks)**
Q. 4.1.4	What is structure ? How user declare structure ? Give example. (Ref. Sec. 4.1)	**(W-17, 4 Marks)**

- Structure is a collection of multiple data elements that can be of different data types.
- Array is a collection of data items of same type, while structure can have data items of different type.
- The memory space required to store one variable of structure is equal to the memory space required by all the elements independently to be stored in memory.
- Let us see the syntax of a structure declaration and declaration of a variable of the structure type.

Syntax of structure declaration :

```
struct structure_name
{
    data_type variable_name;
    data_type variable name;
    -
    -
};
```

- For example a structure to store the some information like name, roll number and fees of a student in college can be declared as shown below.

```
struct student
{
    char name[20];
    int roll_no;
    float fees;
};
```

Syntax of declaration of a variable of the structure :

struct structure_name structure_variable_name;

For example, to declare a variable of the above defined structure student, we can write it as:

struct student s1;

- The variables declared inside the structure are called as the member elements of the structure. For example name, roll number and fees are member elements of the structure student.
- Structure is normally used when a set of data items corresponding to a common thing are to be stored together. For example, when information of a student is to be stored, we need to store his name, roll number, fees etc. In this case we can make a structure element and store the corresponding data of every student in a separate structure element.
- To access a member element of a structure variable we use the dot operator (“.”), also called as the period operator. The syntax of accessing a structure element is given below:

  structure_variable_name.varaible_name;

  For example, in the above case to access the fees of the student “s1”, the period operator will be used as shown below:

  s1.fees;
- The concept of structure will be clear with an example.

Program 4.1.1 : Write a program to store and display the name, roll number and fees of a student using structure.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct student
{
    char name[20];
    int roll_no;
    float fees;
};
void main ()
{
    clrscr();
    struct student s1;
    cout<<"Enter the student's name, roll number and fees
    paid:";
    gets(s1.name);
    cin>>s1.roll_no>>s1.fees;
    cout<<"The student details are as follows:"
    <<endl<<"Name:"<<s1.name<<endl<<"Roll
number:"
    <<s1.roll_no<<endl<<"Fees:"<<s1.fees
    <<endl;
    getch();
}
```

Output :

```
Enter the student's name, roll number and fees paid:
John Patrick
24
106000
The student details are as follows:
Name : John Patrick
Roll number : 24
Fees : 106000
```

Explanation :

- The declaration of a structure for student was already discussed; the same structure is used here.
- Also the declaration of a variable of structure was discussed; the variable s1 is declared here in the same manner.
- The student’s name is accepted using gets() function, while other parameters using “cin”. The parameters are accessed using the period operator as just discussed. Also to display the variables of the structure elements we have used the period operator.

Program 4.1.2 : Write a program to store and display the name, runs scored and wickets taken of a cricket player using structure.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct cricketer
{
```

```
    char name[20];
    int runs, wickets;
};
void main ()
{
    clrscr();
    struct cricketer c1;
    cout<<"Enter the cricketer's name, runs scored and
    wickets taken:";
    gets(c1.name);
    cin>>c1.runs>>c1.wickets;
    cout<<"The cricketer details are as follows:"
    <<endl<<"Name:"<<c1.name<<endl
    <<"Runs scored:"<<c1.runs<<endl<<"Wickets
taken:"
    <<c1.wickets<<endl;
    getch();
}
```

Output :

```
Enter the cricketer's name, runs scored and wickets
taken:Sachin Tendulkar
20000
150
The cricketer details are as follows:
Name:Sachin Tendulkar
Runs scored:20000
Wickets taken:150
```

Explanation :

– A similar program to the previous one, only the data types are different.

➢ **Program 4.1.3 : Write a program to store and display the name, matches played and goals scored of a hockey player using structure.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct hockey
{
    char name[20];
    int matches, goals;
};
void main ()
{
    clrscr();
    struct hockey c1;
    cout<<"Enter the hockey player's name, matches played
and goals scored:";
    gets(c1.name);
    cin>>c1.matches>>c1.goals;
    cout<<"The hockey player's details are as
follows:"<<endl<<"Name:"<<c1.name<<endl
<<"Matches played:"<<c1.matches<<endl<<"Goals
scored:"<<c1.goals<<endl;
    getch();
}
```

Output :

```
Enter the hockey player's name, matches played and goals
scored:V. Raghunath
67
41
The hockey player's details are as follows:
Name:V. Raghunath
Matches played:67
Goals scored:41
```

➢ **Program 4.1.4 : Write a program to define a structure 'Tender' having data members tender-no., cost and company-name. Accept and display this data for two variable of this structure.** **S-14, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct Tender
{
    int tender_no,cost;
    char co_name{20];
};
```

```
void main()
{
    struct Tender t1,t2;
    clrscr();
    cout<<"Enter company name, tender number and cost:";
    cin>>t1.co_name>>t1.tender_no>>t1.cost;
    cout<<"Enter another company name, tender number
and cost:";
    cin>>t2.co_name>>t2.tender_no>>t2.cost;
    cout<<"Number\tCost\tCompany
Name\n"<<t1.tender_no<<"\t"<<t1.cost<<"\t"<<t1.co_
name<<"\n"<<t2.tender_no<<"\t"<<t2.cost<<"\t"<<t2
.co_name<<"\n";
    getch();
}
```

Output :

```
Enter company name, tender number and cost:abc
1
12000
Enter another company name, tender number and cost:xyz
2
5000
Number  Cost   Company Name
1       12000  abc
2       5000   xyz
```

4.2 Arrays of Structure Variable

- An array of structure can be declared in the same way as declaring array of any other data type.
- For example, an array of the structure student can be declared as shown below:

  struct student s[100];

  This array can now store information of 100 students.
- Array of structure must be used when many variables of a structure are required. For example, to store information of 60 students or information of 11 cricket players, etc.
- Let us see some program examples for array of structure.

➢ **Program 4.2.1 : Write a program to store and display the name, marks in three subjects and roll number of 'n' students using structure. Display the output in tabular form.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct student
{
    char name[20];
    int roll_no;
    int physics,chem,maths;
};
void main ()
{
    clrscr();
    struct student s[100];
    int n,i;
    cout<<"Enter the number of students";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter the student's name, roll number
        and marks in three subjects:";
        gets(s[i].name);

cin>>s[i].roll_no>>s[i].physics>>s[i].chem>>s[i].maths;
    }
    cout<<"Name\tRoll No\tPhysics\tChem\tMaths\n";
    cout<<"-----------------------------------------
"<<endl;
    for(i=0;i<=n-1;i++)
    {
    cout<<s[i].name<<"\t"<<s[i].roll_no<<"\t"
<<s[i].physics<<"\t"<<s[i].chem<<"\t"<<s[i].maths<<
endl;
    }
    getch();
}
```

Output :

```
Enter the number of students 4
Enter the student's name, roll number and marks in three subjects:Jay
1
40
40
80
Enter the student's name, roll number and marks in three subjects:Ajay
12
49
49
99
Enter the student's name, roll number and marks in three subjects:Vijay
24
45
45
90
Enter the student's name, roll number and marks in three subjects:Sujoy
5
50
50
100
Name    Roll No    Physics  Chem    Maths
-------------------------------------------------------------
Jay     1          40       40      80
Ajay    12         49       49      99
Vijay   24         45       45      90
Sujoy   5          50       50      100
```

Explanation :

- The elements of an array of structure are accessed in a similar manner as the elements of any other array. The member elements are accessed using the period operator (".").

➢ **Program 4.2.2 : Write a program to store the name, runs scored and wickets taken of 'n' cricketers using structure. Display the output in tabular form.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct cricketer
{
    char name[20];
    int runs, wickets;
};
void main ()
{
    clrscr();
    struct cricketer c[100];
    int n,i;
    cout<<"Enter the number of cricketers";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter the cricketer's name, runs scored
and wickets taken:";
        gets(c[i].name);
        cin>>c[i].runs>>c[i].wickets;
    }
        cout<<"Name\tRuns\tWickets"<<endl;
        cout<<"-----------------------------------"<<endl;
        for(i=0;i<=n-1;i++)
    {

    cout<<c[i].name<<"\t"<<c[i].runs<<"\t"<<c[i].wic
kets<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of cricketers3
Enter the cricketer's name, runs scored and wickets
taken:Sachin
20000
150
Enter the cricketer's name, runs scored and wickets
taken:Zaheer
500
500
Enter the cricketer's name, runs scored and wickets
taken:Dhoni
10000
0
Name      Runs      Wickets
----------------------------------------
Sachin    20000     150
Zaheer    500       500
Dhoni     10000     0
```

- **Program 4.2.3 : Write a program to store the name, roll number and marks in three subjects of 'n' students using structure. Generate a merit list with respect to the total marks scored i.e. display the output in tabular form in order of maximum total marks to minimum total marks in three subjects.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct student
{
    char name[20];
    int roll_no;
    int physics,chem,maths,total;
};
void main ()
{
    clrscr();
    struct student s[100],temp;
    int n,i,j;
    cout<<"Enter the number of students";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter the student's name, roll number
and marks in three subjects:";
        gets(s[i].name);

cin>>s[i].roll_no>>s[i].physics>>s[i].chem>>s[i].maths;
        s[i].total=s[i].physics+s[i].chem+s[i].maths;
    }
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=n-2;j++)
        {
            if(s[j].total<s[j+1].total)
            {
                temp=s[j];
                s[j]=s[j+1];
                s[j+1]=temp;
            }
        }
    }
    cout<<"Name\tRoll No\tPhysics\tChem\tMaths\tTotal\n";
    cout<<"----------------------------------------------
"<<endl;
    for(i=0;i<=n-1;i++)
    {
    cout<<s[i].name<<"\t"<<s[i].roll_no<<"\t"<<s[i].p
hysics<<"\t"<<s[i].chem<<"\t"<<s[i].maths<<"\t"<<s[
i].total<<endl;
    }
    getch();
}
```

Output :

```
Enter the number of students4
Enter the student's name, roll number and marks in three
subjects:Jay
```

```
1
40
40
80
Enter the student's name, roll number and marks in three subjects:Ajay
12
49
49
99
Enter the student's name, roll number and marks in three subjects:Vijay
24
45
45
90
Enter the student's name, roll number and marks in three subjects:Sujoy
5
50
50
100
Name    Roll No    Physics    Chem    Maths    Total
-----------------------------------------------------------
Sujoy     5          50         50      100      200
Ajay      12         49         49      99       197
Vijay     24         45         45      90       180
Jay       1          40         40      80       160
```

Explanation :

– Here, to generate merit list we sort the elements of the structure array in descending order. The sorting of structure elements array is done in a similar manner as done for integer array elements.

Note : The "temp" variable used for swapping should be an element of the same structure. So that the entire structure element is swapped. We have to swap the entire structure element and not just the total marks. Hence, the "temp" variable should not be just an integer, instead it should have the variable to be of structure student type.

➢ **Program 4.2.4 : Write a program to store the name, matches played and goals scored by 'n' hockey players using structure. Generate a list with goals scored in descending order. i.e. display the output in tabular form in order of maximum goals scored to minimum goals scored.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct hockey
{
    char name[20];
    int matches,goals;
};
void main ()
{
    clrscr();
    struct hockey s[100],temp;
    int n,i,j;
    cout<<"Enter the number of hockey players:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter the player's name, matches played and goals scored:";
        gets(s[i].name);
        cin>>s[i].matches>>s[i].goals;
    }
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=n-2;j++)
        {
            if(s[j].goals<s[j+1].goals)
            {
                temp=s[j];
                s[j]=s[j+1];
                s[j+1]=temp;
            }
```

```
            }
        }
        cout<<"Name\t\tMatches\tGoals\n";
        cout<<"----------------------------------"<<endl;
        for(i=0;i<=n-1;i++)
        {

        cout<<s[i].name<<"\t"<<s[i].matches<<"\t"<<
                s[i].goals<<endl;
        }
        getch();
}
```

Output :

```
Enter the number of hockey players:4
Enter the player's name, matches played and goals
scored:Dhyan Chand
100
500
Enter the player's name, matches played and goals
scored:Balbir Singh
75
450
Enter the player's name, matches played and goals scored:Ajit
Pal Singh
67
761
Enter the player's name, matches played and goals
scored:Dhanraj Pillay
120
1058

Name                    Matches    Goals
-----------------------------------------------
Dhanraj Pillay          120        1058
Ajit Pal Singh          67         761
Dhyan Chand             100        500
Balbir Singh            75         450
```

Explanation :

– The logic is exactly similar to the previous program. The only difference is with respect to the elements of the structure.

➢ **Program 4.2.5 : Write a program to store the name, matches played and runs scored by 'n' cricket players using structure. Generate a list with runs scored in descending order i.e. display the output in tabular form in order of maximum runs scored to minimum runs scored.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct cricketer
{
    char name[20];
    int matches,runs;
};
void main ()
{
    clrscr();
    struct cricketer s[100],temp;
    int n,i,j;
    cout<<"Enter the number of cricketers:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter the player's name, matches played
and runs scored:";
        gets(s[i].name);
        cin>>s[i].matches>>s[i].runs;
    }
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=n-2;j++)
        {
            if(s[j].runs<s[j+1].runs)
            {
                temp=s[j];
                s[j]=s[j+1];
                s[j+1]=temp;
```

```
                }
            }
        }
        cout<<"Name\t\tMatches\tRuns\n";
        cout<<"--------------------------------"<<endl;
        for(i=0;i<=n-1;i++)
        {

        cout<<s[i].name<<"\t"<<s[i].matches<<"\t"<<s[i].
runs<<endl;
        }
        getch();
}
```

Output :

```
Enter the number of cricketers:5
Enter the player's name, matches played and runs
scored:Sachin T.
171
14006
Enter the player's name, matches played and runs scored:Sunil
Gavaskar
125
10122
Enter the player's name, matches played and runs
scored:Rahul Dravid
144
11581
Enter the player's name, matches played and runs
scored:Brian Lara
131
11953
Enter the player's name, matches played and runs
scored:Allan Border
156
11174
Name            Matches     9Runs
-----------------------------------------------
Sachin T.       171         14006
Brian Lara      131         11953
Rahul Dravid    144         11581
Allan Border    156         11174
Sunil Gavaskar  125         10122
```

Explanation :

- The logic is exactly similar to the previous program. The only difference is with respect to the elements of the structure.

4.3 Nested Structures

Q. 4.3.1 What do you mean by nested structures? explain with an example . (Ref. Sec. 4.3) **(3 Marks)**

- If one of the members of structure is also a structure, then such a structure is called as a nested structure.
- The syntax of a nested structure can be as given below:

```
struct structure_name
{
    data_type variable_name;
    -
    struct
    {
        data_type variable_name;
        -
        -
    } internal_structure_name;
    -
    -
};
```

For example to describe a circle we need its radius and centre. The centre requires x and y co-ordinates. Hence, this can be described by a nested structure as shown below:

```
struct circle
{
    float radius;
    struct
    {
        float x,y;
    }centre;
}
```

- To access the element of an internal structure the syntax is as given below:

  structure_variable name.internal_structure_name.element_name;

  For example if we make an variable of structure circle as struct circle c, then to access the "x" co-ordinate of the structure we will have the syntax as "c.centre.x"

- Let us see some examples corresponding to nested structure to understand these concepts.

➢ **Program 4.3.1 : Write a program to store the radius and centre of a circle using a nested structure. Also display this information.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct circle
{
    float radius;
    struct
    {
        float x,y;
    }centre;
};
void main ()
{
    clrscr();
    struct circle c;
    cout<<"Enter the radius and x and y co-ordinates of
circle:";
    cin>>c.radius>>c.centre.x>>c.centre.y;
    cout<<"Circle
Information\nRadius="<<c.radius<<"\nCentre co-
ordinates:("<<c.centre.x<<","<<c.centre.y<<")";
    getch();
}
```

Output :

```
Enter the radius and x and y co-ordinates of circle:5.6
2.3
3.6
Circle Information
Radius=5.6
Centre co-ordinates:(2.3,3.6)
```

Explanation :

- You can notice the access of the elements of the internal structure "centre". The outer structure is named as "circle" while the internal structure is named as "centre".
- The access of the elements of the internal structure is as discussed before this program.

➢ **Program 4.3.2 : Write a nested structure to store the information of a book i.e. its title, author's name, publisher's name, edition and ISBN number. The author's name and publisher's name must be a structure with two elements each namely first name and surname.**

```
struct book
{
    char title[20];
    struct
    {
        char f_name[20],surname[20];
    }author;
   struct
    {
        char f_name[20],surname[20];
    }publisher;
    int edition;
    float ISBN;
};
```

Explanation :

- This is just the structure of the book information.
- The program to this can be written in the similar manner as written for the structure circle in program 4.3.1.

4.4 Unions

Q. 4.4.1	Define Union and also give syntax for union (Ref. Sec. 4.4)	**(4 Marks)**
Q. 4.4.2	Compare structure and union (Ref. Sec. 4.4)	**(3 Marks)**

- Union is a collection of multiple data elements that can be of different data types. But, only one of these data items can be stored in the union variable.
- The memory space required to store one variable of union is equal to the memory space required by the largest element in the union.
- Let us see the syntax of a union declaration.

```
Syntax of structure declaration:
union union_name
{
    data_type variable_name;
    data_type variable name;
    -
    -
};
```

- For example, a union to store the some information like name, roll number or id of a student in college can be declared as shown below.

```
union student
{
    char name[20];
    int roll_no;
    float id;
};
```

Syntax of declaration of a variable of the union:

union union_name union_variable_name;

For example, to declare a variable of the above defined union student, we can write it as:

union student s1;

- The memory space required by the variable "name" of the union is 20 bytes (since; one byte of one char data); for "roll_no", the memory space required is 2 bytes while for "id" the space required is 4 bytes.
- Hence, the space allotted to the variable of the union is 20 bytes as the maximum space is required to store the "name", which requires 20 bytes.
- Let us see the differentiation of a structure and union.

Sr. No.	Structure	Union
1.	Memory allotted for a structure is equal to the space require collectively by all the members of that structure.	Memory allotted for a union is equal to the space required by the largest member of that union.
2.	Data is more secure in structures.	Data can be corrupted in a union.
3.	Structure provide ease of programming.	Unions are comparatively difficult for programming.
4.	Structure requires more memory.	Union requires less memory.
5.	Structures must be used when information of all the member elements of a structure are to be stored.	Unions must be used when only one of the member elements of the union is to be stored.

- Let us see a program example of Union.

➢ **Program 4.4.1 : Write a program to store the information of a person as his name or ID number using union. Ask the user for the information choice.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
union info
{
    char name[20];
    long id;
};
void main ()
{
    clrscr();
    union info i1;
    int choice;
    cout<<"Enter your information\n1. Name\n2.ID
number\nEnter your choice:";
```

```
    cin>>choice;
    switch(choice)
    {
        case 1:     cout<<"Enter your name:";
            gets(i1.name);
            break;
        case 2:     cout<<"Enter your ID number:";
            cin>>i1.id;
            break;
        default:cout<<"Invalid Choice";
    }
    if(choice==1)
    cout<<"Your Name entered is "<<i1.name;
    else if(choice==2)
    cout<<"Your ID number entered is "<<i1.id;
    getch();
}
```

Output :

```
Enter your information
1. Name
2.ID number
Enter your choice:1
Enter your name:Harish Narula
Your Name entered is Harish Narula
```

Explanation :

- In this case we have to store your name or ID number as an information about the person. Hence, we use a union over here.
- The user is asked for the choice and accordingly is asked for the information. If the user enters choice for name, the name is accepted in the string; while if the user enters choice for ID, the integer ID is accepted from the user.
- The same is then displayed according to the choice.

4.5 Exam Pack (MSBTE and Review Questions)

☞ **Syllabus Topic : Structures in C++**

Q. 1 Define a structure with it's syntax.
(Refer Section 4.1) **(W-14, 2 Marks)**

Q. 2 Define structure. Give syntax of defining structure.
(Refer Section 4.1) **(W-15, 4 Marks)**

Q. 3 Give syntax and example of defining structure and declaring structure variables.
(Refer Section 4.1) **(S-17, 4 Marks)**

Q. 4 What is structure ? How user declare structure ? Give example. **(Refer Section 4.1) (W-17, 4 Marks)**

Q. 5 Write a program to define a structure 'Tender' having data members tender-no., cost and company-name. Accept and display this data for two variable of this structure.
(Refer Program 4.1.4) **(S-14, 4 Marks)**

Q. 6 What do you mean by nested structures? explain with an example. **(Refer Section 4.3)** **(3 Marks)**

Q. 7 Define Union and also give syntax for union.
(Refer Section 4.4) **(4 Marks)**

Q. 8 Compare structure and union.
(Refer Section 4.4) **(3 Marks)**

❑❑❑

UNIT II

CHAPTER 5

Classes and Objects

Syllabus

Classes and Objects

- **Unit Outcomes**
  - Develop relevant friend functions to solve the given problem.
  - Write C++ program to use array of given objects.
  - Write C++ program to create the given object using constructor.
  - Write program to delete the given object using destructor in C++ program.
- **Topics and Sub-Topics**
  - Class and Object : Introduction, Specifying a Class, Access Specifiers, Defining Member Functions, Creating Objects and Memory Allocation for Objects.
  - Static Data Members, Static Member Function, Friend Function.
  - Array of Objects, Objects as Function Arguments.
  - Concepts of Constructors, Types of Constructors.
  - Multiple Constructors in a Class, Constructors with Default Arguments.
  - Destructors.

5.1 Comparison of Procedure Oriented Programming and Object Oriented Programming

→ **(W-14, S-16, W-17)**

Q. 5.1.1 What is object oriented programming? How it is different from procedure oriented programming? (Ref. Sec. 5.1) **(4 Marks)**

Q. 5.1.2 Differentiate between structure and class. (Ref. Sec. 5.1) **(W-14, 4 Marks)**

Q. 5.1.3 Compare structure and class. (Ref. Sec. 5.1) **(S-16, 4 Marks)**

Q. 5.1.4 What are the differences between structure and class ? (Ref. Sec. 5.1) **(W-17, 4 Marks)**

Q. 5.1.5 What is difference between procedure oriented programming and object oriented programming ? (Ref. Sec. 5.1) **(W-17, 4 Marks)**

- As we have discussed in chapter 1, procedure oriented programming gives significance to procedure i.e. how to do a task.
- The structure of a procedure oriented programming as discussed in chapter 1 is made up of functions. If a variable is to be accessed by multiple functions then such a variable is known as global variable.
- Disadvantages of procedure oriented programming language :
  1. Global data is vulnerable to inadvertent changes. In other words, the global variables may be changed unintentionally.

PDF 3103-oop.pdf

Q. 5.1.3 Compare structure and class. (Ref. Sec. 5.1) (S-16, 4 Marks)

Q. 5.1.4 What are the [illegible] between structure and class ? (Ref. Sec. 5.1) (W-17, 4 Marks)

known as global variable.

– Disadvantages of procedure oriented programming langua[illegible]

1. Global data is vulnerable to inadvertent change[illegible] other words, the global variables may be chang[illegible] unintentionally.

2. If a data structure is changed then all the functions accessing this structure have to be altered i.e. if the data variables in the structure are changed, then all the functions that use this structure will have to be changed.
3. Procedure oriented programs cannot model the real world problems very well.

– Object oriented programming on the other hand gives more importance to data rather than a procedure.

– The structure of object oriented programs as discussed in chapter 1, is also shown in Fig. 5.1.1.

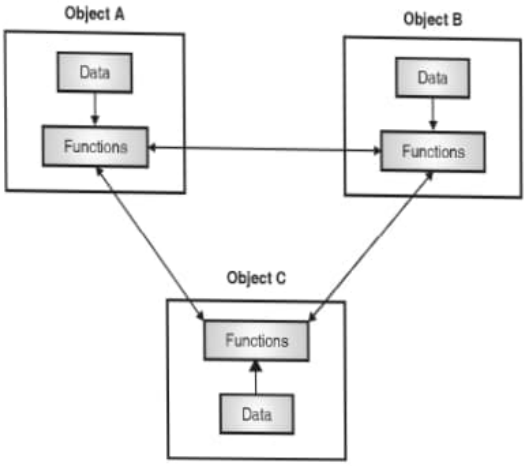


Fig. 5.1.1 : Structure of an object oriented program

☞ **Difference between structure and class**

Sr. No.	Class	Structure
1.	Class is a reference type and its object is created on the heap memory.	Structure is a value type that is why its variable is created on the stack memory.
2.	Class can inherit the another class.	Structure does not support the inheritance.
3.	Class can have the all types of constructor and destructor.	Structure doesn't have constructors or destructors
	The member variable of class can be initialized directly.	The member variable of structure can not be initialized directly.

Syllabus Topic : Class and Object : Introduction

5.1.1 Class and Object : Introduction

➔ **(S-14, W-14, W-16, W-17)**

Q. 5.1.6 What is object-oriented programming ? List basic concepts of it ? Explain each concept. (Ref. Sec. 5.1.1) **(10 Marks)**

Q. 5.1.7 Explain the benefits of object oriented programming. (Ref. Sec. 5.1.1) **(8 Marks)**

Q. 5.1.8 Distinguish between Data abstraction and data encapsulation. (Ref. Sec. 5.1.1) **(5 Marks)**

Q. 5.1.9 What is Data abstraction ? (Ref. Sec. 5.1.1) **(S-14, 2 Marks)**

Q. 5.1.10 Define the following terms : (1) Object (2) Class (Ref. Sec. 5.1.1) **(S-14, 2 Marks)**

Q. 5.1.11 State any four features of OOP. Ref. Sec. 5.1.1) **(S-14, 4 Marks)**

Q. 5.1.12 Define class with it's syntax. (Ref. Sec. 5.1.1) **(W-14, W-16, 2 Marks)**

Q. 5.1.13 Enlist any four concepts of OOP. (Ref. Sec. 5.1.1) **(W-16, 2 Marks)**

Q. 5.1.14 What are the features of object oriented programming ? (Ref. Sec. 5.1.1) **(W-16, 4 Marks)**

Q. 5.1.15 Write down characteristics of object oriented programming. (Ref. Sec. 5.1.1) **(W-17, 4 Marks)**

– Object oriented programming gives significance to objects or data rather than the procedure. Let us see the features of OOP.

1. More emphasis is given to data rather than procedure. It is seen in the real world problems that the data or the objects are more important than the procedure or the method to perform a task. Hence, in object oriented programming, object is given more importance compared to the procedure of doing it.
2. Programs are divided into objects as shown in Fig. 5.1.1. As shown in Fig. 5.1.1, the objects contain data and functions required to access them. Thus, in an object oriented program, you will notice the objects as shown in the structure.

3. Data structures are designed such that they characterize the object i.e. all the information required for an object are stored in the variables of that object.
4. Functions that operate on the data of an object are tied together in data structures i.e. the functions that operate on a data are associated with them as shown in Fig. 5.1.1.
5. Data is hidden or cannot be access by external function. The data of an object can be accessed only by the functions of the same object.
6. Objects communicate with each other through functions. If a function of an object wants the data of another object then it can be accessed only through the functions.
7. New data and functions are easily added when required. Whenever new data is to be added, all the functions need not be changed; only that functions are to be changed, which require to access the data.
8. It follows a bottom-up approach. In this type of approach, each element is first specified in detail and then the entire system is made using these elements. You will notice in the programming of C++, that this approach of bottom-up approach makes the programming very simple.

– There are few important terms related to Object Oriented Programming that we need to understand. First the concept of objects and classes and then the specialties of OOPs viz. Data Abstraction, Encapsulation, Inheritance, Polymorphism, etc. Let us understand them one by one.

1. Class
2. Object
3. Data abstraction
4. Data encapsulation
5. Inheritance
6. Polymorphism

➜ **1. Class**

It is a type or a category of things. It is similar to a structure with the difference that it can also have functions besides data items. A structure, we have seen, can have only data variables but a class can have data members as well as function members.

➜ **2. Object**

It is an instance or example of a class. You can imagine it to be similar to a variable of class like we have a variable of a structure.

➜ **3. Data abstraction**

Data abstraction is like defining or abstracting the object according to the required parameters. For example; if there is a class for circle we need to just define the radius of the object of this class. We need not bother about anything else of that object.

➜ **4. Data encapsulation**

The data of an object is hidden from other objects. This is called as encapsulation. Encapsulation is achieved by putting data and functions associated with it into a class.

➜ **5. Inheritance**

The mechanism of deriving the properties of one class into another class is known as inheritance. We will see in detail about this concept in a special section dedicated on Inheritance.

➜ **6. Polymorphism**

Poly refers to multiple and morph refers to different forms. Hence, polymorphism means multiple forms of the same thing. This topic will also be covered in detail in the later sections

Syllabus Topic : Access Specifiers

5.1.2 Access Specifiers

– In C++, we have 3 access specifiers viz. public, protected and private.
– These access specifiers are used to specify the access of the data and function members of a class
– As the name says, access specifier will indicate who can access the data or function.
– A "public" access specifier indicates that every function can access this member.
– A "protected" access specifier indicates that only classes derieved can use this member. A derieved class concept will be dealt later in the topic called as Inheritance.

- A "private" access specifier indicates that only the functions of the same class can access the member.

Syllabus Topic : Specifying a Class

5.1.3 Specifying a Class

- A class can be specified or created and then objects of that class can be made.
- A class will have data members and function members.
- All the objects of a class will have the same properties as that of specified in the class
- Syntax of specifying a class

```
class class_name
{
    access_specifier:
    data_members;
    access_specifier:
    function members
};
```

Syllabus Topic : Creating Objects and Memory Allocation of Objects

5.1.4 Creating Objects and Memory Allocation of Objects

➔ **(W-14, S-15, W-16, S-17)**

Q. 5.1.16	Explain how memory is allocated to an object of a class with diagram. (Ref. Sec. 5.1.4) **(W-14, W-16, 4 Marks)**
Q. 5.1.17	How memory is allocated when multiple object of class are created ? Explain with example. (Ref. Sec. 5.1.4) **(S-17, 4 Marks)**
Q. 5.1.18	Explain memory allocation for object with example. (Ref. Sec. 5.1.4) **(S-15, 4 Marks)**

An object of a class will be allocated the memory space as required by the data members specified in the class. The object is an instance (example) of the class.

For example, if human being is a class then we all are objects (instances or examples) of this class.

Syntax for creating an object:

class_name object_name;

Note : The memory space is allocated to an object and not to a class. Hence, by writing the statement, circle c; we are actually allocating the memory space required for the data members of the class 'circle'. This statement allocates space as required for two float type elements i.e. for the variables 'r' and 'a'.

Each object will be allocated separate space in memory. Hence, if we declare another object, the memory space for the same will be different than the first object.

Example: Calculate the size of object B1 defined in following class :

```
class Book
{
char B_name[15]; int B_id;
int Price;
};
Book B1;
```

Ans: B_name: 15 bytes

B_id: 2 bytes

Price: 2 bytes

Total : 19 bytes

Syllabus Topic : Defining Member Functions

5.2 Defining Member Functions of a Class

➔ **(W-14)**

Q. 5.2.1	How many ways we can define member function in class ? Give it's syntax. (Ref. Sec. 5.2) **(W-14, 4 Marks)**

- To define the data members of a class the logic is similar to that of the data members of a structure. But the new thing as discussed some time back is the member functions of a class. Let us see how this can be done and what are the different methods to do this.

- There are three methods of defining member functions of a class. They are :

1. Internally defined Functions
2. Externally defined functions
3. Inline functions

- Let us see each of these in details.

5.2.1 Internally Defined Functions

➔ **(S-15, W-16)**

Q. 5.2.2 Explain syntax for declaring the function inside the class and outside the class with example.
(Ref. Secs. 5.2.1 , 5.2.2 and Prog. 5.2.1)
(S-15, 4 Marks)

Q. 5.2.3 Give syntax for defining a member function inside and nesting of functions in a class with example.
(Ref. Secs. 5.2.1, 5.2.1.1 and Prog. 5.2.1)
(W-16, 4 Marks)

- When the member functions of a class are defined inside the class itself they are called as internally defined member functions.
- Let us see some examples of object oriented programs with internally defined functions. We will see these examples for most of the concepts so as to make the understanding of the concept simple.

➢ **Program 5.2.1 : Write a program to find area of circle using Object Oriented Programming such that the class circle must have three member functions namely :**

(a) read() to accept the radius from the user.
(b) compute() for calculating the area.
(c) display() for displaying the result.

S-15, W-16, 2 Marks

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    void read()
    {
        cout<<"Enter radius:";
        cin>>r;
    }
    void compute()
    {
        a=3.14*r*r;
    }
    void display()
    {
        cout<<"Area="<<a;
    }
};
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

Explanation :

- A class is created with the name 'circle'. Two float type variables are declared in it namely 'r' and 'a' for the radius and area respectively.
- The three functions are written inside the class 'circle' namely read(), compute() and display().
- The read() function accepts the value of radius from the user. The compute() function calculates the area of the circle. The display() function displays the calculated area.
- But all these functions will perform these tasks only when they are called. These functions can be called by the period operator used with an object of this class.

- Hence, an object of this class is created in the main function. The syntax of making an object of a class is very simple.

  Syntax of declaring an object of a class:

  Class_name Object_name;

  For example, in the above case we have made an object of the class "circle" by the statement,

  circle c;

- Thereafter; we have called one by one each of the functions of the class 'circle' in the required sequence. To call the function of the class we have used the period operator('.'). The syntax of accessing a member function of a class is as shown below:

  Object_name.Function_name();

  For example, in the above case to call the function read(), we have the statement,

  c.read();

- Similarly; the compute() function and the display() function are called.
- Hence, the read() function accepts the input from user as seen in the output and the display() function displays the result calculated by the compute() function.

➢ **Program 5.2.2: Write a program to calculate the value of the following series using internal member function :**
$S = 1^2 + 2^2 + 3^2 + 4^2 ..... + n^2$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    void read()
    {
        cout<<"Enter the value of n:";
        cin>>n;
    }
    void compute()
    {
        for(i=1, sum = 0;i<=n;i++)
        {
            sum=sum+i*i;
        }
    }
    void display()
    {
        cout<<"Value of the series="<<sum;
    }
};
void main()
{
    clrscr();
    series s;
    s.read();
    s.compute();
    s.display();
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
```

Explanation :

Note : A variable cannot be initialized in the declaration of a class member variable. It has to be initialized in a function only. Hence, the variable "sum" is not initialized in the declaration statement; instead it is initialized to zero in the compute function.

- The logic and way of writing the program is same as the previous one.

