M.SC. MATHEMATICS

MAL-645

Programming in ‘C’

Directorate of Distance Education
Guru Jambheshwar University of Science & Technology
HISAR-125001

M. Sc. MATHEMATICS

MAL-645

(Programming in ‘C’)

CONTENTS

Author: Dr. Sandeep Dalal
Department of Computer Science
MDU ROHTAK

Course Coordinator: Dr. Vizender Singh
Directorate of Distance
G.J.U.S. &T., Hisar

Vetter: Prof. Dharminder Kumar
Department of Computer Science, G.J.U.S. &T., Hisar

Class	**:**	**M.Sc. (Mathematics)**	**Course Code :**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON 1

OVERVIEW OF PROGRAMMING

STRUCTURE

1.0 OBJECTIVE

After reading this lesson, you should be able to:

a) Understand the Concept programming Languages.

b) Understand the characteristics of good programming language

c) Understand the classification of programming language

1.1 INTRODUCTION TO PROGRAMMING LANGUAGE

Language can be defined as means of communication. Natural languages are used for human interaction like English, French, Hindi etc. whereas programming languages are used for machine and human interaction and is a type of logical communication between them. The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document(for example, C programming language is specified by an ISO standard) while other languages such as Perl etc. have a dominant implementation that is treated as a reference. Some languages have both, with the basic language defined by a standard and extensions taken from the dominant implementation being common. So, a programming language can be defined as a formal language that specifies a set of instructions and is used to control the behavior of a machine (often a computer). To solve a computing problem, its solution must be specified in terms of sequence of computational steps such that they are effectively solved by a human agent or by a digital computer. The earliest programmable machine (that is, a machine that can adjust its capabilities based upon changing its "program") can be said to be the Jacquard Loom, which was developed in 1801. The machine used a series of pasteboard cards with holes punched in them. The hole pattern represented the pattern that the loom had to follow in weaving cloth. The loom could produce entirely different weaves using different sets of cards. This innovation was later refined by Herman Hollerith of IBM in the development of the famous IBM punch card.

Requirements and objectives for a language to be characterized as a programming language are:

Function and Target: A computer programming language is a language that is used to write programs for computer to perform some function or computation or to control some external devices like printer, disk drives, or any other peripheral devices. They are different from natural language as natural languages are used for the interaction between people, whereas, programming languages are used to instruct machines.

Abstraction: Programming languages usually contain abstraction, which is a practical necessity and is expressed by the abstraction principle. Abstraction means to hide the complexity of a program.

Constructs: Programming languages may contain constructs for defining and manipulating data structures or for controlling the flow of execution.

Expressive Power: The theory of computation classifies the languages based on their expressive power, that is the power of computation for a language.

The programming language thus can be defined as a set of grammatical rules and vocabulary to instruct a computer to perform specific tasks. Each programming language has a unique set of keywords, also called as reserve words, which are understand by the computer. Examples of programming languages are BASIC, C, COBOL, Java etc.

1.2 CHARACTERISTICS OF GOOD PROGRAMMING LANGUAGE

A computer program is a set of instructions written in a particular language. The efficiency of the program depends upon the programming language in which it has been written. A program should be developed in a language, which can ensure the proper functionality of the computer with some of the desired features such as correctness, reliability, robustness etc. and is easier to understand the logic underlying the program. A good programming language should have the following characteristics:

Clarity, Simplicity, Unity: A good programming language should be very clear and simple to use so that the user can understand it easily. Unity also hints to understandability. Clear, simple and unified concepts must be provided to the user to grasp the language with ease. It is desired to have the minimum number of different concepts and the rules for their combinations should also be simple. The overall simplicity of the syntax of programming language strongly affects the readability of the programs, which are easier to read and understand, and also easier to test and maintenance later. It is also easy to develop and implement a compiler or an interpreter for a programming language, which is simple. However, the power needed for the language should not be sacrificed for simplicity.

Orthogonality: The term orthogonality refers to the property of combining various features of a programming language in all the possible and meaningful combinations. It means 'changing A would not change B'. For example, if we tune the radio station, volume remains the same and vice-versa. It is easier to learn and program a language if the features are orthogonal as there is less number of exceptions and special cases to be memorized.

Naturalness: A good language should have the attribute of naturalness for the particular application area. It should provide suitable data structures, control structures, operators and a natural syntax to make the coding easy and efficient for the users. The syntax of the language should allow the structure of the program to reflect the logic of the program. The program design should be such that it can be converted into suitable program statements.

Abstraction support: Abstraction means hiding the complexity. It is the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored. It keeps only essential information and depth information for any entity is not required. The degree of abstraction allowed by a programming language directly affects its writ ability. Object oriented language support high degree of abstraction. Hence, writing programs in object-oriented language is much easier.

Program Verification: A good programming language should be easily verifiable with the help of verification method. The reliability of the program is always a matter of big concern. Correctness of the program should be checked by various formal verification methods or by informal desk checking or by executing it with set of input values and checking the output values against these input values. Correctness defines that the program performs its required function.

Programming Environment: The presence of a suitable programming environment in addition to the technical structure of the programming language may make a weak language easier to work then a strong language that has little external support. There are many factors that make strong programming environment such as availability of reliable, efficient and documented implementations of the program, special editors, testing packages etc. They can speed up the overall process of programming and testing.

Portability of Program: A good programming language should be portable. Portable means independent of architecture. A program written for one machine must work on another machine without any modifications in the program.

Cost of Use: Cost is another concerned factor for a good programming language. This includes various factors such as:

- The cost for creation of the program, cost for using and testing of the program should be easy.
- The translation should be efficient in case of using high-level languages.
- The Cost of execution is important where the program needs to execute repeatedly.
- The Cost of maintenance is the largest cost involved in software lifecycle.

Robustness: It is the ability of a programming language to cope with errors and erroneous input during execution. It must have good exception handling techniques. Exceptions are the run time errors.

Reliability: A good programming language should be reliable. The reliability means to behave same for same kind of input. It should give same output under different situations.

Flexibility: Programming language should provide flexibility to the programs. A program should be flexible enough to handle most of the changes without having to rewrite the entire program. Most of the programs are developed for a certain period and they require modifications from time to time.

Efficiency: Programs written in a good programming language are efficiently translated into machine code, are efficiently executed, and acquire as little space in the memory as possible. This means that a good programming language is supported with a good language translator which gives due consideration to space and time efficiency.

Structured: Structured means that the language should have necessary features to allow its users to write their programs based on the concepts of structured programming. This style of programming is less prone to errors while writing the programs.

Compactness: Programmers should be able to express intended operations concisely without writing too many details. Programmers do generally not like a verbose language, because they need to write too much.

Check Your Progress A

Fill in the blanks:

1. Language can be defined as means of ……………...
2. The earliest programmable machine can be said to be the Jacquard Loom, which was developed in ……………….
3. The Term ………… refers to the property of combining various features of a programming language in all the possible and meaningful combinations.
4. C was originally developed by…………………..
5. A …………….is a set of instructions written in a particular language.

1.3 CLASSIFICATION OF PROGRAMMING LANGUAGES

Programming languages fall into two fundamental categories – low level languages and high level languages. Through the first four decades of computing, programming languages evolved in generations. The first two generations were low level programming languages and then the next three are composed of high-level languages. Low-level computer languages are either machine codes or are very close to them. They are machine-dependent, whereas high-level languages are machine-independent and can run on a variety of computers. A computer cannot understand the instructions given to it in high-level languages such as in English. It can only understand and execute instructions given in the form of machine language i.e. binary notations. As the involvement of computer, automation and robotics is growing in our daily lives, programming languages have seen a continuous evolution from low level to high level of programming languages. In the present scenario, programming languages have become a wide area of engineering and research. High-level language provides a sophisticated interaction between the programmer and computer. Higher the level of a programming language, the easier it is to understand and use.

Low Level Language

Low-level languages are machine dependent and are written for the specific architecture and hardware of a particular type of computer. They are close to the notions understood by the computer i.e. the form of electric pulses. They are of two types: Machine Code and Assembly language.

Machine Level Language

Machine level language is the programming language of first generation. It is the lowest and the most elementary level of programming. Computers are operated by following the machine language programs, the only language a computer can understand. It is written as long sequences of binary digits (bits) 0 and 1. All the instructions feed to the computer system must be in the form of binary 0 or 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language. In the 1940s and 1950s, computers were programmed by scientists sitting before control panels equipped with toggle switches so that they could input instructions as strings of zeros and ones. Machine level language is very tough to understand and learn by the humans. They relate to the specific architecture and hardware of a particular type of computer.

Advantages of Machine Level Language

- The computer directly understands the machine language. So, there is no requirement for translators such as compilers or interpreters.
- It makes fast and efficient use of the computer as it takes very less time to execute.
- Disadvantages of Machine Level Language
- Machine language is not a portable language, it is a machine dependent language, and so individual programs are written for each machine.
- It is time consuming to develop new programs.

- All operations codes have to be remembered.
- All memory addresses have to be remembered.
- It is hard to find or correct the errors in a program written in the machine language, so process of debugging is very difficult.

Assembly Language

Assembly language is another low-level language, which was developed to overcome some of the difficulties of machine language. It is the language of second generation. By the late 1950s, this language had become popular. In this language, operation codes and operands are written in the form of alphanumeric symbols instead of 0's and l's. These alphanumeric symbols are called as mnemonic codes. They can be combined in a maximum of five-letter combination e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as 'Symbolic Programming Language.'

This language is also very difficult and needs a lot of practice to master it because there is only a little English support in this language. A language translator called as **assembler** converts the instructions of the assembly language into machine codes and then the computer executes them.

For example:

Machine language: 10110000 01100001

Assembly language: mov a1, #061h

Meaning: Move the hexadecimal value 61 (97 decimal) into the processor register named "a1".

Advantages of Assembly Language

- It is easier to understand and use as compared to machine language.
- It is easily modified.
- It takes less time to develop a program in assembly language.
- It is easy to locate and correct errors.
- It is a portable language.
- It works directly on memory locations.

Disadvantages of Assembly Language

- Like machine language, it is also machine dependent/specific.
- The programmer also needs to understand the hardware, as it is a machine dependent language.
- It is difficult to understand the commands at times.

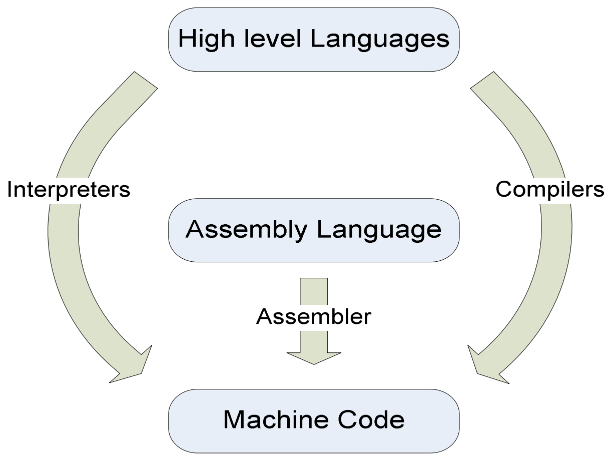


Figure 1.1: Levels of Programming Languages

High Level Language

High-level computer programming languages started to evolve in 1950s after the introduction of compiler. The languages under this category are very close to humans. These languages enable a programmer to create program files using commands that are similar to spoken English (for example: print, if, while), and have become the major means of communication between the digital computer and its user.

Programs written in a high-level language need to be translated into machine language before they can be executed. Tools like complier and interpreter are used for such translation. A **compiler** reads the whole source code and translates it into executable machine code or object code. **Interpreter** is a program that executes instructions written in a high-level language. It reads the source code one instruction or line at a time, converts this line into machine code and executes it. Thus it takes more time to execute the program. It may be possible to execute the same source code either directly by an interpreter or by compiling it and then executing the machine code produced.

Examples: C, C++, Java, FORTRAN, Visual Basic, and Delphi.

Advantages of High Level Language

- High-level languages are programmer-friendly. Instructions or commands are very easy to remember by programmer.
- The structure and logic of the program is easy to understand.
- It takes less time to write a program in high-level language.
- Debugging is easy.

- High-level languages are portable, so no particular knowledge of the hardware is needed.

Disadvantages of High Level Language

- High-level programming language takes more space as compared to machine level or assembly level language. So, they are less efficient.
- Program is first translated into machine code by compiler or interpreter and then it is executed. So, execution time of the whole program is more.

Various generation of programming languages have been summarized in the table below:

Table 1.1: Various Generations of Programming Languages

Generation	**Level**	**Features**	**Example**
First	Low-Level	Machine Language, where sets of commands are represented as a series of 1s and 0s.	Machine Language
Second	Low-Level	Assembly language, where commands are human readable and in the form of alphanumeric symbols	Assembly Language
Third	High-Level	Major improvements than low-level languages and are more programmer-friendly. They are mostly used in business	C, C++, Java, FORTRAN, COBOL, Visual Basic, Ada,

		and scientific applications.	Pascal, Basic
Fourth	High-Level	Languages that consist of statements similar to statements in a human language and are commonly used in database programming and scripts	Perl, PHP, Python, Ruby, SQL, SAS, SPSS
Fifth	High-Level	Artificial Intelligence Languages, programming languages that contain visual tools to help develop a program.	Prolog, Mercury

1.4 COMPUTER PROGRAMMING LANGUAGES

Computer programming is defined as telling a computer what to do through a special sequence of instructions, which are then interpreted by the computer to perform some task(s). These instructions can be specified in one or more programming languages including C, C++, C#, Java, .NET and Visual Basic etc. The task of developing the programs is called programming and the person engaged in programming activity is called programmer. Like natural languages programming languages conform to the rules for syntax and semantics. Syntax is grammar of any programming language. It indicates the structure of the program code. Semantics indicates the logic and meaning of the code. There is a separate set of instructions and grammar for a particular language. Some Examples of computer programming languages are describe below:

C Programming Language: C is a general purpose, middle level computer language, developed by Dennis Ritchie in 1969 at Bell-Labs. It has a rich set of operators, keywords and basic data types. Speed of execution is very fast as it is only complied and not interpreted. That's why, many embedded systems have the processor which is written in C. It is also used to build various online games. Unix operating system has been written in C.

C++ Programming Language: C++ is object oriented programming language. It is a middle-level programming language developed by Bjarne Stroustrup in 1979. It is an extension to C language with additional features from Simula language. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language. It supports the four pillars of object-oriented development – Encapsulation, Data Hiding, Inheritance and Polymorphism. C++ models the real world programs with the help of classes and objects. It is being highly used to write device drivers and other software that rely on direct manipulation of hardware under real time constraints.

C# Programming Language: C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO). It is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

JAVA Programming Language: Java was developed by Sun Microsystems in 1995. Java SE 10 is the latest version of Java. It is general purpose computer language that is concurrent, class based and object-oriented. It is compiled and interpreted, both. It is a portable language with the feature Write Once, Run Anywhere. This feature comes with the fact that it is complied only once and the source code get converted into an intermediate Java byte code. Java JVM is used to interpreted these byte codes into machine language. These universal bytes codes makes Java portable. Java is robust and secure. It is multithreaded and dynamic language.

VISUAL BASIC Programming Language: Visual Basic or VB is an event-driven programming language. It is based on BASIC language. All the versions of VB from 1.0 to 6.0 have been declared as legacy and are no longer supported. Micorsoft has developed various derivatives of VB to use in scripting for example, VBA, VBScript, VB.NET, LotusScript etc.

PHP Programming Language: PHP is an abbreviation for Hypertext Preprocessor. It is a server side scripting language and is designed for web development. It may be embedded into HTML code. The code may also be executed on command-line interface. The PHP language can be deployed on most of the web servers and is supported by a large number of operating systems and platforms. It also supports many databases such as MySQL, Sybase, Oracle etc. PHP is free and an open source software project.

PYTHON Programming Language: Python is interpreted, object-oriented and dynamic language. It was created by Guido Can Rossum. The source code of Python is available under GNU General Public License. It supports functional and structured programming methods and can be used as a scripting language. It can be compiled to byte-code for building large applications. It can be easily integrated with Java, CORBA, C, C++, COM etc. Automatic garbage collection is supported by Python. It also provides interface to all the major databases. Python is easy to read, learn and maintain. It has a broad library, which provides various inbuilt modules. The library is portable and cross-platform compatible.