➢ **Program 5.2.3 : Write a program to define a class student having data members name and roll no. Accept and display data for one object.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Student
```

```
{
    private:
    char name[20];
    int roll;
    public:
    void accept()
    {
        cout<<"Enter name and roll number:";
        cin>>name>>roll;
    }
    void display()
    {
        cout<<"Name:"<<name<<"\nRoll
number:"<<roll<<endl;
    }
};
void main()
{
    clrscr();
    Student s;
    s.accept();
    s.display();
    getch();
}
```

Output :

```
Enter name and roll number:Ajay
24
Name:Ajay
Roll number:24
```

➢ **Program 5.2.4 : Write a program to declare a class 'student' having data members as name and percentage. Write a constructor to initialize these data members accept and display data for one student.** **W-16, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Student
{
    private:
    char name[20];
    int percentage;
    public:
    void accept()
    {
        cout<<"Enter name and percentage:";
        cin>>name>>percentage;
    }
    void display()
    {

    cout<<"Name:"<<name<<"\nPercentage:"<<percent
age<<endl;
    }
};
void main()
{
    clrscr();
    Student s;
    s.accept();
    s.display();
    getch();
}
```

Output :

```
Enter name and percentage:Ajay
85
Name:Ajay
Percentage:85
```

5.2.1.1 Nesting of Member Function

➔ **(S-16, W-16)**

Q. 5.2.4 What is nesting of member function ? Give one example.
(Ref. Sec. 5.2.1.1 and Prog. 5.2.5)
(S-16, 4 Marks)

Q. 5.2.5 Give syntax for defining a member function inside and nesting of functions in a class with example.

(Ref. Secs. 5.2.1, 5.2.1.1 and Prog. 5.2.5)

(W-16, 4 Marks)

➢ **Program 5.2.5 : A member function of a class calling another member function of the same class is called as nesting of member function. The following program example demonstrates this concept.**

S-16, W-16, 2 Marks

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    void read()
    {
        cout<<"Enter the value of n:";
        cin>>n;
    }
    int compute()
    {
        for(i=1, sum = 0;i<=n;i++)
        {
            sum=sum+i*i;
        }
        return sum;
    }
    void display()
    {
        compute();
        cout<<"Value of the series="<<sum;
    }
};
void main()
{
    clrscr();
    series s;
    s.read();
    s.display();
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
```

5.2.2 Externally Defined Functions

➔ **(S-15, W-15, S-16, W-17)**

Q. 5.2.6 Explain syntax for declaring the function inside the class and outside the class with example.

(Ref. Secs. 5.2.1 , 5.2.2 and Prog. 5.2.1)

(S-15, 4 Marks)

Q. 5.2.7 How function is defined outside of class, write general syntax and example of same. (Ref. Sec. 5.2.2 and Prog. 5.2.6)

(W-15, 4 Marks)

Q. 5.2.8 Write general form of member function, definition out of class.

(Ref. Sec. 5.2.2) **(S-16, 2 Marks)**

Q. 5.2.9 How user can declare member function outside the class ?

(Ref. Sec. 5.2.2) **(W-17, 2 Marks)**

- When the member functions of a class are defined outside the class they are called as externally defined member functions.
- The functions when defined outside must use scope resolution operator to define the scope of the function to be within the class. The syntax of defining a function outside the class using the scope resolution operator is as given below:

```
Return_type Class_name :: Function_name( argument list )
{
    -
    Statements;
    -
}
```

- Also in this case the prototype declaration has to be done inside the class for the externally defined functions.
- Let us see these in details.

➢ **Program 5.2.6 : Write a program to find area of circle using Object Oriented Programming such that the class circle must have three externally defined member functions namely:**

(a) read() to accept the radius from the user.

(b) compute() for calculating the area.

(c) display() for displaying the result.

W-15, 2 Marks

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    void read();
    void compute();
    void display();
};
void circle::read()
{
    cout<<"Enter radius:";
    cin>>r;
}
void circle::compute()
{
    a=3.14*r*r;
}
void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

Explanation :

- A class is created with the name 'circle'. Two float type variables are declared in it namely 'r' and 'a' for the radius and area respectively.
- The prototype of the three functions are declared inside the class "circle" namely read(), compute() and display(). The definition of these functions are written outside the class. But, with the help of the scope resolution operator the scope of the functions is declared in the class "circle".
- The read() function accepts the value of radius from the user. The compute() function calculates the area of the circle. The display() function displays the calculated area.
- But all these functions will perform these tasks only when they are called. These functions can be called by the period operator used with an object of this class.
- Hence, an object of this class is made in the main function.
- Thereafter we have called one by one each of the functions of the class "circle in the required sequence.

➢ **Program 5.2.7 : Write a program to calculate the value of the following series using external member function :**

$$S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    void read();
    void compute();
    void display();
};
void series::read()
```

```
{
    cout<<"Enter the value of n:";
    cin>>n;
}
void series::compute()
{
    for(i=1, sum = 0;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
void series::display()
{
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    series s;
    s.read();
    s.compute();
    s.display();
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
```

Explanation :

Note :	A variable cannot be initialized in the declaration of a class member variable. It has to be initialized in a function only. Hence, the variable 'sum' is not initialized in the declaration statement; instead it is initialized to zero in the compute function.

- The logic and way of writing the program is same as the previous one.

5.2.3 Inline Member Functions

➔ **(W-16)**

Q. 5.2.10	What are inline functions ? Discuss its advantages and disadvantages. Write a program to demonstrate inline function. (Ref. Sec. 5.2.3)	**(10 Marks)**
Q. 5.2.11	Write short note on : Inline function. (Ref. Sec. 5.2.3)	**(5 Marks)**
Q. 5.2.12	Explain inline function with an example. (Ref. Sec. 5.2.3)	**(10 Marks)**
Q. 5.2.13	What do you mean by inline function? Write its syntax and example. (Ref. Sec. 5.2.3 and Prog. 5.2.8)	**(W-16, 4 Marks)**

- We have seen that a function can be defined inside a class or outside a class.
- If a function is to be defined outside the class, but still be treated as a internally defined function then such a function has to be made "inline" function.
- To make a function inline function the keyword 'inline' before the header line of the function. The syntax of defining a inline function of a class is as given below:

```
inline Return_type Class_name :: Function_name( argument
list )
{
    -
    Statements;
    -
}
```

- Also in this case the prototype declaration has to be done inside the class for the inline functions.
- The advantage of inline function is that the execution is faster even though the functions are defined outside. Also another advantage is a member can be declared inline for two classes and hence access members of two classes.
- Let us see the same examples as seen inline defined functions.

➢ **Program 5.2.8 : Write a program to find area of circle using Object Oriented Programming such that the class circle must have three inline functions namely:**

(a) read() to accept the radius from the user.

(b) compute() for calculating the area.

(c) display() for displaying the result.

W-16, 2 Marks

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    void read();
    void compute();
    void display();
};
inline void circle::read()
{
    cout<<"Enter radius:";
    cin>>r;
}
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

➢ **Program 5.2.9 : Write a program to calculate the value of the following series using inline member function :**

$$S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    void read();
    void compute();
    void display();
};
inline void series::read()
{
    cout<<"Enter the value of n:";
    cin>>n;
}
inline void series::compute()
{
    for(i=1, sum = 0;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
inline void series::display()
{
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    series s;
    s.read();
    s.compute();
    s.display();
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
```

Syllabus Topic : Concepts of Constructors, Types of Constructors

5.3 Constructor

➔ **(S-14, W-15, W-16, W-17)**

Q. 5.3.1	Explain the concept of constructor in C++. (Ref. Sec. 5.3)	**(5 Marks)**
Q. 5.3.2	What is constructor? Why is it called so? What are the syntax rules for writing a constructor? Describe inline constructor. (Ref. Sec. 5.3)	**(10 Marks)**
Q. 5.3.3	Illustrate the concept of constructor with default argument with suitable example. (Ref. Sec. 5.3)	**(S-14, 4 Marks)**
Q. 5.3.4	State any four characteristics of constructor. (Ref. Sec. 5.3)	**(S-14, 4 Marks)**
Q. 5.3.5	List four types of constructors. . (Ref. Sec. 5.3)	**(W-15, 2 Marks)**
Q. 5.3.6	State any four characteristics of constructor. (Ref. Sec. 5.3)	**(W-15, 4 Marks)**
Q. 5.3.7	Define constructor. State any two type of constructor. (Ref. Sec. 5.3)	**(W-16, 2 Marks)**
Q. 5.3.8	Explain the concept of constructor with default argument. (Ref. Secs. 5.3 and 5.3.1)	**(W-16, 4 Marks)**
Q. 5.3.9	Enlist types of constructor. (Ref. Sec. 5.3)	**(W-17, 2 Marks)**
Q. 5.3.10	What is constructor ? How user can declared constructor in derived class ? Explain with example. (Ref. Sec. 5.3 and Prog. 5.3.1)	**(W-17, 8 Marks)**

– They are special member functions used for initializing the data elements of the object. There are some important points we need to understand about the constructor.

– These points are listed below:

1. The name of constructor must always be same as that of the class.
2. It has to be defined only in the public visibility of a class.
3. It should not have any return type; not even void.
4. Any number of constructors can be defined in a class and such a set of constructors implements a concept called as constructor overloading.
5. The constructor is executed the moment an object of that class is created.
6. A constructor cannot be inherited but a derived class can call the constructor of the base class.
7. A constructor cannot be static or virtual.

Note : Some of the terms used above may not be cleared now, but they will be cleared as you study the further sections

– There are three types of constructors based on the parameter list. The types of constructors are:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

– Let us see the implementation of these constructors in the subsequent sub-sections.

5.3.1 Default Constructor

➔ **(W-16)**

Q. 5.3.11	Explain the concept of constructor with default argument. (Ref. Secs. 5.3 and 5.3.1)	**(W-16, 4 Marks)**

– If a constructor does not have any parameter list i.e. it cannot accept any parameters than it is called as a default constructor.

– Thus, the default constructor has the blank brackets indicating no parameters can be accepted.

– Let us initialize the values of the variable radius of the class circle, using default constructor and the value of the variable 'n' for the below programs.

➢ **Program 5.3.1 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the constructor and the class circle must have two inline functions namely :**

(a) compute() for calculating the area.

(b) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    circle()
    {
        cout<<"Enter the value of radius:";
        cin>>r;
    }
    void compute();
    void display();
};
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    circle c;
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

Explanation :

- The special function with the name of the class i.e. 'circle' is a constructor. This constructor is not called in the main function, but as discussed it is automatically called when the object of the class is made.
- When compared to the previous program, the task of initializing the value of the variable 'r' is done in the constructor, while the remaining logic is similar.

➢ **Program 5.3.2 : Write a program to calculate the value of the following series using default constructor and inline member function :**

$$S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    series()
    {
        cout<<"Enter the value of n:";
        cin>>n;
        sum=0;
    }
    void compute();
    void display();
};
inline void series::compute()
{
    for(i=1;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
inline void series::display()
```

```
{
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    series s;
    s.compute();
    s.display();
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
```

Explanation :

- Here, the variables i.e. 'n' and 'sum' are initialized in the constructor. The value of sum is initialized to zero while the value of 'n' is taken from the user. The remaining logic is similar.

5.3.2 Parameterized Constructor

➔ **(S-14, W-17)**

Q. 5.3.12	What is copy constructor ? Give one programming example of it. (Ref. Sec. 5.3.2)	**(8 Marks)**
Q. 5.3.13	What is parameterized constructor ? Give the syntax and example of it. (Ref. Sec. 5.3.2)	**(S-14, 4 Marks)**
Q. 5.3.14	What is parametrized constructor ? Explain with example. (Ref. Sec. 5.3.2 and Prog. 5.3.3)	**(W-17, 4 Marks)**

- If a constructor does have a parameter list i.e. it accepts one or more parameters than it is called as a parameterized constructor.
- Thus, the parameterized constructor has an argument list written in the brackets associated with the function definition header line.
- To pass the variable to the constructor, it has to be mentioned in the brackets along with the object name while making the object. The syntax of creating an object is then:

  Class_name Object_name (arguments);
- Let us see the same examples of programs again with parameterized constructor.

➢ **Program 5.3.3 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the parameterized constructor and the class circle must have two inline functions namely :**

(a) compute() for calculating the area.

(b) display() for displaying the result.

W-17, 2 Marks

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    circle(float x)
    {
        r=x;
    }
    void compute();
    void display();
};
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
```

```
    float p;
    cout<<"Enter the radius of the circle";
    cin>>p;
    circle c(p);
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

Explanation :

- Here, the constructor has an argument to be accepted i.e. a float variable 'x'. This variable is to be passed to the constructor when the object is made.
- To pass the variable to the constructor, it has to be mentioned in the brackets along with the object name while making the object. You will notice that the variable is passed while making the object in the brackets along with the object i.e. in the statement:

  circle c(p);
- Since the variable 'p' is to be passed while making the object, the value of variable 'p' is to be accepted from user first and then the object can be created.

➢ **Program 5.3.4 : Write a program to calculate the value of the following series using parameterized constructor and inline member function:**

$$S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    series(int x)
    {
        n=x;
        sum=0;
    }
    void compute();
    void display();
};
inline void series::compute()
{
    for(i=1;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
inline void series::display()
{
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    int x;
    cout<<"Enter the value of n:";
    cin>>x;
    series s(x);
    s.compute();
    s.display();
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
```

Explanation :

- Similar; to the previous program, here also the value of the variable 'n' is accepted from the user in the main program and this is passed to the constructor while making the object in the statement:

  series s(x);

Syllabus Topic : Multiple Constructors in a Class

5.3.3 Copy Constructor (Multiple Constructors/Constructor Overloading)

➔ **(W-14, S-16, S-17, W-17)**

Q. 5.3.15	What is copy constructor ? (Ref. Sec. 5.3.3)	**(2 Marks)**
Q. 5.3.16	Explain overloaded constructor in a class with suitable example. (Ref. Sec. 5.3.3)	**(W-14, 8 Marks)**
Q. 5.3.17	Explain multiple constructor in class with example. (Ref. Sec. 5.3.3)	**(S-16, 4 Marks)**
Q. 5.3.18	Explain multiple constructor in class. Give suitable example. (Ref. Sec. 5.3.3 and Prog. 5.3.5)	**(S-17, 4 Marks)**
Q. 5.3.19	What is copy constructor ? Explain with example. (Ref. Sec. 5.3.3 and Prog. 5.3.5)	**(W-17, 4 Marks)**
Q. 5.3.20	What do you mean by overloaded constructor ? (Ref. Sec. 5.3.3)	**(W-17, 2 Marks)**

- If a constructor has an object of the same class as a parameter in the argument list than it is called as a copy constructor.
- The copy constructor uses an object of the same class to initialize another object of the class.
- To pass the object to the constructor, it has to be mentioned in the brackets along with another object name while making this object. The syntax of creating an object is similar to parameterized constructor with the only difference that the object is passed as the argument. The syntax is shown again for your reference:

  Class_name Object_name (arguments);
- In short, a copy constructor is used to initialize the variables of an object using another object.
- Let us see the same examples of programs again with copy constructor.

➢ **Program 5.3.5 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the copy constructor and the class circle must have two inline functions namely:**

(a) compute() for calculating the area.

(b) display() for displaying the result.

S-17, W-17, 2 Marks

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    circle(float x)
    {
        r=x;
    }
    circle(circle &c)
    {
        r=c.r;
    }
    void compute();
    void display();
};
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a<<endl;
}
void main()
{
    clrscr();
    float p;
    cout<<"Enter the radius of the circle";
```

```
    cin>>p;
    circle c(p);
    c.compute();
    c.display();
    circle c1(c);
    c1.compute();
    c1.display();
    getch();
}
```

Output :

```
Enter the radius of the circle5
Area=78.5
Area=78.5
```

Explanation :

- There are two constructors, hence this concept is also called as constructor overloading.
- The first constructor has an argument to be accepted i.e. a float variable 'x'. This variable is to be passed to the constructor when the object is made.
- The second constructor has an argument of type 'circle'. This type is the object of the same class i.e. 'circle'. This is the copy constructor. We have passed the earlier created object 'c' to the constructor of the second object 'c1' to initialize its variable 'r'.
- Both the objects are created in the main() function with the respective constructors being called as discussed in the previous point. The other functions are called individually for each object.Remember memory space allocated for each object are different.

➢ **Program 5.3.6 : Write a program to calculate the value of the following series using copy constructor and inline member function :**

$$S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    series(int x)
    {
        n=x;
        sum=0;
    }
    series(series &x)
    {
        n=x.n;
        sum=0;
    }
    void compute();
    void display();
};
inline void series::compute()
{
    for(i=1;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
inline void series::display()
{
    cout<<"Value of the series="<<sum<<endl;
}
void main()
{
    clrscr();
    int x;
    cout<<"Enter the value of n:";
    cin>>x;
    series s(x);
    s.compute();
    s.display();
    series s2(s);
    s2.compute();
    s2.display();
```

```
    getch();
}
```

Output :

```
Enter the value of n:5
Value of the series=55
Value of the series=55
```

Explanation :

- The copy constructor is similarly used for creating the second object as in the previous program.

Syllabus Topic : Constructors with Default Arguments

5.3.4 Constructors with Default Arguments

➔ **(S-16, W-17, S-17)**

Q. 5.3.21	State the use of default parameters in constructor with example. (Ref. Sec. 5.3.4)	**(S-16, 4 Marks)**
Q. 5.3.22	What do you mean by default argument ? Illustrate concept of constructor with default argument using suitable example. (Ref. Sec.5.3.4 and Prog. 5.3.7)	**(S-17, 4 Marks)**
Q. 5.3.23	Explain constructor with default argument. (Ref. Sec. 5.3.4)	**(W-14, 4 Marks)**

- If a constructor has value passed to it, the given values are initialised to the data members.
- But in case if data value is not passed, then some default value should be initialised in the data members by the objects. To achieve this we require constructors with default parameters.

➢ **Program 5.3.7 : Illustrate the concept of constructor with default argument with suitable example.**

S-17, 2 Marks

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Interest
{
    private:
    static float rate;
    float principal,duration,sim_int;
    public:
    Interest(float x=1000,float y=10)
    {
        principal=x;
        duration=y;
    }
    void calculate()
    {
        sim_int=principal*rate*duration/100;
    }
    void display()
    {
        cout<<"Simple Interest="<<sim_int;
    }
};
float Interest::rate=10;
void main()
{
    float x,y;
    clrscr();
    cout<<"Enter principal amount:";
    cin>>x;
    Interest i(x);
    i.calculate();
    i.display();
    getch();
}
```

Output :

```
Enter principal amount:2000
Simple Interest=2000
```

Explanation :

- In this case, the values of principal and duration, if passed then the same will be initialised. If no values are passed then the values 1000 and 10 are initialised.

Syllabus Topic : Destructors

5.4 Destructor

➔ **(S-14, W-16, W-17)**

Q. 5.4.1	Explain the concept of destructor in C++. (Ref. Sec. 5.4)	**(5 Marks)**
Q. 5.4.2	State the characteristics of destructor. (Ref. Sec. 5.4)	**(S-14, 2 Marks)**
Q. 5.4.3	What is destructor? Give its syntax and example. (Ref. Sec. 5.4)	**(W-16, 4 Marks)**
Q. 5.4.4	What is destructor ? How destructor is declared ? When destructor is invoked ? (Ref. Sec. 5.4 and Prog. 5.4.1)	**(W-17, 4 Marks)**

- A destructor is used to destroy the memory allocated by the constructor.
- The following points are to be noted w.r.t. destructor :
  1. It is defined only in the public visibility of a class.
  2. The name of the destructor must be same as that of class prefixed with a tilde (~) sign. The name of the constructor and destructor are same as that of the class. Hence to differentiate between the two, the tilde sign is used.
  3. It cannot return or accept any value.
  4. It is automatically called when the compiler exits from the main program.
- The destructor as discussed can only destroy the memory locations allocated by the constructor and the constructor can allocate memory only using the 'new' operator.
- 'new' operator allocates memory to a pointer. Hence, a pointer is to be declared and using the new operator a memory location can be allocated by the following syntax:

  Pointer_name = new Data_type;
- To destroy this memory allocated by the new operator, we require delete operator. In this case the delete operator must be in the destructor. Hence, the destructor will be destroying the memory allocated. The syntax for the delete operator to destroy the memory space allocated is as shown below :

  delete Pointer_name;

➢ **Program 5.4.1 : Write a program to demonstrate the destructor.** **W-17, 2 Marks**

```
# include<iostream.h>
# include<conio.h>
class test
{
    int *p;
    public:
    test()
    {
        p=new int;
    }
    void read()
    {
        cout<<"Enter a number";
        cin>>*p;
    }
    void display()
    {
        cout<<"Value="<<*p<<endl;
    }
    ~test()
    {
        delete p;
        cout<<"Destroyed";
    }
};
void main()
{
    clrscr();
    test t;
    t.read();
    t.display();
    getch();
}
```

Output :

```
Enter a number5
Value=5
Destroyed
```

Explanation :

- The destructor is the ~test() function. This destructor is automatically called after the end of the execution of the main() function. Hence, in our case after the statement getch(); the destructor is automatically called.
- When the destructor is called, the memory allocated by the constructor is deleted using the delete operator. The statement;

  cout<<"Destroyed";

  is then executed.
- Hence, similar to constructor even the destructor is not to be called explicitly. It is automatically called.

Syllabus Topic : Friend Function

5.5 Friend Function

➔ **(S-14, W-14, S-16, W-16, W-17)**

Q. 5.5.1	What are the characteristics of a friend function ? (Ref. Sec. 5.5)	**(5 Marks)**
Q. 5.5.2	Explain the concept of friend function. (Ref. Sec. 5.5)	**(S-14, 4 Marks)**
Q. 5.5.3	Write any two rules to define friend function. (Ref. Sec. 5.5)	**(W-14, W-16, 2 Marks)**
Q.5.5.4	Define friend function. Write syntax of declaring it. (Ref. Sec. 5.5 and Prog. 5.5.1)	**(W-14, 4 Marks)**
Q. 5.5.5	What is friend function ? Explain with example. (Ref. Sec. 5.5 and Prog. 5.5.1)	**(W-17, 4 Marks)**

- The private members of a class can be accessed only by the member functions of a class and not by external functions i.e. non-member functions.
- If a non-member function is required to access a private member of a class then such a function must be friend to that class.
- Another advantage of a friend function is that, a function can be friend to two different classes and hence access the private members of both the classes. Hence whenever such a requirement of a function accessing the private members of two classes is required, we can use friend function which is friend to multiple classes.
- The function is a friend function to the class is to be specified in the class.
- Let us see how this is done in the program 5.5.1.

➢ **Program 5.5.1 : Write a program to add two complex numbers using a friend function.**

W-14, W-17, 2 Marks

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
    friend Complex operator + (Complex c1, Complex c2);
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
Complex add (Complex c1, Complex c2)
{
    Complex c;
    c.x=c1.x+c2.x;
    c.y=c1.y+c2.y;
    return c;
}
void main()
{
    clrscr();
    Complex c1;
```

```
    Complex c2;
    Complex c3;
    c1.read();
    c2.read();
    c3=c1.add(c2);
    c3.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

- Here in the class "Distance", we have mentioned that the operator overloading function is a friend to this class. This function is then defined outside the class. This function will accept two objects of the class "Complex". This function also returns an object of the same class i.e. "Complex".

➢ **Program 5.5.2 : Write a program to add two complex numbers using a friend function to add the complex numbers.** **S-16, 8 Marks**

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
    friend Complex add (Complex c1, Complex c2);
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
Complex add (Complex c1, Complex c2)
{
    Complex c;
    c.x=c1.x+c2.x;
    c.y=c1.y+c2.y;
    return c;
}
void main()
{
    clrscr();
    Complex c1;
    Complex c2;
    Complex c3;
    c1.read();
    c2.read();
    c3=add(c1,c2);
    c3.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

- Here in the class "Distance", we have mentioned that the operator overloading function is a friend to this class. This function is then defined outside the class. This function will accept two objects of the class "Complex". This function also returns an object of the same class i.e. "Complex".

Syllabus Topic : Static Data Members, Static Member Function

5.6 Static Members

➔ **(W-14, W-16, W-17)**

Q. 5.6.1	Write any two characteristics of static member function. (Ref. Sec. 5.6) **(W-14, W-16, 4 Marks)**
Q. 5.6.2	Explain the need of static member function with example. (Ref. Sec. 5.6 and Prog. 5.6.1) **(W-17, 4 Marks)**

- Static members are common for the entire class and not for each object of the class.
- We can have static data members as well as static function members.
- We will see the static data member in this class and static function members will be dealt with in chapter 7.
- Static data member will have same value for all the objects. Hence if one object alters the value of static data member, its altered value will be available for all other objects
- Let us see a program example of static data member.

➢ **Program 5.6.1 : Write a program to define a class having data members principle, duration and rate-of-interest. Declare rate-of-interest as static member variable. Calculate the simple interest and display it for one object.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Interest
{
    private:
    static float rate;
    float principal,duration,sim_int;
    public:
    void accept()
    {
        cout<<"Enter principal amount and duration:";
        cin>>principal>>duration;
    }
    void calculate()
    {
        sim_int=principal*rate*duration/100;
    }
    void display()
    {
        cout<<"Simple Interest="<<sim_int;
    }
};
float Interest::rate=10;
void main()
{
    Interest i;
    clrscr();
    i.accept();
    i.calculate();
    i.display();
    getch();
}
```

Output

```
Enter principal amount and duration:1000
5
Simple Interest=500
```

Syllabus Topic : Array of Objects

5.7 Array of Objects

➔ **(S-14, W-14, W-16, S-17)**

- When multiple objects of a class are to be made we can use array of objects.
- Creating array of objects is quite similar to creating array of any other data type.
- Syntax of creating array of objects
- class_name object_array_name[size];
- Each object in the array can then be accessed using the indices from 0 to n-1, in the same manner as we have done for integer arrays.
- Let us take an example of array of objects.

➢ **Program 5.7.1 : Write a program to declare a class 'staff' having data members as name and department. Accept this data for 10 staffs and display names of staff that are in cm department.** **S-14, 8 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Staff
{
    private:
    char name[20],dept[20];
    public:
    void accept()
    {
        cout<<"Enter name and department:";
        cin>>name>>dept;
    }
    void display()
    {
        cout<<name<<"\t"<<dept<<"\n";
    }
};
void main()
{
    int i;
    clrscr();
    Staff s[10];
    for(i=0;i<=9;i++)
    {
        s[i].accept();
    }
    cout<<"Name\tDepartment\n";
    for(i=0;i<=9;i++)
    {
        s[i].display();
    }
    getch();
}
```

Output

```
Enter name and department:a
finance
Enter name and department:Ajay
HR
Enter name and department:Vijay
HR
Enter name and department:Jay
Finance
Enter name and department:Digvijay
Finance
Enter name and department:Manoj
Sales
Enter name and department:Manish
Sales
Enter name and department:Suresh
Finance
Enter name and department:Dinesh
HR
Enter name and department:Saurav
HR
Name      Department
a         finance
Ajay      HR
Vijay     HR
Jay       Finance
Digvijay  Finance
Manoj     Sales
Manish    Sales
Suresh    Finance
Dinesh    HR
Saurav    HR
```

➢ **Program 5.7.2 : Write a program to declare a class staff having data members as name and post. Accept and display data for five staff members. (Using array of object)** **W-14, 8 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Staff
{
    private:
    char name[20],post[20];
    public:
    void accept()
    {
        cout<<"Enter name and post:";
        cin>>name>>post;
    }
    void display()
    {
        cout<<name<<"\t"<<post<<"\n";
    }
};
void main()
{
    int i;
    clrscr();
    Staff s[5];
    for(i=0;i<=4;i++)
    {
        s[i].accept();
    }
    cout<<"Name\tPsot\n";
    for(i=0;i<=4;i++)
    {
        s[i].display();
    }
    getch();
}
```

Output :

```
Enter name and post:Jay
Manager
Enter name and post:Ajay
Officer
Enter name and post:Vijay
Officer
Enter name and post:Suresh
Clerk
Enter name and post:Dinesh
Clerk
Name    Psot
Jay     Manager
Ajay    Officer
Vijay   Officer
Suresh  Clerk
Dinesh  Clerk
```

➢ **Program 5.7.3 : Write a program to declare class Account having data member as acc_no and balance. Accept and display data for five object using pointer to array of object.** **S-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Account
{
    private:
    int acc_no,balance;
    public:
    void get_data()
    {
        cout<<"Enter account number and balance:";
        cin>>acc_no>>balance;
    }
    void display()
    {
        cout<<acc_no<<"\t"<<balance<<endl;
    }
};
void main()
{
```

```
    Account b[5];
    int i;
    clrscr();
    for(i=0;i<=4;i++)
    {
        b[i].get_data();
    }
    cout<<"Acc No\t\Balance\n";
    for(i=0;i<=4;i++)
    {
        b[i].display();
    }
    getch();
}
```

Output :

```
Enter account number and balance:2
3
Enter account number and balance:1
2
Enter account number and balance:123
5000
Enter account number and balance:456
10000
Enter account number and balance:789
1000
Acc No   Balance
2          3
1          2
123      5000
456     10000
789      1000
```

➢ **Program 5.7.4 : Write a program to declare a class mobile having data members as price and model number. Accept and display this data for ten objects.**

W-16, 4 Marks

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Mobile
{
    private:
    char model[20];
    int price;
    public:
    void accept()
    {
        cout<<"Enter model name and price:";
        cin>>model>>price;
    }
    void display()
    {
        cout<<model<<"\t"<<price<<"\n";
    }
};
void main()
{
    int i;
    clrscr();
    Mobile s[10];
    for(i=0;i<=9;i++)
    {
        s[i].accept();
    }
    cout<<"Model\tPrice\n";
    for(i=0;i<=9;i++)
    {
        s[i].display();
    }
    getch();
}
```

Output :

```
Enter model name and price:s1
2000
Enter model name and price:s2
```

```
3000
Enter model name and price:s3
4000
Enter model name and price:s4
5000
Enter model name and price:s5
6000
Enter model name and price:s6
7000
Enter model name and price:s7
8000
Enter model name and price:s8
9000
Enter model name and price:s9
10000
Enter model name and price:s10
11000
Model   Price
s1      2000
s2      3000
s3      4000
s4      5000
s5      6000
s6      7000
s7      8000
s8      9000
s9      10000
s10     11000
```

Syllabus Topic : Objects as Function Arguments

5.8 Objects as Function Arguments

➔ **(W-14, S-16)**

Q. 5.8.1	Explain object as function argument. (Ref. Sec. 5.8)	**(W-14, 4 Marks)**
Q. 5.8.2	Explain with example object as member function argument. (Ref. Sec. 5.8)	**(S-16, 4 Marks)**

- Objects can also be passed and returned by a function. The syntax of passing the arguments is same as primitive data types.
- In the following example an object of complex number is passed as well as returned by the function. The function adds two complex numbers and returns the result

➢ **Program 5.8.1: Write a program to add two complex numbers using objects of a class Complex.**

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
    Complex add (Complex c)
    {
        Complex c1;
        c1.x=x+c.x;
        c1.y=y+c.y;
        return c1;

    }
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
void main()
{
```

```
    clrscr();
    Complex c1;
    Complex c2;
    Complex c3;
    c1.read();
    c2.read();
    c3=c1.add(c2);
    c3.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

5.9 Exam Pack (MSBTE and Review Questions)

☞ **Syllabus Topic : Class and Object : Introduction**

Q. 1 What is object oriented programming? How it is different from procedure oriented programming?
(Refer Section 5.1) **(4 Marks)**

Q. 2 Differentiate between structure and class.
(Refer Section 5.1) **(W-14, 4 Marks)**

Q. 3 Compare structure and class.
(Refer Section 5.1) **(S-16, 4 Marks)**

Q. 4 What are the differences between structure and class ? **(Refer Section 5.1)** **(W-17, 4 Marks)**

Q. 5 What is difference between procedure oriented programming and object oriented programming ?
(Refer Section 5.1) **(W-17, 4 Marks)**

☞ **Syllabus Topic : Class and Object : Introduction**

Q. 6 What is object-oriented programming ? List basic concepts of it ? Explain each concept.
(Refer Section 5.1.1) **(10 Marks)**

Q. 7 Explain the benefits of object oriented programming.
(Refer Section 5.1.1) **(8 Marks)**

Q. 8 Distinguish between Data abstraction and data encapsulation.
(Refer Section 5.1.1) **(5 Marks)**

Q. 9 What is Data abstraction?
(Refer Section 5.1.1) **(S-14, 2 Marks)**

Q. 10 Define the following terms :
(1) Object (2) Class
(Refer Section 5.1.1) **(S-14, 2 Marks)**

Q. 11 State any four features of OOP.
(Refer Section 5.1.1) **(S-14, 4 Marks)**

Q.12 Define class with it's syntax.
(Refer Section 5.1.1) **(W-14, W-16, 2 Marks)**

Q. 13 Enlist any four concepts of OOP.
(Refer Section 5.1.1) **(W-16, 2 Marks)**

Q. 14 What are the features of object oriented programming ?
(Refer Section 5.1.1) **(W-16, 4 Marks)**

Q. 15 Write down characteristics of object oriented programming.
(Refer Section 5.1.1) **(W-17, 4 Marks)**

☞ **Syllabus Topic : Creating Objects and Memory Allocation of Objects**

Q. 16 Explain how memory is allocated to an object of a class with diagram.
(Refer Section 5.1.4) **(W-14, W-16, 4 Marks)**

Q. 17 How memory is allocated when multiple object of class are created ? Explain with example.
(Refer Section 5.1.4) **(S-17, 4 Marks)**

Q. 18 Explain memory allocation for object with example.
(Refer Section 5.1.4) **(S-15, 4 Marks)**

☞ **Syllabus Topic : Defining Member Functions**

Q. 19 How many ways we can define member function in class ? Give it's syntax.
(Refer Section 5.2) **(W-14, 4 Marks)**

Q. 20 Explain syntax for declaring the function inside the class and outside the class with example.

(Refer Sections 5.2.1, 5.2.2 and Program 5.2.1)

(S-15, 4 Marks)

Q. 21 Give syntax for defining a member function inside and nesting of functions in a class with example.

(Refer Sections 5.2.1, 5.2.1.1 and Program 5.2.5)

(W-16, 4 Marks)

Q. 22 Write a program to declare a class 'student' having data members as name and percentage. Write a constructor to initialize these data members accept and display data for one student. **(W-16, 4 Marks)**

Q. 23 What is nesting of member function ? Give one example.

(Refer Section 5.2.1.1 and Program 5.2.5)

(S-16, 4 Marks)

Q. 24 How function is defined outside of class, write general syntax and example of same.

(Refer Section 5.2.2 and Program 5.2.6)

(W-15, 4 Marks)

Q. 25 Write general form of member function, definition out of class. **(Refer Section 5.2.2) (S-16, 2 Marks)**

Q. 26 How user can declare member function outside the class ? **(Refer Section 5.2.2) (W-17, 2 Marks)**

Q. 27 What are inline functions ? Discuss its advantages and disadvantages. Write a program to demonstrate inline function. **(Refer Section 5.2.3) (10 Marks)**

Q. 28 Write short note on : Inline function.

(Refer Section 5.2.3) (5 Marks)

Q. 29 Explain inline function with an example.

(Refer Section 5.2.3) (10 Marks)

Q. 30 What do you mean by inline function? Write its syntax and example.

(Refer Section 5.2.3 and Program 5.2.8)

(W-16, 4 Marks)

Q. 31 Write a program to declare a class 'student' having data members as name and percentage. Write a constructor to initialize these data members accept and display data for one student.

(Refer Program 5.2.4) (W-16, 4 Marks)

☞ **Syllabus Topic : Concepts of Constructors, Types of Constructors**

Q. 32 Explain the concept of constructor in C++.

(Refer Section 5.3) (5 Marks)

Q. 33 What is constructor? Why is it called so? What are the syntax rules for writing a constructor? Describe inline constructor.

(Refer Section 5.3) (10 Marks)

Q. 34 Illustrate the concept of constructor with default argument with suitable example.

(Refer Section 5.3) (S-14, 4 Marks)

Q. 35 State any four characteristics of constructor.

(Refer Section 5.3) (S-14, 4 Marks)

Q. 36 List four types of constructors.

(Refer Section 5.3) (W-15, 2 Marks)

Q. 37 State any four characteristics of constructor.

(Refer Section 5.3) (W-15, 4 Marks)

Q. 38 Define constructor. State any two type of constructor. **(Refer Section 5.3) (W-16, 2 Marks)**

Q. 39 Explain the concept of constructor with default argument. **(Refer Sections 5.3 and 5.3.1)**

(W-16, 4 Marks)

Q. 40 Enlist types of constructor.

(Refer Section 5.3) (W-17, 2 Marks)

Q. 41 What is constructor ? How user can declared constructor in derived class ? Explain with example.

(Refer Section 5.3 and Program 5.3.1)

(W-17, 8 Marks)

Q. 42 What is copy constructor ? Give one programming example of it.

(Refer Section 5.3.2) (8 Marks)

Q. 43 What is parameterized constructor ? Give the syntax and example of it.

(Refer Section 5.3.2) (S-14, 4 Marks)

Q. 44 What is parametrized constructor ? Explain with example.

(Refer Section 5.3.2 and Program 5.3.3)

(W-17, 4 Marks)

☞ **Syllabus Topic : Multiple Constructors in a Class**

Q. 45 What is copy constructor ?

(Refer Section 5.3.3) **(2 Marks)**

Q. 46 Explain overloaded constructor in a class with suitable example.

(Refer Section 5.3.3) **(W-14, 8 Marks)**

Q. 47 Explain multiple constructor in class with example.

(Refer Section 5.3.3) **(S-16, 4 Marks)**

Q. 48 Explain multiple constructor in class. Give suitable example.

(Refer Section 5.3.3 and program 5.3.5)

(S-17, 4 Marks)

Q. 49 What is copy constructor ? Explain with example.

(Refer Section 5.3.3 and Program 5.3.5)

(W-17, 4 Marks)

Q. 50 What do you mean by overloaded constructor ?

(Refer Section 5.3.3) **(W-17, 2 Marks)**

☞ **Syllabus Topic : Constructors with Default Arguments**

Q. 51 State the use of default parameters in constructor with example.**(Refer Section 5.3.4) (S-16, 4 Marks)**

Q. 52 What do you mean by default argument ? Illustrate concept of constructor with default argument using suitable example.

(Refer Section 5.3.4 and Program 5.3.7)

(S-17, 4 Marks)

Q. 53 Explain constructor with default argument.

(Refer Section 5.3.4) **(W-14, 4 Marks)**

☞ **Syllabus Topic : Destructors**

Q. 54 Explain the concept of Destructor in C++.

(Refer Section 5.4) **(5 Marks)**

Q. 55 State the characteristics of destructor.

(Refer Section 5.4) **(S-14 , 2 Marks)**

Q. 56 What is destructor? Give its syntax and example.

(Refer Section 5.4) **(W-16, 4 Marks)**

Q. 57 What is destructor ? How destructor is declared ? When destructor is invoked ?

(Refer Section 5.4 and program 5.4.1)

(W-17, 4 Marks)

☞ **Syllabus Topic : Friend Function**

Q. 58 What are the characteristics of a friend function ?

(Refer Section 5.5) **(5 Marks)**

Q. 59 Explain the concept of friend function.

(Refer Section 5.5) **(S-14, 4 Marks)**

Q. 60 Write any two rules to define friend function.

(Refer Section 5.5) **(W-14, W-16, 2 Marks)**

Q. 61 Define friend function. Write syntax of declaring it.

(Refer Section 5.5 and Program 5.5.1)

(W-14, 4 Marks)

Q. 62 What is friend function ? Explain with example.

(Refer Section 5.5 and Program 5.5.1)

(W-17, 4 Marks)

Q. 63 Write a program to declare class 'complex' with data members x and y. Write member function to read and print data members. Read x and y for two objects and add these two objects into third object using friend function.

(Refer Program 5.5.2) **(S-16, 8 Marks)**

☞ **Syllabus Topic : Static Data Members**

Q. 64 Write any two characteristics of static member function.

(Refer Section 5.6) **(W-14, W-16, 4 Marks)**

Q. 65 Explain the need of static member function with example.

(Refer Section 5.6 and Program 5.6.1)

(W-17, 4 Marks)

☞ **Syllabus Topic : Array of Objects**

Q. 66 Write a program to declare a class 'staff' having data members as name and department. Accept this data for 10 staffs and display names of staff that are in cm department.

(Refer Program 5.7.1) **(S-14, 8 Marks)**

Q. 67 Write a program to declare a class staff having data members as name and post. Accept and display data for five staff members. (Using array of object). **(Refer Program 5.7.2)** **(W-14, 8 Marks)**

Q. 68 Write a program to declare a class mobile having data members as price and model number. Accept and display this data for ten objects. **(Refer Program 5.7.4)** **(W-16, 4 Marks)**

Q. 69 Write a program to declare class Account having data member as acc_no and balance. Accept and display data for five object using pointer to array of object. **(Refer Program 5.7.3)** **(S-17, 4 Marks)**

☞ **Syllabus Topic : Objects as Function Arguments**

Q. 70 Explain object as function argument. **(Refer Section 5.8)** **(W-14, 4 Marks)**

Q. 71 Explain with example object as member function argument. **(Refer Section 5.8)** **(S-16, 4 Marks)**

❑❑❑

UNIT III

CHAPTER 6

Inheritance

Syllabus

Extending Classes using Inheritance

- **Unit Outcomes**
  - Explain given type of inheritance based on its characteristic.
  - Implement given type of inheritance in C++ program.
  - Write C++ program using virtual base class.
  - Use constructor in the given derived class.
- **Topics and Sub-Topics**
  - Introduction to Inheritance, defining a derived class, Visibility Modes and Effects .
  - Types of Inheritance – Single, multilevel, multiple, hierarchical, hybrid.
  - Constructors in derived class.

Syllabus Topic : Introduction to Inheritance, Defining a Derived Class

6.1 Inheritance

➔ **(S-14, S-16, W-16)**

Q. 6.1.1 Define inheritance and enlist it's types. (Ref. Sec. 6.1 and 6.1.1) **(S-14, S-16, 2 Marks)**

Q. 6.1.2 What is inheritance? Why inheritance used in C++? (Ref. Sec. 6.1) **(W-16, 2 Marks)**

- The mechanism of deriving a class from another class is known as inheritance.
- The class from which another class is derived is called as the base class while the class which is derived is called as derived class.

Syllabus Topic : Visibility Modes and Effects

6.1.1 Visibility Modes and Effects

➔ **(S-14, W-14, S-15, W-15, W-16, S-16,S-17)**

Q. 6.1.3 Explain different visibility modes used in inheritance. (Ref. Sec. 6.1.1) **(W-14, S-15, W-16, W-17, 4 Marks)**

Q. 6.1.4 State different types of inheritance with diagram. (Ref. Sec. 6.1.1) **(W-14, 4 Marks)**

Q. 6.1.5 State general format of defining derived class. (Ref. Sec. 6.1.1) **(W-15, 2 Marks)**

Q. 6.1.6 State different types of inheritances and describe any one. (Ref. Sec. 6.1.1) **(W-15, 4 Marks)**

Q. 6.1.7 List types of inheritance. (Ref. Sec. 6.1.1) **(S-16, 2 Marks)**

Q. 6.1.8 State and explain various visibility modifiers in inheritance. (Ref. Sec. 6.1.1) **(S-16, W-16, 4 Marks)**

Q. 6.1.9 Define derived class. Give one example. (Ref. Sec. 6.1.1 and Prog. 6.1.1) **(S-17, 2 Marks)**

Q. 6.1.10 State any two access specifier with example. (Ref. Sec. 6.1.1) **(S- 15, W-16, 2 Marks)**

Q. 6.1.11 Explain various types of inheritance with example (Ref. Secs. 6.1.1 to 6.1.5 and 6.1.7) **(W-16, 8 Marks)**

Q. 6.1.12 How protected access specifier is different from private? (Ref. Sec. 6.1.1) **(W-15, 4 Marks)**

Q. 6.1.11 What are the different types of inheritance? Explain each with example. (Ref. Secs. 6.1.1 to 6.1.5 and 6.1.7) **(W-16, 10 Marks)**

- The members of a class may be public, protected or private and the class may also be derived in the following ways :

1. Public
2. Protected
3. Private

- These keywords viz. public, protected and private are called as "access specifiers", as they decide the access of a member.
- The visibility of base class members within the derived class is as shown in Table 6.1.1

Table 6.1.1 : Visibility of base class members in the derived class

Base class member visibility	Derived class visibility		
	Public derivation	Protected derivation	Private derivation
Public members	Public	Protected	Private
Protected members	Protected	Protected	Private
Private members	Not inherited	Not inherited	Not inherited

- Hence, you will notice in the Table 6.1.1, in case of public derivation the public and protected members of the base class remain in the same visibility even in the derived class. The private members are never derived.
- In case of protected derivation the public and protected members of the base class become protected in the derived class. The private members are never derived.
- In case of private derivation the public and protected members of the base class become private in the derived class. The private members are never derived.
- There are various types of inheritance namely :

1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance
4. Hybrid Inheritance
5. Hierarchical Inheritance

- To derive a class from another we need to use the following syntax:

```
class derived_class_name : access_specifier
base_class_name
```

- Let us see these types of inheritance in the subsequent sub-sections.

Syllabus Topic : Single Type of Inheritance

6.1.2 Single Inheritance

➔ (W-15, W-16, S-17)

- In this case only one class is derived from another class.
- The class diagram and a program example is given below.

➢ **Program 6.1.1 : Write a program to add two numbers using single inheritance such that the base class function must accept the two numbers from the user and the derived class function must add these numbers and display the sum.** **W-16, S-17, 4 Marks**

Solution :

The class diagram of this requirement is as shown in Fig. P. 6.1.1. This diagram shows that the class "Sum" is derived from class "Data" in public visibility.

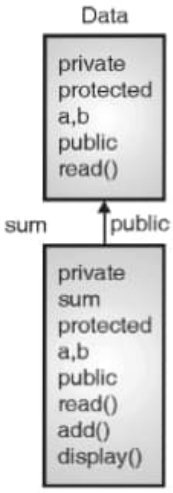


Fig. P. 6.1.1 : Class Diagram of Single Inheritance for Program 6.1.1

```
# include<iostream.h>
# include<conio.h>
class Data
{
    protected:
    int a,b;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>a>>b;
    }
};
class Sum: public Data
{
    private:
    int sum;
    public:
    void add()
    {
        sum=a+b;
    }
    void display()
    {
        cout<<"The sum is "<<sum;
    }
};
void main()
{
    clrscr();
    Sum s;
    s.read();
    s.add();
    s.display();
    getch();
}
```

Output :

```
Enter two numbers4
5
The sum is 9
```

Explanation :

- As shown in the class diagram, the base class data has two protected member variables namely 'a' and 'b'. These members are kept protected, so that they are accessible from the derived class function to calculate the sum.
- The base class also has a function in public visibility to read the inputs from user. The derived class also has public member functions add() and display() to find the sum of the two numbers taken and to display this sum respectively.
- The class "Sum" is derived from the class "Data". This derivation is implemented using the already discussed syntax in the statement:

  class Sum: public Data
- The members of the base class are shown available in the derived class in the class diagram in Fig. P.6.1.1 but these member functions and member variables are not to be defined again in the derived class when actually writing the program. This concept of gaining access to the members of the base class is called as **code reusability**.
- The main() function has first a creation of the object of the class "Sum". The object of the class sum can access all the functions i.e. read(), add() and display(); because the derived class derives the protected members in the same visibility when derived publicly as discussed in Table 6.1.1.

– The functions to read, calculate the sum and display the sum are thereafter called one by one.

➢ **Program 6.1.2 : Write a program to find the area of circle using single inheritance such that the base class function must accept the radius from the user and the derived class function must calculate and display the area.**

Solution : The class diagram of this requirement is as shown in Fig. P. 6.1.2. This diagram shows that the class "Area" is derived from class "Data" in public visibility.

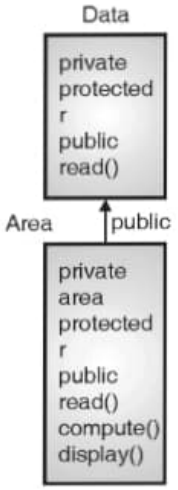


Fig. P. 6.1.2 : Class Diagram of Single Inheritance for Program 6.1.2

```
# include<iostream.h>
# include<conio.h>
class Data
{
    protected:
    int r;
    public:
    void read()
    {
        cout<<"Enter the radius";
        cin>>r;
    }
};
class Area: public Data
{
    private:
    float result;
    public:
    void compute()
    {
        result=3.14*r*r;
    }
    void display()
    {
        cout<<"The area of the circle is "<<result;
    }
};
void main()
{
    clrscr();
    Area a;
    a.read();
    a.compute();
    a.display();
    getch();
}
```

Output :

```
Enter the radius5
The area of the circle is 78.5
```

Explanation :

A program with similar logic, just that instead of calculating the sum we have to calculate the area of a circle.

➢ **Program 6.1.3 : Write a program to implement single inheritance from following Fig. P. 6.1.3 accept and display the data for one table.** **W-15, 4 Marks**

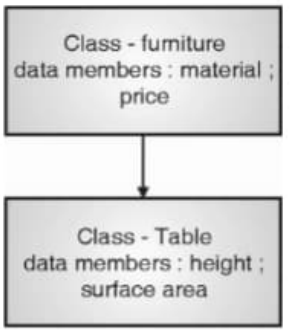


Fig. P. 6.1.3 : Class diagram of single inheritance for Program 6.1.3

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Furniture
{
    protected:
    char material[20];
    int price;
};
class Table:public Furniture
{
    private:
    int height,area;
    public:
    void accept()
    {
        cout<<"Enter material, price, height and area:";
        cin>>material>>price>>height>>area;
    }
    void display()
    {

    cout<<"Material:"<<material<<"\nPrice:"<<price<
<"\nHeight:"<<height<<"\nArea:"<<area;
    }
};
void main()
{
    Table t;
    clrscr();
    t.accept();
    t.display();
    getch();
}
```

Output :

```
Enter material, price, height and area:Plywood
1500
35
50
Material:Plywood
Price:1500
Height:35
Area:50
```

➢ **Program 6.1.4 : Write a program to implement single inheritance. Declare base class 'Employee' with emp_no and emp_name. Declare derived class 'Fitness' with height and weight. Accept and display data for one employee.** **W-16, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Employee
{
    protected:
    int no;
    char name[20];
};
class Fitness:public Employee
{
    private:
    int height,weight;
    public:
    void get_data()
    {
        cout<<"Enter name, number height and weight:";
        cin>>name>>no>>height>>weight;
    }
    void display()
    {

    cout<<"Name:"<<name<<"\nNumber:"<<no<<"\n
Height:"<<height<<"\nWeight:"<<weight;
    }
};
void main()
```

```
{
    Fitness b;
    clrscr();
    b.get_data();
    b.display();
    getch();
}
```

Output :

```
Enter name, number height and weight:Ajay
24
175
67
Name:Ajay
Number:24
Height:175
Weight:67
```

Syllabus Topic : Constructor in Derived Class

6.1.2.1 Constructor in Derived Class

➔ **(S-15, W-15, S-16, W-17)**

➢ **Program 6.1.5 : Write a program to demonstrate constructor in derived class with respect to order of calling constructor and passing parameters to base class constructor.** **S-15, W-15, S-16, W-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Base
{
    public:
    Base(int a)
    {
        cout<<"Base class constructor:"<<a<<endl;
    }
};
class Sub:public Base
{
    public:
    Sub(int b):Base(b)
    {
        cout<<"Sub class constructor:"<<b<<endl;
    }
};
void main()
{
    clrscr();
    Sub s(5);
    getch();
}
```

Output :

```
Base class constructor:5
Sub class constructor:5
```

Syllabus Topic : Multi Level Type of Inheritance

6.1.3 Multi Level Inheritance

➔ **(S-14, W-14, S-15, S-16, W-16)**

Q. 6.1.13 Differentiate between multiple inheritance and multilevel inheritance.
(Ref. Secs. 6.1.3 and 6.1.4) **(S-15, 4 Marks)**

	Multiple inheritance	Multi level inheritance
Definition	Multiple inheritance is an inheritance type were a class inherits from more than one base class.	Multi level inheritance is an inheritance type that inherits from derived class making that derived class a base class for a new class.
Usage	Multiple inheritance is not widely used because it makes the system more complex.	Multi level inheritance is widely used.
Class level	Multiple inheritance has two class levels namely base class and derived class.	Multi level inheritance has three class levels namely base class intermediate class and derived class.

- In this case one class is derived from a class which is also derived from another class.
- The class diagram of such a inheritance can be as shown in Fig. 6.1.1.

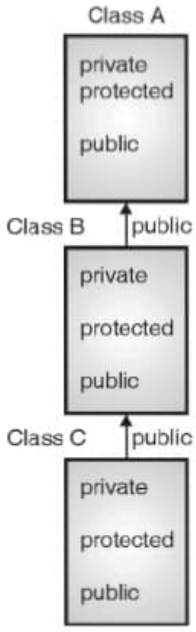


Fig. 6.1.1(a) : Multi level inheritance

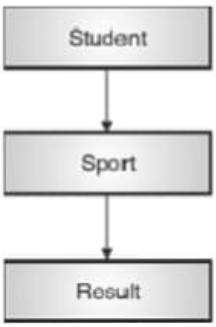


Fig. 6.1.1(b) : Multi level inheritance

- In this case, class 'C' is derived from class 'B' which is derived from class 'A'.
- Let us see some program examples using multilevel inheritance.

➢ **Program 6.1.6 : Write a program to calculate percentage of a student using multi level inheritance. The base class function will accept the marks in three subjects from user. A class will be derived from the above mentioned class that will have a function to find the total marks obtained and another class derived from this will have functions to calculate and display the percentage scored.** **S-16, 4 Marks**

Solution :

The class diagram of this requirement is as shown in Fig. P. 6.1.6.

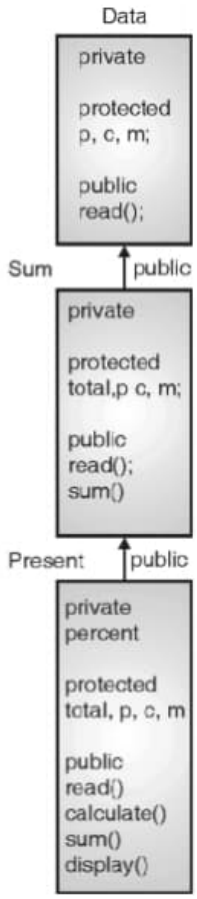


Fig. P. 6.1.6 : Class Diagram of Multi Level Inheritance for Program 6.1.6

```
# include<iostream.h>
# include<conio.h>
class Data
{
    protected:
    int p,c,m;
    public:
    void read()
    {
        cout<<"Enter the marks obtained in Physics,
Chemistry and Maths";
        cin>>p>>c>>m;
    }
};
class Sum: public Data
{
```

```
    protected:
    int total;
    public:
    void sum()
    {
        total=p+c+m;
    }
};
class Percent: public Sum
{
    private:
    float percent;
    public:
    void calculate()
    {
        percent=total/300.0*100;
    }
    void display()
    {
        cout<<"The percentage is "<<percent;
    }
};
void main()
{
    clrscr();
    Percent a;
    a.read();
    a.sum();
    a.calculate();
    a.display();
    getch();
}
```

Output :

```
Enter the marks obtained in Physics, Chemistry and Maths90
98
99
The percentage is 95.666664
```

Explanation :

- The most important part of a program based on inheritance is the class diagram. Once you draw the class diagram then writing the program to implement the diagram is very simple.
- This program is also just an implementation of the class diagram drawn in the Fig. P. 6.1.6. The implementation is similar to that as seen in single inheritance, only that there is one more level of inheritance.

➢ **Program 6.1.7 : Write a program to calculate volume of sphere using multi level inheritance. The base class function will accept the radius from user. A class will be derived from the above mentioned class that will have a function to find the area of a circle and another class derived from this will have functions to calculate and display the volume of the sphere.**

Solution :

The class diagram of this requirement is as shown in Fig. P. 6.1.7.

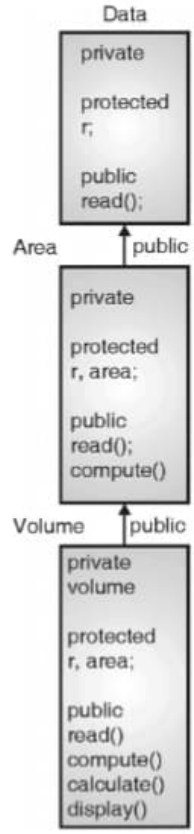


Fig. P. 6.1.7 : Class Diagram of Multi Level Inheritance for Program 6.1.7

```
# include<iostream.h>
# include<conio.h>
class Data
{
    protected:
    float r;
    public:
    void read()
    {
        cout<<"Enter the radius of the circle: ";
        cin>>r;
    }
};
class Area: public Data
{
    protected:
    float area;
    public:
    void compute()
    {
        area=3.14*r*r;
    }
};
class Volume: public Area
{
    private:
    float volume;
    public:
    void calculate()
    {
        volume=4*area*r/3;
    }
    void display()
    {
        cout<<"The volume of the sphere is "<<volume;
    }
};
void main()
{
    clrscr();
    Volume v;
    v.read();
    v.compute();
    v.calculate();
    v.display();
    getch();
}
```

Output :

```
Enter the radius of the circle: 5
The volume of the sphere is 523.333313
```

Explanation :

The implementation of the class diagram shown in Fig. P. 6.1.7 is done in similar manner as for the previous program.

➢ **Program 6.1.8 : Identify the type of inheritance and implement it by writing a program for the following Fig. P. 6.1.8. Assume suitable member functions.**

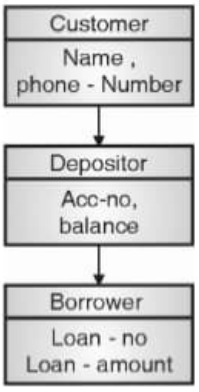


Fig. P. 6.1.8 : Class diagram of multi level inheritance for Program 6.1.8

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Customer
{
    protected:
    char name[20];
    float phone;
```

```
};
class Depositor:public Customer
{
    protected:
    float acc_no,balance;
};
class Burrower:public Depositor
{
    private:
    float loan_no,amount;
    public:
    void accept()
    {
        cout<<"Enter name, phone number,
 account number, balance, loan number and loan amount:";
        cin>>name>>phone>>acc_no>>balance
>>loan_no>>
     amount;
    }
    void display()
    {
    cout<<"Name:"<<name<<"\nPhone
Number:"<<phone
<<"\nAccount
number:"<<acc_no<<"\nBalance:"<<balance
<<"\nLoan number:"<<loan_no<<"\nLoan amount:"
<<amount;
    }
};
void main()
{
    clrscr();
    Burrower b;
    b.accept();
    b.display();
    getch();
}
```

Output :

```
Enter name, phone number, account number, balance, loan
number and loan amount:a
bc
123456
111222
100000
45678
200000
Name:abc
Phone Number:123456
Account number:111222
Balance:100000
Loan number:45678
Loan amount:200000
```

➢ **Program 6.1.9 : Identify the type of inheritance shown in following Fig. P. 6.1.9. Implement it by using suitable member function.** **W-14, S-16, 8 Marks**

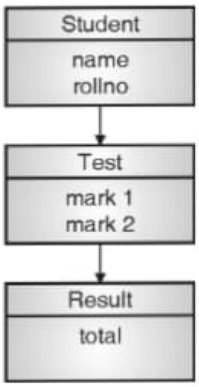


Fig. P. 6.1.9 : Class diagram of multi level inheritance for Program 6.1.9

```
#include<iostream.h>
#include<conio.h>
class student
{
    protected :
        int roll_no ;
    public :
        void get_no (int) ;
        void put_no (void) ;
```

```
};
void student :: get_no ( int a)
{
        roll_no = a ;
}
void student :: put_no ( )
{
        cout << "Roll Number :" << roll_no << "\n" ;
}
class test : public student            // 1st level derivation
{
        protected :
          float sub1 ;
          float sub2;
        public :
          void get_marks (float, float);
          void put_marks (void);
};
void test :: get_marks (float x, float y)
{
        sub1 = x ;
        sub2 = y ;
}
void test :: put_marks ( )
{
   cout << "Marks in subject 1 =" << sub1 << "\n" ;
   cout << "Marks is subject 2 =" << sub2 << "\n" ;
}
class result : public test             // 2nd level derivation
{
          float total ;
        public :
          void display (void) ;
};
void result :: display ( )
{
        total = sub1 + sub2 ;
        put_no ( ) ;
        put_marks ( ) ;
        cout << "Total :" << total  << "\n" ;
}
main ( )
{
        result student1 ;              // student is created
        clrscr( );
        student1.get_no (10) ;
        student1.get_marks (75.0 , 59.5) ;
        student1.display ( ) ;
        getch ( );
        return 0;
}
```

Output :

```
Roll Number :10
Marks in subject 1 =75
Marks is subject 2 =59.5
Total :134.5
```

Syllabus Topic : Multiple Type of Inheritance

6.1.4 Multiple Inheritance

➔ **(S-14, S-15, S-16, W-16, S-17, W-17)**

Q. 6.1.14 Describe multiple inheritance with suitable example. (Ref. Sec. 6.1.4 and Prog. 6.1.10) **(S-14, W-16, 4 Marks)**

Q. 6.1.15 Differentiate between multiple inheritance and multilevel inheritance. (Ref. Secs. 6.1.3 and 6.1.4) **(S-15, 4 Marks)**

Q. 6.1.16 Explain Multiple inheritance. (Ref. Sec. 6.1.4) **(S-14, W-16, S-17, W-17, 4 Marks)**

- In this case one class is derived from multiple classes.
- The class diagram of such a inheritance can be as shown in Fig. 6.1.2.

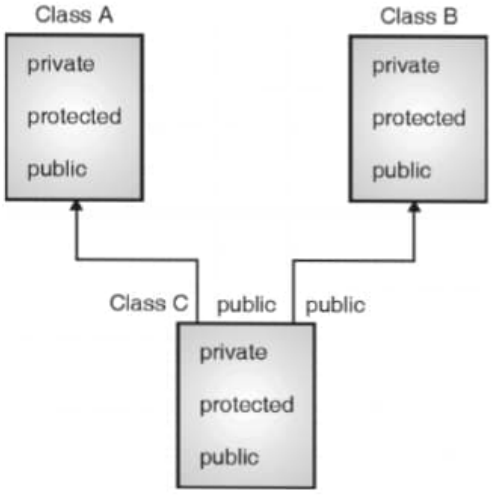


Fig. 6.1.2 : Multiple inheritance

- In this case, class 'C' is derived from class 'B' as well as from class 'A'.
- To derive a class from multiple classes we need to use the following syntax :

class derived_class_name : access_specifier base_class_name,access_specifier base_class_name2,access_specifier base_class_name3.

- Hence, as seen from the above syntax, a class can be derived from whatever number of classes as required.
- Let us see some program examples using multiple inheritance.

➢ **Program 6.1.10 : Write a program to define the following relationship using multiple inheritance.**

S-16, W-16, 4 Marks

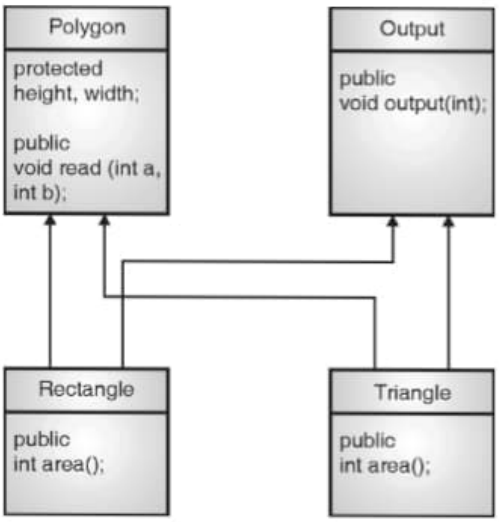


Fig. P. 6.1.10 : Class Diagram of Multiple Inheritance for Program 6.1.10

```
# include<iostream.h>
# include<conio.h>
class Polygon
{
    protected:
    int height, width;
    public:
    void read(int a, int b)
    {
        height=a;
        width=b;
    }
};
class Output
{
    public:
    void output(int x)
    {
        cout<<"Area is "<<x;
    }
};
class Rectangle: public Polygon, public Output
{
    public:
    int area()
    {
        return(height*width);
    }
};
class Triangle: public Polygon, public Output
{
    public:
    int area()
    {
        return(height*width/2);
    }
};
void main()
```

```
{
    clrscr();
    int h,w,choice,a;
    cout<<"1. Area of Rectangle
    \n2. Area of Triangle\nEnter your choice:";
    cin>>choice;
    cout<<"Enter height and width:";
    cin>>h>>w;
    switch(choice)
    {
        case 1:
        Rectangle r;
        r.read(h,w);
        a=r.area();
        r.output(a);
        break;
        case 2:
        Triangle t;
        t.read(h,w);
        a=t.area();
        t.output(a);
        break;
        default: cout<<"Invalid choice";
    }
    getch();
}
```

Output :

```
1. Area of Rectangle
2. Area of Triangle
Enter your choice:2
Enter height and width:5
4
Area is 10
```

Explanation :

- The most important part of a program based on inheritance is the class diagram. Once you draw the class diagram then writing the program based on the diagram is very simple.
- Here, also we have just implemented the class diagram drawn in the Fig. P. 6.1.10.

Syllabus Topic : Hybrid Type of Inheritance

6.1.5 Hybrid Inheritance

➔ **(S-15, S-16)**

Q. 6.1.17 What is hybrid inheritance ? Give one example. (Ref. Sec. 6.1.5 and Prog. 6.1.11) **(S-16, 4 Marks)**

- In this case there is a mixture of multi level and multiple inheritances. Hence the name is hybrid inheritance.
- The class diagram of such a inheritance can be as shown in Fig. 6.1.3.

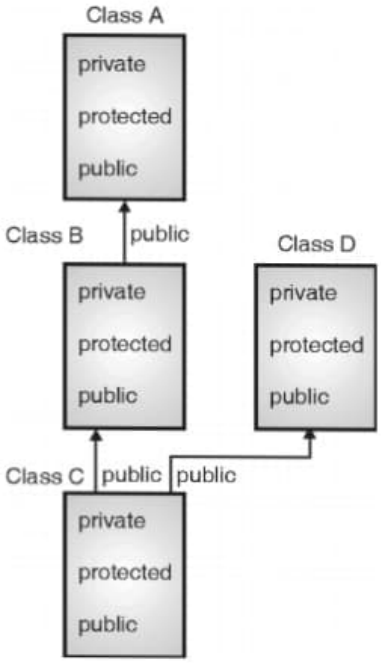


Fig. 6.1.3 : Hybrid inheritance

- In this case, class 'C' is derived from class 'B' which is derived from class 'A'; hence, multi level inheritance. Class 'C' is also derived from class 'D'; hence, multiple inheritance.
- To derive a class from multiple classes we need to use the following syntax:

  class derived_class_name : access_specifier base_class_name,access_specifier base_class_name, access_specifier base_class_name…

- Let us see some program examples using hybrid inheritance.

➢ **Program 6.1.11 : Write a program to define the following relationship using hybrid inheritance.**

S-16, 2 Marks

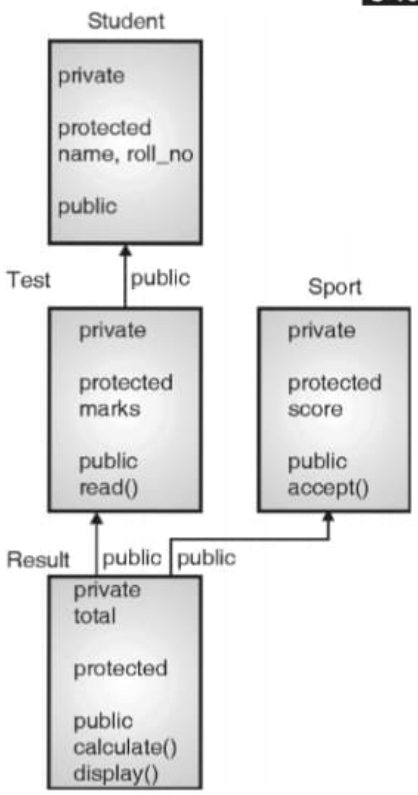


Fig. P. 6.1.11 : Class Diagram of Hybrid Inheritance for Program 6.1.11

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Student
{
    protected:
    char name[20];
    int roll_no;
};
class Test: public Student
{
    protected:
    int marks;
    public:
    void read()
    {
        cout<<"Enter name, roll number and marks
        obtained:";
        gets(name);
        cin>>roll_no>>marks;
    }
};
class Sports
{
    protected:
    int score;
    public:
    void accept()
    {
        cout<<"1. Student has won in national sports
event\n2. Student has not won in any national sports
event\nEnter your choice:";
        cin>>score;
    }
};
class Result: public Test, public Sports
{
    int total;
    public:
    void calculate()
    {
        if(score==1)
        total=marks+15;
        else
        total=marks;
    }
    void display()
    {
        cout<<"The total is "<<total;
    }
};
void main()
{
    clrscr();
    Result r;
    r.read();
    r.accept();
    r.calculate();
    r.display();
    getch();
}
```

Output :

```
Enter name, roll number and marks obtained:Sandeep
24
81
1. Student has won in national sports event
2. Student has not won in any national sports event
Enter your choice:1
The total is 96
```

6.1.6 Problem in Multiple and Hybrid Inheritance

➔ **(W-15)**

- When deriving a class from multiple classes i.e. in Multiple Inheritance or Hybrid Inheritance, there is a problem. The members of the two base classes from which a class is derived, may be same. Hence, there is an ambiguity to which class function is called by the object of the derived class.
- This ambiguity is resolved by using the class name along with the function name when calling it.
- This problem and its resolution is shown in the Program 6.1.12 below.

➢ **Program 6.1.12 : Write a program to define the following relationship using hybrid inheritance.**

W-15, 8 Marks

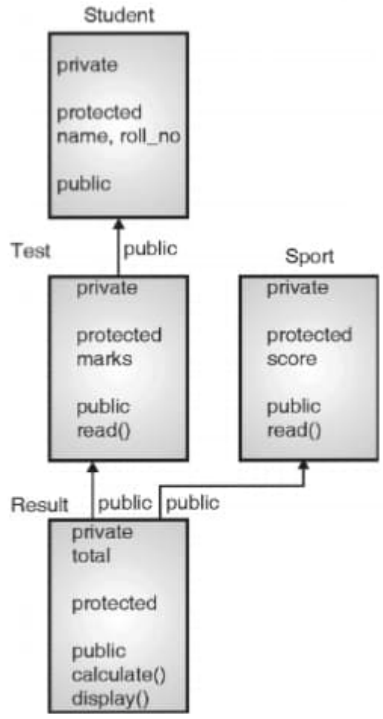


Fig. P. 6.1.12 : Class Diagram of Hybrid Inheritance for Program 6.1.12

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Student
{
    protected:
    char name[20];
    int roll_no;
};
class Test: public Student
{
    protected:
    int marks;
    public:
    void read()
    {
        cout<<"Enter name, roll number and marks
obtained:";
        gets(name);
        cin>>roll_no>>marks;
    }
};
class Sports
{
    protected:
    int score;
    public:
    void read()
    {
        cout<<"1. Student has won in national sports
event\n2. Student has not won in any national sports
event\nEnter your choice:";
        cin>>score;
    }
};
class Result: public Test, public Sports
{
    int total;
```

```
    public:
    void calculate()
    {
        if(score==1)
        total=marks+15;
        else
        total=marks;
    }
    void display()
    {
        cout<<"The total is "<<total;
    }
};
void main()
{
    clrscr();
    Result r;
    r.Test::read();
    r.Sports::read();
    r.calculate();
    r.display();
    getch();
}
```

Output :

```
Enter name, roll number and marks obtained:Satish
24
96
1. Student has won in national sports event
2. Student has not won in any national sports event
Enter your choice:2
The total is 96
```

Explanation :

- You will notice in this program both the classes namely "test" and "Sports" have a member function with the same name i.e. read().
- The resolution to the problem can be seen in the main() function where the two functions are called with specifying the class name along with the function name.

Syllabus Topic : Hierarchical Inheritance

6.1.7 Hierarchical Inheritance

➔ **(S-14,S-15, S-17)**

Q. 6.1.18 Illustrate the hierarchical inheritance. (Ref. Sec. 6.1.7) **(S-14, 4 Marks)**

Q. 6.1.19 How hierarchical inheritance is achieved, explain with example. (Ref. Sec. 6.1.7) **(S-15, 4 Marks)**

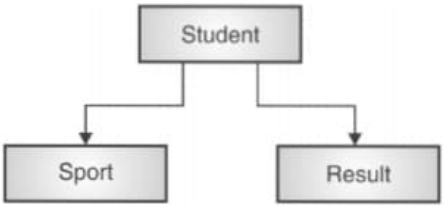


Fig. 6.1.4 : Hierarchical Inheritance

- When multiple classes are derived from a class and further more classes are derived from these derived classes.
- It is like hierarchy father, his children, their children and so on.
- Hence, many a times the derived class is also called as child class while the base class is called as parent class
- Let us see a program example of hierarchical inheritance.

➢ **Program 6.1.13 : Write a program to implement inheritance as shown in Fig. P. 6.1.13. Assume suitable member function.** **S-17, 8 Marks**

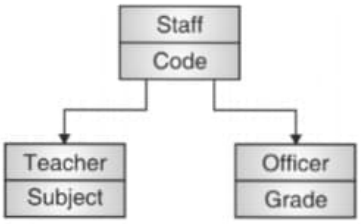


Fig. P. 6.1.13

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

```
class Staff
{
    protected:
    char name[20];
    int code;
};
class Teacher: public Staff
{
    private:
    char subject[20];
    int experience;
    public:
    void read()
    {
        cout<<"Enter name, code, subject and experience
of the teacher:";
        gets(name);
        cin>>code;
        gets(subject);
        cin>>experience;
    }
    void display()
    {
        cout<<"Teacher
Details:\nName:"<<name<<"\nCode:"<<code
<<"\nSubject:"<<subject<<"\nExperience:"
<<experience;
}
};
class Officer: public Staff
{
    private:
    char dept[20];
    int grade;
    public:
    void read()
    {
        cout<<"Enter name, code, department and grade
of the officer:";
        gets(name);
        cin>>code;
        gets(dept);
        cin>>grade;
    }
    void display()
    {
        cout<<"Officer
        Details:\nName:"<<name<<"\nCode:"<<code
    <<"\nDepartment:"<<dept<<"\nGrade:"<<grade;
    }
};
void main()
{
    clrscr();
    int choice;
    cout<<"1. Teacher\n2. Officer\nEnter the choice, whose
details you want to enter:";
    cin>>choice;
    switch(choice)
    {
        case 1:Teacher t;
        t.read();
        t.display();
        break;
        case 2:Officer o;
        o.read();
        o.display();
        break;
        default:cout<<"Invalid choice";
    }
    getch();
}
```

Output :

```
1. Teacher
2. Officer
Enter the choice, whose details you want to enter:1
Enter name, code, subject and experience of the
teacher:Satish
234
Maths
13
Teacher Details:
Name:Satish
Code:234
Subject:Maths
Experience:13
```

➢ **Program 6.1.14 : Write a program to define the following inheritance relationship.**

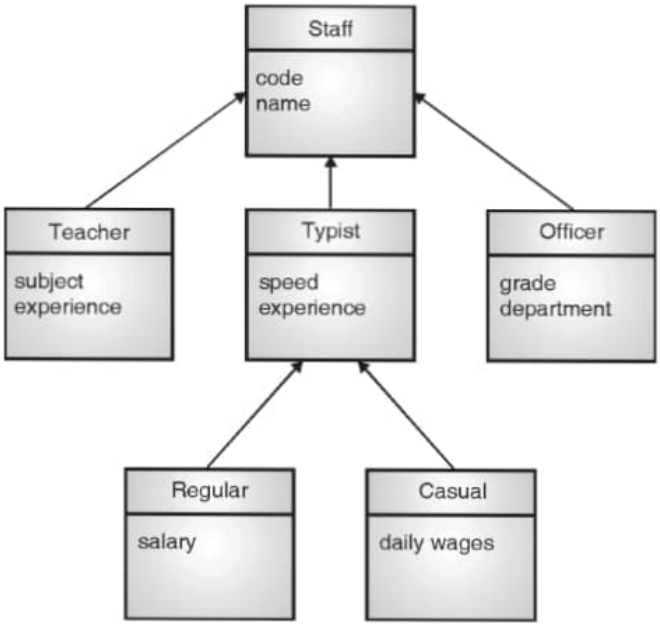


Fig. P. 6.1.14 : Class Diagram of Hierarchical Inheritance for Program 6.1.14

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Staff
{
    protected:
        char name[20];
        int code;
};
class Teacher: public Staff
{
    private:
    char subject[20];
    int experience;
    public:
    void read()
    {
        cout<<"Enter name, code, subject and experience
of the teacher:";
        gets(name);
        cin>>code;
        gets(subject);
        cin>>experience;
    }
    void display()
    {
        cout<<"Teacher
Details:\nName:"<<name<<"\nCode:"<<code
<<"\nSubject:"<<subject<<"\nExperience:"
<<experience;
    }
};
class Officer: public Staff
{
    private:
    char dept[20];
    int grade;
    public:
    void read()
    {
        cout<<"Enter name, code, department and grade
of the officer:";
        gets(name);
        cin>>code;
        gets(dept);
        cin>>grade;
```

```
    }
    void display()
    {
        cout<<"Officer
        Details:\nName:"<<name<<"\nCode:"<<code
    <<"\nDepartment:"<<dept<<"\nGrade:"<<grade;
    }
};
class Typist: public Staff
{
    protected:
    int speed,experience;
};
class Regular: public Typist
{
    private:
    int salary;
    public:
    void read()
    {
        cout<<"Enter name, code, speed, experience and
salary of the regular typist:";
        gets(name);
        cin>>code>>speed>>experience>>salary;
    }
    void display()
    {
        cout<<"Regular Typist
Details:\nName:"<<name<<"\nCode:"<<code<<"\nSpee
d:"<<speed<<"\nExperience:"<<experience<<"\nSalary:
"<<salary;
    }
};
class Casual: public Typist
{
    private:
    int dailywages;
    public:
    void read()
    {
        cout<<"Enter name, code, speed, experience and
dailywages of the Casual typist:";
        gets(name);

    cin>>code>>speed>>experience>>dailywages;
    }
    void display()
    {
        cout<<"Casual Typist
Details:\nName:"<<name<<"\nCode:"
<<code<<"\nSpeed:"<<speed<<"\nExperience:"
<<experience<<"\nDaily Wages:"<<dailywages;
    }
};
void main()
{
    clrscr();
    int choice;
    cout<<"1. Teacher\n2. Officer\n3. Regular Typist
\n4. Casual Typist\nEnter the choice, whose details you want
to enter:";
    cin>>choice;
    switch(choice)
    {
        case 1:Teacher t;
        t.read();
        t.display();
        break;
        case 2:Officer o;
        o.read();
        o.display();
        break;
        case 3:Regular r;
        r.read();
        r.display();
        break;
        case 4:Casual c;
        c.read();
        c.display();
        break;
        default:cout<<"Invalid choice";
```

```
    }
    getch();
}
```

Output :

```
1. Teacher
2. Officer
3. Regular Typist
4. Casual Typist
Enter the choice, whose details you want to enter:1
Enter name, code, subject and experience of the
teacher:Satish
234
Maths
13
Teacher Details:
Name:Satish
Code:234
Subject:Maths
Experience:13
```

6.2 Exam Pack (MSBTE and Review Questions)

☞ **Syllabus Topic : Introduction to Inheritance, Defining a Derived Class**

Q. 1 What is inheritance? Why inheritance used in C++? **(Refer Section 6.1)** **(W-16, 2 Marks)**

Q. 2 Define inheritance and enlist it's types. **(Refer Sections 6.1 and 6.1.1)** **(S-14, S-16, 2 Marks)**

☞ **Syllabus Topic : Visibility Modes and Effects**

Q. 3 Explain different visibility modes used in inheritance. **(Refer Section 6.1.1)** **(W-14, S-15, W-16, W-17, 4 Marks)**

Q. 4 State different types of inheritance with diagram. **(Refer Section 6.1.1)** **(W-14, 4 Marks)**

Q. 5 What are the different types of inheritance? Explain each with example. **(Refer Sections 6.1.1 to 6.1.5 and 6.1.7)** **(W-16, 10 Marks)**

Q. 6 State general format of defining derived class. **(Refer Section 6.1.1)** **(W-15, 2 Marks)**

Q. 7 State different types of inheritances and describe any one. **(Refer Section 6.1.1)** **(W-15, 4 Marks)**

Q. 8 List types of inheritance. **(Refer Section 6.1.1)** **(S-16, 2 Marks)**

Q. 9 State and explain various visibility modifiers in inheritance. **(Refer Section 6.1.1)** **(S-16, W-16, 4 Marks)**

Q. 10 Define derived class. Give one example. **(Refer Section and Program 6.1.1)** **(S-17, 2 Marks)**

Q. 11 State any two access specifier with example. **(Refer Section 6.1.1)** **(S-15, W-16, 2 Marks)**

Q. 12 Explain various types of inheritance with example **(Refer Sections 6.1.1 to 6.1.5 and 6.1.7)** **(W-16, 8 Marks)**

Q. 13 How protected access specifier is different from private? **(Refer Section 6.1.1)** **(W-15, 4 Marks)**

☞ **Syllabus Topic : Single type of Inheritance**

Q. 14 Write a program showing use of single inheritance. **(Refer Program 6.1.1)** **(W-16, 4 Marks)**

Q. 15 Write a program to implement single inheritance from following Fig.1-Q. 15 accept and display the data for one table. **(Refer Program 6.1.3)** **(W-15, 4 Marks)**

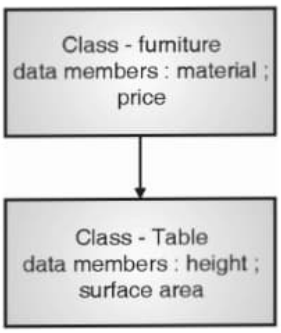


Fig. 1-Q. 15

Q. 16 Write a program to implement inheritance as shown in Fig. 1-Q. 16 given below assume suitable member functions.

(Refer Program 6.1.4) **(W-16, 4 Marks)**

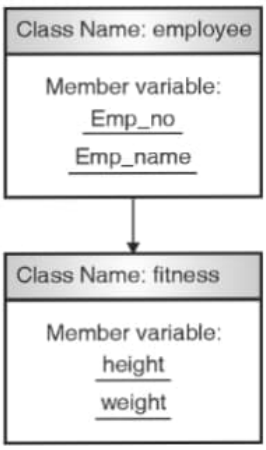


Fig. 1-Q. 16

☞ **Syllabus Topic Constructor in Derived Class**

Q. 17 Write a program to demonstrate constructor in derived class with respect to order of calling constructor and passing parameters to base class constructor. **(Refer Program 6.1.5)**

(W-15, S-16, W-17, 4 Marks)

☞ **Syllabus Topic : Multi Level Type of Inheritance**

Q. 18 Differentiate between multiple inheritance and multilevel inheritance.

(Refer Sections 6.1.3 and 6.1.4) **(4 Marks)**

Q. 19 Write a program that illustrates multilevel inheritance. **(Refer Program 6.1.6) (S-17, 4 Marks)**

Q. 20 Identify the type of inheritance shown in following Fig. 1-Q. 20. Implement it by using suitable member function.

(Refer Program 6.1.9) **(W-14, S-16, 8 Marks)**

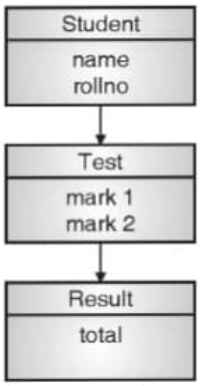


Fig. 1-Q. 20

☞ **Syllabus Topic : Multiple Type of Inheritance**

Q. 21 Explain Multiple inheritance. **(Refer Section 6.1.4)**

(S-14, W-16, S-17, W-17, 4 Marks)

Q. 22 Describe multiple inheritance with suitable example. **(Refer Section 6.1.4 and Program 6.1.10)**

(S-14, W-16, S-17, W-17, 4 Marks)

Q. 23 Write a program to illustrate multiple inheritance. Write suitable data members and member functions. **(Refer Program 6.1.10)**

(S-16, 4 Marks)

Q. 24 Write a program to define the following relationship using multiple inheritance.

(Refer Program 6.1.10) **(S-16, W-16, 8 Marks)**

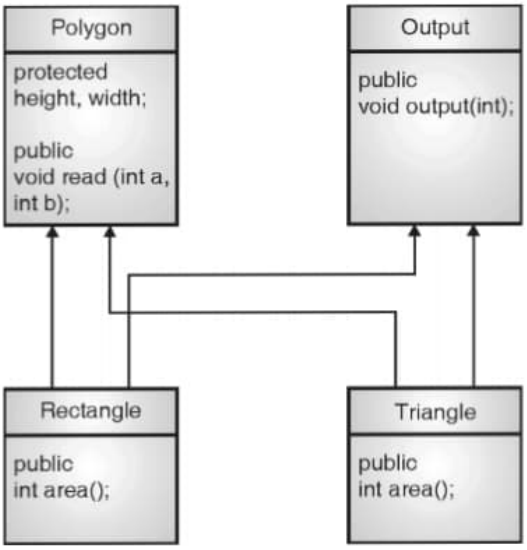


Fig. 1-Q. 24

☞ **Syllabus Topic : Hybrid Type of Inheritance**

Q. 25 What is hybrid inheritance ? Give one example.

(Refer Section 6.1.5 and Program 6.1.11)

(S-16, 4Marks)

Q. 26 Write a program to define the following relationship using hybrid inheritance.

(Refer Program 6.1.12) **(W-15, 8 Marks)**

☞ **Syllabus Topic : Hierarchical Inheritance**

Q. 27 Illustrate the hierarchical inheritance.

(Refer Section 6.1.7) **(S-14, 4 Marks)**

Q. 28 How hierarchical inheritance is achieved, explain with example.

(Refer Section 6.1.7) **(S-15, 4 Marks)**

Q. 29 Write a program to implement inheritance as shown in Fig. 1-Q. 29. Assume suitable member function.

(Refer Program 6.1.13) **(S-17, 8 Marks)**

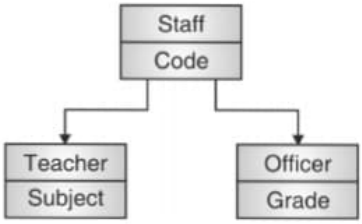


Fig. 1-Q. 29

❑❑❑

UNIT IV

CHAPTER 7

Pointers & Polymorphism in C++

Syllabus

Pointers and Polymorphism in C++

- **Unit Outcomes**
  - Create C++ programs to perform the given arithmetic operations using pointers.
  - Use function overloading to solve the given problem.
  - Use operator overloading to solve the given problem.
  - Implement run time polymorphism using virtual functions in the given C++ program.
- **Topics and Sub-topics**
  - Concepts of Pointer : Pointer declaration, Pointer operator, address operator, Pointer arithmetic.
  - Pointer to Object : Pointer to Object, this pointer, Pointer to derived class. Virtual Base Class and Abstract Class
  - Introduction of Polymorphism, Types of Polymorphism,
  - Compile time Polymorphism : Function overloading, operator overloading, overloading of unary and binary operators, Rules for operator overloading.
  - Run time polymorphism : Virtual functions, rules for virtual functions, pure virtual function.

Syllabus Topic : Concept of Pointer : Pointer Declaration

7.1 Pointers

➔ **(S-14, S-15, W-15, W-17)**

Q. 7.1.1 What is pointer? How will you declare and initialize a pointer? (Ref. Sec. 7.1) **(4 Marks)**

Q. 7.1.2 What is a pointer ? Write down the general syntax of its declaration. (Ref. Sec. 7.1) **(S-14, 2 Marks)**

Q. 7.1.3 What is pointer ? Give any example. (Ref. Sec. 7.1) **(S-15, W-17, 2 Marks)**

Q. 7.1.4 Define pointer. Give syntax of declaration and initialization of pointer. (Ref. Sec. 7.1) **(W-15, 4 Marks)**

- Pointers are variables that are used to store the address of another variable.
- Address of a variable is the memory location number which is allotted to the variable. The memory addresses are 0,1,2,3… and so on up to the capacity of the memory. The address is normally displayed in hexadecimal form.
- Hexadecimal form is a representation of number somewhat similar to binary number system studied in chapter 1. Here four binary digits are combined together to form a hexadecimal number.
- Pointers unlike other variables do not store values. As stated they store the address of other variables.
- It is already mentioned in the first statement that pointers are also variables. Hence, we can also have a pointer that is pointing to another pointer.

- We will discuss about these pointers to pointer in the later sections. To begin with let us see the syntax of declaring pointers, operators required for pointers and some basic programs.
- Syntax of pointer declaration:

  Data_type *ptr_name;

  Wherein "Data_type" is the data type of the variable to which the pointer is supposed to point. If we want a pointer to point to an integer than, we need to have the data type of the pointer as "int", for a float type data pointer should also be of the "float" type and so on.

  The "ptr_name" is an identifier i.e. the name of the pointer. The same rules of identifiers apply to the pointer name as to any other variable declaration.

  The most important difference in the declaration of a pointer is the "*" sign given before the pointer name.
- Hence, according to the syntax seen above, if we want to declare a pointer for "int" type data then we can declare it as given in the example below:

  int *p;

  Here, the pointer name is "p". Hence, "p" can be used as a pointer to point to any of the variable of type "int".

Syllabus Topic : Pointer Operator, Address Operator

7.1.1 Referencing and De-referencing (Operators in Pointers)

- There are two operators required in the pointer based programs viz. Address of operator ("&") and Value of operator ("*"). Let us see the use of these two operators in detail.

1. Address of operator ("&"):

- It is also called as the referencing operator.
- This operator returns the address of the variable associated with the operator.
- For e.g., if we write "&x", it will return the address of the variable "x".
- Hence, if we have a pointer "p", which we want to point to a variable x, then we need to copy the address of the variable "x" in the pointer variable "p". This is implemented by the statement:

  p=&x;

  i.e. the address of the variable "x" is copied into the pointer variable "p", hence the pointer "p" pointing to the variable "x".

2. Value of operator ("*") :

- It is also called as the de-referencing operator.
- This operator returns the value stored in the variable pointed by the specified pointer.
- For e.g., if we write "*p", it will return the value of the variable pointed by the pointer "p".
- Hence, if we want the value of the variable pointed by the pointer "p" to be stored in a variable "y", then the expression can be written as:

  y=*p;

  i.e. the value of the variable pointed by the pointer "p" is stored in the variable "y".
- Let us understand the use of these operators in some sample programs.

7.2 Programs Related to Pointers

7.2.1 Simple Referencing and De-referencing using Pointers

➢ **Program 7.2.1 : Write the output of the following program.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a,*p;
    a=125;
    p=&a;
    cout<<a<<endl;
    cout<<p<<endl;
    cout<<*p<<endl;
    getch();
}
```

Output :

```
125
Address of variable a
125
```

Explanation :

- The declaration of a pointer and a simple integer variable can be seen in the first statement i.e. "int a,*p;". The pointer is the variable "p", as it is associated with a "*" sign.
- The second statement is very simple wherein the value 125 is put into the variable "a".
- The third statement i.e. "p=&a" uses of the address operator. This statement assigns the address of the variable "a" to the pointer variable "p". As already seen address of a variable is the memory location (number), where it is stored.
- The first "cout" statement prints the value of the variable "a" hence the value 125 is displayed as shown in the output.
- The second "cout" statement displays the value of the variable "p". Since the variable "p" is a pointer, the value displayed is an address. This is the address of the variable "a".
- The address is displayed in hexadecimal form as already discussed. Whenever a value is to be displayed in hexadecimal form, it is preceded by "0x" indicating hexadecimal form.
- Note: In actual you will not get such a value as seen here, instead you will get some hexadecimal value like:

  0x8fb7fff4

  But in exams you need not write such an address and hence, in the output shown for the above program we have also mentioned "address of variable a".
- The last "cout" statement is for the value of the variable pointed by the pointer "p". Since; the pointer is initialized to be pointing to the variable "a", this statement will display the value of "a". Hence, it displays the value 125.

7.2.2 Pointer to Pointer

➢ **Program 7.2.2 : Write the output of the following program.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a,*p,**p1;
    a=125;
    p=&a;
    p1=&p;
    cout<<a<<endl;
    cout<<p<<endl;
    cout<<p1<<endl;
    cout<<*p<<endl;
    cout<<*p1<<endl;
    cout<<**p1<<endl;
    getch();
}
```

Output:

```
125
Address of variable a
Address of pointer variable p
125
Address of variable a
125
```

Explanation :

- The declaration of integer variable and a pointer variable is as already seen. The declaration of a pointer to pointer can be seen in the first statement i.e. "int **p1;".
- The pointer to pointer is the variable "p1", as it is associated with a "**" sign.
- A value of 125 is then assigned to the variable "a". The pointer "p" is assigned the address of the variable "a", hence making the pointer "p" point to the variable "a".
- The pointer to pointer variable i.e. "p1" is initialized to point to another pointer variable i.e. "p" remember pointer to pointer should also have the same data type as the pointer to which it has to point. Also the pointer to pointer variable can point to a pointer only.
- Thereafter; the values of "a", "p" and "p1" are printed. It is obvious that the value of "a" will be printed as "125". The value of "p" will be the address of the variable "a" as seen in the previous program.

- The value of "p1" will be the address of the pointer "p", since the address of operator ("&") is used in the program to assign the address of the pointer "p" to the variable p1.
- The next statement displays the value of the variable pointed by the pointer as seen in the previous program.
- Now, the statement "cout<<*p1<<endl;" will display the value of the variable pointed by the pointer "p1". But the variable "p1" points to another address (or variable) i.e. the pointer "p", which has the address of variable "a". Hence, the address of variable "a" stored in the pointer variable "p" is displayed on the screen.
- Finally; to access the actual value pointed by the pointer "p1" through the pointer "p" is displayed. This is the variable "a", hence the value of this variable i.e. 125 is displayed.

Syllabus Topic : Pointer Arithmetic

7.2.3 Pointer Arithmetic: Increment and Decrement Operators Operation on Pointer Variables

➔ **(W-14, S-15, S-16)**

➢ **Program 7.2.3 : Write the output of the following program.** **W-14, S-15, S-16, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a,*a1;
    float b,*b1;
    a1=&a;
    b1=&b;
    cout<<a1<<endl<<b1<<endl;
    a1++;
    b1++;
    cout<<a1<<endl<<b1<<endl;
    getch();
}
```

Output :

```
Address of variable a
Address of variable b
(Address of variable a)+2
(Address of variable b)+4
```

Explanation :

- **Note :** when a pointer is incremented or added by 1, it actually increments by the number equal to the size of the data type for which it is a pointer. For example; if an integer pointer is incremented, it will increment by 2, while if a float type pointer is incremented it will increment by 4.
- The same concept is demonstrated in the above program. Initially; the addresses of "a" and "b" are displayed as those are the contents of the pointers. Thereafter; when the pointers are incremented they increment by 2 and 4 respectively for "a" and "b". Hence, the new values that are displayed are addition of 2 and 4 respectively w.r.t their previous values.

7.2.4 Pointer to Array

➔ **(W-15, W-17)**

➢ **Program 7.2.4 : Write the output of the following program.** **W-15, W-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,a[2]={10,20};
    for(i=0;i<=1;i++)
    {
        cout<<a[i]<<endl;
        cout<<*(a+i)<<endl;
        cout<<*(i+a)<<endl;
    }
    getch();
}
```

Output :

```
10
10
10
20
```

```
20
20
```

Explanation :

- **Note :** The name of the array works as a pointer to the array. The name of the array always points to the first element of the array.
- The values of "a[i], *(a+i) and *(i+a) are same, since "a" is a pointer pointing to the first element in the array. When the value of the variable "i" is "0", the location pointed by *(a+i) and *(i+a) is the first location of the array.
- Similarly, when the value of "i" is "1", the location pointed by *(a+i) and *(i+a) is the second location of the array.
- Hence, for all the three statements the value displayed in the output is the same.

7.2.5 Passing Pointers to Functions

➢ **Program 7.2.5 : Write a program to add two numbers using function. Pass the pointers to the variables as reference to the functions.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a,b;
    cout<<"Enter two numbers";
    cin>>a>>b;
    void sum (int *p1, int *p2);
    sum(&a,&b);
    getch();
}
void sum (int *p1, int *p2)
{
    int c;
    c=*p1+*p2;
    cout<<"The sum is equal to "<<c;
}
```

Output :

```
Enter two numbers4
7
The sum is equal to 11
```

Explanation :

- **Note :** To pass the parameters using pointers we pass the address of the variables using the address of operator ("&"). And hence to accept this address in the function we must have pointers in the argument list. Remember in this case when address of the variable is passed, the value of the actual parameters can be altered. We will see this concept in more details in the subsequent programs.
- As discussed, the pointers are used to accept the address of the variables passed from the main function. the values of these variables are accessed using the pointers in the function "sum".
- The sum is calculated and is redisplayed in the function itself.

7.2.6 Call by Value and Call by Reference

➔ **(W-14, S-15, W-15, S-16, S-17, W-17)**

Q. 7.2.1	Explain Call-by-Value and Call-by-Reference with one programming example each. (Ref. Sec. 7.2.6)	**(10 Marks)**
Q. 7.2.2	Differentiate between call by value and call by reference. (Ref. Sec. 7.2.6)	**(S-16, 4 Marks)**
Q. 7.2.3	What is function ? What is call by value ? What is call by reference ? What is the difference between them ? (Ref. Sec. 7.2.6)	**(W-15, 4 Marks)**

Sr. No.	Call by value	Call by reference
1.	In *call by value*, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.	In *call by reference*, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.

Sr. No.	Call by value	Call by reference
2.	In call by value, actual arguments will remain safe, they cannot be modified accidentally.	In *call by reference*, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.

➢ **Program 7.2.6 : Write a program to swap two numbers using a function. Pass the values to be swapped to this function using call-by-value method.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a,b;
    cout<<"Enter two numbers:";
    cin>>a>>b;
    void swap (int a, int b);
    cout<<"The values of a and b in the main function
before calling the swap function are "<<a<<" and
"<<b<<endl;
    swap(a,b);
    cout<<"The values of a and b in main function after
calling the swap function are "<<a<<" and "<<b<<endl;
    getch();
}
void swap (int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    cout<<"The values of a and b in the swap function after
swapping are "<<a<<" and "<<b<<endl;
}
```

Output :

```
Enter two numbers:4
5
The values of a and b in the main function before calling the
swap function are 4 and 5
The values of a and b in the swap function after swapping are
5 and 4
The values of a and b in main function after calling the swap
function are 4 and 5
```

Explanation :

- In the output you will notice that before calling the function and after coming back from the function the values of actual parameters "a" and "b" in the main function remain the same. You will notice in this program the actual parameter do not get their values change.

> **Note :** We have already discussed, the actual parameters and formal parameters are always different. Even if the name of an actual parameter is same as that of a formal parameter, the memory allocated to it is different. Hence, the function work on different memory locations (or variables) while the main function works on different memory locations. Hence, keeping the actual parameters unchanged.

- We have already discussed, the actual parameters and formal parameters are always different. Even if the name of an actual parameter is same as that of a formal parameter, the memory allocated to it is different.
- Hence, the function work on different memory locations (or variables) while the main function works on different memory locations. Hence, keeping the actual parameters unchanged.
- This is demonstrated in the above program. Here, the values of the variables are passed to the swap function. This is called as "call-by-value".
- In the next program, we will use the concept of "call-by-reference" wherein the address of the parameter is passed instead of passing the value of the variable. Hence, we can access the actual parameter.

➢ **Program 7.2.7 : Write a program to swap two numbers using a function. Pass the values to be swapped to this function using call-by-reference method.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a,b;
    cout<<"Enter two numbers:";
    cin>>a>>b;
    void swap (int *p1, int *p2);
    cout<<"The values of a and b in the main function
    n before calling the swap function are "<<a<<" and
    "<<b<<endl;
    swap(&a,&b);
    cout<<"The values of a and b in main function after
calling the swap function are "<<a<<" and "<<b<<endl;
    getch();
}
void swap (int *p1, int *p2)
{
    int temp;
    temp=*p1;
    *p1=*p2;
    *p2=temp;
    cout<<"The values of a and b in the swap function after
swapping are "<<*p1<<" and "<<*p2<<endl;
}
```

Output :

```
Enter two numbers:4
5
The values of a and b in the main function before calling the
swap function are 4 and 5
The values of a and b in the swap function after swapping are
5 and 4
The values of a and b in main function after calling the swap
function are 5 and 4
```

Explanation :

- In the output, you will notice in this program the actual parameter are also changed i.e. the values of the parameters "a" and "b" before calling the function and after returning from the function are not the same, they are swapped.
- This is because the addresses of the actual parameters are passed to the function. In the function, the parameters are accessed through the pointers; hence the values of these actual parameters are altered.
- This method of passing the parameters to a function is call-by-reference. We have also seen an example program of call by reference in program 7.2.5 to add two numbers.

➢ **Program 7.2.8 : Write a program to accept string from user and count number of vowels in the string using pointer to string.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    char *a;
    int i,n,count;
    clrscr();
    a=new char[20];
    cout<<"Enter a string:";
    gets(a);
    for(i=0,count=0;a[i]!=0;i++)
    {

    if(*(a+i)=='a'||*(a+i)=='A'||*(a+i)=='e'||*(a+i)
==='E'||*(a+i)=='i'||*(a+i)=='I'||*(a+i)=='o'||*(a+i)
==='O'||*(a+i)=='u'||*(a+i)=='U') count++;
    }
    cout<<"Number of vowels:"<<count;
    getch();
}
```

Output :

```
Enter a string:Hi, How are you?
Number of vowels:6
```

➢ **Program 7.2.9 : Write a program to concatenate two strings by using pointers.** **W-15, W-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    char *a,*b;
    int i,n,count;
    clrscr();
    a=new char[20];
    b=new char[20];
    cout<<"Enter a string:";
    gets(a);
    cout<<"Enter another string:";
    gets(b);
    n=0;
    while(a[n]!=0)
    {
        n++;
    }
    for(i=0;b[i]!=0;i++,n++)
    {
        *(a+n)=*(b+i);
    }
    *(a+n)=0;
    cout<<"Concatenated String:"<<a;
    getch();
}
```

Output :

```
Enter a string:Hi
Enter another string:Hello
Concatenated String:HiHello
```

➢ **Program 7.2.10 : Write a program to search a character in a string using pointer.** **S-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    char *a,search;
    int i;
    clrscr();
    a=new char[20];
    cout<<"Enter a string:";
    gets(a);
    cout<<"Enter the character to be searched:";
    cin>>search;
    for(i=0;a[i]!=0;i++)
    {
        if(*(a+i)==search) break;
    }
    if(a[i]==0) cout<<"Not found";
    else cout<<"Found at index"<<i;
    getch();
}
```

Output :

```
Enter a string:Hi, How are you?
Enter the character to be searched:w
Found at index6
```

➢ **Program 7.2.11 : Write a program to find reverse of a string using pointer to string.** **W-14, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    char *str,temp;
```

```
    int i,n;
    clrscr();
    str=new char[20];
    cout<<"Enter a string:";
    gets(str);
    n=0;
    while(str[n]!=0)
    {
        n++;
    }
    for(i=0;i<=(n-1)/2;i++)
    {
        temp=*(str+i);
        *(str+i)=*(str+n-i-1);
        *(str+n-i-1)=temp;
    }
    cout<<"Reverse String:"<<str;
    getch();
}
```

Output :

```
Enter a string:Hello
Reverse String:olleH
```

➢ **Program 7.2.12 : Write a program to search a number from an array using pointer to array.**

W-14, S-17, 4 Marks

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    int i,n,*a,search;
    clrscr();
    cout<<"Enter the number of elements:";
    cin>>n;
    a=new int[n];
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter an element:";
        cin>>*(a+i);
    }
    cout<<"Enter element to be searched:";
    cin>>search;
    for(i=0;i<=n-1;i++)
    {
        if(search==*(a+i)) break;
    }
    if(i==n) cout<<"Element not found";
    else cout<<"Found at position:"<<(i+1);
    getch();
}
```

Output :

```
Enter the number of elements:4
Enter an element:2
Enter an element:45
Enter an element:23
Enter an element:43
Enter element to be searched:23
Found at position:3
```

➢ **Program 7.2.13 : Write a program to find the length of string using pointer.** **S-15, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
    char *a;
    int n;
    clrscr();
    a=new char[20];
    cout<<"Enter a string:";
    gets(a);
    n=0;
    while(*(a+n)!=0)
```

```
    {
        n++;
    }
    cout<<"Length of string:"<<n;
    getch();
}
```

Output :

```
Enter a string:Hi, How are you?
Length of string:16
```

Syllabus Topic : Function Overloading

7.3 Function Overloading or Function Polymorphism

➔ **(S-17)**

Q. 7.3.1 Write short notes on : Function overloading (Ref. Sec. 7.3) **(5 Marks)**

- Function overloading refers to creating multiple functions with the same name but different parameter list.
- Different parameter list can be with reference to the number of parameters or the type of parameters.
- Since, for overloaded functions the functions have same name, a function can be called with the common name but the function that will be invoked is based on the parameter type and number of parameters.
- This concept of function overloading is also called as function polymorphism.
- Let us see some program examples of function overloading.

➢ **Program 7.3.1 : Write a program to add two numbers using function overloading such that one function adds two integers, second function adds two float numbers and the third function adds a float number with an integer.**

```
# include<iostream.h>
# include<conio.h>
float add(float a, int b)
{
    float c;
    c=a+b;
    return c;
}
int add(int a, int b)
{
    int c;
    c=a+b;
    return c;
}

float add(float a, float b)
{
    float c;
    c=a+b;
    return c;
}
void main()
{
    clrscr();
    int x,a=5,b=6;
    float y,p=3.5,q=6.6;
    x=add(a,b);
    cout<<"Sum="<<x<<endl;
    y=add(p,q);
    cout<<"Sum="<<y<<endl;
    y=add(p,a);
    cout<<"Sum="<<y<<endl;
    getch();
}
```

Output :

```
Sum=11
Sum=10.1
Sum=8.5
```

Explanation :

- The first call to function add() in the main() function, has both the integer elements 'a' and 'b'; hence the second add() function is called which has both the integer parameters.

- The second call to function add() in the main() function, has both the float type elements 'p' and 'q'; hence the third add() function is called which has both the float type parameters.
- The third call to function add() in the main() function, has a float type element 'p' and an integer type element 'a'; hence the first add() function is called with a similar type of parameters.
- This is a program example wherein the number of parameters for all the overloaded functions is same but their data types are different.

➢ **Program 7.3.2 : Write a program to calculate the area of triangle, rectangle and circle using function overloading. The program should be menu-driven.**

```
# include<iostream.h>
# include<conio.h>
# include<math.h>
float area(float r)
{
    return (3.14*r*r);
}
float area(float l, float b)
{
    return (l*b);
}
float area(float a, float b,float c)
{
    float s,ar;
    s=(a+b+c)/2;
    ar=s*(s-a)*(s-b)*(s-c);
    ar=pow(ar,0.5);
    return ar;
}
void main()
{
    clrscr();
    int choice;
    float x,y,z,a;
    cout<<"1.Area of Circle\n2.Area of Rectangle\n3.Area of Triangle\nEnter your choice:";
    cin>>choice;
    switch(choice)
    {
        case 1:cout<<"Enter the radius of the circle:";
        cin>>x;
        a=area(x);
        cout<<"The area of the circle="<<a;
        break;
        case 2:cout<<"Enter the length and breadth of the rectangle:";
        cin>>x>>y;
        a=area(x,y);
        cout<<"The area of the rectangle="<<a;
        break;
        case 3:cout<<"Enter the length of the three sides of the triangle:";
        cin>>x>>y>>z;
        a=area(x,y,z);
        cout<<"The area of the triangle="<<a;
        break;
        default:cout<<"Invalid Choice";
    }
    getch();
}
```

Output :

```
1.Area of Circle
2.Area of Rectangle
3.Area of Triangle
Enter your choice:2
Enter the length and breadth of the rectangle:3.5
2
The area of the rectangle=7
```

Explanation :

- The user is asked for the choice of the shape whose area is to be found.

- The first choice is that of circle that calls to function area() that has only one parameter 'x'; hence the first area() function is called which has only one parameter.
- The second choice is for rectangle that calls to function area() that has two float type elements 'x' and 'y'; hence the second area() function is called which has two float type parameters.
- The third choice is for triangle that calls to function area() that has three float type elements 'x', 'y' and 'z'; hence the third area() function is called with a similar number of parameters.
- This is a program example wherein the number of parameters for all the overloaded functions is different while their data types are same.

➢ **Program 7.3.3 : Write a program using function overloading to swap 2 integer numbers and swap 2 float numbers.** **S-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void swap(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
void swap(float *a, float *b)
{
    float temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
void main()
{
    int a,b;
    float c,d;
    clrscr();
    cout<<"Enter two integers:";
    cin>>a>>b;
    cout<<"Numbers entered are: a="<<a<<" and
b="<<b<<endl;
    swap(&a,&b);
    cout<<"After swapping, a="<<a<<" and
b="<<b<<endl;
    cout<<"Enter two float numbers:";
    cin>>c>>d;
    cout<<"Numbers entered are: c="<<c<<" and
d="<<d<<endl;
    swap(&c,&d);
    cout<<"After swapping, c="<<c<<" and
d="<<d<<endl;
    getch();
}
```

Output :

```
Enter two integers:3
4
Numbers entered are: a=3 and b=4
After swapping, a=4 and b=3
Enter two float numbers:3.5
2.75
Numbers entered are: c=3.5 and d=2.75
After swapping, c=2.75 and d=3.5
```

➢ **Program 7.3.4 : Write a program to calculate area of circle and area of rectangle using function overloading.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
float area(float r)
{
    return (3.14*r*r);
}
float area(float l,float b)
{
    return(l*b);
}
```

```
void main()
{
    clrscr();
    cout<<"Area of circle with radius 10 units is
"<<area(10)<<endl;
    cout<<"Area of rectangle with sides 5 units and 7 units
is "<<area(5,7)<<endl;
    getch();
}
```

Output :

```
Area of circle with radius 10 units is 314
Area of rectangle with sides 5 units and 7 units is 35
```

7.3.1 Constructor Overloading

➔ **(S-14, W-14, S-15, W-15, S-16, S-17)**

Q. 7.3.2 Explain overloaded constructor in a class with suitable example.
(Ref. Sec. 7.3.1 and Prog. 7.3.5) **(S-14, W-14, S-16, 8 Marks)**

Q. 7.3.3 Define constructor overloading.
(Ref. Sec. 7.3.1) **(S-15, 2 Marks)**

Q. 7.3.4 Write syntax for overloaded constructor.
(Ref. Sec. 7.3.1) **(S-17, 2 Marks)**

Q. 7.3.5 What is overloaded constructor in a class? Explain with example.
(Ref. Sec. 7.3.1 and Prog. 7.3.5) **(W-15, 4 Marks)**

Q. 7.3.6 Write a program which implement the concept of overloaded constructor.
(Ref. Prog. 7.3.5) **(W-17, 4 Marks)**

Although we have seen examples of constructor overloading in previous section, let us see one more example of constructor overloading.

➢ **Program 7.3.5 : Write a program to calculate the area of triangle, rectangle and circle using constructor overloading. The program should be menu-driven.**

W-14, W-15, S-16, S-17, W-17, 4 Marks