SHELL SCRIPTING Programming Language: Shell Script is basically a computer program that is designed to be run by the Unix shell. Various languages of shell script are called as shell scripting languages.

Check Your Progress B

State whether the following statements are True or False:

1. Languages that consist of statements similar to statements in a human language and are commonly not used in database programming and scripts
2. Machine Language, where sets of commands are represented as a series of 1s and 0s.
3. Artificial Intelligence Languages, programming languages that contain visual tools to help develop a program.

1.4 SUMMARY

A Programming Language is a formal language that specifies a set of instructions that can be used to produce various kinds of output. Programming languages generally consist of instructions for a computer. Programming languages can be used to create programs that implement specific algorithms. There are programmable machines that use a limited set of specific instructions, rather than the general programming languages of modern computers. The efficiency of the program depends upon the programming language in which it has been written. A good programming language should have some characteristics like Clear, simple and unified ,Orthogonality, Naturalness, Abstraction support, Robustness, Reliability, Flexibility and Compactness etc. Programming languages fall into two fundamental categories – low level languages and high level languages. Through the first four decades of computing, programming languages evolved in generations. The first two generations were low level programming languages and then the next three are composed of high-level languages. Low-level computer languages are either machine codes or are very close to them. They are machine-dependent, whereas high-level languages are machine-independent and can run on a variety of computers. High-level language provides a sophisticated interaction between the programmer and computer. Higher the level of a programming language, the easier it is to understand and use.

1.5 KEYWORDS

Communication: The imparting or exchanging of information by speaking, writing, or using some other medium.

Compiler: A compiler reads the whole source code and translates it into executable machine code or object code.

Interpreter: Interpreter is a program that executes instructions written in a high-level language. It reads the source code one instruction or line at a time, converts this line into machine code and executes it.

Low Level Language: The languages which are closely related to instruction set of the computer. They are of two types- Assembly language and Machine language

High Level Language: The programming language, which is close to human languages and need to be translated into machine level language. They are compiled or interpreted.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. Communication
2. 1801
3. Orthogonality
4. Dennis Ritchie
5. Computer Program

Check Your Progress B

1. False
2. True
3. True

1.6 SELF ASSESSMENT QUESTIONS

1. What do you understand by Programming languages? Explain.
2. Explain the Characteristics of Good Programming Language in detail.
3. Explain the Classification of Programming Languages in detail.
4. Differentiate between Low level and high level languages in detail.
5. Briefly explain Compiler and Interpreter.
6. Differentiate between Assembly language and Machine Language.

1.7 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code:**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON – 2

C ESSENTIALS

STRUCTURE

2.0 OBJECTIVE

After reading this lesson, you should be able to:

a) Understand types of Program Development

b) Idea about anatomy of a C program

c) Understand all types of Variables, Constants, Expressions and Assignment Statements of C

d) Idea about Preprocessor and Formatting source files in C

2.1 WHAT IS C

C is a general-purpose, high level programming language developed by Dennis MacAlistair Ritchie in 1972 at Bell Telephone Laboratories. He was an American computer scientist. In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard. C has now become a widely used professional language for various reasons –

- It is a structured language.
- The programs are very efficient in C.
- It can be compiled on a variety of computer platforms
- It is reliable, simple and easy to use.

Major parts of popular operating systems like Windows, UNIX, Linix have been written in C. Various device drivers are also written in C. Mobile devices like Cell phones, palmtops and consumer devices like washing machines and digital cameras have embedded programs which are written in C. Online games are written in C, as the speed of execution for a C program is very high. C language deals with various data types, variables, operators, loops, arrays and pointers.

2.2 PROGRAM DEVELOPMENT

In C, the phases of the program development include following stages:

- Defining a problem
- Analyzing Problem
- Development of algorithm
- Coding & Documentation
- Testing & Debugging
- Maintenance

Defining a problem:

This is the first phase of program development. In this phase, the problem is defined and one should gather the requirements related to solve the problem and analyze the output of the problem.

Analyzing the problem:

This is the second phase of program development. In this phase, the problem is analyzed and required variables and functions etc. to solve the problem. The bounds of the solution will be determined in this phase.

Development of algorithm:

This is the third phase of program development. In this phase, step by step procedure is formed to solve the problem using the requirements framed in the second phase.

Coding & Documentation:

This is the fourth phase of program development. In this phase, the code is implemented step by step to construct the actual program and document it for the future reference.

Testing & Debugging:

This is the sixth phase of program development. In this phase, the code will be tested whether if the problem is getting solved or not also to check whether the code is giving desired output.

Maintenance:

This is the last phase of program development. In this phase, the enhancements to the program are made if the user encounters any problem. The program development life cycle is repeated from the first to last phase and solve the issue where it is found.

Check your progress A

Fill in the blanks:

1. C is developed by……
2. UNIX operating system is written in ………language.
3. Speed of a C program is …….. comparing to other languages.

2.3 ANATOMY OF A C PROGRAM

The C Program consists of

- Preprocessor commands

- Main function
- Type definitions
- Function prototypes
- Variables
- Functions
- Comments

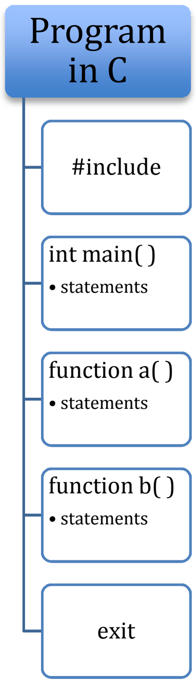


Figure 1 Anatomy of a C program

Each instruction is written as a statement, so, a C program is composed of various statements in series. Although there is no specific rule for the position of the statements, these statements should be written in the same order in which, we want to execute them. To improve the readabilty, blank spaces may be inserted. The syntax of a C program can be seen in the below example-

Example

```
#include<stdio.h>

int main()

/* my first program */

{

printf("Today is Sunday");

return 0;

}
```

Lets see the structure of the C program.

- The first line is a preprocessor command, #include <stdio.h>, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line *int main()* is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been used to readability of the program. They are called as comments statements.
- The next line *printf(...)* is another function available in C which causes the message "*Today is Sunday* " to be displayed on the screen.
- The next line *return 0***;** terminates the main() function and returns the value 0

Lets discuss all these parts in brief-

First of all, preprocessor statements are specified. They are used to include various files in the program and for macro expanxions. Then the main function is defined. Finally comes the sub functions.

Preprocessor

The C preprocessor is a *macro processor* that is used automatically by the C compiler to transform the program before actual compilation. It is called a macro processor because it allows us to define *macros*, which are brief abbreviations for longer constructs. The C preprocessor provides four special functions:

- File Inclusion – Using this directive, a file can be entirely included into another program. This is very helpful in case of large programs, where they can be subdivided into several files and then incorporated into one program.

  Syntax - *#include<stdio.h>*

  #include<conio.h>

- Macro expansion - Macros are defined as the abbreviations for arbitrary fragments of C code, and then the C preprocessor will replace the macros with their definitions throughout the program. They help in program readability and also save the time of manually changes in a value at many places.

  Syntax - *#define UPPER 25*

  This preprocessor directive will replace UPPER in the program with the value 25.

- Conditional compilation. They are special directives to include or exclude some parts of the program based on some conditions.

  Syntax - *#ifdef macroname*

  ```
  #ifdef macroname
          Statement 1;
          Statement 2;
          Statement 3;
  # endif
  ```

Main():

A function is a set of statements. In C, execution of the program begins with main() function. All the statements belonging to main are written inside the curly braces. It always returns an integer value so the type void is definned with it. The integer value that is return is declared as 0.

The curly braces{}:

The { } braces indicates the start and end of the function in the program. Statements are terminated with ‘ ; ‘ .

return 0:

Execution starts when the function returns 0 to the shell.

Comments:

Whenever we want to specify something just for the sake of readability or understanding of the user, comments can be used. Comments are like helping text in the C program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below –

/ my first C program */*

/ function to compute area */*

We cannot have comments within comments, that is comments cannot be nested.

2.4 VARIABLES

Variable is a location in the memory where a program can manipulate the data. The location is used to hold the value of the variable. Variables in C has specific types, which carry the size and layout of the memory of variable, variables range can be stored within the memory. The variable can be composed of letters, digits and characters. As C is case sensitive language, lower and upper-case characters are distinct. Variable should be declared before using and initialization if variable means assigning the value to the variable. A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

Syntax

Variable declaration:

data_type variable_name;

Variable initialization:

data_type variable_name= value;

Example

int x= 120;

int y=140;

Output

```
x=120
y=140
```

Rules for writing Variable in C

- A variable name should be a combination of alphabets, digits and underscore.
- The first character should be an alphabet or underscore.
- Commas or blankspaces are not allowed within a variable name.
- Maximum length of a variable is 31 characters.
- Keywords cannot be used as variables.

2.5 CONSTANTS

The constants are fixed values which cannot be altered during its execution. These values are also known as literals. These can be of any data types such as int, float, char constant.

Syntax:

const type constant name;

In C language, constants are defined in two ways. They are

- Using #define preprocessor directive.
- Using const keyword.

Example

int marks = 25;

2.6 EXPRESSIONS

Expressions are defined as sequence of operators and operands that specifies computation of value. This may contain single entity or multiple entries interconnected by one or more operators.

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int x=456, y;
 y=x+3;
clrscr();
printf("value of x= %d\n", x);
printf ("value of y=%d", y);
getch();
return 0;
```

}

Output