```
# include<iostream.h>
# include<conio.h>
# include<math.h>
class Area
{
    float area;
    public:
    Area(float r)
    {
        area=3.14*r*r;
    }
    Area(float l, float b)
    {
        area=l*b;
    }
    Area(float a, float b,float c)
    {
        float s;
        s=(a+b+c)/2;
        area=s*(s-a)*(s-b)*(s-c);
        area=pow(area,0.5);
    }
    void display()
    {
        cout<<"Area="<<area;
    }
};
void main()
{
    clrscr();
    int choice;
    float x,y,z;
    cout<<"1.Area of Circle\n2.Area of Rectangle\n3.Area of
Triangle\nEnter your choice:";
    cin>>choice;
    switch(choice)
    {
        case 1:cout<<"Enter the radius of the circle:";
        cin>>x;
        Area a(x);
        a.display();

    break;
```

```
            case 2:cout<<"Enter the length and breadth of
the rectangle:";
            cin>>x>>y;
            Area a1(x,y);
            a1.display();
            break;
            case 3:cout<<"Enter the length of the three sides
of the triangle:";
            cin>>x>>y>>z;
            Area a2(x,y,z);
            a2.display();
            break;
            default:cout<<"Invalid Choice";
    }
    getch();
}
```

Output :

```
1.Area of Circle
2.Area of Rectangle
3.Area of Triangle
Enter your choice:1
Enter the radius of the circle:5
Area=78.5
```

Explanation :

- The user is asked for the choice of the shape whose area is to be found and the object of the class is made accordingly with the passing of those variables to the constructor.
- Then the display function is called with respect to the object.

Syllabus Topic : Operator Overloading, Overloading of Unary and Binary operators , Rules for Operator Overloading

7.4 Operator Overloading (Overloading of Unary and Binary Operators)

➔ **(S-14, W-14, W-15, S-16, S-17, W-17)**

Q. 7.4.1 What is operator overloading ? (Ref. Sec. 7.4) **(4 Marks)**

Q. 7.4.2 Enlist any four operators which cannot be overloaded. (Ref. Sec. 7.4) **(S-14, 2 Marks)**

Q. 7.4.3 State any four rules for operator overloading. (Ref. Sec. 7.4) **(S-14, W-14, W-15, S-16, S-17, 4 Marks)**

Q. 7.4.4 Explain various rules for overloading operators. (Ref. Sec. 7.4) **(W-15, W-16, W-17, 4 Marks)**

- The mechanism of assigning a new meaning to an already existing operator is called as operator overloading.
- There are some rules necessary to be known for operator overloading. Below is a list of these rules :

1. Only existing operators can be given a new meaning i.e. only existing operators can be overloaded.
2. Even after overloading the basic meaning of the operator remains the same.
3. Overloaded operators follow the same syntax as that of the original operators.
4. Unary operators overloaded by means of member function can accept no parameters.
5. Unary operators overloaded by means of friend function can accept up to one parameter.
6. Binary operators overloaded by means of member function can accept up to one parameter.
7. Binary operators overloaded by means of friend function can accept up to two parameters.
8. Some of the operators cannot be overloaded viz.
   (a) sizeof()
   (b) member operator (i.e. '.')
   (c) pointer to member operator (i.e. ".*")
   (d) scope resolution operator (i.e. "::")
   (e) conditional operator (i.e. "?:")
9. Some of the operators cannot be overloaded by friend functions viz.
   (a) assignment operator (i.e. '=')
   (b) function call operator (i.e. "( )")
   (c) subscripting operator (i.e. "[ ]")
   (d) class member access operator (i.e. "->")

10. Some of the rules are very simple, but some you may not understand now. We will understand these rules with program examples.
11. The declaration of the function that is meant for operator overloading has a special syntax as given below :

    return_type operator op (argument list);

    where "op" is to be replaced by the symbol of the operator to be overloaded.

➢ **Program 7.4.1 : Write a program to negate the values of two variables contained in an object.**

```
#include<iostream.h>
#include<conio.h>
class Negate
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>x>>y;
    }
    void compute()
    {
        x=-x;
        y=-y;
    }
    void display()
    {
        cout<<"x="<<x<<endl<<"y="<<y;
    }
};
void main()
{
    clrscr();
    Negate n;
    n.read();
    n.compute();
    n.display();
    getch();
}
```

Output :

```
Enter two numbers4
6
x=-4
y=-6
```

Explanation :

- A simple program with three functions respectively to accept the numbers, compute the negative and to display the result.
- An object of the class is made and the three functions are called one after the other.
- Let us see the same operation by overloading the unary minus ('-') operator in the following program.

➢ **Program 7.4.2 : Write a program to negate the values of two variables contained in an object by overloading the operator unary negate i.e. '-'.**

W-16, 4 Marks

```
#include<iostream.h>
#include<conio.h>
class Negate
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>x>>y;
    }
    void operator -()
    {
        x=-x;
        y=-y;
    }
    void display()
    {
        cout<<"x="<<x<<endl<<"y="<<y;
    }
};
void main()
{
```

```
    clrscr();
    Negate n;
    n.read();
    -n;
    n.display();
    getch();
}
```

Output :

```
Enter two numbers4
6
x=-4
y=-6
```

Explanation :

- In this case we have implemented the operator overloading operation by overloading the operator unary minus i.e. '-'.
- Compare to previous program where we had to call the compute() function, here we have to just use the negate operator with the object of the class i.e. "–n;".
- Actually operators can work only with variables of basic data type. The operators cannot be used with the objects of a class. But operator overloading allows those operators that are overloaded in a class to be used with the objects of that class.
- Remember, we cannot use an operator with an object unless we have written an operator overloading function for that operator in the corresponding class.

➢ **Program 7.4.3 : Write a program to overload unary operators i.e. increment and decrement.**

```
#include<iostream.h>
#include<conio.h>
class IncDec
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>x>>y;
    }
    void operator --()
    {
        x--;
        y--;
    }
    void operator ++ ()
    {
        x++;
        y++;
    }
    void display()
    {

    cout<<"x="<<x<<endl<<"y="<<y<<endl;
    }
};
void main()
{
    clrscr();
    IncDec n;
    n.read();
    --n;
    cout<<"After decrementing the object one time\n";
    n.display();
    ++n;
    ++n;
    cout<<"After incrementing the object twice\n";
    n.display();
    getch();
}
```

Output :

```
Enter two numbers4
5
After decrementing the object one time
x=3
y=4
After incrementing the object twice
x=5
y=6
```

Explanation :

- In this case we have implemented the operator overloading operation by overloading the operators unary increment and decrement ("++" and "--").

➢ **Program 7.4.4 : Write a program to overload binary operator "+" to add two complex numbers.**

S-16, 4 Marks

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
    Complex operator + (Complex c)
    {
        Complex c1;
        c1.x=x+c.x;
        c1.y=y+c.y;
        return c1;

    }
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
void main()
{
    clrscr();
    Complex c1;
    Complex c2;
    Complex c3;
    c1.read();
    c2.read();
    c3=c1+c2;
    c3.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

- Here we have overloaded the binary operator "+" by the operator overloading function Complex operator +(Complex c).
- Two objects of the class complex are created. The values of these complex variables are read.
- The two objects of the class "complex" are added using the same syntax of the addition operator i.e. '+'. The result is returned to another object of the same class.

➢ **Program 7.4.5 : Write a program to overload binary operator "+=" to add two complex numbers.**

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
```

```
    void operator += (Complex c)
    {
        x=x+c.x;
        y=y+c.y;
    }
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
void main()
{
    clrscr();
    Complex c1;
    Complex c2;
    c1.read();
    c2.read();
    c1+=c2;
    c1.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

- Similar implementation like the previous program, but the original object of the class "complex" i.e. 'c1' is changed instead of having another object.

➢ **Program 7.4.6 : Write a program to add two distances entered by the user in feet and inches using overload binary operator "+".**

```
#include<iostream.h>
#include<conio.h>
class Distance
{
    int ft,in;
    public:
    void read()
    {
        cout<<"Enter the distance in feet and inches:";
        cin>>ft>>in;
    }
    Distance operator + (Distance d)
    {
        Distance dr;
        dr.ft=ft+d.ft;
        dr.in=in+d.in;
        if(dr.in>=12)
        {
            dr.ft++;
            dr.in=dr.in-12;
        }
        return dr;
    }
    void display()
    {
        cout<<"Distance is "<<ft<<" feet and
"<<in<<" inches.";
    }
};
void main()
{
    clrscr();
    Distance d1;
    Distance d2;
    Distance d3;
    d1.read();
    d2.read();
```

```
    d3=d1+d2;
    d3.display();
    getch();
}
```

Output :

```
Enter the distance in feet and inches:4
3
Enter the distance in feet and inches:5
11
Distance is 10 feet and 2 inches.
```

Explanation :

– Similar operation as in program 7.4.4.

After adding the inches part, if the total inches is greater than or equal to 12, then the feet is to be incremented by one and inches are to be deducted by 12. This is implemented in the "if" statement of the operator overloading function in the operator overloading function of the class "Distance".

➢ **Program 7.4.7 : Write a program to declare a class 'Journal' having data members as journal-name, price and no-of-pages. Accept this data for two objects and display the name of journal having greater price.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Journal
{
    private:
    char name[20];
    int pages,price;
    public:
    void accept()
    {
        cout<<"Enter name, pages and price:";
        cin>>name>>pages>>price;
    }
    void display()
    {
    cout<<"Name:"<<name<<"\nPages:"<<pages<<"\n
Price:"<<price;
    }
    Journal operator >(Journal x)
    {
        if(price>x.price) return *this;
        return x;
    }
};
void main()
{
    Journal a,b,c;
    clrscr();
    a.accept();
    b.accept();
    c=a>b;
    c.display();
    getch();
}
```

Output :

```
Enter name, pages and price:a
4
400
Enter name, pages and price:b
2
200
Name:a
Pages:4
Price:400
```

➢ **Program 7.4.8 : Write a program to declare a class distance having data members feet and inches. Overload unary '-' operator so that when it is used with object of this class, it will decrement values of inches by 1.**

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

```
class Distance
{
    private:
    int ft,inch;
    public:
    void accept()
    {
        cout<<"Enter distance in feet and inches:";
        cin>>ft>>inch;
    }
    void display()
    {
        cout<<ft<<" feet "<<inch<<" inches\n";
    }
    void operator -()
    {
        inch=inch-1;
        if(inch==-1)
        {
            inch=11;
            ft=ft-1;
        }
    }
};
void main()
{
    clrscr();
    Distance d;
    d.accept();
    d.display();
    -d;
    cout<<"After decrementing:\n";
    d.display();
    getch();
}
```

Output :

```
Enter distance in feet and inches:3
0
3 feet 0 inches
After decrementing:
2 feet 11 inches
```

- **Program 7.4.9 : Write a program to overload binary ++ operator.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Distance
{
    private:
    int ft,inch;
    public:
    void accept()
    {
        cout<<"Enter distance in feet and inches:";
        cin>>ft>>inch;
    }
    void display()
    {
        cout<<ft<<" feet "<<inch<<" inches\n";
    }
    void operator ++()
    {
        inch=inch+1;
        if(inch==12)
        {
            inch=0;
            ft=ft+1;
        }
    }
};
void main()
{
```

```
    clrscr();
    Distance d;
    d.accept();
    d.display();
    ++d;
    cout<<"After incrementing:\n";
    d.display();
    getch();
}
```

Output :

```
Enter distance in feet and inches:3
11
3 feet 11 inches
After incrementing:
4 feet 0 inches
```

➢ **Program 7.4.10 : Write a program to add two complex numbers using operator overloaded by a friend function.**

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
    friend Complex operator + (Complex c1, Complex c2);
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
Complex operator + (Complex c1, Complex c2)
{
    Complex c;
    c.x=c1.x+c2.x;
    c.y=c1.y+c2.y;
    return c;
}
void main()
{
    clrscr();
    Complex c1;
    Complex c2;
    Complex c3;
    c1.read();
    c2.read();
    c3=c1+c2;
    c3.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

– Here in the class "Distance", we have mentioned that the operator overloading function is a friend to this class. This function is then defined outside the class. This function will accept two objects of the class "Complex". This function also returns an object of the same class i.e. "Complex".

➢ **Program 7.4.11 : Write a program which overload unary ++ operator by using friend function**

```
include<iostream.h>
#include<conio.h>
class Complex
{
```

```
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a
complex number:";
        cin>>x>>y;
    }
    friend void operator ++ ( Complex c);
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
void operator ++(Complex c)
{
    c.x++;
    c.y++;
}
void main()
{
    clrscr();
    Complex c1;
    c1.read();
    c1++;
    c1.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
3+i4
```

- **Program 7.4.12 : Write a program to overload "+" operator so that it will perform concatenation of two strings.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class String
{
    private:
    char a[100];
    public:
    void accept()
    {
        cout<<"Enter a string:";
        cin>>a;
    }
    void display()
    {
        cout<<a<<"\n";
    }
    String operator +(String x)
    {
        int i,n;
        String temp;
        for(i=0;a[i]!=0;i++)
        {
            temp.a[i]=a[i];
        }
        for(n=0;x.a[n]!=0;n++,i++)
        {
            temp.a[i]=x.a[n];
        }
        temp.a[i]=x.a[n];
        return temp;
    }
};
void main()
{
    clrscr();
    String s1,s2,s3;
    s1.accept();
```

```
    s2.accept();
    s3=s1+s2;
    s3.display();
    getch();
}
```

Output :

```
Enter a string:Hi
Enter a string:Hello
HiHello
```

Syllabus Topic : Introduction to Polymorphism, Types of Polymorphism

7.5 Polymorphism

➔ **(S-14, W-14, S-15, W-15, W-16, S-16, S-17, W-17)**

Q. 7.5.1 Write short notes on: Polymorphism.
(Ref. Sec. 7.5) **(5 Marks)**

Q. 7.5.2 Explain with examples how is polymorphism achieved at compile time and run time.
(Ref. Sec. 7.5) **(10 Marks)**

Q. 7.5.3 Describe the following features of object oriented programming : Polymorphism
(Ref. Sec. 7.5) **(10 Marks)**

Q. 7.5.4 List types of polymorphism.
(Ref. Sec. 7.5) **(S-14, 2 Marks)**

Q. 7.5.5 Distinguish between run-time polymorphism and compile-time polymorphism.
(Ref. Sec. 7.5)
(S-14, W-14, S-15, W-15, W-16, S-17, 4 Marks)

Q. 7.5.6 Define polymorphism. List types of polymorphism.
(Ref. Sec. 7.5) **(W-14, S-16, W-16, 2 Marks)**

Q. 7.5.7 State any two types of polymorphism.
(Ref. Sec. 7.5) **(S-15, 2 Marks)**

Q. 7.5.8 Define polymorphism. Explain any two types with example. (Ref. Sec. 7.5) **(W-15, 4 Marks)**

Q. 7.5.9 What is polymorphism? List its types.
(Ref. Sec. 7.5) **(S-17, 2 Marks)**

Q. 7.5.10 What is static polymorphism?
(Ref. Sec. 7.5) **(W-17, 2 Marks)**

Q. 7.5.11 What is polymorphism ? Enlist different types of polymorphism. What are the differences between them ?
(Ref. Sec. 7.5) **(W-17, 4 Marks)**

- As discussed in chapter 1 and, polymorphism refers to multiple forms of same thing i.e. function, constructors or operators.
- The chart in Fig. 7.5.1 shows the different types of polymorphism. The chart shows two types of polymorphism viz. static and dynamic. Those operations that take place during the compile time are called as "static". Those operations that take place during the run time or execution time are called as "dynamic".
- Static polymorphism is also referred to as early binding while dynamic polymorphism is referred to as late binding.

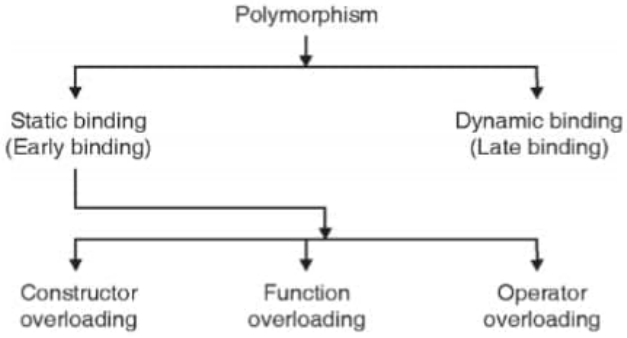


Fig. 7.5.1 : Types of polymorphism

- Static binding or static polymorphism includes Constructor overloading, function overloading and operator overloading i.e. in static polymorphism operators, functions or constructors can take multiple forms or behave differently at different places in the program.
- As already stated, static means compile time; the compiler resolves the reference in this case. Hence, when a function is called and there are multiple functions of the same name, then the compiler decides which function is to be called based on the parameters.
- Thus, the compiler during the compile times resolves the reference. Similarly; it is done for constructor and operator overloading.
- We have already seen the static polymorphism types i.e. constructor overloading, function overloading and operator overloading.
- We will see dynamic polymorphism in the subsequent sub section.

Sr. No.	Compiler time polymorphism	Run time polymorphism
1.	In compile time polymorphism call is resolved by the compiler.	In run time polymorphism call is not resolved by the compiler.
2.	It is also known as static binding, Early binding and overloading as well.	It is also known as dynamic binding late binding and over riding as well.
3.	Overloading is compile time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Over riding is run time polymorphism having same method with same parameters or signature but associated in a class and its subclass.
4.	It is achieved by function over loading and operator over loading.	It is achieved by virtual functions and pointers.
5.	It provides fast execution because known early compile time.	It provides slow execution as compare to early binding because it is known at runtime.
6.	Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.

Syllabus Topic : Virtual Functions, Rules for Virtual Functions, Pure Virtual Functions

7.5.1 Dynamic Binding using Virtual Function or Compile/Run time Polymorphism

➔ **(S-14, W-15, S-16, W-16, W-17)**

Q. 7.5.12 Write short notes on : Virtual function.
(Ref. Secs. 7.5.1 and 7.5.2) **(5 Marks)**

Q. 7.5.13 What is function overriding? Give example.
(Ref. Sec. 7.5.1 and Prog. 7.5.1) **(10 Marks)**

Q. 7.5.14 Explain pure virtual function with example.
(Ref. Sec. 7.5.1 and Prog. 7.5.3) **(5 Marks)**

Q. 7.5.15 Explain virtual function with suitable example.
(Ref. Sec. 7.5.1 and Prog. 7.5.2) **(S-14, S-16, 4 Marks)**

Q. 7.5.16 What is pure virtual function?
(Ref. Sec. 7.5.1) **(W-16, 2 Marks)**

Q. 7.5.17 What is virtual function ? Why we need virtual function ?
(Ref. Secs. 7.5.1 and 7.5.2) **(S-17, 4 Marks)**

Q. 7.5.18 What is pure virtual functions ? Write down the rules used for pure virtual function.
(Ref. Secs. 7.5.1 and 7.5.2) **(W-17, 4 Marks)**

Q. 7.5.19 How to define virtual function ? Give example.
(Ref. Sec. 7.5.1 and Prog. 7.5.2) **(W-17, 4 Marks)**

- Dynamic binding of a member function is achieved in C++ with the use of the keyword "virtual".
- Those member functions that are preceded by the keyword "virtual" are known as **virtual functions**.

7.5.2 Rules for Virtual Functions

➔ **(W-14, W-15)**

Q. 7.5.20 Why do we need virtual functions ?
(Ref. Sec. 7.5.2) **(W-14, 4 Marks)**

Q. 7.5.21 What is pure virtual function ?
(Ref. Sec. 7.5.2) **(W-14, 2 Marks)**

Q. 7.5.22 Write short note on: virtual function.
(Ref. Secs. 7.5.1 and 7.5.2) **(5 Marks)**

Q. 7.5.23 Explain any four rules for virtual function.
(Ref. Sec. 7.5.2) **(W-15, 4 Marks)**

Q. 7.5.24 What is need of virtual function? Explain with example. (Ref. Sec. 7.5.2 and Prog. 7.5.2) **(W-15, 4 Marks)**

- If a base class function has the same name as that of the derived class then the function called depends on the parameters, this is called as function overloading. But if the base class function and derived class function have same name and same parameter list then it is said to be **function overriding**.
- If the keyword "virtual" is preceded to the header line of the function in the base class then the binding of such a function with its call becomes dynamic i.e. runtime. Thus, if this function is called then whether the base class function is to be invoked or the subclass function is to be invoked is decided during the runtime. This is called as **dynamic binding** or **late binding**.

- The question arises, why pure virtual functions? The reason is that, in practical applications the base class member functions are rarely used to perform any function; in such cases the functions are to be declared as pure virtual function and their definitions in the derived classes.
- Virtual function may be a friend function of another class.
- The virtual functions must be non-static members of same class accessed by using object pointers only. The object pointer must be of the base class i.e. the object pointer of the base class can access the object of derived class but vice versa is not true.
- If a function in a base class is left blank i.e. without any definition and only declaration then;
  1. The function must be declared "virtual".
  2. This function is called as "**pure virtual function**" or "do nothing function" or "dummy function".
  3. This function must be overridden in the sub class i.e. the function with same name and parameters must be declared and defined in the sub class of the virtual class.
  4. A pure virtual function can be defined with the following syntax:

     virtual return_type function_name () =0;

     For e.g.

     virtual void read() = 0;
- Let us see examples for each of these concepts i.e.

1.	Function overriding
2.	Virtual function
3.	Pure virtual function

➢ **Program 7.5.1 : Write a program to demonstrate function overriding.**

```
#include<iostream.h>
#include<conio.h>
class Base
{
    protected:
    int a,b;
    public:
    void read()
    {
        cout<<"Enter two values:";
        cin>>a>>b;
    }
    void display()
    {
        cout<<"The values are:"<<a<<"\n"<<b;
    }
};
class Sub: public Base
{
    protected:
    int c,d;
    public:
    void read()
    {
        cout<<"Enter 4 values:";
        cin>>a>>b>>c>>d;
    }
    void display()
    {
cout<<"The values are:"
    <<a<<"\n"<<b<<"\n"<<c<<"\n"<< d;
    }
};
void main()
{
    clrscr();
    Sub s;
    s.read();
    s.display();
    getch();
}
```

Output :

```
Enter 4 values:4
3
2
1
```

```
The values are:4
3
2
1
```

Explanation :

- The class "Sub" is derived from the class "Base". The functions read() and display() are declared and defined in the base class and derived class. The name and parameter list of the functions in base class and derived class are same. This is called as function overriding.
- In this case, the functions in the class "Sub" override the functions read() and display() of the class "Base". Hence when the object of the class "Sub", created in the main() function will call the respective functions of the sub class and not the base class.

Syllabus Topic : Pointer to Derived Class

7.5.2.1 Pointer to Derived Class

➔ **(S-14, W-15, S-16, S-17, W-17)**

➢ **Program 7.5.2 : Write a program to demonstrate dynamic binding using virtual function.**

S-14, W-15, S-16, W-17, 4 Marks

```
#include<iostream.h>
#include<conio.h>
class Base
{
    protected:
    int a,b;
    public:
    virtual void read()
    {
        cout<<"Enter two values:";
        cin>>a>>b;
    }
    virtual void display()
    {
        cout<<"\nThe values
are:"<<a<<"\n"<<b<<endl;
    }
};
class Sub: public Base
{
    protected:
    int c,d;
    public:
    virtual void read()
    {
        cout<<"Enter 4 values:";
        cin>>a>>b>>c>>d;
    }
    virtual void display()
    {
        cout<<"\nThe values are:"
<<a<<"\n"<<b<<"\n"<<c<<"\n"<<d<<endl;
    }
};
void main()
{
    clrscr();
    Base *ptr;
    Base b;
    Sub s;
    ptr=&b;
    ptr->read();
    ptr->display();
    ptr=&s;
    ptr->read();
    ptr->display();

    getch();
}
```

Output :

```
Enter 2 values:1
2

The values are:1
2
```

```
Enter 4 values:1
2
3
4

The values are:1
2
3
4
```

Explanation :

- The class "Sub" is derived from the class "Base". The functions read() and display() are declared and defined in the base class and derived class.
- The name and parameter list of the functions in base class and derived class are same. This is called as function overriding.
- But, here the functions are preceded by the keyword "virtual", hence the binding with the call to these functions are done during the runtime. This is called as dynamic binding or late binding.
- In the main() function, we have made objects for both the classes and a pointer for the base class. The base class pointer initially points to the object of the base class and hence when the functions are called, the base class functions are invoked.
- For the second time the same pointer is pointed to the object of the sub class and hence at this time when the functions are called, the derived class functions are invoked.
- This can also be seen from the output and comparing it with the program. The same pointer when calls the function at different places calls different functions decided during the run time.

➢ **Program 7.5.3 : Write a program to demonstrate pure virtual function.**

```
#include<iostream.h>
#include<conio.h>
class Base
{
    protected:
    int a,b;
    public:
    virtual void read()
    {
    }
    virtual void display() = 0;
};
class Sub: public Base
{
    protected:
    int c,d;
    public:
    void read()
    {
        cout<<"Enter 4 values:";
        cin>>a>>b>>c>>d;
    }
    void display()
    {
        cout<<"The values are:"
        <<a<<"\n"<<b<<"\n"<<c<<"\n"<<d;
    }
};
void main()
{
    clrscr();
    Sub s;
    Base *ptr;
    ptr=&s;
    ptr->read();
    ptr->display();
    getch();
}
```

Output :

```
Enter 4 values:1
2
3
4
The values are:1
2
3
4
```

```
    ptr->display();
    getch();
}
```

Output :

```
Enter base and height2
10
Area=10
Enter length and breadth5
4
Area=20
```

Syllabus Topic : Virtual Base Class and Abstract Class

7.5.3 Virtual Base Class and Abstract Class

➔ **(W-14, S-16,S-17)**

Q. 7.5. 25 What is the purpose of using Virtual base classes ? Shows the hierarchy after using the concept of virtual base class. Give proper example.
(Ref. Sec. 7.5.3 and Prog. 7.5.5) **(10 Marks)**

Q. 7.5.26 What is abstract class ?
(Ref. Sec. 7.5.3) **(W-14, 2 Marks)**

Q. 7.5.27 What is virtual base class ? Explain with example.
(Ref. Sec. 7.5.3 and Prog. 7.5.5) **(S-16, 4 Marks)**

Q. 7.5.28 Explain abstract class with suitable example.
(Ref. Sec. 7.5.3 and Prog. 7.5.5) **(S-17, 4 Marks)**

- A class can be inherited as a virtual class by another class. In this case when the base class is inherited by another class as virtual class, then the base class is called a virtual base class.
- To inherit a class as virtual, we need to mention the keyword "virtual" while deriving the class. The syntax for virtual derivation of a class is as given below:

  class sub_class : public virtual base_class;

  For example in the program 7.5.3, we can write class Sub: virtual public Base;
- In the syntax mentioned above the keywords "virtual" and public" are interchangeable. Hence the syntax can also be written as:

  class sub_class : virtual public base_class;
- A class which is used only as a virtual derivation and no object is made of that class is called as abstract class i.e. it is not used to create objects.
- Abstract classes are used as a template for the designing purposes.

➢ **Program 7.5.5 : Write a program to demonstrate virtual class or abstract class.** **S-16, S-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
class Base
{
    protected:
    int a,b;
    public:
    void read()
    {
        cout<<"Enter two values:";
        cin>>a>>b;
    }
    void display()
    {
        cout<<"The values are:"<<a<<"\n"<<b;
    }
};
class Sub: public virtual Base
{
    protected:
    int c,d;
    public:
    void read()
    {
        cout<<"Enter 4 values:";
        cin>>a>>b>>c>>d;
```

```
    }
    void display()
    {
        cout<<"The values are:"
        <<a<<"\n"<<b<<"\n"<<c<<"\n"<< d;
    }
};
void main()
{
    clrscr();
    Sub s;
    s.read();
    s.display();
    getch();
}
```

Output :

```
Enter 4 values:4
3
2
1
The values are:4
3
2
1
```

Explanation :

- The class "Sub" is derived from the class "Base". The functions read() and display() are declared and defined in the base class and derived class. The name and parameter list of the functions in base class and derived class are same. This is called as function overriding.

  In this case, the functions in the class "Sub" override the functions read() and display() of the class "Base".
- The keyword "virtual" while deriving the class "sub", makes the class "base" asabstract.

Syllabus Topic : this Pointer

7.6 this Pointer

➔ **(S-14, W-14, W-15, S-16, S-17, W-17)**

Q. 7.6.1	Explain the concept of 'this' pointer. (Ref. Sec. 7.6)	**(S-14, W-14, 4 Marks)**
Q. 7.6.2	Describe 'this' pointer with respect to its use only. (Ref. Sec. 7.6)	**(W-15, 2 Marks)**
Q. 7.6.3	Describe the concept of' this' pointer. Give one example. (Ref. Sec. 7.6 and Prog. 7.6.1)	**(S-16, 4 Marks)**
Q. 7.6.4	What is 'this' pointer ? Give suitable example. (Ref. Sec. 7.6 and Prog. 7.6.1)	**(S-17, W-17, 4 Marks)**

- "this" is a keyword used to indicate the current object. It is used to access the members of the current object.
- It is not much used until now because of the ease of access of variables and assigning values to variables.
- Let us see an example wherein we return an object using the keyword "this".

➢ **Program 7.6.1 : Write a program to demonstrate the use of "this" pointer.** **S,-16, S-17, W-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
class Compare
{
    protected:
    int a;
    public:
    void read()
    {
        cout<<"Enter a value:";
        cin>>a;
    }
    Compare compare (Compare c)
    {
        if(a>c.a)
        return *this;
```

```
            else
            return c;
    }
    void display()
    {
            cout<<"The value is:"<<a;
    }
};
void main()
{
    clrscr();
    Compare c1;
    Compare c2;
    c1.read();
    c2.read();
    Compare c3;
    c3=c1.compare(c2);
    c3.display();
    getch();
}
```

Output :

```
Enter a value:3
Enter a value:5
The value is:5
```

Explanation :

- The variable "counter" is declared static and of type integer. The value of this variable is incremented in the constructor i.e. whenever the object is made. Hence the variable keeps a track of the number of objects created.
- The statement declaring the static variable "counter" as a member of the class "Count" is stated after the class. The syntax of the statement is "data_type class_name::variable_name;". This statement is necessary for the compiler to resolve the scope of the static variables. This statement is used to initialize the value of variable "counter" to zero.

Syllabus Topic : Pointer to Object

7.7 Pointer to Object

➔ **(W-14, S-15, W-15, S-16, S-17, W-17)**

Q. 7.7.1	Write syntax to create a pointer for object. (Ref. Sec. 7.7)	**(S-15, S-17, 2 Marks)**
Q. 7.7.2	How pointer is assigned to object ? Explain with simple example. (Ref. Sec. 7.7)	**(S-16, 2 Marks)**
Q. 7.7.3	Give syntax of create pointer to object. (Ref. Sec. 7.7 and Prog. 7.7.1)	**(W-17, 2 Marks)**

- A pointer to object can be create as we create pointer for primitive data type.
- This pointer can then point to any object of that class as well as objects of class derived from that class.
- Hence an object of a base class can point to the object of same class or that of derived class.
- The following example demonstrates the pointer object of base class pointing to object of base as well as derived class
- To select a member of an object using the pointer object we need the object pointer operator i.e. "->". This operator is used in the same manner as the dot(.) operator is used for other objects.

➢ **Program 7.7.1 : Write a program to demonstrate use of pointer to object.** **W-17, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
class Base
{
    protected:
    int a,b;
    public:
    virtual void read()
    {
            cout<<"Enter two values:";
            cin>>a>>b;
    }
    virtual void display()
    {
```

```
            cout<<"\nThe values
are:"<<a<<"\n"<<b<<endl;
      }
};
class Sub: public Base
{
      protected:
      int c,d;
      public:
      virtual void read()
      {
            cout<<"Enter 4 values:";
            cin>>a>>b>>c>>d;
      }
      virtual void display()
      {
            cout<<"\nThe values are:"
      <<a<<"\n"<<b<<"\n"<<c<<"\n"<<d<<endl;
      }
};
void main()
{
      clrscr();
      Base *ptr;
      Base b;
      Sub s;
      ptr=&b;
      ptr->read();
      ptr->display();
      ptr=&s;
      ptr->read();
      ptr->display();

      getch();
}
```

Output :

```
Enter 2 values:1
2

The values are:1
2
Enter 4 values:1
2
3
4

The values are:1
2
3
4
```

➢ **Program 7.7.2 : Write a program to declare a class birthday having data members day, month and year. Accept this information for 5 objects using pointer to the array of object.** **W-14, 4 Marks**

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

class Birthday
{
   int day, month, year;
public:
  void getdata()
  {
    cout<<"\n\t\tEnter birthdate ( day month year)  ::";
    cin>>day>>month>>year;
  }
    void putdata()
    {
      cout<<" "<<day<<"-"<<month<<"-"<<year;
    }
};
void main()
{
 Birthday B[5],*ptr;
 clrscr();
 ptr = &B[0];
```

```
for(int i=0;i<5;i++)
{
cout<<"\n\t\tEnter birthdate for object "<<i+1;
ptr->getdata();
ptr++;
}
ptr = &B[0];
for(i=0;i<5;i++)
{
cout<<"\n\t\tBirthdate for object ::: "<<i+1;
ptr->putdata();
ptr++;
}
getch();
}
```

Output :

```
Enter birthdate for object 1
  Enter birthdate ( day month year)  :: 16 11 2005

  Enter birthdate for object 2
  Enter birthdate ( day month year)  :: 15 10 1975
  Enter birthdate for object 3
  Enter birthdate ( day month year)  :: 21 12 1979

  Enter birthdate for object 4
  Enter birthdate ( day month year)  :: 10 03 1978
  Enter birthdate for object 5
  Enter birthdate ( day month year)  :: 03 12 1987
  Birthdate for object 1 :: 16-11-2005
  Birthdate for object 2 :: 15-10-1975
  Birthdate for object 3 :: 21-12-1979
  Birthdate for object 4 :: 10-3-1978
  Birthdate for object 5 :: 3-12-1987
```

- **Program 7.7.3 : Write a program to declare class product having data members as product-id and price. Accept and display data for one object using pointer to the object.** **W-15, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Product
{
    private:
    int id,price;
    public:
    void accept()
    {
        cout<<"Enter product ID and price:";
        cin>>id>>price;
    }
    void display()
    {
        cout<<"Product
ID:"<<id<<"\nPrice:"<<price;
    }
};
void main()
{
    Product *p;
    clrscr();
    Product p1;
    p=&p1;
    p->accept();
    p->display();
    getch();
}
```

Output :

```
Enter product ID and price:24
1200
Product ID:24
Price:1200
```

- **Program 7.7.4 : Write a program to declare class 'city' with data members cityname and state. Create array of object of size 5. Read and print data for array using pointer to object.** **S-16, 4 Marks**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class City
{
    private:
    char name[20],state[20];
    public:
    void accept()
    {
        cout<<"Enter city name and state:";
        cin>>name>>state;
    }
    void display()
    {
        cout<<name<<"\t"<<state<<endl;
    }
};
void main()
{
    City *c,cc[5];
    int i;
    clrscr();
    for(i=0;i<=4;i++)
    {
        c=&cc[i];
        c->accept();
    }
    cout<<"City\tState\n";
    for(i=0;i<=4;i++)
    {
        c=&cc[i];
        c->display();
    }
    getch();
}
```

Output :

```
Enter city name and state:Mumbai
Maharashtra
Enter city name and state:Pune
Maharashtra
Enter city name and state:Indore
MadhyaPradesh
Enter city name and state:Nashik
Maharashtra
Enter city name and state:Nagpur
Maharashtra
City    State
Mumbai  Maharashtra
Pune    Maharashtra
Indore  MadhyaPradesh
Nashik  Maharashtra
Nagpur  Maharashtra
```

7.8 Exam Pack (MSBTE and Review Questions)

☞ **Syllabus Topic : Concept of Pointer : Pointer Declaration**

Q. 1 What is pointer? How will you declare and initialize a pointer ? **(Refer Section 7.1)** **(4 Marks)**

Q. 2 What is a pointer ? Write down the general syntax of its declaration.
(Refer Section 7.1) **(S-14, 2 Marks)**

Q. 3 What is pointer ? Give any example.
(Refer Section 7.1) **(S-15, W-17, 2 Marks)**

Q. 4 Define pointer. Give syntax of declaration and initialization of pointer.
(Refer Section 7.1) **(W-15, 4 Marks)**

☞ **Syllabus Topic : Pointer Arithmetic**

Q. 5 Explain pointer arithmetic with example.
(Refer Program 7.2.3) **(W-14, S-16, 4 Marks)**

Q. 6 Explain various pointer arithmetic operations with examples. **(Refer Program 7.2.3) (S-15, 4 Marks)**

Q. 7 What is pointer to array? Explain with example.
(Refer Program 7.2.4) (W-15, 4 Marks)

Q. 8 Write a program to display elements of array using pointer to array of integers.
(Refer Program 7.2.4) (W-15, 4 Marks)

Q. 9 Write a program which implement the concept of pointer to array.
(Refer Program 7.2.4) (W-17, 4 Marks)

Q. 10 Explain Call-by-Value and Call-by-Reference with one programming example each.
(Refer Section 7.2.6) (10 Marks)

Q. 11 Differentiate between call by value and call by reference. **(Refer Section 7.2.6) (S-16, 4 Marks)**

Q. 12 What is function ? What is call by value ? What is call by reference ? What is the difference between them ?
(Refer Section 7.2.6) (W-17, 4 Marks)

Q. 13 Write a program to concatenate two strings using pointer to string.
(Refer Program 7.2.9) (W-15, 8 Marks)

Q. 14 Write a program to search a character in a string using pointer.
(Refer Program 7.2.10) (S-17, 4 Marks)

Q. 15 Write a program to find reverse of a string using pointer to string.
(Refer Program 7.2.11) (W-14, 4 Marks)

Q. 16 Write a program to search a number from an array using pointer to array.
(Refer Program 7.2.12) (W-14, S-17, 4 Marks)

Q. 17 Write a program to find the length of string using pointer. **(Refer Program 7.2.13) (S-15, 4 Marks)**

☞ **Syllabus Topic : Function Overloading**

Q. 18 Write short notes on: Function overloading
(Refer Section 7.3) (5 Marks)

Q. 19 Explain overloaded constructor in a class with suitable example.
(Refer Section 7.3.1 and Program 7.3.5)
(S-14, W-14, 8 Marks)

Q. 20 Define constructor overloading.
(Refer Section 7.3.1) (S-15, 2 Marks)

Q. 21 Write syntax for overloaded constructor.
(Refer Section 7.3.1) (S-17, 2 Marks)

Q. 22 What is overloaded constructor in a class? Explain with example.
(Refer Section 7.3.1 and Program 7.3.5)
(W-15, S-16, 4 Marks)

Q. 23 Write a program using function overloading to swap two integer number and swap two float number.
(Refer Program 7.3.3) (S-17, 4 Marks)

Q. 24 Write a program which implement the concept of overloaded constructor.
(Refer Program 7.3.5) (W-17, 4 Marks)

☞ **Syllabus Topic : Operator Overloading, Overloading of Unary and Binary operators Rules for Operator Overloading**

Q. 25 What is operator overloading ?
(Refer Section 7.4) (4 Marks)

Q. 26 Enlist any four operators which cannot be overloaded. **(Refer Section 7.4) (S-14, 2 Marks)**

Q. 27 State any four rules for operator overloading.
(Refer Section 7.4)
(S-14, W-14, W-15, S-16, S-17, 4 Marks)

Q. 28 Explain various rules for overloading operators.
(Refer Section 7.4) (W-15, W-16, W-17, 4 Marks)

Q. 29 Write a program to negate the values of two variables contained in an object by overloading the operator unary negate i.e. '-'.
(Refer Program 7.4.2) (W-16, 4 Marks)

Q. 30 Write a program to overload operator '+' to add two complex numbers.
(Refer Program 7.4.4) (S-16, 4 Marks)

Syllabus Topic : Introduction to Polymorphism, Types of Polymorphism

Q. 31 Write short notes on : Polymorphism.
(Refer Section 7.5) **(5 Marks)**

Q. 32 Explain with examples how is polymorphism achieved at compile time and run time.
(Refer Section 7.5) **(10 Marks)**

Q. 33 Describe the following features of object oriented programming : Polymorphism
(Refer Section 7.5) **(10 Marks)**

Q. 34 List types of polymorphism.
(Refer Section 7.5) **(S-14, 2 Marks)**

Q. 35 Distinguish between run-time polymorphism and compile-time polymorphism.
(Refer Section 7.5)
(S-14, W-14, S-15, W-15, W-16, S-17, 4 Marks)

Q. 36 Define polymorphism. List types of polymorphism.
(Refer Section 7.5) **(W-14, S-16, W-16, 2 Marks)**

Q. 37 State any two types of polymorphism.
(Refer Section 7.5) **(S-15, 2 Marks)**

Q. 38 Define polymorphism. Explain any two types with example. **(Refer Section 7.5)** **(W-15, 4 Marks)**

Q. 39 What is polymorphism? List its types.
(Refer Section 7.5) **(S-17, 2 Marks)**

Q. 40 What is static polymorphism?
(Refer Section 7.5) **(W-17, 2 Marks)**

Q. 41 What is polymorphism ? Enlist different types of polymorphism. What are the differences between them ? **(Refer Section 7.5)** **(W-17, 4 Marks)**

Syllabus Topic : Virtual Functions, Rules for Virtual Functions, Pure Virtual Functions

Q. 42 Write short notes on : Virtual function.
(Refer Sections 7.5.1 and 7.5.2) **(5 Marks)**

Q. 43 What is function overriding. Give example. **(Refer Section 7.5.1 and Program 7.5.1)**
(10 Marks)

Q. 44 Explain virtual function with suitable example.
(Refer Section 7.5.1 and Program 7.5.2)
(S-14, S-16, 4 Marks)

Q. 45 Explain pure virtual function with example.
(Refer Section 7.5.1 and Program 7.5.3)(5 Marks)

Q. 46 Explain any four rules for virtual function.
(Refer Section 7.5.1) **(W-15, 4 Marks)**

Q. 47 What is pure virtual function?
(Refer Section 7.5.1) **(W-16, 2 Marks)**

Q. 48 What is virtual function ? Why we need virtual function ? **(Refer Section 7.5.1)** **(S-17, 4 Marks)**

Q. 49 How to define virtual function ? Give example.
(Refer Section 7.5.1 and Program 7.5.2)
(W-17, 4 Marks)

Q. 50 What is pure virtual functions ? Write down the rules used for pure virtual function.
(Refer Sections 7.5.1 and 7.5.2) **(W-17, 4 Marks)**

Q. 51 What is need of virtual function? Explain with example.
(Refer Section 7.5.2 and Program 7.5.2)
(W-15, 4 Marks)

Q. 52 Why do we need virtual functions ?
(Refer Section 7.5.2) **(W-14, 4 Marks)**

Q. 53 What is pure virtual function ?
(Refer Section 7.5.2) **(W-14, 2 Marks)**

Q. 54 Create class shape. Derive two class triangle and rectangle from class shape. Write appropriate functions in both classes to accept dimensions and calculate area. Here make area ( )function virtual which is common to all and will calculate area of both rectangle and triangle. Display area of both.
(Refer Program 7.5.4) **(W-15, S-17, 4 Marks)**

Syllabus Topic : Pointer to Derived Class

Q. 55 Give any example where runtime polymorphism can be used. **(Refer Program 7.5.2)** **(W-15, 2 Marks)**

Q. 56 Write a program to demonstrate dynamic binding using virtual function. **(Refer Program 7.5.2)**

(S-14, W-15, S-16, W-17, 4 Marks)

☞ **Syllabus Topic : Virtual Base Class and Abstract Class**

Q. 57 What is abstract class ?

(Refer Section 7.5.3) **(W-14, 2 Marks)**

Q. 58 What is the purpose of using Virtual base classes ? Shows the hierarchy after using the concept of virtual base class. Give proper example.

(Refer Section 7.5.3 and Program 7.5.5)

(10 Marks)

Q. 59 Explain abstract class with suitable example.

(Refer Section 7.5.3 and Program 7.5.5)

(S-17, 4 Marks)

Q. 60 What is virtual base class ? Explain with example.

(Refer Section 7.5.3 and Program 7.5.5)

(S-16, 4 Marks)

Q. 61 Write a program to demonstrate virtual class or abstract class.

(Refer Program 7.5.5) **(S-16, S-17, 4 Marks)**

☞ **Syllabus Topic : This Pointer**

Q. 62 Explain the concept of 'this' pointer.

(Refer Section 7.6) **(S-14, W-14, 4 Marks)**

Q. 63 Describe 'this' pointer with respect to its use only.

(Refer Section 7.6) **(W-15, 2 Marks)**

Q. 64 Describe the concept of' this' pointer. Give one example.

(Refer Section 7.6 and Program 7.6.1)

(S-16, 4 Marks)

Q. 65 What is 'this' pointer ? Give suitable example.

(Refer Section 7.6 and Program 7.6.1)

(S-17, W-17, 4 Marks)

Q. 66 Write a program to demonstrate the use of "this" pointer. **(Refer Program 7.6.1)**

(S,-16, S-17, W-17, 4 Marks)

☞ **Syllabus Topic : Pointer to Object**

Q. 67 Write syntax to create a pointer forobject.

(Refer Section 7.7) **(S-15, S-17, 2 Marks)**

Q. 68 How pointer is assigned to object ? Explain with simple example.

(Refer Section 7.7) **(S-16, 2 Marks)**

Q. 69 Give syntax of create pointer to object.

(Refer Section 7.7 and Program 7.7.1)

(W-17, 2 Marks)

Q. 70 Write a program to demonstrate use of pointer to object. **(Refer Program 7.7.1)** **(W-17, 4 Marks)**

Q. 71 Write a program to declare a class birthday having data members day, month and year. Accept this information for 5 objects using pointer to the array of object. **(Refer Program 7.7.2)** **(W-14, 4 Marks)**

Q. 72 Write a program to declare class product having data members as product-id and price. Accept and display data for one object using pointer to the object.

(Refer Program 7.7.3) **(W-15, 4 Marks)**

Q. 73 Write a program to declare class 'city' with data members cityname and state. Create array of object of size 5. Read and print data for array using pointer to object. **(Refer Program 7.7.4)** **(S-16, 4 Marks)**

❑❑❑

UNIT V

CHAPTER 8

File Operations

Syllabus

File Operation

- **Unit Outcomes**
  - Identify relevant class for performing the given file operation.
  - Write statement to open and close the given file in C++.
  - Develop C++ program to perform read/write operation from/to the given file.
- **Topics and Sub-Topics**
  - C++ Stream Classes, Classes for File Stream Operations.
  - Opening Files, Closing Files, Reading from and Writing to Files.
  - Detection of End of File , File Modes

Syllabus Topic : C++ Stream Classes

8.1 Concept of Streams and C++ stream Classes

Q. 8.1.1 What is a stream ? (Ref. Sec. 8.1) **(2 Marks)**

Q. 8.1.2 Explain various constructors of of stream. (Ref. Sec. 8.1) **(4 Marks)**

- The earlier C++ programs used the "stream.h" header files for accessing the files, but the newer versions have the "iostream.h" which has better support.
- The file handling is done using the stream objects predefined in the header file "iostream.h".
- Each of the stream has an associated class that contains various functions for dealing a particular type of data.

☞ **Constructors of stream**

There are two constructors in the class of stream.

They are :

1. **Default constructor :** This creates an object of the stream class which is not associated with any file.
2. **Initialization constructor :** The first argument passed is the file name with which the object is to be associated. The second parameter indicates the file open mode.

Syllabus Topic : Classes for File Stream Operations

8.1.1 File Stream Classes

Q. 8.1.3 Explain the any two file stream classes needed for the file manipulation. (Ref. Sec. 8.1.1) **(4 Marks)**

Q. 8.1.4 What are stream extraction and stream insertion operators ? (Ref. Sec. 8.1.1) **(4 Marks)**

- The three classes for handling files are :

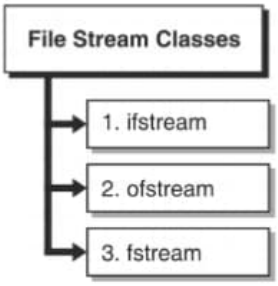


Fig. C.8.1 : Classes for handling files

1. **ifstream :** This class is used to handle the input files.
2. **ofstream :** This class is used to handle output files.
3. **fstream :** This class is used for files in which both input as well as output is to be done.

- A stream is a sequence of bytes. The iostream as the name says has input stream and output stream to read a stream or sequence of bytes and similarly write a stream i.e. sequence of bytes.
- The stream class has an hierarchy as shown in the Fig. 8.1.1.

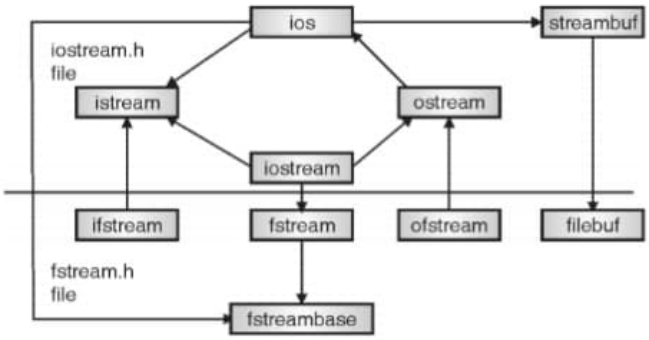


Fig. 8.1.1 : Stream class hierarchy

- We have been using some of these in this book with the help of the extraction (>>) and insertion (<<) operators i.e. with the "cin" and "cout" statements. These are derived from the "ios" class.
- The ios class is virtual class for the "istream" and "ostream" i.e. input stream and output stream respectively. The "istream" is a class derived from "ios" for input while "ostream" is for output.
- The class "iostream" has a multiple inheritance i.e. it is derived from classes "istream" and "ostream", hence used for both input as well as output.
- Similarly the classes "ifstream" is for input from files and "ofstream" is for output to files. The "fstream" class is for both input and output.

8.1.2 Advantages of Stream Classes

- Stream classes have various functions that make it easy for the programmer for reading or writing a sequence of bytes.
- These classes have good error handling capabilities
- The classes work as an abstraction for the user i.e. the internal operation is encapsulated from the user.
- The stream classes are buffered and hence do not use the memory disk space.

Syllabus Topic : File Modes, Opening Files, Closing Files

8.2 File Management Functions and File Modes (Opening and Closing Files)

Q. 8.2.1	List file mode operation. (Ref. Sec. 8.2)	**(3 Marks)**
Q. 8.2.2	Explain flags ios::binary and ios::ate. (Ref. Sec. 8.2)	**(4 Marks)**
Q. 8.2.3	Explain tellg and seekp functions. (Ref. Sec. 8.2)	**(4 Marks)**
Q. 8.2.4	Explain various file mode parameters in C++. Write a program to copy the contents of a source file **student1.txt** to a destination file **student 2.txt** character by character. (Ref. Sec. 8.2)	**(7 Marks)**
Q. 8.2.5	Explain *tellg( )* function in file streams. (Ref. Sec. 8.2)	**(1 Mark)**
Q. 8.2.6	Explain *ios::app* and *ios::ate* flags. (Ref. Sec. 8.2)	**(3 Marks)**
Q. 8.2.7	Explain the functions used to read and write data in binary file. (Ref. Sec. 8.2)	**(4 Marks)**

Tellg and Seekp functions

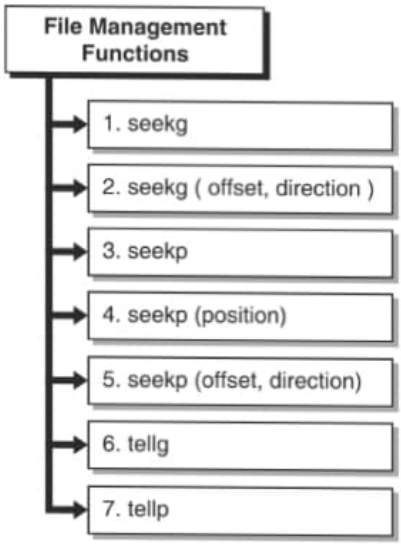


Fig. C.8.2 : Tellg and seekp functions

1. seekg

Sets the position of the *get pointer*. The *get pointer* determines the next location to be read in the source associated to the stream.

Syntax : seekg (position)

Using this function the stream pointer is changed to the absolute position (counting from the beginning of the file).

2. seekg (offset, direction)

Using this function, the position of the get pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off_type, which is also an integer type. And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from.

3. seekp

The *seekp* method changes the location of a stream object's file pointer for output (put or write.) In most cases, seekp also changes the location of a stream object's file pointer for input (get or read.)

4. seekp (position)

Using this function the stream pointer is changed to the absolute position (counting from the beginning of the file).

5. seekp (offset, direction)

- Using this function, the position of the put pointer is set to an offset value relative to some specific point determined by the parameter direction.
- Offset is of the member type off_type, which is also an integer type. And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from.

6. tellg

The tellg() function is used with input streams, and returns the current "get" position of the pointer in the stream.

Syntax : pos_type tellg()

It has no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer.

7. tellp

Returns the absolute position of the *put pointer*. The put pointer determines the location in the output sequence where the next output operation is going to take place.

Syntax : pos_type tellp()

The tellp() function is used with output streams, and returns the current "put" position of the pointer in the stream. It has no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the put stream pointer.

- To work with files as discussed in the earlier section we need to include the header file "fstream.h".
- We need to first create an object of the stream class before opening a file. This stream object will be used to open the file as well as to perform various operations like read and write on the file.
- The object of that class may be created based on the operation required to be done on the file. Thus if input operation is to be done then an object of the class "ifstream" is to be created.
- Thus, the object can be created as below :
  - ifstream i;
  - ofstream o;
  - fstream f;

- Once the object of stream class is created, the file can be opened using the open() function. The prototype of open() function is as given below :

  void open(const char *filename, int mode, int access = filebuf::openprot);

- The filename is the name of the file to be opened, which can be given along the path. The mode can be one of the following based on the operation to be performed :

Sr. No.	Mode	Function
1.	ios::app	Append to a file
2.	ios::ate	Move the pointer/cursor to the end of file
3.	ios::binary	File open in binary mode
4.	ios::in	Open file in read mode
5.	ios::nocreate	This will cause to fail the open function if file doesn't exists
6.	ios::norepace	This will cause function to fail open function if file exists
7.	ios::out	Open file in write mode
8.	ios::trunc	This will cause earlier files (if any) with same name to be destroyed.

- Thus, for example if a file "test" is to be opened in out mode, it will be done by the following two statements :

```
ofstream o;
o.open("test",ios::out);
```

- Another example to open a file "test" in input mode, the statements can be as given below :

```
ifstream i;
i.open("test",ios::in);
```

- For opening a file in input and output mode, the statements can be as given below:

```
ifstream i;
i.open("test",ios::in | ios::out);
```

- To close a file the function close() is to be used. This can be simply done as shown below:

```
i.close();
```

Syllabus Topic : Reading from and Writing to Files

8.2.1 Reading from and Writing to Files

- The functions to read and write the data from a file can be done by fprintf() and fscanf() or simply using the insertion operator (<<) and extraction operator (>>).
- For the insertion and extraction operators the objects created are required for reading or writing data from/to file; instead of the objects "cout" and "cin" that are required for the reading or writing data from/to the console.
- Let us see some programs to access a file.

➢ **Program 8.2.1 : Write a program to write and read a string from/to a file.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char msg[20];
    ofstream o;
    o.open("test0",ios::in);
    o<<"HelloFriends!!"<<endl;
    o<<"Bye!!"<<endl;
    o.close();
    ifstream i;
    i.open("test0",ios::out);
    i>>msg;
    cout<<msg<<endl;
    i>>msg;
    cout<<msg<<endl;
    getch();
}
```

Output :

```
HelloFriends!!
Bye!!
```

➢ **Program 8.2.2 : Write a program to write and read string, integer and float from/to a file.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char name[20];
    int roll;
```

```
    float fee;
    ofstream o;
    o.open("test0",ios::in);
    o<<"Ajay"<<endl;
    o<<24<<endl;
    o<<55988.45<<endl;
    o.close();
    ifstream i;
    i.open("test0",ios::out);
    i>>name;
    i>>roll;
    i>>fee;
    cout<<"Name:"<<name<<endl;
    cout<<"Roll No."<<roll<<endl;
    cout<<"Fee:"<<fee<<endl;
    getch();
}
```

Output

```
Name:Ajay
Roll No.24
Fee:55988.449219
```

Syllabus Topic : Detection of end of File

8.2.2 Detecting End of File

Q. 8.2.8 What is the basic difference between manipulators and ios member functions in implementation ? Give examples. (Ref. Sec. 8.2.2) **(4 Marks)**

- To detect an end of file we have the function eof() that can be called by the ifstream class object. This function returns a true of false based on the pointer pointing to the end of file or not respectively.
- Let us see a program example for the same.

➢ **Program 8.2.3 : Write a program that reads a text file and creates another text file that is identical except that every letter must be converted to lower case irrespective of its original case (e.g 'a' or 'A' will become 'a').**

```
#include<conio.h>
#include<iostream.h>
#include<fstream.h>
void main()
{
    clrscr();
    char msg;
    ofstream o;
    o.open("test123",ios::in);
    ifstream in;
    in.open("test0",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        if(msg>='A' && msg<='Z')
        msg=msg+'a'-'A';
        o<<msg;
    }
    o.close();
    in.close();
    in.open("test123",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        cout<<msg;
    }
    getch();
}
```

Output

```
this is a bench.
```

➢ **Program 8.2.4 : Write a program that opens two text files for reading data. It creates a third file that contains the text of first file and then that of second file (text of second file to be appended after text of the first file, to produce the third file).**

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

```
#include<conio.h>
void main()
{
    char msg;
    int i;
    ofstream o;
    o.open("test11",ios::in);
    ifstream in;
    in.open("test0",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        o<<msg;
    }
    o.close();
    in.close();
    o.open("test11",ios::app);
    in.open("test00",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        o<<msg;
    }
    in.close();
    o.close();
    cout<<"\nData from concatenated file:\n";
    in.open("test11",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        cout<<(msg);
    }
    getch();
}
```

Output :

```
Data from concatenated file:
This is a bench.   This is a car.
```

- **Program 8.2.5 : Write a program to copy the contents of a source file student1.txt to a destination file student2.txt character by character.**

```
#include<conio.h>
#include<iostream.h>
#include<fstream.h>
void main()
{
    clrscr();
    char msg;
    ofstream o;
    o.open("student1.txt",ios::in);
    ifstream in;
    in.open("student2.txt",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        o<<msg;
    }
    o.close();
    in.close();
    in.open("student2.txt",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        cout<<msg;
}
    getch();
}
```

- **Program 8.2.6 : Explain the output of the following programs.**

```
#include<iostream.h>
#include<conio.h>
    typedef void(*Fptr)(int,int);
    void Add(int i,int j)
    {
        cout<<i<<"+"<<j<<"="<<i+j;
```

```
}
void subtract(int i,int j)
{
    cout<<i<<"-"<<j<<"="<<i-j;
}
void main()
{
    clrscr();
    Fptr ptr;
    ptr=& Add;
    ptr(1,2);
    cout<<endl;
    ptr=&subtract;
    ptr(3,2);
    getch();
}
```

Output :

```
1+2 = 3
3-2 = 1
```

The function pointer ptr, initially points to the Add function and then to the subtract function. Accordingly the parameters are passed to those functions and the function is called and executed.

Sr. No.	Manipulators	ios member function
1.	Manipulators do not return value.	ios member functions can return value.
2.	We can create our own manipulators.	We cannot create our own functions for ios.
3.	Multiple manipulators can be combined.	ios member functions cannot be combined.
4.	For manipulators we need to include the "iomanip.h" header file.	For ios functions we need to include the "iostream.h" header file.
5.	These are non-member functions.	These are member functions.

8.3 Random and Binary File Handling

Q. 8.3.1 Explain the use of binary files in C++ with a program. (Ref. Sec. 8.3) **(7 Marks)**

- In the previous sections we have seen how to read or write integer, float numbers and string into the file. In this section we will see how to read or write a binary data from/to the file.
- To read and write the binary data we have the functions get() and put() respectively.

➢ **Program 8.3.1 : Write a program to write string, integer and float to a file and read it as binary data.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char name[20],ch;
    int roll;
    float fee;
    ofstream o;
    o.open("test0",ios::in);
    o<<"Ajay"<<endl;
    o<<24<<endl;
    o<<55988.45<<endl;
    o.close();
    ifstream i;
    i.open("test0",ios::out);
    while(i)
    {
        i.get(ch);
        cout<<ch;
    }
    getch();
}
```

Output :

```
Ajay
24
```

```
55988.45
!
```

- **Program 8.3.2 : Write a program to store employee names with their designation and net pay to a file on console. Also display employee details from file. Create a class 'employee'.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<stdio.h>
class employee
{
    char name[20], designation[20];
    int pay;
    public:
    void accept()
    {
        cout<<"Enter name, designation and net pay:";
        gets(name);
        gets(designation);
        cin>>pay;
        ofstream o;
        o.open("test0",ios::in);
        o<<name<<endl<<designation<<endl<<pay;
        o.close();
    }
    void display()
    {
        char n[20],d[20];
        int p;
        ifstream i;
        i.open("test0",ios::out);
        i>>n;
        i>>d;
        i>>p;
        cout<<n<<endl<<d<<endl<<p;
        i.close();
    }
};
void main()
{
    employee e;
    e.accept();
    e.display();
    getch();
}
```

Output :

```
Enter name, designation and net pay:Ajay
Manager
20000
Ajay
Manager
20000
```

- **Program 8.3.3 : Write a program to display the contents of a text file in the reverse order (use pointer manipulation).**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char text[200],c;
    int i=0;
    ifstream in;
    in.open("test0",ios::out);
    while(!in.eof())
    {
        in.get(c);
        text[i]=c;
        i++;
    }
    i--;
    while(i>=0)
    {
        cout<<text[i];
```

```
        i--;
    }
    in.close();
    getch();
}
```

Output

```
!00002
reganaM
yajA
```

➢ **Program 8.3.4 : Write a C++ program to read character data from a file. Create one file to store all capital alphabets and another file to store all small case alphabets. Also display contents of both files.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char msg;
    int i;
    ofstream o;
    o.open("test0",ios::in);
    for(i=0;i<=25;i++)
    {
        o<<char('A'+i)<<endl;
    }
    o.close();
    o.open("test01",ios::in);
    for(i=0;i<=25;i++)
    {
        o<<char('a'+i)<<endl;
    }
    o.close();
    ifstream in;

in.open("test0",ios::out);
    cout<<"Reading from file 1:\n";
    for(i=1;i<=26;i++)
    {
        in>>msg;
        cout<<msg<<"\t";
    }
    cout<<"\n\n\nReading from file 2:"<<endl;
    in.close();
    in.open("test01",ios::out);
    for(i=1;i<=26;i++)
    {
        in>>msg;
        cout<<msg<<"\t";
    }
    in.close();;
    getch();
}
```

Output :

```
Reading from file 1:
A   B   C   D   E   F   G   H   I   J   K
L   M   N   O   P   Q   R   S   T   U   V
W   X   Y   Z
Reading from file 2:
a   b   c   d   e   f   g   h   i   j   k
l   m   n   o   p   q   r   s   t   u   v
w   x   y   z
```

➢ **Program 8.3.5 : Write a c++ program which opens two text files (only if they exist) and concatenates the contents of the second file to the first. Display an error message if the file do not exist.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char msg;
    int i;
    ofstream o;
    o.open("test0",ios::app);
    ifstream in;
    in.open("test01",ios::out);
    for(i=1;i<=26;i++)
```

```
    {
        in>>msg;
        o<<msg<<"\t";
    }
    in.close();
    o.close();
    in.open("test0",ios::out);
    cout<<"\n\n\nReading from concatenated file "<<endl;
    for(i=1;i<=52;i++)
    {
        in>>msg;
        cout<<msg<<"\t";
    }
    in.close();;
    getch();
}
```

Output :

```
Reading from concatenated file
A    B    C    D    E    F    G    H    I    J
K    L    M    N    O    P    Q    R    S    T
U    V    W    X    Y    Z    a    b    c    d
e    f    g    h    i    j    k    l    m    n
o    p    q    r    s    t    u    v    w    x
y         z
```

- **Program 8.3.6 : A file "student.txt" contains roll numbers and names. Write a C++ program to read the contents of this file and search for a student having a specific roll number.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char name[20];
    int roll,search,i;
    ifstream in;
    in.open("test02",ios::out);
    for(i=1;i<=4;i++)
    {
        in>>roll;
        cout<<roll<<"\t";
        in>>name;
        cout<<name<<endl;
    }
    in.close();
    cout<<"Enter roll number of student to search:";
    cin>>search;
    in.open("test02",ios::out);
    do
    {
        in>>roll;
        in>>name;
    }while(search!=roll);
    cout<<"roll number "<<search<<" is of student with
name "<<name;
    getch();
}
```

Output

```
24    Ajay
16    Jay
48    Vijay
64    Digvijay
Enter roll number of student to search:16
roll number 16 is of student with name Jay
```

- **Program 8.3.7 : A file contains a list of telephone numbers in the following form :**

John	**23406**
Ahmed	**56789**
...	...

The names contain only one word and the names and telephone numbers are separated by white spaces. Write a program to read the file and output the list in two columns. The names should be left justified and numbers should be right-justified in a suitable field width.

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

```
    char name[20];
    int phone,i;
    ifstream in;
    in.open("test02.txt",ios::out);
    for(i=1;i<=4;i++)
    {
        in>>name;
        cout<<name<<"\t";
        in>>phone;
        cout<<phone<<endl;

}
    in.close();
    getch();
}
```

Output :

```
Ajay    12345
Jay     01267
```

8.4 Exceptions and Error Handling During File Operations

Q. 8.4.1 List and explain error handling functions during file operations. (Ref. Sec. 8.4) **(2 Marks)**

There are various cases wherein there can be problems while accessing files, such as :

1. Attempting to access a non-existing file.
2. Some disk error.
3. Opening a read-only file in write mode.
4. Reading beyond the end of the file.
5. Opening a file with an invalid name.

There are functions in "ios" class to handle some of these errors are listed in the table below :

Sr. No.	Function	Operation
1.	eof()	This function returns "true" if it reaches the end of file while accessing the file; else it returns "false".
2.	bad()	This functions returns "true" if attempted unrecoverable errors, else it returns "false".
3.	good()	This function returs "true" if the last operation was sucessful else returns "false".
4.	clear()	This function clears all the previous error states.
5.	fail()	This function returns "true" if the last read or write operation has failed. else returns "false"".
6.	rdstate()	This function returns the status data of the class ios.

8.5 Solved Example

➢ **Program 8.5.1 : Write a program which uses command line argument to copy the contents of a file *A.txt* into another file *B.txt* by reversing case of the characters. *E.g. File A.txt: aBCd File B.txt : AbcD.***

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char msg[20];
    ofstream o;
    ifstream i;
    o.open("B.txt",ios::in);
    i.open("A.txt",ios::out);
    i>>msg;
    for(int l=0;l!=20;l++)
    {
        if(msg[l]>=65 && msg[l]<=90) msg[l]=msg[l]+32;
        else if(msg[l]>=97 && msg[l]<=112)
        msg[l]=msg[l]-32;
    }
    o<<msg;
    o.close();
    i.close();
    getch();
}
```

8.6 Exam Pack (MSBTE and Review Questions)

☞ **Syllabus Topic : C++ Stream Classes**

Q. 1 Explain various constructors of stream. **(Refer Section 8.1)** **(4 Marks)**

Q. 2 What is a stream ? **(Refer Section 8.1)** **(2 Marks)**

☞ **Syllabus Topic : Classes for File Stream Operations**

Q. 3 What are stream extraction and stream insertion operators ? **(Refer Section 8.1.1)** **(4 Mark)**

Q. 4 Explain the any two file stream classes needed for the file manipulation. **(Refer Section 8.1.1)** **(4 Marks)**

Q. 5 Write a statement using seekg() to go backward by 20 bytes from the end. **(1 Mark)**

Ans. : seekg(20,ios _base::end);

☞ **Syllabus Topic : File Modes, Opening Files, Closing Files**

Q. 6 List file mode operation. **(Refer Section 8.2)** **(3 Marks)**

Q. 7 Explain flags ios::binary and ios::ate. **(Refer Section 8.2)** **(4 Marks)**

Q. 8 Explain tellg and seekp functions. **(Refer Section 8.2)** **(4 Marks)**

Q. 9 Explain various file mode parameters in C++. Write a program to copy the contents of a source file **student1.txt** to a destination file **student 2.txt** character by character. **(Refer Section 8.2)** **(7 Marks)**

Q. 10 Explain *tellg( )* function in file streams. **(Refer Section 8.2)** **(1 Mark)**

Q. 11 Explain *ios::app* and *ios::ate* flags. **(Refer Section 8.2)** **(3 Marks)**

Q. 12 Explain the functions used to read and write data in binary file. **(Refer Section 8.2)** **(4 Marks)**

☞ **Syllabus Topic : Detecting End of File**

Q. 13 What is the basic difference between manipulators and ios member functions in implementation ? Give examples. **(Refer Section 8.2.2)** **(4 Marks)**

Q. 14 Write a program that reads a text file and creates another text file that is identical except that every letter must be converted to lower case irrespective of its original case (e.g 'a' or 'A' will become 'a'). **(Refer Program 8.2.3)** **(7 Marks)**

Q. 15 Write a program that opens two text files for reading data. It creates a third file that contains the text of first file and then that of second file (text of second file to be appended after text of the first file, to produce the third file). **(Refer Program 8.2.4)** **(7 Marks)**

Q. 16 Write a program to copy the contents of a source file student1.txt to a destination file student2.txt character by character. **(Refer Program 8.2.5)** **(4 Marks)**

Q. 17 Explain the use of binary files in C++ with a program. **(Refer Section 8.3)** **(7 Marks)**

Q. 18 List and explain error handling functions during file operations. **(Refer Section 8.4)** **(2 Marks)**

Q. 19 A file contains a list of telephone numbers in the following form :

```
John     23406
Ahmed    56789
...      ...
```

The names contain only one word and the names and telephone numbers are separated by white spaces. Write a program to read the file and output the list in two columns. The names should be left justified and numbers should be right-justified in a suitable field width. **(Refer Program 8.3.7)** **(7 Marks)**

Q. 20 Write a program to store employee names with their designation and net pay to a file on console. Also display employee details from file. Create a class 'employee'. **(Refer Program 8.3.2)** **(5 Marks)**

Q. 21 Explain the output of the following programs.

(Refer Program 8.2.6) **(2 Marks)**

```
#include<iostream.h>
#include<conio.h>
  typedef void(*Fptr)(int,int);
  void Add(int i,int j)
  {
      cout<<i<<"+"<<j<<"="<<i+j;
  }
  void subtract(int i,int j)
  {
      cout<<i<<"-"<<j<<"="<<i-j;
  }
  void main()
  {
      clrscr();
      Fptr ptr;
      ptr=& Add;
      ptr(1,2);
      cout<<endl;
      ptr=&subtract;
      ptr(3,2);
      getch();
  }
```

❑❑❑

LAB MANUAL

Lab Manual

➔ **Statement 1 : Develop minimum 2 programs using constants, variables, arithmetic expression, operators, exhibiting data type conversion.**

➢ **Program 1 : Write a C / C++ program to accept a number and display its square.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int x, y;
    cout<<"Enter a number:";
    cin>>x;
    y=x*x;
    cout<<"The square of "<<x<<" is "<<y;
    getch();
}
```

Output:

```
Enter a number:6
The square of 6 is 36
```

Explanation :

- In this program another header file is included i.e. conio.h for the two functions used in the program namely clrscr() and getch().
- The function clrscr() is used to clear the user screen and the function getch() as already discussed accepts a character from user.
- Here, we use this function, so that the processor waits for us to enter the character before closing the output window after displaying the result.
- As the processor operation is very fast it displays the result on the output screen and closes the output screen, before we can see the output. Hence, to make the processor to wait for us to see the result we give a getch() function.
- The variables x and y are declared as integer type data.
- Using the first cout statement the user is asked to provide a number whose square is to be found.
- The cin operator is used to accept the input from the user. The statement cin is associated with the extraction operator (>>), which indicates that the cin statement extracts the input from the standard input device and provides it in the variable x. The standard input device is keyboard.
- The next statement assigns the value of x multiplied by itself to provide the square in the variable y.
- The next cout statement is said to have concatenated insertion operators. The insertion operator is used along with the cout statement to display a data on the monitor.