```
Value of x= 456
Value of y = 459
```

Check Your Progress B

Fill in the blanks:

1) Curly braces are denoted as ……..
2) Constants are also known as ………..
3) Preprocessor for a file inclusion is………
4) Preprocessor for a macro defination is ……….

2.7 ASSIGNMENT STATEMENTS

The definition of an entity is defined as the completion of execution of an assignment statement. C uses "==" equivalent relational operator, as the simple statement to avoid confusion with their assignment statement. Compound assignment operator is an easy and shorthand method to specify commonly needed form of assignment. For compound assignment, in the right side, the destination variable appears as the first operand in the expression. C language consists of two unary operators namely "++" and "**--**". These appear as prefix operators.

Syntax

Variable = Expression

Types of assignment statements:

There are four types of assignment statements. They are

- Arithmetic
- Logical
- Statement label
- Character

Check Your Progress C

Give Short Answers

1) What are the types of assignment statements?
2) What are the various types of preprocessors?
3) With which functions, execution in a program is started?

2.8 FORMATTING SOURCE FILES

The source code written in source file is the human readable code of the program. It needs to be "compiled", into machine language so that the CPU can actually execute the program as per the instructions given. The compiler compiles the source codes into final executable programs. The most frequently used and free available compiler is the GNU C/C++ compiler.

The appearance of program in C language changes by inserting or deleting white space by indent program. This is used to make code easier. This also converts one style to another style of the program in C. The substantial amount of C syntax can be understood by indent and it comes to attempt incomplete and mis formed syntax. There are different versions of invoking indent in C language. The format of indent command as per version 1.3 is

indent [options] [input - files]

In this case, we can specify one or more input files. Indent makes backup of every file and the original file is replaced with indented version. The value of the environment variable **VERSION_CONTROL** controls the type of backup file. If the string is **simple**, only simple backup files are made and if the string is **numbered**, then numbered backup files are made. If the value of the string is **numbered-existing**, then the numbered backups will be made and if they are already existing, then numbered backups are intended else a simple backup is made. Indent assumes the

numbered - existing, if the **version_control** is not set. In naming backup files, the indent use the suffix as **.BAK.** This can be emulated by setting " **SIMPLE_BACKUP_SUFFIX** " to

"**.BAK**".

Either **simple** or **numbered** backup files can be made.

Simple:

By adding suffix to the original file name, **simple** backup file names are generated. The default suffix which is used for simple backup files are "~ ".

Example

"java.c" is the original name and the simple backup file name of it will be "java.c~".

You can use specify any string as a suffix by setting environment variable ***SIMPLE_BACKUP_SUFFIX*** to your preferred suffix.

Numbered:

The backup versions of numbered files are represented with version number to the original file name. Here our file name is “src/momewrath.c”, the backup name of the file will be named as “src/momewrath.c.~V~”.

Here V is the version number and the greatest in the existing directory “src”.

Example:

“momewraths” is the original file name and the numbered backup file name will be

“momewraths.c.~23~”.

indent [options] [single - input - file] [-o output - file]

In this form, only one input file is specified. In this case, you may specify an output file after the -o function when the standard input is used.

“slithy_toves.c “is intended to “slithy_toves.out”.

In the other version of indent i.e. 1.2, it also recognizes a long name for each option name. Long options are prefixed by “-- “or “+ “.

Indent uses blank lines. It has number of options to insert or delete blank lines in specific places.

bad:

This causes indent to force a blank line after every block of declarations.

nbad:

This option causes indent not to force such blank lines.

bap:

This forces blank line after every procedure body.

nbap:

This forces no such blank lines.

sob:

This causes indent to swallow optional blank lines.

nsob:

This causes any blank lines present in the input file will be copied to the output file.

Usage of comments

The comments in C begin with ‘/* ‘and ends with ‘*/ ‘. Indent handles comments depending upon their context. Indent attempts to distinguish among comments which follow statements, declarations, Preprocessor directives. Indent leave boxed comments without modifying. This is enclosed in a rectangular shape.

Check Your Progress D

Give short Answers

1) What are the types of backup files which can be created through the indent?
2) How a comment can be written in C?

3) Which is the most used compiler for C?

2.9 THE PREPROCESSOR

This is a program which automatically executes before passing the source program to compiler. This is known as pre-processing. This is not a part of compiler but it's a program invoked by the compiler as the starting part of translation. This is used to run on computer's hardware and application programs and the preprocessor translates the high-level language to middle level language. The commands used in Preprocessor are called as preprocessor directives. These commands begin with '# 'and won't end with ' ; '. The Preprocessor directives can be used anywhere in the program but in C language these are being placed on the starting of the program before defining the function.

Types of Preprocessor Directives

There are different types of Preprocessor directives. They are

- File inclusion directives
- Macro expansion directives
- Conditional compilation directives
- Miscellaneous directives

File Inclusion Directives

The file Inclusion Di helps in giving information to the compiler to include a file in source code program. There are two types of files which can be included.

- Header file

- User defined

Header File

Header file consists of C declarations which are shared between several source files. These files contain definitions of pre-defined functions like printf()& scanf(). Different header files declare different functions.

Syntax:

#include<file_name>

User Defined File

If a program becomes large, then it can be divided into smaller parts and include whenever needed. These are known as user defined files.

#include "file_name"

Macro Expansion Directives

This is a sort of abbreviation which can be defined once and use later. There are many features associated with macros in C Preprocessor. The #define directive is used to define a macro.

Macros with Arguments

The arguments can also be passed to macros. The macros defined with arguments works similar to functions.

Conditional Compilation Directives

These directives help in compiling specific part of program or to skip compilation of some part of the program based on the existing conditions. There are set of commands which are part of conditionals. They are #ifdef, #endif ,

#if, #else, #ifndef . These commands are included or excluded in source program before compilation with respect to condition.

Miscellaneous Directives

There are few directives which are not regularly used. They are

#undef

#pragma

#undef

This is used to undefine a defined macro variable.

#pragma

This is used to call a function before and after main function in C program.

Various directives that can be defined as a preprocessor statements are shown in the table.

Sr. No.	Directive and Description
1	**#define** Substitutes a preprocessor macro.
2	**#include** Inserts another file.
3	**#undef** Undefines a preprocessor macro.
4	**#ifdef** Returns true if this macro is defined.

5	**#ifndef** Returns true if this macro is not defined.
6	**#if** Tests if a compile time condition is true.
7	**#else** The alternative for #if.
8	**#elif** #else and #if in one statement.
9	**#endif** Ends preprocessor conditional.
10	**#error** Prints error message on stderr
11	**#pragma** Issues special commands to the compiler, using a standardized method.

2.10 RULES OF COMMENTS IN C

In C programming, comments provide clarity in source code of the program as it will be useful to other users to understand the program. Comments are important in huge projects which have thousand lines of source code or in projects in which two or more contributors are working on source code. In general, comments start with “/* “and ends with “*/ “. Comments can be used anywhere in program.

Syntax:

/*   comment   */

2.11 SUMMARY

In C, the development of program happens in stages. The anatomy of C consists of comments, functions, preprocessor commands, variables and sub functions etc. C deals with various types of preprocessor directives and formatting of source files. Expressions are evaluated based on their precedence and order of associativity. Rules for commenting in C are followed in a format for clear understanding. The execution of the program starts with main function. It returns an integer type value in C. Input and outpur can be achieved with printf() and scanf() functions.

2.12 KEYWORDS

Preprocessor directives: They are used to direct the compiler to perform some special functions before starting the compilation of the source code.

Macros: Macros are used for abbreviation. They are preprocessing directives.

Compiler:It is used to translate a source file written in a high level language into machine langauge.

Comments: Comments are used to increase the readabilty of a program. They are not executed by the compiler.

Variables: Variable is the name of storage area and can be composed of letters, digits, and the underscore character.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1) Dennies MacAlister Ritchie
2) C
3) High

Check Your Progress B

1) {}
2) Literals
3) #include
4) #define

Check Your Progress C

1) Arithmetic
2) File Inclusion directives, Macro Expansion Directives, Conditional Compilation Directives, Miscellaneous Directives.
3) Main

Check Your Progress D

1) Simple, Numbered
2) /* this is a comment */
3) GNU C/C++ compiler

2.13 SELF-ASSESMENT QUESTIONS:

1) Describe program development life cycle?
2) Describe anatomy of C?
3) Define arithmetic statements?
4) Describe about preprocessor directives?
5) Describe formatting of source files?
6) What are the rules for C Comments?
7) What are the rules for defining a variable in C?

2.14 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code**	**:**
	MAL-645			
Subject	**:**	**Programming in C**		

LESSON 3

DATA TYPES

STRUCTURE

3.0 OBJECTIVE

After reading this lesson, you should be able to:

(a) Understand the basic data types in C

(b) Understand the concepts for integer, floating point and character data types.

(c) Understand the rules for integer constants

(d) Understand various mixing types and explicit conversions

(e) Understand typedefs and void data types

3.1 INTRODUCTION TO DATA TYPES IN C

Data types define how we enter the data in any program for a particular language. C language has some predefined set of data types to handle various kinds of data that we can use in the program. These data types have different storage capacities. Data type defines the type of data a variable can hold. It also determines how to interpreted the stored data.

C language supports different types of data types as classified below:

- **Basic or Primary data types**: They are the primitive or fundamental data types in C. They can be classified further as Integer(int), floating point(float), character(char) and void. The void type specifies that no value is available. It is used in the situation when a function returns no value or it does not accept the parameters.
- **Derived data types**: They are a twisted group of primary data types. For example, array, structure, union and pointer.
- **User Defined data types**: These data types are defined by the user. They can be of two types- Enum and Typedef.

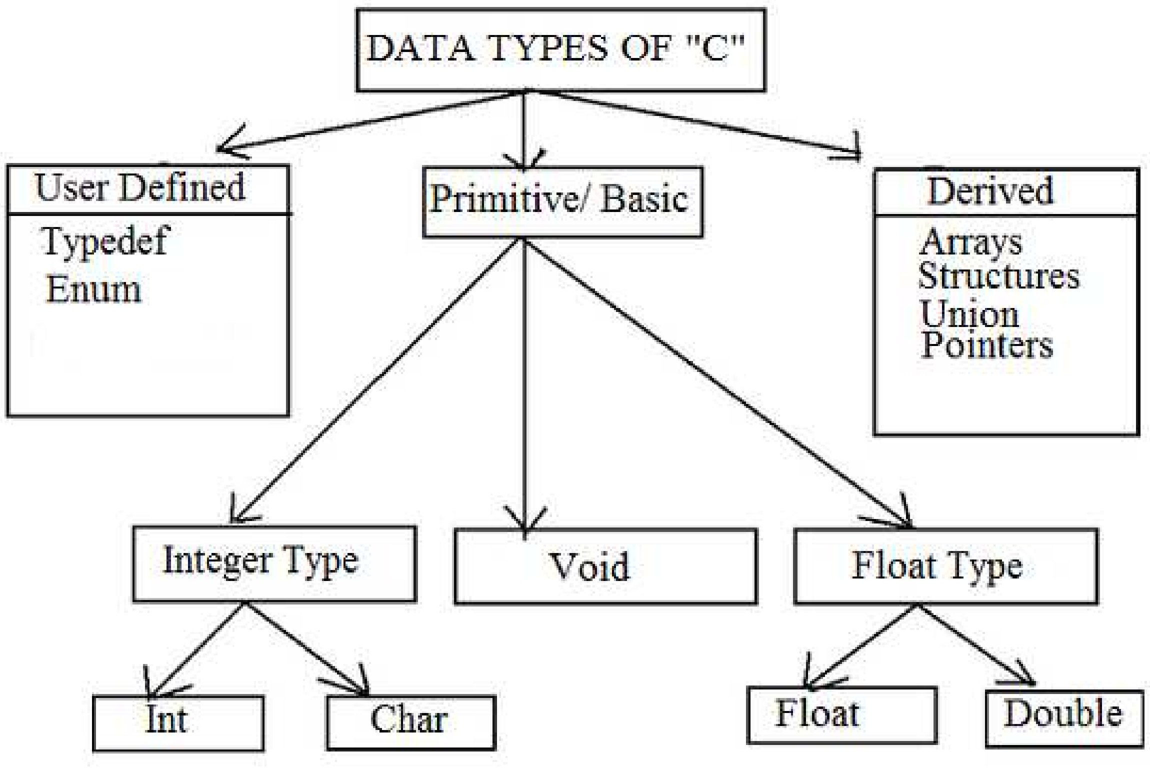


Figure 1 Data Types in C

The data types are briefly described as follows:

Primary Data Types

The primary data types are of four types:

Integer

Keyword int is used for declaring integer variable. The size of int is either 2 bytes or 4 bytes depending upon the type of compiler.

Example:

int id;

int id, age;

Floating Point

In programming, floating points are used to represent fractions.

Example:

float num;

Character

Character data type allows a variable to store only one character. Storage size of character data type is 1 byte. “char” keyword is used to refer character data type.

For Example, ’A’ can be stored using char data type. We can’t store more than one character using char data type.

Example:

char c;

Void

Void is an empty data type that has no value. This can be used in functions and pointers. When a pointer variable is declared with keyword void, it turns into a general-purpose pointer variable. Now the address of any variable of any data type such as char, int, and float can be assigned to a void pointer variable.

Example:

void func(int n)	//It indicates that the function returns no result.

int func(void)	//It indicates that the function has no parameters.

Derived data types

C language provides the facility to construct some data types from primary data types; they are called as secondary data types. Derived data types are nothing but primary data types grouped together like: Array, Structure, Union, Pointer, Enumerator or Enum.

Array

Array is a collection of variables belonging to the same data type. We can store group of data of same type of data in an array. Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values. Array might be belonging to any of the primary data types. Its size must be a constant value. Always, contiguous (adjacent) memory locations are used to store array elements in memory.

Example:

Syntax of array

int a [2];

a [0] =10;

a[1]=20;

a[2]=30;

Structure

It is a collection of variables of different types under a single name. This is done to handle data efficiently. “struct” keyword is used to define structure. Each element in a structure is called member of the structure and to access the members in C, structure variable should be declared.

Example:

Syntax of structure:

struct student

{

int a;

char b[10];

}

Union

Union allows storing of various data types in the same memory location. Programmers can define a union with different members, but only a single member can contain a value at given time. Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called members. Structure allocates storage space for all its members separately, whereas, union allocates one common storage space for all its members. Many union variables can be created in a program and memory will be allocated for each union separately.

Pointers

Pointers in C language is a variable that stores/points the address of another variable. A pointer is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Example:

Pointer syntax

*data_type *variable_name;*

*int *p;*

*char *p;*

Where, * is used to denote that “p” is pointer variable and not a normal variable.

User Defined data types

The user according to the need defines these data types. They can be of the two types.

Enumeration

Enumeration is a user defined data type. It is mainly used to assign names to integral constants that make a program easy to read and maintain. The keyword “enum” is used to declare enumeration types. The name of enumeration is flag and the constant are the values of the flag. By default, the values of the constants are as follows.

Constant 1=0, Constant 2=1, ...

Example:

#include<stdio.h>

```
 enum months{Jan, Feb ,Mar ,Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};
int main()
{
enum=month;
month= Mar;
printf("%d", month);
return 0;
}
```

Output:

```
2
```

Typedef

It is an advance data type in C, which allows to create an alternate name for an existing data type or for a user defined data type. It is useful in the situations when there is a big declaration for a data type.

Syntax

```
typedef data_type new_name;
typedef unsigned long int ULI;
```

The above typedef creates a new name ULI for the unsigned long int data type.

Declarations

A variable can be declared in C by using following syntax:

data_type variable_name;

or

data_type variable1, variable2 ..., variable n;

Where *data_type* is any valid C *data_type* and *variable_name* is any valid identifier.

Example:

int a;

float variable;

float a, b;

Check Your Progress A

Fill in the blanks.

1. Array is a collection of elements of ……….………. data types.
2. Storage capacity of character is ……………byte/bytes.
3. The data type of void is …………….
4. Array is a type of ……………….(primary/derived) data type.
5. Typedef is used to give …………names for other data types.

3.2 INTEGER DATA TYPES AND CONSTANTS

Integer is the set of numbers that include all the whole numbers and their negative values. In programming, integer is a data type used to represent integer values. A variable declared as an integer type occupies a specified amount of memory space in computer.

A variable defined as integer type cannot store any other types of data such as character type or floating type.

Integer Data Types

There are following integer types available in C language:

signed short int

unsigned short int

signed int

unsigned int

signed long int

unsigned long int

signed short int

It is also known as short int or short. The size of signed short int is 2 bytes and ranges from -32768 to 32767. Short integers require less value space so speed up the program to execute faster. C allows declaration of short int to short. In C language declaration short int is specified using "%u" format specifier.

Example:

signed short int a;

or

short int a;

or

short a;

unsigned short int

The size of unsigned short int is 2 bytes and ranges from 0 to 65,535. C allows declaring unsigned short int as unsigned short. This is declared using “%u” format specifier in programs.

Example:

unsigned short int a;

or

unsigned short a;

signed int

Signed integer is known as signed int. It can represent both positive and negative values. Signed integers have undefined behaviour of the overflow. Signed int uses left most bit to identify whether the number is positive or negative. The size of signed integer is of 2 or 4 bytes (depends on the type of compiler) and ranges from -32768 to 32767 or -2,147,483,648 to 2,147,483,647. In c programs “%i” and “%d” format specifiers are used to declare signed int.

Example:

signed int a;

unsigned int

Unsigned integer is known as unsigned int. Unsigned variable of int type can hold positive numbers and zero. It cannot represent negative values & overflow of unsigned integers wrap around. The size of unsigned integer is of 2 or 4 bytes (depends on the type of compiler) and ranges from 0 to 65535 or 0 to 4,294,967,295. “%u” format specifier is used to represent unsigned int in c programs.

Example:

unsigned int a;

signed long int

It is also known as long int or long. The size of signed long int is 4 bytes and ranges from -2,147,483,648 to 2,147,483,647. Long integers help to increase the value of the assigned variable tremendously. But sometimes cause program to run slowly. The size of long int is high enough to accommodate the needs of very large numbers. C allows declaration of long int or signed long int to long. In programs it is declared as “%ld” format specifier.

Example:

signed long int a;

or

long int a;

or

long a;

unsigned long int

The size of unsigned long int is of 4 bytes and ranges from 0 to 4,294,967,295. The overflow of the unsigned long int can be wrap around. C allows declaration of unsigned long int as unsigned long. It is specified as “%lu” format specifier in programs.

Example:

unsigned long int a;

or

unsigned long a;

The following table provides the details of standard integer types with their storage and value ranges-

Table 1 Integer Data Types with their Storage and Range Values

TYPE	**SIZE (Bytes)**	**RANGE**
signed short int	2	-32768 to 32767
unsigned shot int	2	0 to 65535
signed int or int	2 or 4	-32768 to 32767 or -2,147,483,648 to 2,147,483,647

unsigned int	2 or 4	0 to 65535 or 0 to 4,294,967,295
signed long int or lon int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Example for integer data types

```
/*Program to add two numbers*/
#include < stdio.h>
int main()
{
int num_1, num_2, addnums;
printf (“enter two integers:”);
scanf(“%d %d”, &num_1, & num_2);  // two integers are entered by user
addnums = num_1 + num_2; //sum of numbers is stored in addnums variable
printf(“%d + %d = %d”, num_1, num_2, addnums);
return 0;
}
```

Output

```
enter two numbers : 10

20

10 + 20 = 30
```

To get the exact size of a type or a variable on a particular platform, we can use the sizeof operator. The expressions sizeof(type) yields the storage size of the object or type in bytes.

Example:

```
#include<stdio.h>
#include<limits.h>
int main ()

{

printf (“storage size for int: %d\n”, sizeof(int));

return 0;

}
```

On compilation and execution of the above program, it produces the following result:

```
storage size for int: 4
```

Integer Constants

A constant is an entity that does not change. For example, if we declare *int x;*

x=3;

Then, 3 is a constant value whereas x is a variable.

There are certain rules for constructing integer constants.

- An integer constant must have at least one digit.
- An integer constant refers to a sequence of digits without a decimal point.
- Any whole number value is an integer.
- It can be positive or negative.
- An integer preceded by a unary minus may be considered to represent a negative constant.
- No commas or blank spaces are allowed within an integer constant.
- Allowable range for integer constants depends on the type of compiler.

For example, for Visual Studio complier, gcc, the range for integer constant is -2147483648 to +2147483647, whereas for compliers like Turbo C, the range is -32768 to +32767.

Example:	0
	-33
	32767

There are three types of integer constants in C. They can be defined as follows:

Decimal integer constant (base 10)

It consists of any combinations of digits taken from the set 0 through 9, preceded by an optional – or + sign. The first digit must be other than 0. Embedded spaces, commas, and non-digit characters are not permitted between digits.

Example: -

Valid:	0		
	32456		
	-9999		
	-95		
Invalid:	15,768	-	illegal character
	12 23 3	-	illegal character (blank space)

Octal integer constant (base 8)

It consists of any combinations of the digits taken from the set 0 through 7. If a constant contained two or more digits, the first digit must be 0.

Example

Valid:	047		
	0		
	0546		
Invalid:	0786	-	illegal digit 8

564	-	does not begin with zero
02.1	-	illegal character (.)

Hexadecimal integer constant (base 16)

It consists of any combinations of digits taken from the 0 through 7 and also **a** through **f (**either uppercase or lower case). The letters **a** through **f** (or A through F) represent the decimal quantities 10 through 15 respectively. This constant must begin with either 0x or 0X.

Example: -

Valid:	0x		
	0X1		
	0x5F		
Invalid:	0xefh	-	illegal character h
	243	-	does not begin with 0x

Example:

```
/*Integer Constants program*/
#include<stdio.h>
#include<conio.h>
int main()
{
const int a = 5678;
clrscr();
printf("value of a = %d", a);
getch();
return 0;
}
```

Output:

```
value of a = 5678
```

Check Your Progress B

Fill in the blanks.

1. The size of signed short int is. …………bytes and ranges from …………...
2. An integer constant must have at least. ………digit/digits.
3. No …………….are allowed within an integer constant.
4. Decimal integer constant consists of any combinations of digits from…………
5. Hexadecimal integer constant must begin with …………….

3.3 FLOATING POINT DATA TYPES

Floating Point (float) stores inexact representations of real numbers, both integer and non-integer values. Floating-point are numbers that contain decimal points that can float left or right. In programming, floating points is used to represent fractions. It can be used with numbers that are much greater than the greatest possible int. Some numbers like 0, cannot be represented exactly as floats, then it will pop an error. Very large and very small numbers will have less precision and arithmetic operations and sometimes not associative or distributive because of a lack of precision. Floating-point numbers are most commonly used for approximating real numbers; operations are efficient on modern microprocessors. They can be written in two forms- Factorial form and Exponential form. In factorial form, it consists of at least one digit in left and right of the decimal point. In exponential form, it is represented as composed of mantissa and exponent. The mantissa represents the decimal part and the letter E or e represents the exponent for the mantissa.

The following are the valid floating point numbers-

0.134

2E-8

0.1122e4

The table provides the details of standard floating-point types with storage sizes and value ranges and their precisions.

Table 2 Floating Data Types with their Storage and Range Values

TYPE	**SIZE (Bytes)**	**VALUE RANGE**	**PRECISION**
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places

double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	10 bytes	3.4E-4932 to 1.1E+4932	19 decimal places

Floating point data type consists of three types. They are, *float, double*, *long double.*

float

Float data types allow a variable to store decimal values. Storage size of float data type is 4 bytes in memory and can range from -3.4e38 to +3.4e38. The precision can be up-to six digits after decimal using float data types. The specifier used for declaring float is “%f”.

Example:

11.678954 can be stored in a variable using float data type.

double

If range for the float data type is insufficient, C offers a data type called as double which can have 8 bytes of memory. Double data type is also same as float data type which allows up-to ten digits after decimal. Its value ranges from -1.7e4932 to +1.7e4932. The specifier used for declaring double is “%lf”.

Example:

45.7686546785 can be stored in a variable using double datatype.

long double

Storage size of long double is 10 bytes. The range for long double is -1.7e4932 to +1.7e4932. Its precision is up to 19 decimal places and the specifier used is "%Lf".

The example for printing storage space taken by a float type and its range values is shown below:

/*The header file float.h defines macros that allow us to use these values and other details about the binary representation of real numbers in the C programs. */

```
#include <stdio.h>
#include <float.h>
int main ()
{
printf ("storage size for float: %d \n", sizeof(float));
printf ("minimum float positive value: %e\n", flt_min);
printf ("maximum float positive value: %e\n", flt_max);
printf ("precision value: %d\n", flt_dig);
return 0;
}
```

Output:

```
storage size for float: 4
minimum float positive value: 1.175494E-38
```

maximum float positive value: 3.402823E+38

precision value: 6

Check Your Progress C

Give one word for the following:

1. What is the storage capacity of double?
2. What is the precision for long double?
3. Which type of floating point data type has maximum storage capacity?
4. What are the names of two parts for the floating data type?

3.4 CHARACTER DATA TYPE

Character data type allows a variable to store only one character. Storage size of character datat ype is 1 byte. “char” keyword is used to refer character data type.

For Example, ’A’ can be stored using char data type. We can’t store more than one character using char data type.

Example:

char c;

char a;

Character data type has two sub types, both of them has a storage value of 1 byte.

signed char

It has a range from -128 to +127. The most significant bit is consumed in specifying the sign.

unsigned char

It has a range from 0 to 255 as all the eight digits are availble to store the values.

Signed and unsigned char, both occupy 1 byte, but they have different ranges.

Consider char='A';

Here what gets stored in char is the binary equivalent of the ASCII/Unicode value of 'A' (i.e. binary of 65). And if 65's binary can be stored, then -54's binary can also be stored (in a signed char).

The following table provides the details of char data type with size and ranges.

Table 3 Character Data Types with Storage and Range Values

TYPE	SIZE (Bytes)	RANGE
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

Example

```
/*Program to print the value of a character*/
int main ()
{
char ch;
unsigned char uch;
ch=231;
uch=(char)ch;
std::cout<<(int)ch<<std::endl<<(int)uch;
return 0;
}
```

This gives the output:

```
-25
231
```

3.5 MIXED TYPES AND EXPLICIT CONVERSATIONS

Mixed Data Types

Various data types can be specified in a single printf or scanf statement. Combination of one or more data types used in a single printf() requires appropriate format specifiers for the control string with appropriate arguments. Arguments must be aligned with their format specifiers. It can be seen in the example below:

Example:

```
#include<stdio.h>
int main()
{
int dd=05, mm=07, yy=2018;
char title [25] = “Machines”;
int pages = 500;
float price = 450.98;
char currency=’$’;
long int isbn=9543276548769;
printf(“\n\nBooks List : “);
printf (“\n\n| Title\t\t\t | Pages, Date (pub.): ISBN#\t | price”);
printf(“\n----------------------”);
printf(“\n\n| %-25s | %5d |  %2d-%2d-%4d | %-13lli | %c-%-4.2f
|”,title,pages,dd,mm,yy,isbn, currency,price);
printf(“\n\n| %25s | %5d | %2d-%2d-%4d | %-13lli |  %c-%-4.2f |”,”C
Programming”,145,02,07,2018, 9543276548769,$,150.99);
printf (“\n\n”);
 return 0;
}
```

Output

```
Books List :

Title                  Pages, Date (pub.):  ISBN           price

Machines.       500     05-7-2018     9543276548769.   $ 450.98


C Programming     145   02-07-2018  9543276548769    $ 150.99
```

Explicit Type Conversion

This process is also called as Type Casting and it is user defined. The user can type cast the result to make it of a particular data type. Explicit type conversion can also be achieved with separately defined conversion routines such as an overloaded object constructor.

Syntax:

(datatype) expression indicates the data type to which the final result is converted.

Example:

```
#include<stdio.h>

int main ()

{

double X = 1.2;

int (sum) = (into)x+1;
printf("sum=%d", sum);
return 0;
}
```

Output

```
sum = 2
```

Example:

```
float f;
int a=20, b=3;
f=a/b;
```

The value of f will be 6.000000 instead of 6.666666 because operation between two integers yields an integer value. To solve this problem, we shall use mixed mode arithmetic and change the type of either a or b to double or float. Changing the type of variable is not always feasible.

If we write the above statement as

```
f=(float)a/b;
```

then we will get the correct answer as 6.666666.

Working of Cast operator

Firstly, it converts the variable type int to type float temporarily. Here we should notice, in the previous statement data type of variable ‘a’ is float till the execution of the statement only, after that it will be treated as int.

Now we shall see the following statement,

f=(float)(a/b);

You may get doubt that the statement is same as the previous one (i.e. (f=(float)a/b;) but it is not.

Here the expression a/b is evaluated then results are converted to float because of typecasting and eventually assigned to f.

Example:

```
#include<stdio.h>
int main()
{
int a=25, b=13;
float result;
result=a/b;
printf("(without typecasting)25/13=%.2f\n",result);
result=(float)a/b;
printf("(with typecasting)25/13=%.2f\n", result);
return 0;
}
```

Output

```
(Without typecasting) 25/13 = 1.00
(With typecasting) 25/13 = 1.92
```

Initializations

C variables declared can be initialized with the help of assignment operators '='.

Syntax

data_type variable_name= constant/literal/expression

or

Variable_name=constant/literal/expression

Example:

1. int a=10;
2. int a=b+c;
3. a=10;
4. a=b+c;

Multiple variables can be initialized in a single statement by a single value.

Example:

int a, b, c, d, e;

a=b=c=d=e=10;

Constant Variables

C variables have the same or unchanged value during the execution of the program is called Constant Variables. The keyword **const** declares the variable.

Example:

const int a=100;

Volatile Variables

The variables that can be changed at any time by the same program or by external sources are called volatile Variables.

Syntax:

volatile data_type variable_name;

3.6 TYPEDEFS AND VOID DATA TYPES

Typedef

Typedef is a keyword used in C language to assign alternative names to the existing data types. It is mostly used with user defined data types, when names of the data types become slightly complicated to use in programs. To define a term **BYTE** for one-byte numbers, syntax is:

typedef unsigned char **BYTE**;

The identifier Byte can be used as an abbreviation for the type **unsigned char**.

Example

Byte b1, b2;

Upper case letters are used for these definitions to remind the user that the type name is really a symbolic abbreviation; lowercase letters can also be used.

typedef unsigned char **byte**;

typedef can also be used to give a name to your user defined data types as well.

Syntax

typedef <existing_name> <alias_name>

Example:

```
#include<stdio.h>
#include<string.h>
typedef struct Books
{
char title [25];
char author [25];
char subject [50];
int book_id;
} Book;
int main ()
{
Book book;
strcpy (book.title,” C Programming” );
strcpy( book.author, “Ananya”);
strcpy(book.subject, “C Programming Tutorial”);
book.book_id=698502;
printf (“Book Title : %s\n”,book.title);
printf (“Book Author : %s\n”,book.author);
printf(“Book Subject : %s\n”,book.subject);
printf(“Book Book_id : %d\n”,book.book_id);
return 0;
}
```

Output

```
Book Title : C Programming
Book Author : Ananya
Book Subject : C Programming Tutorial
Book Book_id : 698502
```

Void Data Type

Void is an empty data type that has no value. This can be used in functions and pointers. Void data type is used to refer an object or variable that does not have any value of any type. Keyword void is used within function prototype and header to signify that no information is passed to the function.

Example

```
void func(int n)        //It indicates that the function returns no result.
int func(void)          //It indicates that the function has no parameters.
```

Void is used when declaring a pointer variable with a specific data type that cannot hold the other data type variable’s address.

Example:

```
char*p;
int v1;
p=&v1;
```

This is invalid because ‘p’ is a character pointer variable and causes compilation error.

In C to overcome this problem, void is used to declare the pointer. When a pointer variable is declared with keyword void, it turns into a general-purpose pointer variable. Now the address of any variable of any data type such as char, int, and float can be assigned to a void pointer variable.

Example:

void *p;   //Now p is a general-purpose pointer variable

Void data type is used in three different scenarios given below.

Table 4 Different Uses of Void Data Types

Sr. No.	**Types & Descriptions**
1	**Function returns as void:** There are various functions in C, which do not return any value. A function with no return value has the return type as void. Example: *void exit (int status);*
2	**Function arguments as void:** There are various functions in C, which do not accept any parameter. A function with no parameter can accept a void. Example: i*nt and(void);*
	Pointers to void: A pointer of type void * represents the address of an object,

3	but not its type. Example: A memory allocation function *void *malloc(size_t size);* returns a pointer to void which can be casted to any data type.

The use of void for a function returning no value is not necessary. The function can be declared as being type (int) and simply not return any value and never use the function in an expression. The void declaration makes the nature of the function explicit.

Example:

```
#include<stdio.h>
void main ()
{
int a=25;
float b=7.5;
void *ptr;
ptr=&a;
printf (“The value of integer variable is=%d”, *((int*) ptr);
ptr=&b;
printf (“The value of float variable is=%f”, *((float*) ptr);
}
```

Output

```
The value of integer variable is 25

The value of float variable is 7.5
```

3.7 SUMMARY

Data types specify the way of entering the data into our programs. Data types are declarations for variables and memory locations, which determine the features of the data that may be stored. C language has some predefined set of data types to handle various kinds of data, such as integer, real number types, character etc, which have different storage capacities. There are certain secondary or derived data types also such as array, union, structure etc. Several headers in the C standard library contain definitions of support types that have additional properties, such as providing storage with an exact size, independent of the implementation etc. Various other keywords have been defined such as typedef, void etc.

3.8 KEYWORDS

Data type: The type associated with the data in C, which represents the storage capacity.

Keywords: They are predefined and reserved words, which have special meanings. They cannot be used as identifiers.

Sign Qualifiers: A data type can hold positive and negative values. Sign qualifier is used to declare that with the help of signed and unsigned keywords.

Constant qualifier: Const keyword is used to declare an identifier as constant whose value cannot be changed further in a program.

Character set: It is a set of alphabets, digits and special symbols that can be used in a C program

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. Similar
2. One
3. Empty
4. Derived

Alternate/alias

Check Your Progress B

1. 2 , -32768 to 32767.
2. one digit.
3. Commas or blank spaces
4. 0-9
5. 0x or 0X.

Check Your Progress C

1. 8 bytes
2. 19 decimals
3. long double

4. mantissa, exponent

3.9 SELF ASSESSMENT QUESTIONS

1. Describe various types of primary data types.
2. Describe various types of integers.
3. What are various integer constants?
4. What is a void data type? Define its various uses?
5. What are various floating-point data types?
6. What is typedef operator?

3.10 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code**	**:**	**MAL-645**
Subject	**:**	**Programming in C**			

LESSON 4

DATA INPUT AND OUTPUT

STRUCTURE

4.0 OBJECTIVES

After reading this chapter, you should be able to:

a) Understand the input and output functions in C

b) Undersatnd the difference between formatted and non-formatted input-output functions

4.1 INPUT AND OUTPUT FUNCTIONS

Input means to provide the program with some data to be used in the program. Output means to display data on screen and write the data to a printer or a file. C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result. All these built in functions are present in C- header files. There are various library functions available for I/O. They can be classified into two categories-

Console I/O Functions - These functions enable the transfer of data between the C programs and standard input output devices. They receive input from Keyboard and write them on visual display unit.

File I/O Functions- These functions perform Input/Output operations on a floppy disk or a hard disk.

Here, in this chapter, we would be discussing only Console I/O functions.

4.2 CONSOLE INPUT- OUTPUT FUNCTIONS

C always treats all input output data, regardless of where they originate and where they go, as a stream of characters. The operating system makes the input and output devices available to C program if these devices were files. when a C program sends output data to the console, it is in effect reading from the file associated with the keyboard device. When a C program sends output data to the console. It is in effect writing to the file associated with the console device. A stream of characters is a sequence of characters divided into lines. Each line consists of various characters followed by a newline character (\n). Keyboard and screen together are called as console. Console input/output functions are of two types- formatted input output functions and non-formatted input output functions.

The standard input and output devices are referred as :

- stdin
- stdout
- stderr

stdin:

Standard input file, normally connected to keyboard

stdout:

Standard output file, normally connected to the screen/console

stderr:

Standard error display device file normally connected to the screen/console.

Categories of Input/Output functions:

The input - output functions fall into two categories. They are

- Formatted Input/Output Functions
- Non-formatted Input/Output Functions

4.3 FORMATTED INPUT-OUTPUT FUNCTIONS

The formatted input - output functions consists of printf () and scanf (). The header file stdio.h is included in a C program to use these functions.

printf ()

The printf () is used to display output data onto the standard output device. In general, the printf function is written as

Printf (<control string>, arg2,......argn);

Control string refers to the string containing required formatting information as in scanf and arg1, arg2...argn are individual data variables whose values are to be printed. Unlike scanf, these data variable names are not preceded by the & symbol. This is because printf is expected only to output the values of these variables and not addresses.

A full format of the format specifier in printf would be like -

%[-][width][flags][format]

[ ] - implies specifiers are optional

% - denotes the beginning of a format specifier.

"-" : indicates that data to be displayed as left justified, if the "-" missing, the data display is right justified.

Width - the width of field or number of spaces to use for displaying

Flags - precision of output to be displayed

Format - the format specifier itself.

Example

```
#include<stdio.h>

main ()

{
```

```
int n;
n=27;
printf (“the value of n=%d”, \n\n);
}
```

Output

```
The value of n = 27
```

scanf ()

The scanf function is used to read formatted input data. The format in which input data is to be provided is specified by the scanf function itself as its first parameter.

Syntax

scanf (<control string>, &address1, &address2,.&addressn);

Control string contains list of format specifiers indicating the format and type of data. The remaining parameters are addresses of the variables where the read data will be stored.

Scanf reads the input data as per the format specifiers and stores them to the corresponding addresses. An & is prefixed to the variable name to denote its address.

scanf(“%d%f”,&x,&y);

The first argument is a string that contains two format specifiers -%d and %f. The first format specifier (%d) is for argument &x and the second format specifier (%f) is for argument &y. The two pieces of data will be read, the first piece is treated according to %d and the second piece is treated according to %f. Format specifiers are always specified for the remaining arguments in order from left to right.

Scanf treats all whitespace characters as delimiters to separate one data input from the other. While entering your data inputs, you can separate them out with a blank space or a tab space character or a newline character.

Table 1 Formatted Functions

Type	Input	Output
char	scanf( )	printf()
int	scanf( )	printf()
float	scanf( )	printf()
string	scanf( )	printf()

Syntax of Format Specifier

Each format specifier begins with the percentage (%) followed by a conversion character which indicates the type of the corresponding data item. Similar control string is also used with the printf function.

The following table indicates few format specifiers which apply to both printf() and scanf().

Table 2 Syntax for format specifiers

Format specifiers	**Input Data (scanf)**	**Output Data (printf)**
%c	reads a single character	Single Character
%d	Reads a numeric value as signed int. Treats the input data as a decimal number i.e a number in the base 10 number system.	Signed decimal integer
%i	Can read data value provided either as a decimal int, hexadecimal int, octal int	Print as signed decimal integer
%o	Reads data value as an octal number	Prints data as an octal integer without leading zero.
%x	Reads data value as a hexadecimal number	Prints data as hexadecimal integer without leading 0x.
%u	Reads data value as unsigned integer	Prints as an unsigned integer
%h	Reads data value as a short integer	N/A
%f	Reads data as a floating point value without the exponent.	Prints as a floating point value(without exponent)
%e	Reads data as a floating point value. The input data can also have an exponent part.	Prints as a floating point value in exponent form.

%E	This is same as %e, except that in the case the exponent is represented with a capital E.	This is same as %e, except that in the case the exponent is represented with a capital E.
%g	Reads data as a floating point value in either of the format specified for %f and %e.	Prints as a floating point value. Depending upon the magnitude of the value, it will be displayed either e-type or f-type conversion. If the value does not have any fractional part, the trailing decimal point is not printed.
%G	This is same as %g, except that in this case if an exponent is there, it is represented with capital E.	This is same as %g, that in this case if an exponent is there, it is represented with capital E.
%s	Reads data as a string of characters. Reading terminates when a whitespace character is encountered. The read string of characters is appended with a null character "\\0" at the end.	Prints a character string
%[ ]	Reads data as a string which may even include whitespace characters as well.	N

Check Your Progress A

Fill in the blanks :

1. Printf and scanf are ………… functions
2. The standard input output devices are referred as ……
3. Format specifier for printing an unsigned integer using printf is……..
4. %d is the format specifier for………….
5. What is the name of header file for formatted input/output functions?

4.4 NON - FORMATTED INPUT - OUTPUT FUNCTIONS

The non formatted input output functions consists of getchar(), putchar(), gets() and puts().The header file used for these functions is conio.h.

getchar ()

This is a single character input function. getchar() reads a single character from stdin (the standard input data stream) and returns an integer. It only reads a single character at one time and a loop can be used if you need to read more than one character.

Example

```
#include<stdio.h>
main()
{
char c;
printf (“\n continue(Y/N)?”);
c=getchar ();
if(c=='Y’)
{
-do-something-
}
else
{
-do something else-
}
}
```

When you input any single character for the C program to read, you must indicate end of the data stream or end of the input by pressing the return/enter enter key without inputting any value first, the getchar function will return the symbolic constant EOF which typically stores the value 1, when there is no input. The EOF is an integer constant defined in stdio.h header file.

putchar ()

This is a single character output function. Putchar () writes a single character to stdout. If you want to display more than one character, you should you putchar() inside a loop.

Example:

```
#include<stdio.h>
```

```
main ()
{
char c;
printf (“\n continue(Y/N));
c=getchar ();
putchar(c);
}
```

You can declare the variable c either as a char or as an int, a single character is internally represented as it's corresponding numeric ASCII code.

gets () and puts ()

The standard library function gets accepts inputs in the form of a string. The Character string may even include whitespace characters. Each call to gets will read all the characters from the input steam until an end of the line character is encountered. The end of the line character is represented as \n and gets generated when you press the enter key. gets assigns the read string to the variable that is passed as it's parameters. Gets assigns NULL when an error occurs.

The standard library function puts sends the passed string to stdout. After the output, outs send out a carriage return and a line feed character. This takes the cursor to the next line automatically.

Example

```
#include<stdio.h>
```

```
main ()
{
char name [71];
printf("\nEnteryourname");
gets(name);
puts(name);
}
```

Output

```
NAME
```

Table 3 Non-Formatted Functions

Type	**Input**	**Output**
char	getchar( )	putchar()
int	-	-
float	-	-
string	gets( )	puts()

Check Your Progress B

Fill In The Blanks :

1. The getchar() function accepts………..(single/multiple) characters as input.

2. What is the name of header file for non-formatted input-output functions?
3. Which function is used for multiword string input?

4.5 SUMMARY

Input means to provide data to the computer. Output means to display that data. C uses various input output functions for console and file operations. Console input output functions are related to keyboard and displaying screen such as monitor. There are various functions in header files stdio.h and conio.h which are used for character and string input-output.

4.6 KEYWORDS

Input: Input means to feed the data into a program. Input can be given in the file form or directly from the command line.

Output: Output means displaying the data on the screen, printer or in a file.

Stdio.h: Stdio.h is the header file, which is included in a C program to use the formatted input output functions.

Conio.h: Conio.h is the header file, which is included in a C program for non-formatted input output functions.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. Formatted input output functions
2. stdin, stdout, stderr

3. %u
4. Signed decimal integer
5. stdio.h

Check Your Progress B

1. single
2. conio.h
3. gets( )

4.7 SELF ASSESSMENT QUESTIONS

1. What do you understand by input output functions?
2. Explain formatted input output functions in detail.
3. Write the differences between printf and scanf functions.
4. Explain non-formatted input output functions.
5. Differenciate between stdio.h and conio.h header files.

4.8 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code**	**:**	**MAL-645**
Subject	**:**	**Programming in C**			

LESSON 5

OPERATORS AND EXPRESSIONS

STRUCTURE

5.0 OBJECTIVES

After reading this chapter, you should be able to:

a) Understand types of Operators in C
b) Understand all types of arithmetic, Logical, Relational and Bitwise operators
c) Understand various conditional operators, memory operators, cast operators and Size of Operators
d) Get clear idea about Input/output Functions, Precedence and associativity
e) Clearly understand Unary, increment and decrement operators

5.1 OPERATORS

Operators

C language supports a rich set of built in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate the data and variables.

Types of Operators

C operators can be classified into following types. They are

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators

- Assignment Operators
- Conditional Operators
- Special Operators

5.2 PRECEDENCE AND ASSOCIATIVITY

If more than one operator is involved in an expression, c language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.

In C language, the precedence of arithmetic operators (*, %, /, +, -) is higher than the relational operators (==, != , >, <, >= ,<=) and precedence of relational operators is higher than the logical operators (&& , || , !).

Example:

(1> 2 + 3 && 4)

This expression is equivalent to

((1> (2+3)) &&4)

i.e. (2+3) executes first and results 5

first part of expression (1>5) executes and results 0 (false)

Now, (0 && 4) executes and results into 0

(false)

Output

0

Associativity of operators

Associativity is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left.

Example:

1 == 2! = 3

In the above example, the operators “== “and “! =” have same precedence. The associativity of both == and != is left to right i.e. the expression on the left is executed first and moves towards the right.

The above expression is equivalent to

((1 == 2)! = 3

1 == 2 executes first and results 0 (false)

Later 0! = 3 executes and results 1 (true)

Output

```
1
```

The following table shows all the operators in C with precedence and associativity.

Table 1 Precedence and Associativity Table for C Operators

OPERATORS	DESCRIPTION	ASSOCIATIVITY
()	Parentheses	Left to Right
[]	Brackets	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment / decrement	
++ --	Prefix increment / decrement	Right to Left
+-	Unary plus / minus	
!~	Logical negative / Bitwise complement	
(type)	Cast	
*	Dereference	
&	Address	
sizeof	Determine size in bytes in this implementation	
*	Multiplication	Left to Right
/	Division	
%	Modulus	
+	Addition	Left to Right
-	Subtraction	
<<	Bitwise shift left	Left to Right
>>	Bitwise shift right	
<	Relational less than	Left to Right
<=	Relational less than equal to	
>	Relational greater than	
>=	Relational greater than or equal to	
==	Relational is equal to	Left to Right

!=	Relational is not equal to	
&	Bitwise AND	Left to Right
^	Bitwise Exclusive OR	Left to Right
|	Bitwise Inclusive OR	Left to Right
&&	Logical AND	Left to Right
||	Logical OR	Left to Right
?:	Ternary Conditional	Right to Left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition Assignment Subtraction Assignment Multiplication Assignment Division Assignment Modulus AND Assignment Bitwise AND Assignment Bitwise Exclusive OR Assignment Bitwise Inclusive OR Assignment Bitwise Shift Left Assignment Bitwise Shift Right Assignment	Right to Left
,	Comma	Left to Right

Check Your Progress A

Fill in the blanks:

1. Associativity of operators can be either from ………. and ………
2. In C, the precedence of arithmetic operators is ……. than the relational operators.

3. Precedence of comma operator is …….
4. ?: is called as ……… operator.
5. Operator is defined as ………….
6. Reference operator is denoted as ……
7. Comma operator acts as…..
8. Bitwise operator works on……. and perform ……. operations.

5.3 UNARY OPERATORS

A unary operator in C is an operator that takes a single operand in an expression or a statement. These operators act upon a single operand to produce new value.

Types of Unary Operator

There are different types of unary operators. They are

- Unary plus (+)
- Unary minus (-)
- Increment (++)
- Decrement (--)
- Not (!)
- Addressof (&) operator
- sizeof operator
- Dereferencing (*)
- Cast operator

All unary operators have associativity from right to left.

Unary Plus:

The result of the unary plus operator is the value of its operand. The operand to the unary plus operator must be of an arithmetic type. Integral promotion is performed on integral operands. The resultant type is the type which the operand is promoted.

Example:

```
#include<stdio.h>

int main ()

{

int a = +8;

printf (“a=%d\n”, a);

return 0;

}
```

Output

```
a=8
```

Unary Minus:

The minus operator changes the sign of its argument. A positive number becomes negative and a negative number becomes positive. Unary minus is different from subtraction operator, as subtraction requires two operands.

Example:

```
#include<stdio.h>
 int main ()
{
int a= 20;
int b= -10;
printf (“value of -a: %d\n”, -a);
printf (“value of -b: %d\n”, -b);
}
```

Output

```
value of -a: -20
value of -b: 10
```

Increment and Decrement Operator

Increment Operators are used to increase the value of the variable by one. Both increment and decrement Operators are used on single operand or variable. C offers two unusual and unique operators (++) and (--) are known as increment and decrement operators. The use of these two operators will results in the value of the variable being incremented or decremented by unity. i=i+1 can be written as i++ and j=j-1 can be written as j--.

The operators can be placed before or after the operand. If the operator is placed before the variable (++i or --i) it is known as pre- incrementing or pre-decrementing. If the operator is placed after the variable (i++ or i--) is known as post incrementing or post decrementing.

Pre-incrementing or pre-decrementing and post incrementing or post decrementing have different effect when used in expression. In the pre-increment or pre-decrement, the value of the operand is incremented or decremented first and then assigned.

Increment:

It is used to increment the value of the variable by 1. The increment can be done in two ways.

Prefix increment:

In this method, the operator precedes the operand(++a). The value of operand will be altered before it is used.

Example:

int a=2;

int b = ++a;

Output

b=3

Example:

```
#include<stdio.h>

#include<conio.h>

void main ()

{
```

```
int x, i;
i=5;
x=++i;
printf ("x: %d", X);
printf ("i: %d", i);
getch ();
}
```

Output

```
x=6
i=6
```

In the above program, the value of i is first increased and then value of i is used into expression.

Postfix increment:

In this method, the operator follows the operand (ex: a++). The value of operand will be altered after it is used.

Example:

```
int a=1;
int b= a++;
```

Output

```
b=1, a=2
```

Example:

```
#include<stdio.h>

#include<conio.h>

void main ()

{

int x, i;

i=5;

x=i++;

printf (“x: %d”, x);

getch ();

}
```

Output

```
x=6;

i=6;
```

Decrement:

It is used to decrement the value of the variable by 1. The increment can be done in two ways.

Prefix Decrement:

In this method, the operator precedes the operand (ex: ++a). The value of operand will be altered before it is used.

```
int a=1;

int b= --a;
```

Output

```
b=0
```

Example:

In this program, the value of i is used in the expression and then increase the value by 1.

```
#include<stdio.h>

#include<conio.h>

void main ()

{

int x, i;

i=5;

x=--i;

printf (“x: %d”, x);

printf (“i: %d”, i);

getch ();

}
```

Output

```
x=4;

i=4;
```

Postfix Decrement:

In this method, the operator follows the operand (ex: a--). The value of operand will be altered after it is used.

```
int a=1;
int b=a--;
int c=a;
```

Output

```
b=1
c=0
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int x, i;
i=5;
x=i--;
printf (“x: %d”, x);
printf (“i: %d”, i);
getch ();
}
```

Output

```
x=5;

i=4;
```

In the above program, the value of X is first used in the expression and the value of i is decreased by 1.

Example for Inrement and Decrement operators-

Example

```
 #include<stdio.h>

#include<conio.h>

void main ()

{

int p, q, r, s;

clrscr ();

printf (“Enter the value of r”, \n);

scanf (“%d”, &r);

printf (“Enter the value of s”, \n);

scanf (“%d”, &s);

printf (“r=%d\ns=%d\n”, r, s);

p=r++;

q=s++;

printf (“r=%d\ts=%d\n”, r, s);
printf (“r=%d\ts=%d\n”, p, q);

p=--r;

q=--s;

printf (“x=%d\ty=%d\n”, r, s);

printf (”p=%d\tq=%d\n”, p, q);

getch ();

}
```

Output

```
Enter the value of r 10

 Enter the value of s 10

r=10

s= 20

r=11, s=21

p = 10 q= 20

r=10, s= 20

p=10, q= 20
```

Not (!):

It is used to reverse the logical state of its operand. If a condition is true, then logical NOT operator will make it false.

if x is true, then! x is false

if x is false, then! x is true

Addressof operator (&)

It gives an address of variable.it is used to return the memory address of a variable. These addresses returned by the address of operator are known as pointers because they point to the variable in memory.

Example:

int a;

*int *ptr;*

ptr=&a;

Sizeof ():

This operator returns the size of its operands in bytes. The sizeof operator always precedes its operand. The operand is an expression or may be cast.

Cast Operator:

The cast operator can be used to explicitly convert the value of an expression to a different type. To do this the name of the data type to which the conversion is to be made is enclosed in parentheses and placed directly to the left of the expression whose value is to be converted.

Syntax :

(data type) expression

Example:

```
#include<stdio.h >

#include<conio.h>

void main ()

{

int a=100;

float b= 3.1652;

double c= 38.426518

char d= 'c’;

clrscr ();

printf (“a=%d\nb=%d\fc=%if\nd=%c\n”, a, b, c, d);

printf(“(float)a: %f\n”, (float)a);

printf(“(float)b: %f\n”, (float)b);

printf(“(float)c: %f\n”, (float)c);

printf(“(float)d: %f\n”, (float)d);

printf(“(int)a: %d\n”, (int)a);

printf(“(int)b: %d\n”, (int)b);

printf(“(int)c: %d\n”, (int)c);

printf (“(int)d: %d\n”, (int)d);

printf(“(char)a: %c\n”, (char)a);
printf("(char)b: %c\n", (char)b);
printf("(char)c: %c\n", (char)c);
printf("(char)d: %c\n", (char)d);
getch ();
}
```

Output

```
a = 100
b = 3.1652
c = 38.426518
d = c
(float) a = 100.000000
(float)b = 3.165200
(float)c= 38.426518
(float)d= 99.0000
(int) a = 100
(int)b = 3
(int)c = 38
(int)d = 99
(char)a: d
(char)b:
(char)c:
```

(char)d: c

Check Your Progress B

Fill in the blanks:

1. All unary operators have associativity from …… to ……
2. Prefix increment is denoted by …...

5.4 BINARY ARITHMETIC OPERATORS

The arithmetic operators perform arithmetic operations. C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators. Assume variable A holds 20 and variable B holds 40.

Table 2 Binary Arithmetic Operators in C

OPERATORS	**DESCRIPTION**	**EXAMPLE**
+	Adds two operands	A + B = 60
-	Subtract second operands from first	A - B = 20
*	Multiply two operands	A * B = 800
/	Divide numerator by denominator	B / A = 2
%	Remainder after an integer division	B % A = 8
++	Increment operator - increases	A + + = 21

	integer value by one	
--	Decrement operator - decreases integer value by one	A - - = 19

Example