➢ **Program 2 : Write a program to accept basic salary from the keyboard. Calculate the gross salary that includes basic salary, 50% DA and 40% HRA.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    float basic,hra,da,gross;
    cout<<"Enter the basic salary:";
    cin>>basic;
    hra=40*basic/100;
    da = 50*basic/100;
    gross=basic+da+hra;
    cout<<"The total salary is "<<gross;
    getch();
}
```

Output :

```
Enter the basic salary:10000
The total salary is 19000
```

Explanation :

- To calculate HRA we cannot write hra=40/100 * basic; as the division 40/100 will return the quotient of this division which is 0 and hence the hra will become 0.

- Once 40 is multiplied with basic the result is automatically type casted (upgraded) to the type of higher size i.e. float as one of the data was float (the variable basic is float type).

➢ **Program 3 : Write a program to accept a number and display its equivalent ASCII using type casting.**

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    int a;
    cout<<"Enter a number:";
    cin>>a;
    cout<<"This is ASCII value of: "<<(char)a;
    getch();
}
```

Output :

```
Enter a number:65
This is ASCII value of: A
```

Explanation :

This program also uses type casting operator to go to a lower data type size.

➔ **Statement 2 : Develop a program to implement decision making statements (If-else, switch).**

➢ **Program 4 : Write a program to check if the entered number is prime number or not.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i=2,n;
    cout<<"Enter a number:";
    cin>>n;
    while(n%i!=0)
    {
        i++;
    }
    if(n==i)
    {
        cout<<"Prime Number";
    }
    else
    {
        cout<<"Not a prime number";
    }
    getch();
}
```

Output :

```
Enter a number:17
Prime Number
```

Explanation :

- A simple logic is used to find out whether the entered number is prime or not. The logic is similar to the one used by us to check the number being prime or not.
- The number entered by user is checked for divisibility by 2, 3 and so on upto the entered number. If there is a divisibility then the further checking stops and the control comes out of the while loop. In worst case the control will come out of the while loop when the value of i is equal to n, because a number is definitely divisible by itself.
- Finally when the control comes out of the while loop, if n is equal to i, then the number is a prime number else it is not. This is checked using the if-else statement.

➢ **Program 5 : Write a program to check if the entered number is Armstrong or not.**

Note: A number is said to be Armstrong number if the sum of the cube of its digits is equal to the number itself. For e.g. 153, the sum of the cubes is $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$ i.e. the original number itself.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int sum=0,digit,x,copy;
```

```
    cout<<"Enter a number:";
    cin>>x;
    copy=x;
    while(x!=0)
    {
        digit=x%10;
        sum=sum+digit*digit*digit;
        x=x/10;
    }
    if(sum==copy)
    {
        cout<<"Armstrong Number";
    }
    else
    {
        cout<<"Not an Armstrong Number";
    }
    getch();
}
```

Output :

```
Enter a number:789
Not an Armstrong Number
```

Explanation :

- The variable sum is initialized to 0, and then the cubes of the user entered number are added to this variable sum.
- The method of separating the digits is same as seen in quite a lot previous programs. But in this case the cube of the digit is taken and then added to the variable sum
- Another variable called as copy is declared in this program to keep the copy of the user entered number as the number entered by the user will become 0 after dividing it by 10 in every iteration. Hence, we would not have any copy of the original number without the variable copy. To avoid this problem the variable copy is declared and keeps the value of the user entered number.

➢ **Program 6 : Write a program to display the class according to the marks scored by a student. The marks scored is taken as input and the class is displayed according to the following range :**

Marks	Class
70-100	Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int marks;
    cout<<"Enter marks scored (0-100):";
    cin>>marks;
    if(marks>=70)
    {
        cout<<"Distinction";
    }
    else
    {
        if (marks>=60)
        {
            cout<<"First Class";
        }
        else
        {
            if (marks>=50)
            {
                cout<<"Second Class";
            }
            else
            {
                if(marks>=40)
                {
                    cout<<"Pass Class";
                }
                else
                {
                    cout<<"Fail";
                }
```

```
            }
        }
    }
    getch();
}
```

Output :

```
Enter marks scored (0-100):89
Distinction
```

Explanation :

- The above program is said to implement if else ladder with the operation as explained just before the program.

➢ **Program 7 : Write a program to display a user entered number in words.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,digit,rev=0;
    cout<<"Enter a number:";
    cin>>n;
    while(n!=0)
    {
        digit=n%10;
        rev=rev*10+digit;
        n=n/10;
    }
    while(rev!=0)
    {
        digit=rev%10;
        rev=rev/10;
        switch(digit)
        {
            case 0:cout<<"Zero ";
            break;
            case 1:cout<<"One ";
            break;
            case 2:cout<<"Two ";
            break;
            case 3:cout<<"Three ";
            break;
            case 4:cout<<"Four ";
            break;
            case 5:cout<<"Five ";
            break;
            case 6:cout<<"Six ";
            break;
            case 7:cout<<"Seven ";
            break;
            case 8:cout<<"Eight ";
            break;
            case 9:cout<<"Nine ";
            break;
        }
    }
    getch();
}
```

Output :

```
Enter a number:513
Five One Three
```

Explanation :

The number is first reversed using the logic we have already done in one of the previous program. This is done because the last digit is separated first using the technique we have already studied. Using the switch case statement the corresponding words of the digits are selected and printed.

➔ **Statement 3 : Develop a program to demonstrate control structures (for, while, do-while).**

➢ **Program 8 : Write a program to display first n elements of Fibonacci series.**

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

```
    clrscr();
    int a=0,b=1,c,i,n;
    cout<<"Enter a number: ";
    cin>>n;
    cout<<"Fibonacci Series\n0\n1\n";
    for(i=1;i<=n-2;i++)
    {
        c=a+b;
        cout<<c<<endl;
        a=b;
        b=c;
    }
    getch();
}
```

Output :

```
Enter a number: 10
Fibonacci Series
0
1
1
2
3
5
8
13
21
34
```

Note : Fibonacci series is a series wherein the first two elements are 0 and 1. Thereafter the next values are the sum of previous two elements. Hence it can be said that the series is 0,1,1,2,3,5,8,13,21,34,55 . . .

Explanation :

- We first display the values 0 and 1 i.e. the first two elements of Fibonacci series. Also these two values are initialized in the variables a and b.
- In every iteration, the next value is calculated and stored in the variable c and is printed. Also the values of a and b are updated accordingly i.e. the latest two values calculated must always be in a and b.
- The condition statement of the for loop is important in this case. Since, the first two elements are already printed the counting of printing the elements of Fibonacci series in the loop must go only upto n-2 i.e. two values less which are printed outside the loop.

➢ **Program 9 : Write a program to calculate the value of the following series.**

$$1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}$$

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int i,n,fact=1;
    float sum=0.0;
    cout<<"Enter a number: ";
    cin>>n;
    for(i=1;i<=n;i++)
    {
        fact=fact*i;
        sum=sum+1.0/fact;
    }
    cout<<"The value of the series is:"<<sum;
    getch();
}
```

Output :

```
Enter a number: 5
The value of the series is:1.716667
```

Explanation :

- The factorial is calculated for every term in the variable fact. The previous value of fact is just updated by multiplying the next number to the previous factorial. For example if the variable fact has the factorial of 4, when multiplied by 5 (i), the resultant is the factorial of 5.
- Remaining logic is same as the previous program.

- **Program 10 : Write a program to count the number of digits in a user entered number.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,count=0;
    cout<<"Enter a number:";
    cin>>n;
    while(n!=0)
    {
        n=n/10;
        count++;
    }
    cout<<"Number of digits="<<count;
    getch();
}
```

Output :

```
Enter a number:7567
Number of digits=4
```

- **Program 11 : Write a program to find the sum and product of all the digits of a user entered number using the do-while loop.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,digit,sum=0,product=1;
    cout<<"Enter a number:";
    cin>>n;
    do
    {
        digit=n%10;
        sum=sum+digit;
        product=product*digit;
        n=n/10;
    }while(n!=0);
    cout<<"Sum="<<sum<<endl<<"Product="
<<product;
    getch();
}
```

Output :

```
Enter a number:1345
Sum=13
Product=60
```

➔ Statement 4 : Develop a program to implement 1-dimension array.

- **Program 12 : Write a program to sort numbers in ascending order.**

  OR Write a program to implement bubble sorting algorithm for sorting numbers in ascending order.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n,i,a[100],x,index;
    cout<<"Enter the number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    void ascend (int a[], int n);
    ascend(a,n);
    getch();
}
void ascend (int a[], int n)
{
    int i,j,temp;
    for(i=0;i<=n-2;i++)
    {
```

```
        for(j=0;j<=n-2;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    cout<<"After sorting\n";
    for(i=0;i<=n-1;i++)
    {
        cout<<a[i]<<endl;
    }
}
```

Output :

```
Enter the number of elements:5
Enter a value:4
Enter a value:6
Enter a value:1
Enter a value:85
Enter a value:9
After sorting
1
4
6
9
85
```

Explanation :

- The method of accepting and passing the elements of the array is similar to the previous programs. Now let us see the bubble sorting algorithm in detail.
- Bubble sorting algorithm says that compare the consecutive elements from the beginning of array till the end. Wherever the proper sorting is not found, swap the numbers. And this entire set of comparisons is to be repeated for n-1 times. Let us take the example of numbers as given in the output of the program to understand this logic.
- The first two elements are compared i.e. 4 and 6, since they are already sorted i.e. 4<6, no swapping takes place. The next comparison is the next two elements i.e. 6 and 1; now these are not sorted i.e. 6>1, hence these are swapped. This continues until the last two elements are compared as shown in Fig. P.12.
- You will notice in Fig. P.12, after all the comparisons done in the first iteration; the largest number has reached to the last position. But the numbers are still not sorted properly. To sort these numbers we need to repeat the above process for n-1 times.

Output :

- Note : Since the "j" and the "j+1" elements are compared the value of "j" must go maximum upto n-2. When the value of "j" is n-2, we will be comparing the n-2 and n-1 i.e. the last two elements. Also since the value of "i" is starting from 0, we have to go upto n-2, as we have to perform n-1 comparisons

	Initial values	After first comparision	After second comparision	After third comparision	After fourth comparision
Iteration 1		4>6, No, hence no swap	6>1, Yes, hence swap	6>85, No, hence no swap	85>9, Yes, hence swap
	4	4	4	4	4
	6	6	1	1	1
	1	1	6	6	6
	85	85	85	85	9
	9	9	9	9	85
Iteration 2		4>1, Yes, hence swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85
Iteration 3		1>4, No, hence no swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap

	Initial values	After first comparision	After second comparision	After third comparision	After fourth comparision
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85
Iteration 4		1>4, No, hence no swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85

Fig. P. 12 : Bubble sorting example

- You may also notice in the Fig. P. 12, that the sorting is already completed after the second iteration. The remaining two iterations do no swapping. But, we have to consider the worst case condition i.e. if all the numbers entered by the user were in exactly reverse order. In this case we would require all the four iterations. This is explained in the Fig. P. 12(a).
- Here, you will notice that till the last iteration the swapping is required. The worst case condition is one in which the numbers are exactly in the reverse order as expected to be. You will notice in this case the numbers are in descending order and they are to be sorted in ascending order.

	Initial values	After first comparision	After second comparision	After third comparision	After fourth comparision
Iteration 1		5>4, Yes, hence swap	5>3, Yes, hence swap	5>2, Yes, hence swap	5>1, Yes, hence swap
	5	4	4	4	4
	4	5	3	3	3
	3	3	5	2	2
	2	2	2	5	1
	1	1	1	1	5
Iteration 2		4>3, Yes, hence swap	4>2, Yes, hence swap	4>1, Yes, hence swap	4>5, No, hence no swap
	4	3	3	3	3
	3	4	2	2	2
	2	2	4	1	1
	1	1	1	4	4
	5	5	5	5	5
Iteration 3		3>2, Yes, hence swap	3>1, Yes, hence swap	3>4, No, hence no swap	4>5, No, hence no swap
	3	2	2	2	2
	2	3	1	1	1
	1	1	3	3	3
	4	4	4	4	4
	5	5	5	5	5
Iteration 4		2>1, Yes, hence swap	2>3, No, hence no swap	3>4, No, hence no swap	4>5, No, hence no swap
	2	1	1	1	1
	1	2	2	2	2
	3	3	3	3	3
	4	4	4	4	4
	5	5	5	5	5

Fig. P. 12(a) : Worst case condition of sorting numbers using Bubble sorting

➔ **Statement 5 : Develop a program to perform matrix operations using multi-dimensional array.**

➢ **Program 13 : Write a program to multiply two matrices using a function.**

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int m,n,p,i,j,a[10][10],b[10][10];
```

```
    cout<<"Enter the number of rows and columns of matrix
1:";
    cin>>m>>n;
    cout<<"Enter the values of matrix 1\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=n-1;j++)
        {
            cout<<"Enter a value:";
            cin>>a[i][j];
        }
    }
    cout<<"The number of rows for matrix 2 will
be"<<n<<endl;
    cout<<"Enter the columns of matrix 2:";
    cin>>p;
    cout<<"Enter the elements of matrix 2:\n";
    for(i=0;i<=n-1;i++)
    {
        for(j=0;j<=p-1;j++)
        {
            cout<<"Enter a value:";
            cin>>b[i][j];
        }
    }
void MatMul (int a[10][10],int b[10][10], int m, int n, int p);
    MatMul(a,b,m,n,p);
    getch();
}
void MatMul (int a[10][10], int b[10][10], int m, int n, int p)
{
    int i,j,k,c[10][10];
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=p-1;j++)
        {
            c[i][j]=0;
            for(k=0;k<=n-1;k++)
            {
                c[i][j]=c[i][j]+a[i][k]*b[k][j];
            }
        }
    }
    cout<<"The resultant matrix is:\n";
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=p-1;j++)
        {
            cout<<c[i][j]<<"\t";
        }
        cout<<endl;
    }
}
```

Output :

```
Enter the number of rows and columns of matrix 1:3
2
Enter the values of matrix 1
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
The number of rows for matrix 2 will be 2
Enter the columns of matrix 2:3
Enter the elements of matrix 2:
Enter a value:1
Enter a value:2
Enter a value:3
Enter a value:4
Enter a value:5
Enter a value:6
The resultant matrix is:
9     12    15
19    26    33
29    40    51
```

Explanation :

- The method to accept the elements of the matrices is same as the previous program.
- To multiply two matrices their sizes must be m × n and n × p respectively. Hence the number of rows of the matrix 2 is kept same as the number of columns of matrix 1. The size of the resultant matrix will be m × p.
- To calculate the resultant matrix the two matrices and their sizes are passed to the function.
- The function has three level of nested "for" loops. The outer two loops are used to select an element of the resultant matrix whose value is to be calculated. Initially the value is made zero. Then, the corresponding elements of matrix 1 and 2 are multiplied and added to get the terms in an element of the resultant matrix. The terms required to get an element of the resultant matrix are shown in the Fig.P.13 considering the example of the matrices to be multiplied to be of sizes m × n and n × p respectively.

$$\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{bmatrix} \times \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{bmatrix} = \begin{bmatrix} (a_{0,0}\times b_{0,0}) + (a_{0,1}\times b_{1,0}) & (a_{0,0}\times b_{0,1}) + (a_{0,1}\times b_{1,1}) & (a_{0,0}\times b_{0,2}) + (a_{0,1}\times b_{1,2}) \\ (a_{1,0}\times b_{0,0}) + (a_{1,1}\times b_{1,0}) & (a_{1,0}\times b_{0,1}) + (a_{1,1}\times b_{1,1}) & (a_{1,0}\times b_{0,2}) + (a_{1,1}\times b_{1,2}) \\ (a_{2,0}\times b_{0,0}) + (a_{2,1}\times b_{1,0}) & (a_{2,0}\times b_{0,1}) + (a_{2,1}\times b_{1,1}) & (a_{2,0}\times b_{0,2}) + (a_{2,1}\times b_{1,2}) \end{bmatrix}$$

Fig. P. 13 : Multiplication of matrices

- Hence, in the Fig. P. 13 you will notice that to calculate each term of the resultant matrix, we need to add many product terms. This is what is implemented by the three level nested loops. Initially the element of the resultant matrix is made zero, then each of the products are added to this element and hence obtain the entire expression value.
- To find each element, you will notice that the column number of matrix 1 is changed while the row number for matrix 2. For example in the first term, $(a_{0,0}\times b_{0,0}) + (a_{0,1}\times b_{1,0})$, as seen the column number of matrix 1 is 0 for the first term and 1 for the second term, while the row number of matrix 2 is 0 for the first term and 1 for the second term. Hence, the third nested loop varies the value of k from 0 to n-1 i.e. the number of columns for matrix 1 or number of rows for matrix 2.

➔ Statement 6 : Develop programs that implements a class and use it with objects.

➢ **Program 14 : Write a program to find area of circle using Object Oriented Programming such that the class circle must have three member functions namely :**

(a) read() to accept the radius from the user.

(b) compute() for calculating the area.

(c) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    void read()
    {
        cout<<"Enter radius:";
        cin>>r;
    }
    void compute()
    {
        a=3.14*r*r;
    }
    void display()
    {
        cout<<"Area="<<a;
    }
};
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

Explanation :

- A class is created with the name 'circle'. Two float type variables are declared in it namely 'r' and 'a' for the radius and area respectively.
- The three functions are written inside the class 'circle' namely read(), compute() and display().
- The read() function accepts the value of radius from the user. The compute() function calculates the area of the circle. The display() function displays the calculated area.
- But all these functions will perform these tasks only when they are called. These functions can be called by the period operator used with an object of this class.
- Hence, an object of this class is created in the main function. The syntax of making an object of a class is very simple.

  Syntax of declaring an object of a class:

  Class_name Object_name;

  For example, in the above case we have made an object of the class "circle" by the statement,

  circle c;
- Thereafter; we have called one by one each of the functions of the class 'circle' in the required sequence. To call the function of the class we have used the period operator('.'). The syntax of accessing a member function of a class is as shown below:

  Object_name.Function_name();

  For example, in the above case to call the function read(), we have the statement,

  c.read();
- Similarly; the compute() function and the display() function are called.
- Hence, the read() function accepts the input from user as seen in the output and the display() function displays the result calculated by the compute() function.

➔ Statement 7 : Develop programs that implements a class and create array of objects.

➢ **Program 15 : Write a program to declare a class 'staff' having data members as name and department. Accept this data for 10 staffs and display names of staff that are in cm department.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Staff
{
    private:
    char name[20],dept[20];
    public:
    void accept()
    {
        cout<<"Enter name and department:";
        cin>>name>>dept;
    }
    void display()
    {
        cout<<name<<"\t"<<dept<<"\n";
    }
};
void main()
{
    int i;
    clrscr();
    Staff s[10];
    for(i=0;i<=9;i++)
    {
        s[i].accept();
    }
    cout<<"Name\tDepartment\n";
    for(i=0;i<=9;i++)
    {
        s[i].display();
    }
    getch();
}
```

Output :

```
Enter name and department:a
finance
Enter name and department:Ajay
HR
Enter name and department:Vijay
HR
Enter name and department:Jay
Finance
Enter name and department:Digvijay
Finance
Enter name and department:Manoj
Sales
Enter name and department:Manish
Sales
Enter name and department:Suresh
Finance
Enter name and department:Dinesh
HR
Enter name and department:Saurav
HR
Name      Department
a         finance
Ajay      HR
Vijay     HR
Jay       Finance
Digvijay  Finance
Manoj     Sales
Manish    Sales
Suresh    Finance
Dinesh    HR
Saurav    HR
```

➔ **Statement 8 : Write a program to implement friend function.**

➢ **Program 16 : Write a program to add two complex numbers using a friend function.**

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a complex number:";
        cin>>x>>y;
    }
    friend Complex operator + (Complex c1, Complex c2);
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
Complex add (Complex c1, Complex c2)
{
    Complex c;
    c.x=c1.x+c2.x;
    c.y=c1.y+c2.y;
    return c;
}
void main()
{
    clrscr();
    Complex c1;
    Complex c2;
    Complex c3;
    c1.read();
    c2.read();
    c3=c1.add(c2);
    c3.display();
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

- Here in the class "Distance", we have mentioned that the operator overloading function is a friend to this class. This function is then defined outside the class. This function will accept two objects of the class "Complex". This function also returns an object of the same class i.e. "Complex".

➔ Statement 9 : Write a program to implement inline function.

➢ **Program 17 : Write a program to find area of circle using Object Oriented Programming such that the class circle must have three inline functions namely :**

(a) read() to accept the radius from the user.

(b) compute() for calculating the area.

(c) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    void read();
    void compute();
    void display();
};
inline void circle::read()
{
    cout<<"Enter radius:";
    cin>>r;
}
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

➔ Statement 10 : Write a program to implement all types of constructors (constructor overloading) with destructor.

➢ **Program 18 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the parameterized constructor and the class circle must have two inline functions namely :**

(a) compute() for calculating the area.

(b) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    circle(float x)
    {
        r=x;
    }
    void compute();
    void display();
```

```
};
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    float p;
    cout<<"Enter the radius of the circle";
    cin>>p;
    circle c(p);
    c.compute();
    c.display();
    getch();
}
```

Output :

```
Enter radius:5
Area=78.5
```

Explanation :

- Here, the constructor has an argument to be accepted i.e. a float variable 'x'. This variable is to be passed to the constructor when the object is made.
- To pass the variable to the constructor, it has to be mentioned in the brackets along with the object name while making the object.You will notice that the variable is passed while making the object in the brackets along with the object i.e. in the statement:

  circle c(p);
- Since the variable 'p' is to be passed while making the object, the value of variable 'p' is to be accepted from user first and then the object can be created.

➢ **Program 19 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the copy constructor and the class circle must have two inline functions namely :**

(a) compute() for calculating the area.

(b) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    circle(float x)
    {
        r=x;
    }
    circle(circle &c)
    {
        r=c.r;
    }
    void compute();
    void display();
};
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
    cout<<"Area="<<a<<endl;
}
void main()
{
    clrscr();
    float p;
    cout<<"Enter the radius of the circle";
    cin>>p;
    circle c(p);
    c.compute();
    c.display();
    circle c1(c);
    c1.compute();
    c1.display();
    getch();
}
```

Output :

```
Enter the radius of the circle5
Area=78.5
Area=78.5
```

Explanation :

- There are two constructors, hence this concept is also called as constructor overloading.
- The first constructor has an argument to be accepted i.e. a float variable 'x'. This variable is to be passed to the constructor when the object is made.
- The second constructor has an argument of type 'circle'. This type is the object of the same class i.e. 'circle'. This is the copy constructor. We have passed the earlier created object 'c' to the constructor of the second object 'c1' to initialize its variable 'r'.
- Both the objects are created in the main() function with the respective constructors being called as discussed in the previous point. The other functions are called individually for each object.Remember memory space allocated for each object are different.

➢ **Program 20 : Write a program to demonstrate the destructor.**

```
# include<iostream.h>
# include<conio.h>
class test
{
    int *p;
    public:
    test()
    {
        p=new int;
    }
    void read()
    {
        cout<<"Enter a number";
        cin>>*p;
    }
    void display()
    {
        cout<<"Value="<<*p<<endl;
    }
    ~test()
    {
        delete p;
        cout<<"Destroyed";
    }
};
void main()
{
    clrscr();
    test t;
    t.read();
    t.display();
    getch();
}
```

Output :

```
Enter a number5
Value=5
Destroyed
```

Explanation :

- The destructor is the ~test() function. This destructor is automatically called after the end of the execution of the main() function. Hence, in our case after the statement getch(); the destructor is automatically called.
- When the destructor is called, the memory allocated by the constructor is deleted using the delete operator. The statement;

  cout<<"Destroyed";

  is then executed.
- Hence, similar to constructor even the destructor is not to be called explicitly. It is automatically called.

➔ **Statement 11 : Write a program for implementing single inheritance.**

➢ **Program 21 : Write a program to add two numbers using single inheritance such that the base class function must accept the two numbers from the user and the derived class function must add these numbers and display the sum.**

Solution :

The class diagram of this requirement is as shown in Fig. P. 21. This diagram shows that the class "Sum" is derived from class "Data" in public visibility.

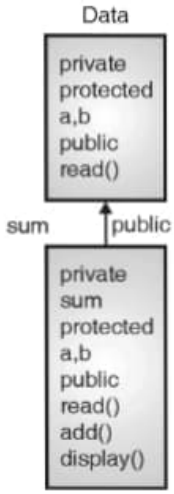


Fig. P. 21 : Class Diagram of Single Inheritance for Program 21

```
# include<iostream.h>
# include<conio.h>
class Data
{
    protected:
    int a,b;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>a>>b;
    }
};
class Sum: public Data
{
    private:
    int sum;
    public:
    void add()
    {
        sum=a+b;
    }
    void display()
    {
        cout<<"The sum is "<<sum;
    }
};
void main()
{
    clrscr();
    Sum s;
    s.read();
    s.add();
    s.display();
    getch();
}
```

Output :

```
Enter two numbers4
5
The sum is 9
```

Explanation :

- As shown in the class diagram, the base class data has two protected member variables namely 'a' and 'b'. These members are kept protected, so that they are accessible from the derived class function to calculate the sum.
- The base class also has a function in public visibility to read the inputs from user. The derived class also has public member functions add() and display() to find the sum of the two numbers taken and to display this sum respectively.
- The class "Sum" is derived from the class "Data". This derivation is implemented using the already discussed syntax in the statement:

  class Sum: public Data

- The members of the base class are shown available in the derived class in the class diagram in Fig.P.21 but these member functions and member variables are not to be defined again in the derived class when actually writing the program. This concept of gaining access to the members of the base class is called as **code reusability**.

- The main() function has first a creation of the object of the class "Sum". The object of the class sum can access all the functions i.e. read(), add() and display(); because the derived class derives the protected members in the same visibility.
- The functions to read, calculate the sum and display the sum are thereafter called one by one.

➔ Statement 12 : Write a program for implementing multi level inheritance.

➢ **Program 22 : Write a program to calculate percentage of a student using multi level inheritance. The base class function will accept the marks in three subjects from user. A class will be derived from the above mentioned class that will have a function to find the total marks obtained and another class derived from this will have functions to calculate and display the percentage scored.**

Solution :

The class diagram of this requirement is as shown in Fig. P. 22.

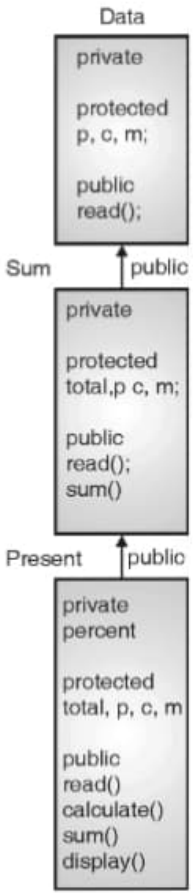


Fig. P. 22 : Class Diagram of Multi Level Inheritance for Program 22

```
# include<iostream.h>
# include<conio.h>
class Data
{
    protected:
    int p,c,m;
    public:
    void read()
    {
        cout<<"Enter the marks obtained in Physics,
Chemistry and Maths";
        cin>>p>>c>>m;
    }
};
class Sum: public Data
{
    protected:
    int total;
    public:
    void sum()
    {
        total=p+c+m;
    }
};
class Percent: public Sum
{
    private:
    float percent;
    public:
    void calculate()
    {
        percent=total/300.0*100;
    }
    void display()
    {
        cout<<"The percentage is "<<percent;
    }
};
void main()
{
```

```
    clrscr();
    Percent a;
    a.read();
    a.sum();
    a.calculate();
    a.display();
    getch();
}
```

Output :

```
Enter the marks obtained in Physics, Chemistry and Maths90
98
99
The percentage is 95.666664
```

Explanation :

- The most important part of a program based on inheritance is the class diagram. Once you draw the class diagram then writing the program to implement the diagram is very simple.
- This program is also just an implementation of the class diagram drawn in the Fig. P. 22. The implementation is similar to that as seen in single inheritance, only that there is one more level of inheritance.

➔ Statement 13 : Write a program for implementing multiple inheritance.

➢ **Program 23 : Write a program to define the following relationship using multiple inheritance.**

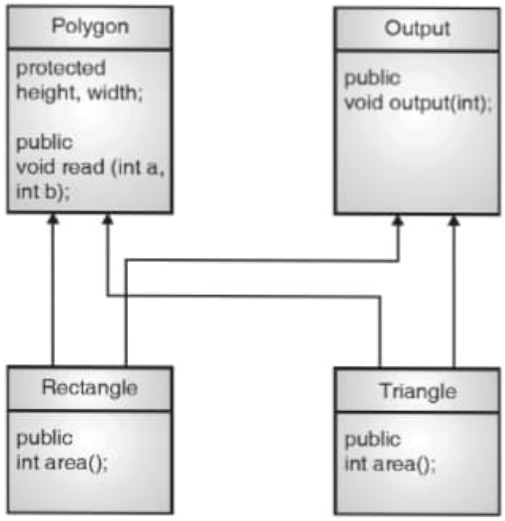


Fig. P. 23 : Class Diagram of Multiple Inheritance for Program 23

```
# include<iostream.h>
# include<conio.h>
class Polygon
{
    protected:
    int height, width;
    public:
    void read(int a, int b)
    {
        height=a;
        width=b;
    }
};
class Output
{
    public:
    void output(int x)
    {
        cout<<"Area is "<<x;
    }
};
class Rectangle: public Polygon, public Output
{
    public:
    int area()
    {
        return(height*width);
    }
};
class Triangle: public Polygon, public Output
{
    public:
    int area()
    {
        return(height*width/2);
    }
};
void main()
```

```
{
    clrscr();
    int h,w,choice,a;
    cout<<"1. Area of Rectangle\n2. Area of Triangle\nEnter your choice:";
    cin>>choice;
    cout<<"Enter height and width:";
    cin>>h>>w;
    switch(choice)
    {
        case 1:
        Rectangle r;
        r.read(h,w);
        a=r.area();
        r.output(a);
        break;
        case 2:
        Triangle t;
        t.read(h,w);
        a=t.area();
        t.output(a);
        break;
        default: cout<<"Invalid choice";
    }
    getch();
}
```

Output :

```
1. Area of Rectangle
2. Area of Triangle
Enter your choice:2
Enter height and width:5
4
Area is 10
```

Explanation :

- The most important part of a program based on inheritance is the class diagram. Once you draw the class diagram then writing the program based on the diagram is very simple.
- Here, also we have just implemented the class diagram drawn in the Fig. P. 23.

➔ **Statement 14 : Develop minimum I program to demonstrate Pointer to object.**

➢ **Program 24 : Write a program to declare a class birthday having data members day, month and year. Accept this information for 5 objects using pointer to the array of object**

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

class Birthday
{
  int day, month, year;
public:
  void getdata()
  {
    cout<<"\n\t\tEnter birthdate ( day month year)  ::";
    cin>>day>>month>>year;
  }
    void putdata()
    {
      cout<<" "<<day<<"-"<<month<<"-"<<year;
    }
};
void main()
{
 Birthday B[5],*ptr;
 clrscr();
 ptr = &B[0];
 for(int i=0;i<5;i++)
 {
 cout<<"\n\t\tEnter birthdate for object "<<i+1;
 ptr->getdata();
 ptr++;
 }
 ptr = &B[0];
 for(i=0;i<5;i++)
```

```
{
cout<<"\n\t\tBirthdate for object ::: "<<i+1;
ptr->putdata();
ptr++;
}
getch();
}
```

Output :

```
Enter birthdate for object 1
Enter birthdate ( day month year) :: 16 11 2005

Enter birthdate for object 2
Enter birthdate ( day month year) :: 15 10 1975
Enter birthdate for object 3
Enter birthdate ( day month year) :: 21 12 1979

Enter birthdate for object 4
Enter birthdate ( day month year) :: 10 03 1978
Enter birthdate for object 5
Enter birthdate ( day month year) :: 03 12 1987
Birthdate for object 1 :: 16-11-2005
Birthdate for object 2 :: 15-10-1975
Birthdate for object 3 :: 21-12-1979
Birthdate for object 4 :: 10-3-1978
Birthdate for object 5 :: 3-12-1987
```

➔ Statement 15 : Develop minimum 1 program to demonstrate Pointer to derived class.

➢ Program 25 : Write a program to demonstrate use of pointer to object.