```
#include<stdio.h>
int main()
{
int a=20, b=40, add, sub, mul, div, mod;
add=a+b;
sub=a-b;
mul=a*b;
div=a/b;
mod=a%b;
printf (“Addition of a, b is: %d\n, add);
printf (“subtraction of a, b is: %d\n, sub);
printf (“multiplication of a, b is: %d\n, mul);
printf (“division of a, b is: %d\n, div);
printf (“modulus of a, b is: %d\n, mod);
}
```

Output

```
Addition of a, b is: 60

Subtraction of a, b is: 20

Multiplication of a, b is: 800

Division of a, b is: 2

Modulus of a, b is: 8
```

5.5 ARITHMETIC ASSIGNMENT OPERATORS

The assignment operator assigns the values to an identifier. The assignment operator is denoted by “=”. Assignment operator is a binary operator which operates on two operands. This operator will have two values i.e. R value and L value. This operator assigns the values on the right-hand side of the operand to the left-hand side. An assignment operator requires the value of expressions to be well defined and ensures that variables represent an entity that can be modified. Assignment operators also permit the same variable to hold different values at the different stages of program execution. This makes assignment operators devoid of referential transparency, where procedures are supposed to return the same results for a partial set of inputs at a given time. Assignment operators are basic components where several values are associated with variable names at different stages of program’s execution. The assignment operator has lower precedence than other operators but has higher precedence than comma operator.

For instance, consider

```
int x= 5;
```

double y ;

x= 10;

In this set of statements, the variable x is initially assigned the value 5. Internally, a memory location is reserved for x, holding value 5. In the third statement, however same variable x is assigned another value. Therefore, the output of value x after the execution of all the three statements represents 10. At the machine level, assignment is executed using operations such as MOVE and STORE.

Assignment operators also allow chained assignment of values. For instance, the expression a=b=c=10, assigns the value 10 to a, b and c. This is referred as chained assignment.

The assignment operators used along with binary operators are called compound assignment operators. They perform binary operator operation on both operands and store the output of the operation in the left operand.

The following table shows the list of assignment operators supported by C language.

Table 3 Assignment Operators in C

OPERATORS	DESCRIPTION	EXAMPLE
=	Simple assignment operator,assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C.
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the	C += A is equivalent to C = C + A.

	left operand.	
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C - = A is equivalent to C = C - A.
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A.
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C / * A is equivalent to C = C / A.
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C % = A is equivalent to C = C % A.
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C & = 2 is same as C = C & 2
^=	Bitwise XOR assignment operator	C^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR assignment operator	C | = 2 is same as C = C | 2

Different ways of using assignment operator:

- Assignment operator used to assign values

```
int main ()
{
int value;
value = 10;
return 0;
}
```

In the above example, we have assigned 10 to the variable.

- Assignment Operator used to cast:

```
int value;
value = 10;
printf (“%d”, value);
```

Assignment operator can typecast higher values to lower values. It can also cast lower values to higher values.

- Assignment operator in if statement:

```
if (value = 10)
printf(“True”);
else
printf(“False”);
```

Above program will always execute True condition because Assignment operator is used to inside if statement not comparison Operator.

5.6 SPECIAL OPERATORS

C language supports some special operators. They are

- Comma Operator (,)
- Reference Operator (&)
- Dereference Operator (*)
- Size of () operator

Comma Operator (,):

The comma operator can be used to link the related expressions together. Comma Operator is a set of expression separated by comma is a valid constant in the C language. This is a special kind of operator which is widely used in programming to separate the declaration of multiple variable.

Comma Operator has lowest precedence i.e. it is having lowest priority so it is evaluated at last. The comma Operator returns the value of the right operand when multiple comma operators are used inside an expression.

Comma Operator can act as

Operator: it acts as operator in the expression.

Separator: It acts as separator in declaring variable, in function call, function definitions, variable declarations, enum declarations.

Comma as operator:

res = (num1, num2)

In this case the value of the rightmost operator will be assigned to the variable.

Comma as separator:

Int num1=1, num2=2;

Example:

/*Using comma operator inside printf statement*/

Printf (“computer”, “programming”);

Reference Operator (&):

Reference Operator is denoted by “&”. It is also a unary operator in c language that's used for assigning address of the variables. It returns the pointer address of the variable. This is known as Reference Operator.

Dereference Operator:

The dereference operator is also known as indirection operator. This is denoted by “*”. This is also a unary operator in c language that is used for pointer variables. This operates on pointer variable and returns i-value equivalent to the value at the pointer address. This is known as dereferencing operator.

Example:

#include<stdio.h>

int main ()

{

*int*pt;*

int var;

```
var=1;

printf (“Address of var: %d\n”, &var);

printf (“Value of var: %d\n”, var);

pt=&var;

printf (“Address of pointer pt: %d\n”, pt);

printf (“Content of pointer pt: %d\n\n”, *pt);

var=2;

printf (“Address of pointer pt: %d\n”, pt);

printf (“Content of pointer pt: %d\n\n”, *pt);

*pt=3;

Printf (“Address of var: %d\n”, &var);

Printf (“value of var: %d\n\n”, var);

return 0;

}
```

Output

```
Address of var: 1565276140

Value of var: 1

Address of pointer pt: 1565276140

Content of pointer pt: 1

Address of pointer pt: 1565276140

Content of pointer pt: 2
```

```
Address of var: 1565276140

Value of var: 3
```

Sizeof () Operator:

Sizeof is much used operator in C language. It is a compile time unary operator which can be used to compute the size of its operand. The result of sizeof is of unsigned integral type which is usually denoted by size_t. sizeof can be applied to any data type, including primitive types such as integer and floating-point types, pointer types or compound data types such as structure, union etc. When the sizeof () is used with an operand, it returns the number of bytes that the operand occupies. The operand may be a variable, constant or data type qualifier. A sizeof operator is normally used to determine the lengths of arrays and structures. It is also used to allocate memory space dynamically to variables during execution of a program.

Syntax:

Sizeof (data type)

Sizeof is used to find out number of elements in an array.

Example:

```
#include<stdio.h>

int main ()

{

int arr [] = {1,2,3,4,5,6,7};

printf (“Number of elements: %d”, sizeof(arr)/sizeof (arr [0]));
return 0;

}
```

Output

```
Number of elements = 7
```

Sizeof is used more in dynamic memory allocation.

Example:

```
int*ptr = malloc(10*sizeof(int));
```

Sizeof data type :

```
#include<stdio.h>

int main ()

{

Printf ("%d", sizeof(int));

Printf ("%d", sizeof(char));

Printf ("%d", sizeof(float));

 return 0;

}
```

Sizeof constant:

```
#include<stdio.h>
```

```
int main ()
{
 Printf (“%d”, sizeof(10));
 printf(“%d”, sizeof(A));
 printf(“%d”, sizeof(10.10));
 return 0;
 }
```

Nested Sizeof operator:

```
 #include<stdio.h>
 int main()
 {
 int num = 10;
 printf (“%d”, sizeof(num));
 return 0;
}
```

We can use nested Sizeof in C language, inner sizeof will be executed normally and the result of inner sizeof will be passed as input to outer sizeof operator.

The inner sizeof operator will evaluate size of variable “num” i.e. 2 bytes. The outer sizeof operator will evaluate size of constant “2” i.e. 2 bytes.

Example :

```
 #include<stdio.h>
 #include<conio.h>
 int main ()
  {
 int a;
 char b;
 float c;
 double d;
 printf (“storage size for int data type: %d\n”, sizeof(a));
 printf (“storage size for char data type: %d\n”, sizeof (b));
printf (“storage size for float data type: %d\n”, sizeof(c));
printf (“storage size for double data type: %d\n”, sizeof (d));
return 0;
}
```

Output :

```
storage size for int data type: 4
storage size for float data type: 1
storage size for char data type: 4
storage size for double data type: 8
```

5.7 RELATIONAL OPERATORS

Relational Operators are used to compare logical, arithmetic and character expression. Each of these six relational operators takes two operands. Each of these operators compares their left side with their right side. The whole expression involving the relation operator then evaluate to an integer. It evaluates to 0 if the condition is false and 1 if the condition is true.

The following table shows all the relational operators. Assume variable A holds 20 and B holds 40.

Table 4 Relational Operators

OPERATORS	**DESCRIPTION**	**EXAMPLE**
==	Checks if the values of two operands are equal or not. If yes then the condition becomes true.	(A = = B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal then the condition becomes true.	(A! = B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes then the condition becomes true.	(A > B) is not true
<	Check if the value of the left operand is lesser than the value of the right operand. If yes then the condition becomes true.	(A < B) is true
	Check if the value of left operand is greater	(A > = B) is not true

>=	than or equal to the value of right operand. If yes then the condition becomes true.	
<=	Check if the value of left operand is lesser than or equal to the value of right operand. If yes then the condition becomes true.	(A< = B) is true

Example

```
#include<stdio.h>
int main()
{
int a=20,b=30;
if(a==b)
{
printf("a and b are equal");
}
else
{
printf("a and b are not equal");
}
}
```

Output

```
a and b are not equal
```

5.8 Logical Operators

A logical operator is used to compare and evaluate logical and relational expression.

The following table shows all the logical operators.

Assume variable A holds 0 and variable B holds 1.

Table 5 Logical Operators in C

OPERATORS	**DESCRIPTION**	**EXAMPLE**
&&	This is called as Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	This is called Logical OR operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	This is called Logical NOT operator. It is used to reverse the logical state of its operand. If a condition is true, then logical NOT operator will make it false.	!(A && B) is true

Example:

```
#include<stdio.h>

int main()

{

int a= 20,b=40;

int c=60,d=80;

if(a>b && b!=0)

{

printf("&& operator : Both conditions are true\n");

}

if(c>d || d!=20)

{

printf("|| operator : only one condition is true\n");

}

if(!(a>b && a!=0)

{

printf("! operator : Both conditions are true\n");

}

else

{
```

```
printf("! operator : Both conditions are true" "but status is inverted as
false\n");
 }
 }
```

Output

```
&& operator : Both conditions are true
|| operator : only one condition is true
! operator : Both conditions are true. But status is inverted as false.
```

5.9 BITWISE ASSIGNMENT OPERATORS

Bitwise Operators works on bits and perform bit - by - bit operations. These operators operate on individual bits of integer (int and long) values. This is useful for writing low level hardware or OS code where the ordinary abstractions of numbers, characters and pointers are insufficient. Bit manipulation code tends to be less portable, if without any programmer intervention it compiles and runs correctly on different types of computers. The Bitwise operations are commonly used with unsigned types. These operators perform bitwise logical operations on values. Both operands must be of the same type and width, so resultant value will also be of same type and width.

Table 6 Bitwise Assisgnment Opertors In C

OPERATORS	DESCRIPTION
&	Binary AND operator copies a bit to the result if it exists in both the operands.
!	Binary OR operator copies a bit if it exists in either operand.
^	Binary XOR operator copies the bit if it is set in one operand but not the both.
~	Binary Ones complement operator is unary and has the effect of flipping bits.
<<	Binary Left shift operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right shift operator. The left operands value is moved right by the number of bits specified by the right operand.

Now, let's see the truth table for bitwise ‘**&**’ ‘|’ and ‘^’

p	**Q**	**p & q**	**p | q**	**p ^ q**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise shift operator shifts the bit value. The left operand specifies the value to be shifted and the right operand specified the number of positions that the bits in the value has to be specified. Both operands have the same precedence.

Example

```
#include<stdio.h>

int main ()

{

int a=40, b=80, AND_opr,OR_opr, XOR_opr,NOT_opr;

AND_opr = (m&n);

OR_opr = (m|n);

XOR_opr = (m^n);
NOT_opr = (~m);
Printf ("AND_opr value = %d\n",AND_opr);
Printf ("OR_opr value = %d\n",OR_opr);
Printf ("XOR_opr value = %d\n",XOR_opr);
printf ("NOT_opr value = %d\n",NOT_opr);
printf ("left_shift value = %d\n",a<<1);
printf ("right_shift value = %d\n",a>>1);
}
```

Output

```
AND_opr value = 0
OR_opr value = 120
NOT_opr value= -41
XOR_opr value = 120
left_shift_ value = 80
right _shift_value = 20
```

5.10 BIT MANIPULATION OPERATORS

Bit manipulation is the act of algorithmically manipulating bits or other pieces of data shorter than a word. Source code that does bit manipulation makes use of the bitwise operations.

AND, OR ,XOR, NOT and bit shifts

Example:

```
int computeXOR
{
if(n%4==0
return n;
if(n%4==0
return 1;
if(n%4==2)
return n+1;
else
return 0;
}
```

Output:

```
Input : 6
Output : 7
```

5.11 CONDITIONAL OPERATORS

The conditional operators in C language are also known by the two more names.

They are

- Ternary Operator

- ? : Operator

Syntax

The syntax of conditional Operator is

Conditional Expression? expression1 :   expression2

The conditional operator works as follows:

The first expression conditional expression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false. If conditional expression is true, expression1 is evaluated. If conditional expression is false, expression2 is evaluated.

The conditional expression can be used as shorthand for some if-else statements. This operator consists of two symbols: question mark (?) And colon (:).

This is an expression, not a statement so it represents a value. The operator works by evaluating test expression.

The classic example of the ternary Operator is to return the smaller of two variables.

```
if (p < q)
{
min=p;
}
else
{
```

```
min = q;
}
```

Example:

```
#include<stdio.h>
#include<conio.h >
int main ()
{
int x, y, result, choice;
clrscr ();
printf (“Enter first number\n”);
scanf (“%d”, &x);
printf (“Enter second number”);
  scanf (“%d”, &y);
  printf (“Enter 1 for addition or 2 for multiplication \n”);
 scanf (“%d”, & choice);
  result=(choice==1)?x+y:(choice==2)?x*y:printf (“invalid input”);
 if(choice==1||choice==2);
printf (“The result is %d\n\n”, result);
getch ();
}
```

Output

```
Enter first number
10
Enter second number
3
Enter 1 for addition or 2 for multiplication
2
The result is 30
```

5.12 MEMORY OPERATORS

The C language provides several functions for memory allocation and management. These functions can be found in the<stdlib.h>.

Functions of Memory Operators

C language supports four functions of which deals with memory allocation. They are

- The Malloc Function
- The Calloc Function
- The Realloc Function
- The Free Function

The Malloc Function:

The standard C function "malloc" is the means of implementing dynamic memory allocation. It is defined in stdlib.h or malloc.h depending on operating system. Malloc.h contains only the definitions for the memory allocation functions and not the rest of the other functions defined in stdlib.h. The corresponding call to release allocated memory back to the operating system is free. When dynamically allocated memory is no longer needed, free Should be called to release it back to the memory pool. Overwriting the pointer that points to dynamically allocated memory can result in particular data becoming inaccessible. If this happens frequently, eventually the operating system will no longer be able to allocate more memory for the process. Once the process exists, the operating system is able to free all dynamically allocated memory associated with the process.

The Calloc Function:

The Calloc Function allocates the space for an array of items and initializes the memory to zeros. The call mArray = calloc (count, sizeof ((struct v)) allocates count objects each of whose size is sufficient to contain an instance of the structure. The space is initialized to all bits zero. The function returns either a pointer to the allocated memory or if the allocation fails null.

The Realloc Function:

*void *realloc (void *ptr, size_t size);*

The Realloc Function changes the size of the object pointed to the size specified. The contents of the object shall be unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate. If pointer is a null pointer then the realloc function behaves like the malloc function for the specified size. Otherwise if pointer doesn't match a pointer earlier returned by the Calloc, malloc or realloc function or if the space has been deallocated by the call to free or realloc function, the behavior is undefined. If the space cannot be allocated the object pointed is unchanged. If size is zero and pointer is not null pointer, the object pointed to is freed. The realloc function returns either a null pointer or a pointer to the possibly moved allocated project.

The Free Function :

Memory that has been allocated using malloc, Calloc or realloc must be released back to the system memory pool once it is no longer needed. This is done to avoid perpetually allocating more and more memory, which could result in an eventual memory allocation failure.

Memory that is not released is released when the current program terminates on most operating systems.

Syntax

```
#include<stdlib.h>

void *calloc (size_t nmemb, size_t size);

void free (void *ptr);

void *malloc (size_t size);

void *realloc (void *ptr, size_t, size);
```

Example

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>
```

```
int main ()
{
char name [100];
char *description;
strcpy (name, “Buddy”);
description = malloc (30 *sizeof(char));
if(description==NULL)
{
Fprintf (stderr,” Error - unable to allocate required memory \n”);
 }
else
{
Strcpy (description, “Buddy a dps student”);
}
description=realloc (description,100* sizeof(char));
if(description==NULL)
{
 Fprintf (stderr,” Error - Unable to allocate required memory\n”);
}
else
strcat (description:” she is in class 4th”);
}
Printf (“Name=%s\n”, name);
printf (“description: %s\n”, description);
free (description);
}
```

Output

```
Name: Buddy
Description: Buddy a DPS student, she is in class 4th.
```

Same program can be written using Calloc (), need to replace malloc () with Calloc ().

Calloc (200, size of char));

So, you have complete control and you can pass any value while allocating memory, unlike arrays where once the size defined, you cannot change it.

Check Your Progress C

Give short answers:

1. What are the various functions of memory operators ?
2. Where can we find the functions in C ?

5.13 SUMMARY

Operators are specifically used to perform certain mathematical and logical operations in C language. The expressions are nothing but mix of all the variables, operators and constants. The Precedence and associativity of operators will be based on the priority of operator. All the mathematical calculation programs are operated using arithmetic, increment, decrement and unary operators. Logical calculations in C are performed using relational, Logical, Binary, Bit wise and conditional operators. Memory operators and Input/output operators are used to retrieve messages or values. Finally, without using operators no calculations can be made in C.

5.14 KEYWORDS

Operator: Operators are used to perfrom mathematical and logical operations.

Precedence: Precedence is the priorty of evaluating an opertor in an expression.

Memory Opertor: Memory operators are used to assign memory to the variables in a program.

Associativity: Associativity is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left.

ANSWERS TO CHECK YOUR PROGRESS

Check your progress A

Fill In The Blanks :

1. Left to right or Right to Left
2. Higher
3. Left to Right
4. Ternary conditional
5. a symbol that tells the compiler to perform certain mathematical and logical manipulation.
6. &
7. Operator, Separator
8. Bits & Bit - by - Bit operations

Check your progress B

Fill In The Blanks :

1. Right to left
2. ++a

Check Your Progress C

1. Malloc, Calloc, Realloc, Free
2. <stdlib.h>

5.15 SELF-ASSESSMENT QUESTIONS

1. Describe various types of operators?
2. Describe about precedence and associativity of operators?
3. Describe about increment and decrement operators?
4. Describe about Memory Operators ?

5.16 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code:**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON - 6

DECISION CONTROL STRUCTURE

STRUCTURE

6.0 OBJECTIVES

After reading this chapter, you should be able to:

a) Understand flow of control statements.

b) Usage of switch statements.

c) Understand the concept of if, if else and nested if statements.

d) Usage of Goto statements.

6.1 CONTROL FLOW AND TYPES

Control flow in C is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated. The emphasis on explicit control flow distinguishes an imperative programming language from a declarative programming language.

Within an imperative programming language, a control flow statement is a statement the execution of results in choice being made as to which of two or more paths. For non-strict functional languages, functions and language constructs exist to achieve the same result, but they are usually not termed as control flow statements.

Set of statements in turn generally structured as block, in addition to grouping also defines a lexical scope. Interrupts and signals are low level mechanism that can alter the flow of control in a way similar to subroutine, but usually occur as a response to some external stimulus or event, rather than execution of an in-line control flow statement.

At the level of machine language or assembly language, control flow instructions usually work by altering the program counter. To handle such type of statements some flow controls are needed. These flow controls are called Control Statements. The control statements are used to control the cursor in a program according to the condition or to the requirement in the loop.

Types of Control Statements:

There are three types of Control Statements. They are

- Branching
- Looping
- Jumping

Decision Control Statements

C supports two types of decision control statements that can alter the flow of a sequence of instructions. These include conditional type branching & unconditional type branching.

The conditional branching statements help to jump from one art of the program to another depending on whether the particular condition is satisfied.

Decision control statements are of following categories. They are

- **If statement**
- **The If else statement**
- **Nested if statement**
- **Switch statement**

6.2 IF STATEMENT

This is the simplest form of the control statement. It is very frequently used in allowing the flow of program execution and decision making.

Syntax

```
if (test expression)
{
statements;
}
```

The **if** statement evaluates the test expression inside the parenthesis. If the test statement is evaluated to true (non-zero), statements inside the body of **if** is executed. If the test expression is evaluated to false (0), statements inside the body of **if** is skipped from execution. The test expressions are written with the help of operators, which have already been described earlier in this book.

Example

```
int main()
{
int number;
printf(“Enter an integer: “);
scanf(“%d”, & number);
if (number<0)
{
printf (“you entered %d.\n”, number);
}
printf (“We are out of the if block.”);
return 0;
}
```

Output 1

```
Enter an integer: -4
You entered -4
We are out of the if block.
```

When user enters -4, the test expression (number<0) becomes true. Hence -4 is displayed on the screen.

Output 2

```
Enter an integer: 6
We are out of the if block.
```

When user enters 6 in the test expression (number<0) becomes false and the statement inside the body of **if** is skipped.

The flowchart for the **if** statement is presented here in figure 1 to help you understand the flow of control in a **if** statement.

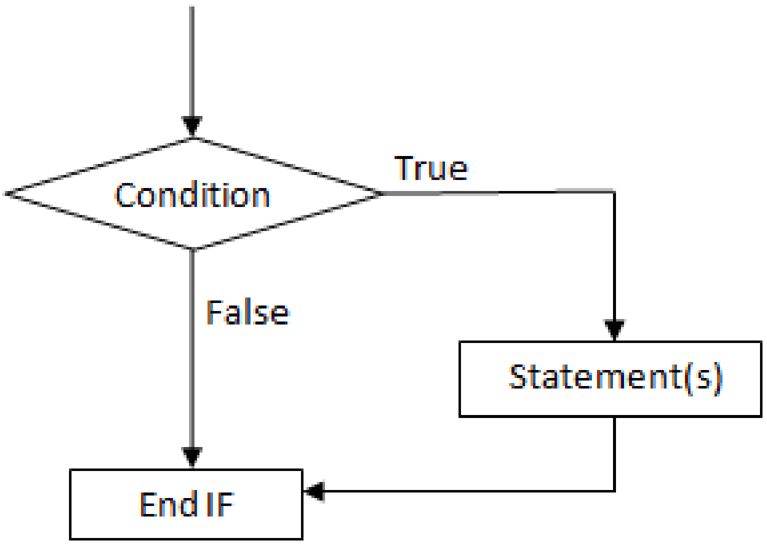


Figure 1 Flow chat for IF statement

In C, a non-zero value is considered as true and a 0 is considered as false. For example, if there is a value inside the **if** statement other than zero, the expression is considered to give a true value. Look at the following examples-

```
if(5+7*4)
{
printf("If statement is true.")
}


if(-10)
{
printf("If statement is true.")
}
```

In above examples, both the **if** statements are executed as true as there is some value inside the **if** function. The condition of **if** will false only when there is a 0 inside it.

6.3 IF-ELSE STATEMENT

The **if-else** statement executes the code if the test expression is true (non-zero) inside **if** and if the **if** statement is executed as false (0), code after the else statement is executed.

Syntax

```
If(test expression)
{
statement1;
}
else
{
statement2;
}
```

If the test expression is true, codes inside the body of the **if** statement is executed and codes inside the body of **else** statement is skipped. If the test expression is false, codes inside the body of the **else** statement is executed and codes inside the body of **if** statement is skipped. The sequence of execution for the flow of control for **if-else** is shown in the figure 2.

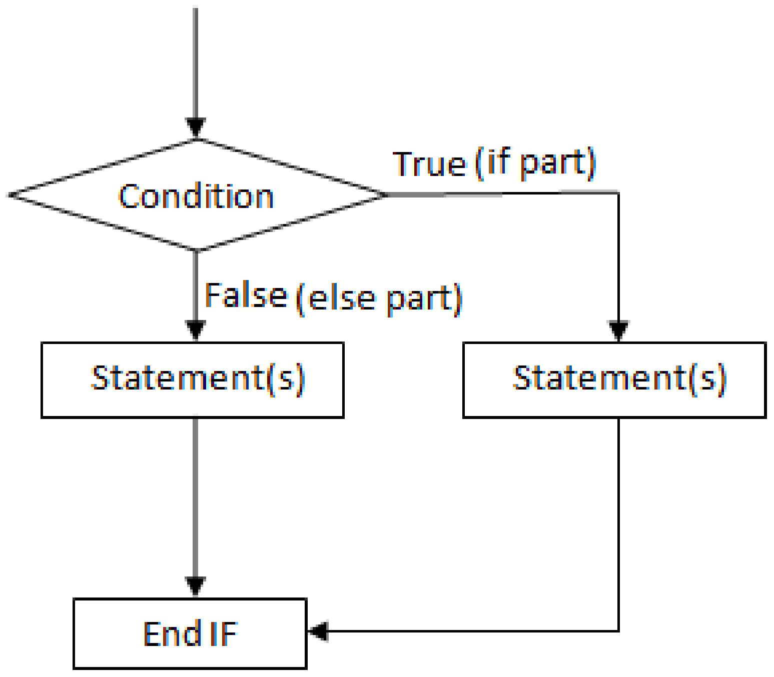


Figure 2 Flow chart for If-Else Statement

Example

```
int main ()
{
int number;
printf (“Enter an integer: “);
scanf (“%d”, & number);
if (number%2==0)
printf(“%d is an even integer, “number);
else
printf (“%d is an odd integer, “number);
return 0;
}
```

Output

```
Enter an integer: 5
5 is an odd integer.
```

When user enters 5, the test expression (number%2==0) is evaluated to false. Hence the statement inside the body of **else** statement printf ("%d is an odd integer"); is executed and statement inside the body of **'if'** is skipped.

6.4 NESTED IF-ELSE STATEMENT

The if else statement executes two different codes depending upon whether the test expression is true or false.

The nested if else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

Syntax

```
if(testExpression1)
{
statement1;
}
else if (testExpression2)
{
statement2;
}
else if(testExpression3)
{
statement3;
}
.
.
else
{
statementn;
}
```

Example

```
int main ()
{
int number1, number2;
printf("Enter two integers: ");
scanf ("%d %d", &number1, & number2);
if(number1 == number2)
{
printf (“Result: %d = %d”, number1, number2);
}
else if (number1 > number2)
{
 printf(“Result: %d > %d”, number1, number2);
}
else
{
printf(“Result: %d < %d”, number1, number2);
}
return 0;
}
```

Output

```
 Enter two integers: 2 4
 Result: 2 < 4
```

Check Your Progress A

Give short answers

1. What are the types of Control Flow statements?
2. What are the categories of Decision control statements?
3. For what values of a if statement, the statements get executed?

6.5 SWITCH STATEMENT

Switch case statements are used as substitute in case of long if statements that compare a variable to several integer values.

Switch statement is a multiway branch statement. It provides an easy way of execution to different parts of code based on the value of the expression. It is a control statement that allows a value to change control of execution.

The value of the variable given into switch is compared to the value following each of the cases and one value matches the value of the variable and the execution of the program is continued from the point. The condition of switch statement is a value.

Syntax

```
switch (expression);
{
case value 1:
break;
case value 2:
break;
default:
```

break;

}

In the switch case, the selection is determined by the value of an expression which is specified and enclosed in the parentheses after the keyword switch. The data type of the value that's the result of the expression must be an integer or else compiler will issue an error. Immediate exit is caused from the switch when used break statement. Though usage of break statement is not mandatory, the statements in the next case will be executed if you don't add break statement at the end of the previous case block. If the expression doesn't match any cases the default statement is the only choice of the switch case statement. The break statement after the default statement is not necessary unless you put another case statement below.

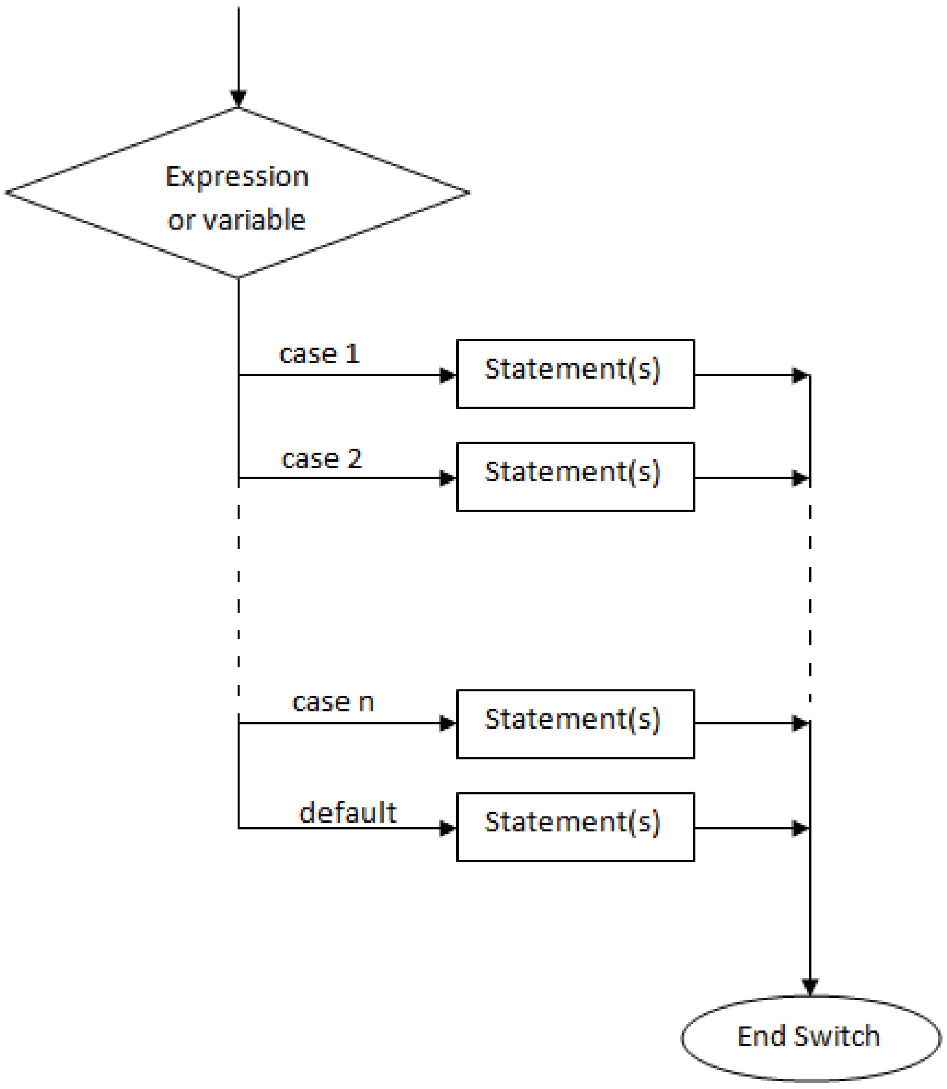


Figure 3 Flowchart for Swich Statement

Example

int main()

{

int color =1;

```
printf (“please choose a color (1: red, 2: green, 3: blue):\n);

scanf (“%d”, &color);

switch (color)

{

case 1:

printf (“you chose red color \n”);

break;

printf(“you chose green color \n”);

break;

printf (“you chose blue color \n”);

break;

default:

printf (“you did not choose any color \n”);

break;

}

return 0;

}
```

Output

```
Please choose a color (1: red, 2: green, 3: blue)

3
```

```
You chose blue color
```

Few more switch case scenarios with examples

Switch

The decision made from the available number of choices by the control statement is known as switch.