```
#include<iostream.h>
#include<conio.h>
class Base
{
    protected:
    int a,b;
    public:
    virtual void read()
    {
        cout<<"Enter two values:";
        cin>>a>>b;
    }
    virtual void display()
    {
        cout<<"\nThe values
are:"<<a<<"\n"<<b<<endl;
    }
};
class Sub: public Base
{
    protected:
    int c,d;
    public:
    virtual void read()
    {
        cout<<"Enter 4 values:";
        cin>>a>>b>>c>>d;
    }
    virtual void display()
    {
        cout<<"\nThe values are:"
    <<a<<"\n"<<b<<"\n"<<c<<"\n"<<d<<endl;
    }
};
void main()
{
    clrscr();
    Base *ptr;
    Base b;
    Sub s;
    ptr=&b;
    ptr->read();
    ptr->display();
    ptr=&s;
    ptr->read();
    ptr->display();
```

```
    getch();
}
```

Output :

```
Enter 2 values:1
2

The values are:1
2
Enter 4 values:1
2
3
4

The values are:1
2
3
4
```

➔ Statement 16 : Write a program to demonstrate operator overloading for Unary operator.

➢ **Program 26 : Write a program to negate the values of two variables contained in an object.**

```
#include<iostream.h>
#include<conio.h>
class Negate
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>x>>y;
    }
    void compute()
    {
        x=-x;
        y=-y;
    }
    void display()
    {
        cout<<"x="<<x<<endl<<"y="<<y;
    }
};
void main()
{
    clrscr();
    Negate n;
    n.read();
    n.compute();
    n.display();
    getch();
}
```

Output :

```
Enter two numbers4
6
x=-4
y=-6
```

Explanation :

- A simple program with three functions respectively to accept the numbers, compute the negative and to display the result.
- An object of the class is made and the three functions are called one after the other.
- Let us see the same operation by overloading the unary minus ('-') operator in the following program.

➢ **Program 27 : Write a program to negate the values of two variables contained in an object by overloading the operator unary negate i.e. '-'.**

```
#include<iostream.h>
#include<conio.h>
class Negate
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter two numbers";
        cin>>x>>y;
```

```
    }
    void operator -()
    {
        x=-x;
        y=-y;
    }
    void display()
    {
        cout<<"x="<<x<<endl<<"y="<<y;
    }
};
void main()
{
    clrscr();
    Negate n;
    n.read();
    -n;
    n.display();
    getch();
}
```

Output :

```
Enter two numbers4
6
x=-4
y=-6
```

Explanation :

- In this case we have implemented the operator overloading operation by overloading the operator unary minus i.e. '-'.
- Compare to previous program where we had to call the compute() function, here we have to just use the negate operator with the object of the class i.e. "–n;".
- Actually operators can work only with variables of basic data type. The operators cannot be used with the objects of a class. But operator overloading allows those operators that are overloaded in a class to be used with the objects of that class.
- Remember, we cannot use an operator with an object unless we have written an operator overloading function for that operator in the corresponding class.

➔ Statement 17 : Write a program to demonstrate operator overloading for Binary operator.

➢ **Program 28 : Write a program to overload binary operator "+=" to add two complex numbers.**

```
#include<iostream.h>
#include<conio.h>
class Complex
{
    int x,y;
    public:
    void read()
    {
        cout<<"Enter the real and imaginary parts of a
complex number:";
        cin>>x>>y;
    }
    void operator += (Complex c)
    {
        x=x+c.x;
        y=y+c.y;
    }
    void display()
    {
        if(y<0)
        cout<<x<<y<<"i";
        else
        cout<<x<<"+i"<<y;
    }
};
void main()
{
    clrscr();
    Complex c1;
    Complex c2;
    c1.read();
    c2.read();
    c1+=c2;
    c1.display();
```

```
    getch();
}
```

Output :

```
Enter the real and imaginary parts of a complex number:2
3
Enter the real and imaginary parts of a complex number:4
5
6+i8
```

Explanation :

- Here we have overloaded the binary operator "+" by the operator overloading function Complex operator + (Complex c).
- Two objects of the class complex are created. The values of these complex variables are read.
- The two objects of the class "complex" are added using the same syntax of the addition operator i.e. '+'. The result is returned to another object of the same class.

➔ Statement 18 : Write a program to demonstrate function overloading.

➢ **Program 29 : Write a program to add two numbers using function overloading such that one function adds two integers, second function adds two float numbers and the third function adds a float number with an integer.**

```
# include<iostream.h>
# include<conio.h>
float add(float a, int b)
{
    float c;
    c=a+b;
    return c;
}
int add(int a, int b)
{
    int c;
    c=a+b;
    return c;
}
float add(float a, float b)
{
    float c;
    c=a+b;
    return c;
}
void main()
{
    clrscr();
    int x,a=5,b=6;
    float y,p=3.5,q=6.6;
    x=add(a,b);
    cout<<"Sum="<<x<<endl;
    y=add(p,q);
    cout<<"Sum="<<y<<endl;
    y=add(p,a);
    cout<<"Sum="<<y<<endl;
    getch();
}
```

Output :

```
Sum=11
Sum=10.1
Sum=8.5
```

Explanation :

- The first call to function add() in the main() function, has both the integer elements 'a' and 'b'; hence the second add() function is called which has both the integer parameters.
- The second call to function add() in the main() function, has both the float type elements 'p' and 'q'; hence the third add() function is called which has both the float type parameters
- The third call to function add() in the main() function, has a float type element 'p' and an integer type element 'a'; hence the first add() function is called with a similar type of parameters.
- This is a program example wherein the number of parameters for all the overloaded functions is same but their data types are different.

➔ Statement 19 : Write a program to read and write data to and from a file.

➢ **Program 30 : Write a program to write and read string, integer and float from/to a file.**

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char name[20];
    int roll;
    float fee;
    ofstream o;
    o.open("test0",ios::in);
    o<<"Ajay"<<endl;
    o<<24<<endl;
    o<<55988.45<<endl;
    o.close();
    ifstream i;
    i.open("test0",ios::out);
    i>>name;
    i>>roll;
    i>>fee;
    cout<<"Name:"<<name<<endl;
    cout<<"Roll No."<<roll<<endl;
    cout<<"Fee:"<<fee<<endl;
    getch();
}
```

Output :

```
Name:Ajay
Roll No.24
Fee:55988.449219
```

❑❑❑

Appendix A

Solved MSBTE Question Papers of Winter 2018 and Summer 2019

Object Oriented Programming Using C++

Winter 2018

Q. 1(a) State any four object oriented languages. **(02 Marks)**

Ans. : The following are Object Oriented programming languages :

1. C++
2. Java
3. Python
4. Visual Basic
5. .NET

Q. 1(b) Describe use of protected access specifier used in the class. **(Section 6.1.1)** **(02 Marks)**

Q. 1(c) Differentiate between OOP and POP. **(Section 1.6.1)** **(02 Marks)**

Q. 1(d) Write any two characteristics of destructor. **(Section 5.4)** **(02 Marks)**

Q. 1(e) Describe meaning of following:

(i) ios : : in

(ii) ios : : out **(Section 8.2)** **(02 Marks)**

Q. 1(f) Give output for following code : **(02 Marks)**

```
class student
{
        int roll no;
        char name [14];
} s[6];
void main( )
{
    cout<<sizeof(s);
}
```

Ans. : Output : 96

Q. 1(g) Write syntax to define a derived class. **(Section 6.1.1)** **(02 Marks)**

Q. 2(a) Write a C++ program to accept array of five elements, find and display smallest number from an array. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n=5,i,a[5],small;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    int smallest (int a[], int n);
    small=smallest(a,n);
    cout<<"The smallest number is "<<small;
    getch();
}
int largest (int a[], int n)
{
    int i,small;
    small=a[0];
    for(i=1;i<=n-1;i++)
    {
        if(small>a[i])
        small=a[i];
    }
    return small;
}
```

Appendix A

Solved MSBTE Question Papers of Winter 2018 and Summer 2019

Object Oriented Programming Using C++

Winter 2018

Q. 1(a) State any four object oriented languages. **(02 Marks)**

Ans. : The following are Object Oriented programming languages :

1. C++
2. Java
3. Python
4. Visual Basic
5. .NET

Q. 1(b) Describe use of protected access specifier used in the class. **(Section 6.1.1)** **(02 Marks)**

Q. 1(c) Differentiate between OOP and POP. **(Section 1.6.1)** **(02 Marks)**

Q. 1(d) Write any two characteristics of destructor. **(Section 5.4)** **(02 Marks)**

Q. 1(e) Describe meaning of following:

(i) ios : : in

(ii) ios : : out **(Section 8.2)** **(02 Marks)**

Q. 1(f) Give output for following code : **(02 Marks)**

```
class student
{
        int roll no;
        char name [14];
} s[6];
void main( )
{
    cout<<sizeof(s);
}
```

Ans. : Output : 96

Q. 1(g) Write syntax to define a derived class. **(Section 6.1.1)** **(02 Marks)**

Q. 2(a) Write a C++ program to accept array of five elements, find and display smallest number from an array. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int n=5,i,a[5],small;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    int smallest (int a[], int n);
    small=smallest(a,n);
    cout<<"The smallest number is "<<small;
    getch();
}
int largest (int a[], int n)
{
    int i,small;
    small=a[0];
    for(i=1;i<=n-1;i++)
    {
        if(small>a[i])
        small=a[i];
    }
    return small;
}
```

Output :

```
Enter the number of elements:5
Enter a value:21
Enter a value:33
Enter a value:12
Enter a value:24
Enter a value:11
The smallest number is 12
```

Q. 2(b) Write a C++ program to declare a class 'College' with data members as name and college code. Derive a new class 'student' from the class college with data members as sname and roll no. Accept and display details of one student with college data. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class College
{
    protected:
        char name[20];
        int code;
};
class Student
{
    char name[20];
    int roll_no;
    public:
    void accept()
    {
        cout<<"Enter college name, code, student name
                and roll number:";
        cin>>name>>code>>sname>>roll_no;
    }
    void display()
    {
        cout<<"College name:"<<name <<endl<<
        "College code:"<<code<<endl<<"Student name:"
        << sname<<endl<<"Roll Number:"<<roll_no;
    }
};
void main()
{
    Student s;
    clrscr();
    s.accept();
    s.display();
    getch();
}
```

Output :

```
Enter college name, code, student name and roll number:
Xaviers
123
Vijaya
24
College name:Xaviers
College code:123
Student name: Vijaya
Roll Number:24
```

Q. 2(c) Write a C++ program to declare a class 'circle' with data members as radius and area. Declare a function getdata to accept radius and putdata to calculate and display area of circle.
(Program 5.2.1) **(04 Marks)**

Q. 2(d) With suitable example, describe effect of ++ and – – operators used with pointer in pointer arithmetic.
(Section 7.2.3) **(04 Marks)**

Q. 3(a) Write a C++ program to declare a class addition with data members as x and y. Initialize value of x and y with constructor. Calculate addition and display it using function 'display.' **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class Addition
{
    int x,y,sum;
    public:
    Addition()
    {
```

```
        cout<<"Enter two numbers:";
        cin>>x>>y;
    }
    void calculate()
    {
        sum=x+y;
    }
    void display()
    {
        cout<<"Sum:"<<sum;
    }
};
void main()
{
    clrscr();
    Addition s;
    s.calculate();
    s.display();
    getch();
}
```

Output :

```
Enter two numbers: 5
9
Sum: 14
```

Q. 3(b) With suitable diagram describe structure of C++ program. **(Section 1.10)** **(04 Marks)**

Q. 3(c) Describe the concept of virtual base class with suitable example. **(Section 7.5.3)** **(04 Marks)**

Q. 3(d) Describe use of static data member in C++ with example. **(Section 5.6)** **(04 Marks)**

Q. 4(a) Write a C++ program to implement inheritance shown in Fig. 1-Q. 4(a) :

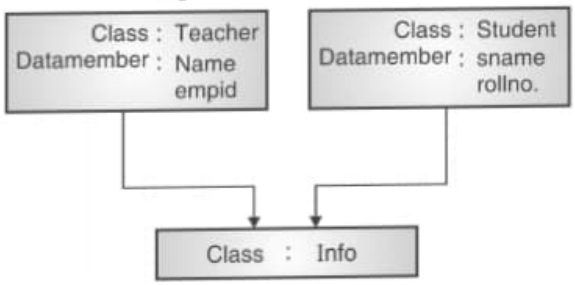


Fig. 1-Q. 4(a)

Accept and display data of one teacher and one student using object of class 'Info'. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class Teacher
{
    protected:
        char name[20];
        int empid;
};
class Student
{
    protected:
    char name[20];
    int roll_no;
};
class Info : public Teacher, public Student
{
    public:
    void accept()
    {
        cout<<"Enter teacher name, employee ID,
        student name and roll number:";
        cin>>name>>empid>>sname>>roll_no;
    }
    void display()
    {
        cout<<"Teacher name:"<<name<<endl<<
        "Employee ID:"<<empid<<endl<<"Student
        name:"<<sname<<endl<<"Roll Number:"
        <<roll_no;
    }
};
void main()
{
    info s;
    clrscr();
    s.accept();
    s.display();
    getch();
}
```

Output :

```
Enter teacher name, employee ID, student name and roll
number: Ajay
123
Vijaya
24
Teacher name:Ajay
Employee ID:123
Student name: Vijaya
Roll Number:24
```

Q. 4(b) Write a C++ program to print multiplication table of 7. (example: 7 × 1 = 7 ..... 7 × 10 = 70) **(Section 2.2.2)** **(04 Marks)**

Q. 4(c) Write a C++ program to swap two integer numbers and swap two float numbers using function overloading. (Hint : overload swap function) **(Section 7.3.3)** **(04 Marks)**

Q. 4(d) Write a C++ program to count number of spaces present in contents of file. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char temp[20];
    int count=0;
    ifstream in;
    in.open("test.txt",ios::out);
    while(!in.eof())
    {
        in>>name;
        count++;
    }
    in.close();
    cout<<"The file contains "<<count<<" words";
    getch();
}
```

Output :

```
The file contains 5 words
```

Q. 4(e) Write a C++ program to find greatest number among two numbers from two different classes using friend function. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class C1
{
    int x;
    public:
    void read()
    {
        cout<<"Enter a number:";
        cin>>x;
    }
    friend int Greatest (C1 c1, C2 c2);
};
class C2
{
    int x;
    public:
    void read()
    {
        cout<<"Enter a number:";
        cin>>x;
    }
    friend int Greatest (C1 c1, C2 c2);
};
int Greatest (C1 c1, C2 c2)
{
    if(c1.x > c2.x) return c1.x;
    else return c2.x;
}
void main()
{
    clrscr();
    C1 c1;
    C2 c2;
    c1.read();
    c2.read();
    cout<<"Greatest number is:"<<Greatest(c1,c2);
    getch();
}
```

Output :

```
Enter a number:2
Enter a number:4
Greatest number is:4
```

Q. 5(a) Write a C++ program to overload binary operator '+' to concatenate two strings.

(Program 7.4.12) **(06 Marks)**

Q. 5(b) Write a C++ program to write 'Welcome to poly' in a file. Then read the data from file and display it on screen. **(06 Marks)**

Ans. :

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    charmsg[20];
    ofstream o;
        o.open("test0",ios::in);
        o<<"Welcome to poly"<<endl;
    o.close();
    ifstream i;
        i.open("test0",ios::out);
    i>>msg;
        cout<<msg<<endl;
    i>>msg;
        cout<<msg<<endl;
    getch();
}
```

Output :

```
Welcome to poly
```

Q. 5(c) Write a C++ program to declare a class 'Account' with data members as acc no., name and bal. Accept data for eight accounts and display details of accounts having balance less than 10,000.

(06 Marks)

Ans. :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Account
{   private:
        int acc_no,balance;
        char name[20];
    public:
    void get_data()
    {
    cout<<"Enter name, account number and balance:";
    cin>>name>>acc_no>>balance;
    }
    void display()
    {
    if(balance<10000)
    cout<<name<<"\t"<<acc_no<<"\t"<<balance<<endl;
    }
};
void main()
{   Account b[8];
    int i;
    clrscr();
    for(i=0;i<=7;i++)
    {
        b[i].get_data();
    }
    cout<<"Name\tAcc No\tBalance\n";
    for(i=0;i<=4;i++)
    {
        b[i].display();
    }
    getch();
}
```

Output :

```
Enter name, account number and balance:Jay 1 50000
Enter name, account number and balance:Ajay 2 1000
Enter name, account number and balance:Vijay 3 20000
Enter name, account number and balance:Sujay 4 30000
Enter name, account number and balance:Harsh 5 400
Enter name, account number and balance:Meet 6 70000
Enter name, account number and balance:Sita 7 6000
```

```
Enter name, account number and balance:Rita 8 90500
Enter name, account number and balance:Mita 9 15000
Enter name, account number and balance:Gita 10 10000
Name     Acc No     Balance
Ajay     2          1000
Harsh    5          4
Sita     7          6000
```

Q. 6(a)(i) Write a C++ program to find whether the entered number is even or odd. **(03 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
void main()
{   int x;
    clrscr();
    cout<<"Enter a number:";
    cin>>x;
    if(x%2 = = 0)
        cout<<"Number is even";
    else
        cout<<"Number is odd";
    getch();
}
```

Output :

```
Enter a number:4
Number is even
```

Q. 6(a)(ii) Write a C++ program to declare a structure employee with members as empid and empname. Accept and display data for one employee using structure variable. **(03 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
struct Employee
{
    char name[20];
    int empid;
};
void main ()
{
    clrscr();
    struct Employee s1;
    cout<<"Enter the employee's name and ID:";
    gets(s1.name);
    cin>>s1.empid;
    cout<<"The employee details are as follows:"
    <<endl << "Name:"<<s1.name<< endl
    <<"Employee ID:"<<s1.empid;
    getch();
}
```

Output :

```
Enter the employee's name and ID:John Patrick
24
The employee details are as follows:
Name:John Patrick
EmployeeID :24
```

Q. 6(b) Write a C++ program to implement following inheritance.

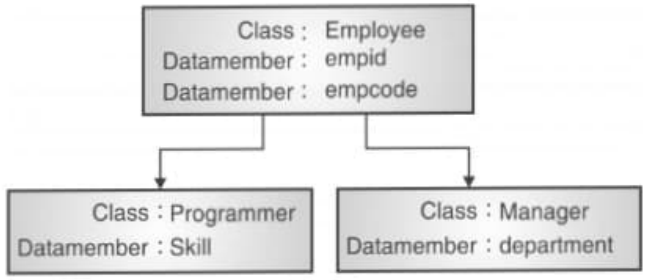


Fig. 1-Q. 6 (b)

Accept and display data for one programmer and one manager. Make display function virtual. **(06 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Employee
{
    protected:
        int empid,code;
    virtual void display()
    { }
};
class Programmer:public Employee
{   private:
```

```
        char skill[20];
    public:
    void get_data()
    {
        cout<<"Enter skill, employee ID and code:";
        cin>>skill>>empid>>code;
    }
    void display()
    {
        cout<<"Employee ID:"<<empid<<"\nCode:"
        <<code<<"\nSkill:"<<skill;
    }
};
class Manager:public Employee
{   private:
        char department[20];
    public:
    void get_data()
    {
     cout<<"Enter department, employee ID and code:";
    cin>>department>>empid>>code;
    }
    void display()
    {
        cout<<"Employee ID:"<<empid<<"\nCode:"
        <<code<<"\nDepartment:"<<department;
    }
};
void main()
{   clrscr();
    Programmer b;
        b.get_data();
        b.display();
    Manager m;
        m.get_data();
        m.display();
    getch();
}
```

Output :

```
Enter name, number height and weight:Ajay
24
175
67
Name:Ajay
Number:24
Height:175
Weight:67
```

Q. 6(c) Write C++ program for following multilevel inheritance.

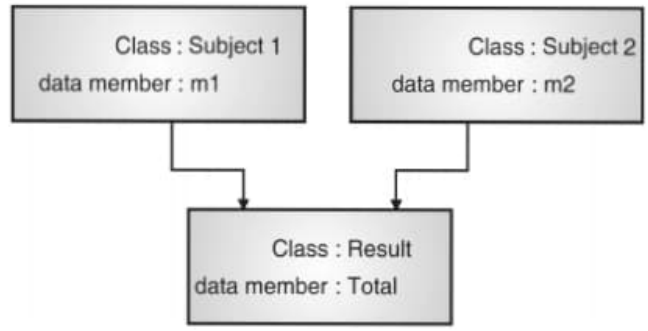


Fig. 1-Q. 6 (c)

Accept and display data for one car with all details.

(06 Marks)

Ans. :

```
# include<iostream.h>
# include<conio.h>
class CarManufacturer
{
    protected:
        char name[20];
};
class CarModel: public CarManufacturer
{
    protected:
        int model_no;
        char model_name[20]
    public:
};
class Car: public CarModel
{
```

```
    private:
    char color[20];
    int no;
    public:
    void read()
    {
        cout<<"Enter the car manufacturer name, model
        name, model number, color and number:";
        cin>>name>>model_name>>model_no>>
        color>>no;
    }
    void display()
    {
    cout<<"Car Manufacturer:"<<name<<endl
    <<"Model Name:"<<model_name<<endl
    << Model number:" <<model_no << endl << "Color:"
    <<color<<endl<<"Number:"<<no;
    }
};
void main()
{
    clrscr();
    Car a;
    a.read();
    a.display();
    getch();
}
```

Output :

```
Enter the car manufacturer name, model name, model
number, color and number:Maruti
Eritga
2
Blue
2476
Car Manufacturer:Maruti
Model Name:Ertiga
Model number:2
Color:Blue
Number:2476
```

Summer 2019

Q. 1(a) State the use of cin and cout.
(Section 1.16) **(02 Marks)**

Q. 1(b) Describe derived class with example.
(Section 6.1.2) **(02 Marks)**

Q. 1(c) State use of scope resolution operator.
(Section 1.12.1) **(02 Marks)**

Q. 1(d) Define class and object.
(Section 5.1.1) **(02 Marks)**

Q. 1(e) Write the use of ios::in and ios::out.
(Section 8.2) **(02 Marks)**

Q. 1(f) Describe use of static data member.
(Section 5.6) **(02 Marks)**

Q. 1(g) Give meaning of following statements :
(02 Marks)

```
int *ptr, a=5;
ptr = &a;
cout<< *ptr;
cout<< (*ptr) + 1;
```

Ans. :

```
int *ptr, a=5;
    /* An integer pointer 'ptr' is declared as well as an integer
    variable 'a', initialized to 5 */
ptr = &a;
    /*The pointer 'ptr' is initialized to point to the
    variable 'a' */
cout<<*ptr;
    /*The value of the variable pointed by pointer 'ptr'
    i.e. value of variable 'a' is displayed. Thus value displayed
    will be 5*/
cout<<(*ptr)+1;
    /*The value of the variable pointed by pointer 'ptr'
     i.e. value of variable 'a' +1 is displayed. Thus the value
    displayed will be 6*/
```

Q. 2(a) Write a 'C++' program to find factorial of given number using loop.
(Section 2.2.1) **(04 Marks)**

Q. 2(b) Write a 'C++' program to declare a class COLLEGE with members as college code. Derive a new class as STUDENT with members as stupid. Accept and display details of student along with college for one object of student. **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class College
{   protected:
    char name[20];
    int code;
};
class Student
{
    char sname[20];
    int studid;
    public:
    void accept()
    {
        cout<<"Enter college name, code, student name
        and student ID:";
        cin>>name>>code>>sname>>studid;
    }
    void display()
    {
        cout<<"College name:"<<name<<endl
        <<"College code:"<<code<<endl<<"Student
        name:"<<sname<<endl<<"Student ID:"<<studid;
    }
};
void main()
{   Student s;
    clrscr();
    s.accept();
    s.display();
    getch();
}
```

Output :

```
Enter college name, code, student name and student ID:
Xaviers
123
Vijaya
24
College name:Xaviers
College code:123
Student name: Vijaya
Student ID:24
```

Q. 2(c) Write a C++ program to find smallest number from two numbers using friend function
(Hint : use two classes).
(Section 3.6.3) **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class B;
class A
{   int x;
    public:
    void accept()
    {
        cout<<"Enter a number:";
        cin>>x;
    }
    void display()
    {
        cout<<x;
    }
    friend void small(A a, B b);
};
class B
{   int x;
    public:
    void accept()
    {
        cout<<"Enter a number:";
        cin>>x;
    }
```

```
    void display()
    {   cout<<x;
    }
    friend void small(A a, B b);
};
void small (A a, B b)
{
    if(a.x<b.x)
        a.display();
    else
        b.display();
}
void main()
{
    clrscr();
    A a;
    B b;
    a.accept();
    b.accept();
    cout<<"Smaller number is:";
    small(a,b);
    getch();
}
```

Output :

```
Enter a number:4
Enter a number:6
Smaller number is:4
```

Q. 2(d) Differentiate between run time and compile time polymorphism. **(Section 5.4.1)** **(04 Marks)**

Q. 3(a) Write a C++ program to create a class STUDENT. The data members of STUDENT class.

Roll_No.

Name

Marks **(Section 2.1.5)** **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class Student
{
    int roll_no, marks;
    char name[20];
    public:
    void accept()
    {
        cout<<"Enter name, roll number and marks:";
        cin>>name>>roll_no>>marks;
    }
    void display()
    {
        cout<<"\nName:"<<name<<"\tRoll No.:"
        <<roll_no<<"\tMarks:"<<marks;
    }
};
void main()
{
    clrscr();
    Student s[5];
    int i;
    for(i=0;i<=4;i++)
    {
        s[i].accept();
    }
    for(i=0;i<=4;i++)
    {
        s[i].display();
    }
    getch();
}
```

Output :

```
Enter name, roll number and marks : Jay 12 90
Enter name, roll number and marks : Ajay 24 94
Enter name, roll number and marks : Vijay 48 88
Enter name, roll number and marks : Sujay 96 80
Enter name, roll number and marks : Digvijay 192 84

Name:Jay        Roll No.:12     Marks:90
Name:Ajay       Roll No.:24     Marks:94
Name:Vijay      Roll No.:48     Marks:88
Name:Sujay      Roll No.:96     Marks:80
Name:Digvijay   Roll No.:192    Marks:84
```

Q. 3(b) Accept data for five students and display it. Write a C++ program to display sum of array elements of array size n*. **(Section 2.10)** **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int a[100],i,n,sum=0;
    clrscr();
    cout<<"Enter number of elements:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
        cout<<"Enter a value:";
        cin>>a[i];
    }
    for(i=0;i<=n-1;i++)
    {
        sum+=a[i];
    }
        cout<<"Sum of all elements="<<sum;
    getch();
}
```

Output :

```
Enter number of elements:4
Enter a value:12
Enter a value:10
Enter a value:9
Enter a value:11
Sum of all elements=42
```

Q. 3(c) Describe with examples, passing parameters to base class constructor and derived class constructor by creating object derived class **(Sections 5.5.1 and 5.5.2)** **(04 Marks)**

Q. 3(d) Describe how memory is allocated to objects of class with suitable diagram. **(Sections 6.3 and 6.3.1)** **(04 Marks)**

Q. 4(a) Write a program to implement multiple inheritance as shown in Fig.1-Q.4(a) :

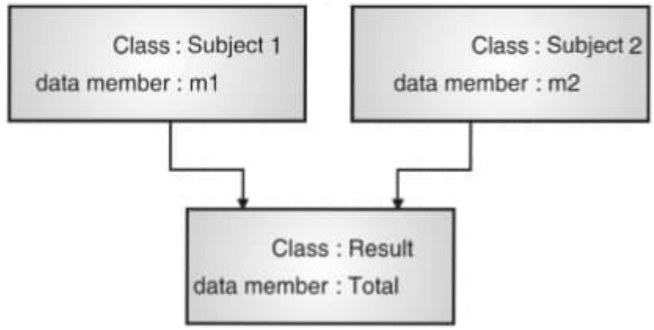


Fig. 1-Q. 4(a)

Accept and display data for one object of class result.

(Sections 2.17, 2.17.1 and 2.17.2) **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<conio.h>
class Subject1
{
    protected:
    int m1;
};
class Subject2
{
    protected:
    int m2;
};
class Result: public Subject1, public Subject2
{
    int total;
    public:
    void accept()
    {
        cout<<"Enter marks in two subjects:";
        cin>>m1>>m2;
    }
    void display()
    {
        total=m1+m2;
        cout<<"Total marks:"<<total;
    }
};
void main()
{
    Result s;
```

```
    clrscr();
    s.accept();
    s.display();
    getch();
}
```

Output :

```
Enter marks in two subjects: 95
90
Total marks:185
```

Q. 4(b) Describe following terms : Inheritance, data abstraction, data encapsulation, dynamic binding. **(Section 3.7.3)** **(04 Marks)**

Q. 4(c) State and describe visibility modes and its effects used in inheritance. **(Section 3.1.1)** **(04 Marks)**

Q. 4(d) Write a C++ program to count number of spaces in text file. **(Section 6.14.4)** **(04 Marks)**

Ans. :

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char temp[20];
    int count=0;
    ifstream in;
    in.open("test.txt",ios::out);
    while(!in.eof())
    {
        in>>name;
        count++;
    }
    in.close();
    cout<<"The file contains "<<count-1<<" spaces";
    getch();
}
```

Output :

```
The file contains 5 spaces
```

Q. 4(e) Differentiate between constructor and destructor. **(Section 6.4)** **(04 Marks)**

Ans. :

Sr. No.	Constructor	Destructor
1.	The name of constructor is same as that of the class.	The name of destructor is also same as that of the class but preceded by tilde (~) sign.
2.	It can accept arguments.	It cannot accept arguments.
3.	There are three types of constructors : default, parameterized and copy constructors.	There is no any types of destructors.
4.	Constructor is automatically called when an object is made.	Destructor is automatically called when the control exits from the function where the object was created.
5.	Constructor overloading is possible.	Destructor overloading is not possible.

Q. 5(a)(i) Write any three rules of operator overloading. **(Section 7.4)** **(03 Marks)**

Q. 5(a)(ii) Write a program in C++ to overload unary '–' operator to negate values of data members of class. **(Program 7.4.1)** **(03 Marks)**

Q. 5(b) Write a C++ program to append data from abc.txt to xyz.txt file. **(Program 8.3.5)** **(06 Marks)**

Q. 5(c) Write a C++ program to declare a class student with members as roll no., name and department. Declare a parameterized constructor with default value for department as 'CO' to initialize members of object. Initialize and display data for two students. **(06 Marks)**

Ans. :

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class CollegeStudent
{
    protected:
        char name[20];
        int id,code;
};
class Test: public virtual CollegeStudent
```

```
{
    protected:
        float percentage;
    public:
    void read()
    {
    cout<<"Enter name, id, college code and
            percentage:";
    gets(name);
    cin>>id>>code>>percentage;
    }
};
class Sports: public virtual CollegeStudent
{
    protected:
    int grade;
    public:
    void accept()
    {
        cout<<"1. Student has won in national sports
        event\n2. Student has not won in any national
        sports event\nEnter your choice:";
        cin>>grade;
    }
};
class Result: public Test, public Sports
{   public:
    void calculate()
    {
        if(grade==1)
        percentage=percentage+10;
    }
    void display()
    {
        cout<<"The percentage is "<<percentage;
    }
};
void main()
{   clrscr();
    Result r;
    r.read();
    r.accept();
    r.calculate();
    r.display();
    getch();
}
```

Output :

```
Enter name, id, college code and percentage: Sandeep
24
81
92
1. Student has won in national sports event
2. Student has not won in any national sports event
Enter your choice:1
The percentage is 97
```

Q. 6(a)(i) Describe structure of C++ program with diagram. **(Section 1.10)** **(03 Marks)**

Q. 6(a)(ii) Write a C++ program to add two 3 x 3 matrices and display addition. **(Section 3.5.2)** **(03 Marks)**

Q. 6(b) Write a program to swap two integers using call by references method. **(Section 7.2.7)** **(06 Marks)**

Q. 6(c) Write a C++ program to implement following inheritance Refer Fig. 1-Q. 6(c). **(Section 6.11)** **(06 Marks)**

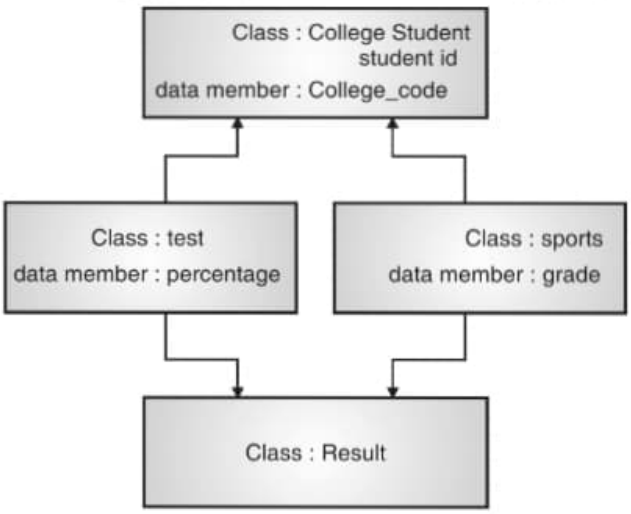


Fig. 1-Q. 6(c)

Accept and display data for one object of class result (Hint : use virtual base class). **(06 Marks)**

Ans. :

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class CollegeStudent
{
```

```
    protected:
        char name[20];
        int id,code;
};
class Test: public virtual CollegeStudent
{
    protected:
        float percentage;
    public:
    void read()
    {
    cout<<"Enter name, id, college code and
            percentage:";
    gets(name);
    cin>>id>>code>>percentage;
    }
};
class Sports: public virtual CollegeStudent
{
    protected:
        int grade;
    public:
    void accept()
    {
        cout<<"1. Student has won in national sports
        event\n2. Student has not won in any national
        sports event\nEnter your choice:";
        cin>>grade;
    }
};
class Result: public Test, public Sports
{
    public:void calculate()
    {
        if(grade==1)
        percentage=percentage+10;
    }
    void display()
    {
        cout<<"The percentage is "<<percentage;
    }
};
void main()
{
    clrscr();
    Result r;
    r.read();
    r.accept();
    r.calculate();
    r.display();
    getch();
}
```

Output :

```
Enter name, id, college code and percentage: Sandeep
24
81
92
1. Student has won in national sports event
2. Student has not won in any national sports event
Enter your choice:1
The percentage is 97
```

❑❑❑

Note

Appendix - B

Solved University Question Paper of Winter 2019

Winter 2019

Q. 1 Attempt any FIVE of the following: (10 Marks)

(a) State the difference between OOP and POP.

(Section 1.6.1) (2 Marks)

(b) What is a class? Give its example.

(Section 5.1.3 and Ex. 5.1.4) (2 Marks)

(c) What is multilevel inheritance? Draw the diagram to show multilevel inheritance using classes with data member and member function.

(Section 6.1.3) (2 Marks)

(d) Explain use of scope resolution operator.

(Section 1.12.1(6)) (2 Marks)

(e) Write two properties of static member function.

(Section 5.6) (2 Marks)

(f) Explain virtual base class with suitable example.

(Section 7.5.3) (2 Marks)

(g) Give syntax and use of fclose ( ) function.

(Section 8.2) (2 Marks)

Q. 2 Attempt any THREE of the following: (12 Marks)

(a) Describe memory allocation for objects.

(Section 5.1.4) (4 Marks)

(b) Write a program to implement single inheritance from the following refer Fig. 1 - Q. 2(b).

(Section 6.1) (4 Marks)

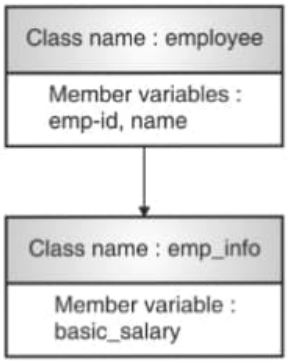


Fig. 1 - Q. 2(b)

Ans. :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Employee
{
        protected:
        int emp_id;
        char name[20];
};
class Emp_info:public Employee
{
        private:
        float basic_salary;
        public:
        void get_data()
        {
                cout<<"Enter employee name, ID and
                  basic salary:";
            cin>>name>>emp_id>>basic_salary;
        }
        void display()
        {
            cout<<"Name:"<<name<<"\nEmployee
                ID:"<<emp_id<<"\nBasic
                Salary:"<<basic_salary;
        }
};
void main()
{
        Emp_info b;
        clrscr();
        b.get_data();
        b.display();
        getch();
}
```

(c) Write any four benefits of OOP.

(Section 5.1.1) (4 Marks)

(d) Describe 'this' pointer with an example.

(Section 7.6) (4 Marks)

Q. 3 Attempt any THREE of the following: (12 Marks)

(a) Write the applications of object oriented programming.

(Section 1.6.2) (4 Marks)

(b) State the rules for writing destructor function.

(Section 5.4) (4 Marks)

(c) What is inheritance? Give different types of inheritance. **(Sections 6.1 and 6.1.1) (4 Marks)**

(d) What are the rules for virtual function ?

(Section 7.5.2) (4 Marks)

Q. 4 Attempt any THREE of the following: (12 Marks)

(a) What is parameterized constructor ?

(Section 5.3.2) (4 Marks)

(b) Write a program to sort an 1-d array in ascending order.

(Program 12 Lab Manual Pg. L6) (4 Marks)

(c) Explain the friend function with proper example. **(Section 5.5) (4 Marks)**

(d) Write a program to count the number of lines in file.

Ans. :

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
        char temp[20];
        int count=0;
        ifstream in;
        in.open("test.txt",ios::out);
        while(getline(in,temp))
        {
                count++;
        }
        in.close();
        cout<<"The file contains "<<count<<" lines";
        getch();
}
```

Output :

```
The file contains 5 lines
```

Q. 5 Attempt any TWO of the following: (12 Marks)

(a) Write a program to declare a class 'student' having data members as 'stud_name' and 'roll_no'. Accept and display this data for 5 students.

Ans. :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Student
{
        private:
        char name[20];
        int roll;
        public:
        void accept()
        {
                cout<<"Enter name and roll number:";
                cin>>name>>post;
        }
        void display()
        {
                cout<<name<<"\t"<<roll<<"\n";
        }
};
void main()
{
        int i;
        clrscr();
        Student s[5];
        for(i=0;i<=4;i++)
        {
                s[i].accept();
        }
        cout<<"Name\tRoll Number\n";
        for(i=0;i<=4;i++)
        {
                s[i].display();
        }
        getch();
}
```

Output :

```
Enter name and roll number:Jay
24
Enter name and roll number:Ajay
48
Enter name and roll number:Vijay
72
Enter name and roll number:Suresh
12
Enter name and roll number:Dinesh
```

```
15
Name            Roll Number
Jay             24
Ajay            48
Vijay  72
Suresh          12
Dinesh          15
```

(b) State and explain the visibility modes used in inheritance. **(Section 6.1.1)** **(4 Marks)**

(c) Write a program to declare a class 'book' containing data members as 'title', 'author-name', 'publication', 'price'. Accept and display the information for one object using pointer to that object.

Ans. :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Book
{
        private:
        char title[20],author_name[20],publication[20];
        int price;
        public:
        void accept()
        {
        cout<<"Enter title, author's name, publication and
                        price of the book:";
        cin>>title>>author_name>>publication>>price;
        }
        void display()
        {
        cout<<title<<"\t"<<author_name<<"\t"
                        <<publication<<"\t"<<price<<e
                        ndl;
        }
};
void main()
{
        Book *c,cc;
        clrscr();
        c=&cc;
        c->accept();
        c->display();
        getch();
}
```

Q. 6 Attempt any TWO of the following: (12 Marks)

(a) Write a program that copies contents of one file into another file. **(Program 8.2.5)** **(4 Marks)**

(b) Write a program to implement the following hierarchy using suitable member functions. Refer Fig. 1 – Q.6 (b). **(Program 6.1.11)** **(4 Marks)**

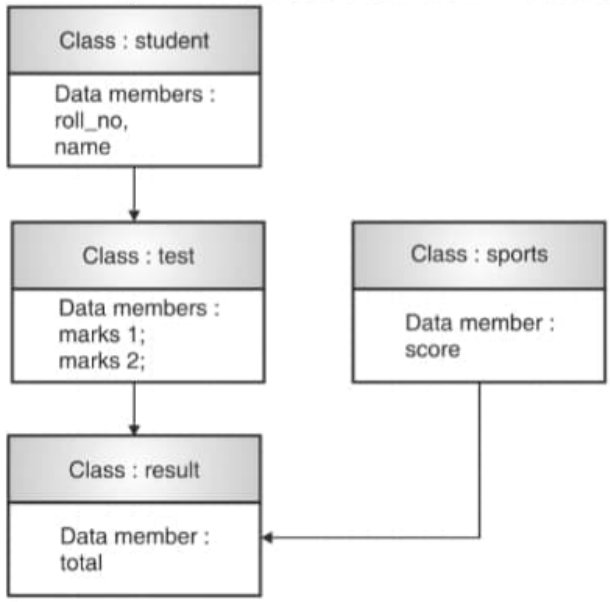


Fig. 1 – Q.6 (b)

(c) Write a program to overload the '—' unary operator to negate the values. **(Program 7.4.2)** **(4 Marks)**

❑❑❑

Note