Example

```
#include<stdio.h>
int main()
{
 int j = 2;
 switch(j)
 {
 case 1:
 printf("sheet 1 \n");
 case 2:
 printf("sheet 2 \n");
 case 3:
 printf("sheet 3 \n");
 default:
```

```
 printf(“sheet default \n”);
 }
 return 0;
 }
```

Output

```
sheet 2
sheet 3
sheet default
```

In this case switch executes the case where in similarity is found and remaining subsequent cases too.

If we use break statement after the required switch case then the control will go out of the loop.

Example

```
#include<stdio.h>
int main()
{
 int j = 2;
 switch(j)
 {
 case 1:
```

```
 printf("sheet 1\n");
break;
case 2:
 printf("sheet 2\n");
 break;
case 3:
printf(*sheet 3\n");
break;
default:
printf("default sheet");
}
return 0;
}
```

Output

```
sheet 2
```

Scrambled Case Order:

Switch cases can be taken not only in ascending order but in any order.

Example

```
 #include<stdio.h>
 int main()
 {
 int j = 180;
 switch (j)
 {
 case 250:
 printf(“sheet number is 250 \n”);
 break;
 case 120:
printf(“sheet number is 120 \n”);
break;
case 180:
printf(“sheet number is 180 \n”);
break;
default;
printf(“default sheet number is \n”);
}
return 0;
}
```

Output:

```
Sheet number is 180
```

Char values can also be used in switch cases. They get converted into their ASCII values internally.

Example

```
#include<stdio.h>
int main()
{
char c = 'A’;
switch(c)
{
case B:
printf(“sheet name is 25 \n”);
break;
case C:
printf(“sheet name is 26 \n”);
break;
case A:
printf(“sheet name is 27 \n”);
break;
default;
printf(“default sheet name is \n”);
}
return 0;
}
```

Output:

```
sheet name is 27
```

Execution of common set of statements in multiple cases:

When there are multiple case for a switch case, the statements in the cases are executed till they don't encounter a break statement.

Example

```
#include<stdio.h>
int main()
{
char ch;
printf(“Enter any of the characters d,e,f);
scanf(“%d”,&ch);
switch(ch);
{
case 'd’:
case D:
printf(“d is dog \n”);
break;
case 'e’:
case E:
printf(“e is eagle \n”);
break;
case 'f’:
case F:
printf(“f is frog \n”);
break;
default;
printf(“these are alphabets”);
return 0;
}
}
```

Though there are multiple statements to be executed in each case, there is no need to close them in braces. The statements in switch should belong to some case. The compiler won't report error if a statement doesn't belong to any of the case.

Example

```
#include<stdio.h>
int main()
```

```
{
int a,b;
printf("Enter value of a");
scanf("%d",&a);
switch (a)
{
printf("Hai \n");
case 1:
b= 20;
break;
case 2:
b=40;
break;
 }
return 0;
}
```

The program falls through the entire switch and continues with the next instruction that follows the closing brace of switch if there's no default case.

The switch is replacement to if in few cases, because it provides better way to write a program when compared to IF. In some cases, we don't have chance to use switch but only if.

Advantages of Switch:

Switch leads to more structured program.

The value of any expression can be checked in switch.

Disadvantage of Switch:

The switch is not allowed for float values, we can use only int or char or an expression which evaluates to one of the constants.

The value of any expression can be checked in switch. The expressions which are used in the case provided are called as constant expressions. In switch, if the break statement is used it will take control outside the switch. As switch is not a looping statement, continue will not take the control of the beginning not the switch. A switch within the other switch is known as Nested switch statements. The switch statements are useful to write menu driven programs.

6.6 GOTO STATEMENT

The **goto** statement is used to alter the normal sequence of program execution by transferring control to some other part of the program, which is provided by a label, written next to it. **Goto** statement is also called as jump statement which is sometimes referred to as unconditional jump statement. The usage of goto should be avoided in a program as it usually violates the normal flow of execution. Multiple goto statements can trasfer the control to a single label.

Syntax

```
Goto label;
…………
………….
label:
statement;
```

Example

```
int main()
{
int i==10;
do
{
if (i==15)
{
i=i+1;
goto LOOP;
}
printf(“value of i: %d\n”, i);
i++;
}
while (i<20);
return 0;
}
```

Output

```
value of i: 10
value of i: 11
value of i: 12
value of i: 13
value of i: 14
value of i: 15
value of i: 16
value of i: 17
value of i: 18
value of i: 19
```

Check Your Progress B

Fill in the blanks

1. Switch case is also known as …………….
2. Switch can be used for only ……..and ………values.
3. …….statement inside a switch takes control the control……the switch.

6.7 SUMMARY

C language must be able to perform different actions based on the circumstances. Decision control statements are used to make a decision by checking the condition, whether true for false. The flow of control for a program based on the value of test expression inside the if statement. If, if else, nested if and goto statements are decision making statements available in C language helping us to write programs based on the decision making/control.

6.8 KEYWORDS

If: It is a decion control statement of C language. It is always executed whenever there is a non zero value for a test expression.

If-else: If else is a decision control statement where one of the part get executed either from the if block or from the else block.

Switch: It is a type of control statement, which allows to make a decision from multiple choices.

ANSWERS TO CHECK YOUR PROGRESS

Check your progress A

1. Conditional & Unconditional
2. If statement, if - else statement, nested if statement, switch statement
3. Any value other than 0.
4.

Check your progress B

1. Multiway branch statement.
2. Character, integer
3. Break, outside

6.9 SELF-ASSESSMENT QUESTIONS:

1. What is a control statement?
2. Describe decision control statements?
3. Explain nested if in detail?
4. What is the purpose of Switch case?
5. What is the difference between a switch and if decision control flow?
6. Describe goto statement with example.

6.10 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code:**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON - 7

LOOP CONTROL STATEMENT

STRUCTURE

7.0 OBJECTIVES

After reading this chapter, you should be able to:

a) Understand types of loops

b) Usage of For, While and Do-While loop statements.

c) Understand the concept of nested loops.

d) Usage of infinite loops.

7.1 LOOPS

The process of repeatedly executing a collection of statements is called as Looping. The statements are executed in sequential order generally from the first followed by second and the process continues till the end. When a block of code needs to be executed several number of times you may encounter worst situations. The looping helps in executing a statement or group of statements multiple times.

Types of Loop:

C language consists of three different loops. They are

- For Loop
- While Loop
- Do while Loop

7.2 FOR LOOP

This is used to execute set of statements in a repeat mode until a particular condition is true. This is also known as open ended tool.

Syntax

for (initialization; condition; increment/decrement)

{

statement - *block;*

}

In for loop, there exists two semicolons (;), one for initialization and the other for condition. In this loop there exists more than one initialization or increment/decrement, separated using comma operator but it can have only one condition.

The step - by - step procedure of for loop execution:

- Evaluates the initialization code
- Then checks the condition expression
- If it is true, the body of the for loop is executed
- Then increment/decrement condition is evaluated
- The loop exists if the condition statement becomes false.

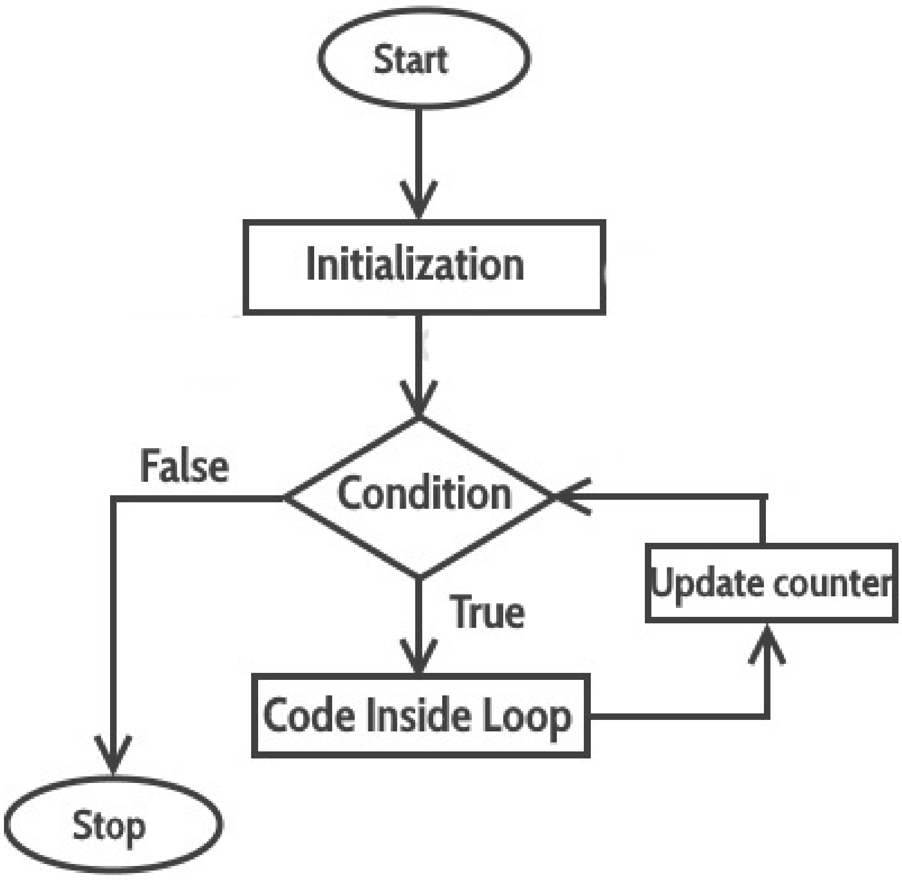


Figure 1 Flowchart for For Loop

Example

```
void main ()
{
int x;
for (x=1; x<=5; x++)
{
```

```
printf("%d\t", x);
}
}
```

Output:

```
1
2
3
4
5
```

7.3 WHILE LOOP

The while loop statement in C allows repeatedly to run the same block of statements until condition is met. This is the basic loop of C language. The while loop consists of one control condition and executes as long as the condition is true. The body of the loop is executed after the condition of the loop is executed. This is also called as an Entry controlled loop.

Syntax:

```
While (condition);
{
loop body;
increment or decrement;
```

}

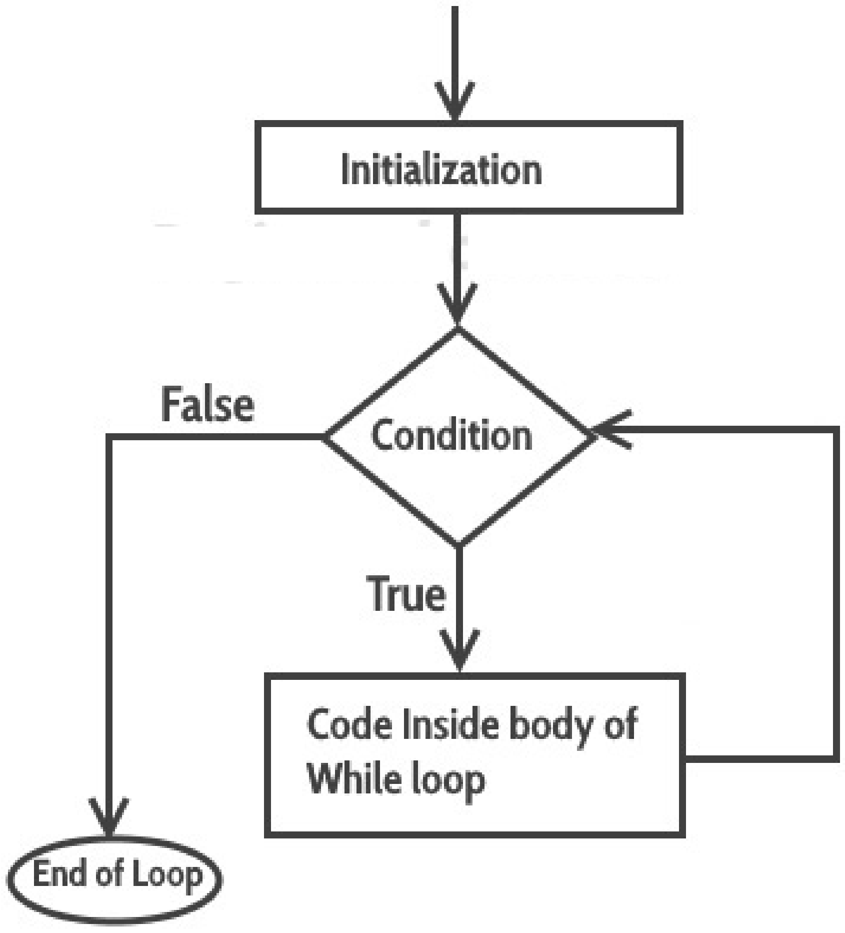


Figure 2 Flowchart for While Loop

Example

```
void main ()
{
int i;
clrscr();
i=1;
while(i<6)
{
printf ("\n%d", i);
i++;
}
getch();
}
```

Output

```
 1
 2
 3
 4
 5
```

7.4 DO-WHILE LOOP

This is similar to while loop with one execution. In this the stage the statements inside the body of loop are executed first and before checking the condition. Do-while will run once if the condition is false.

Syntax

```
do
{
while (condition test);
}
```

Example:

```
int main ()
{
int i=0;
do
{
printf(“Value of variable i is: %d\n”, i);
i++;
}
while (i<=3);
return 0;
}
```

Output

```
value of variable i is: 0
value of variable i is: 1
value of variable i is: 2
```

value of variable i is: 3

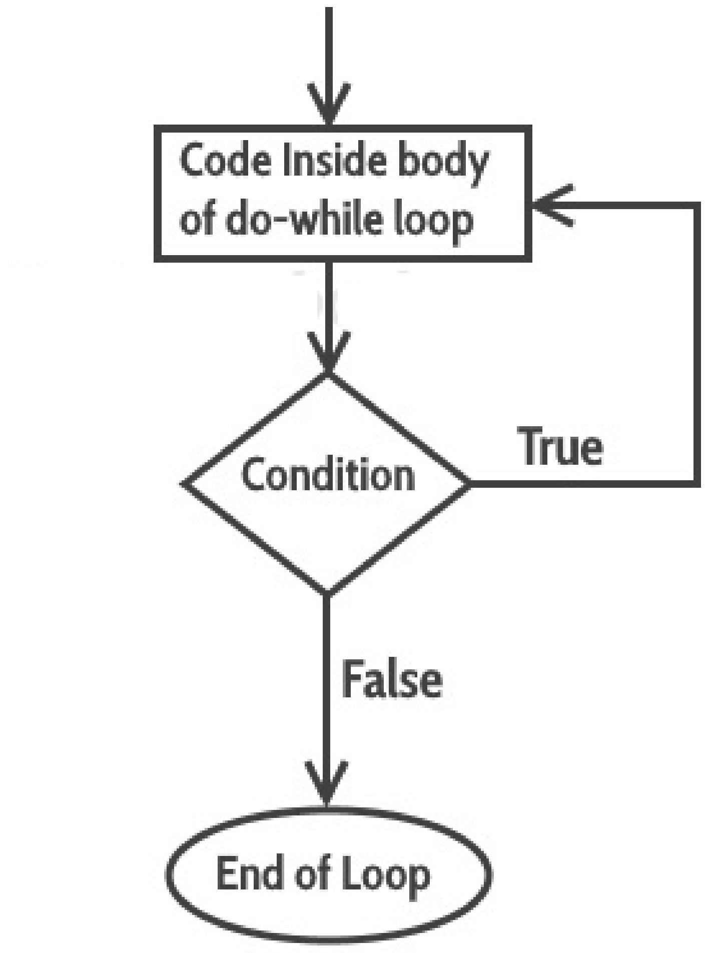


Figure 3Flowchart for Do While Loop

7.5 NESTED LOOPS

Nested loop is defined as the loop inside the loop. Depth of nested loop defines on the complexity of the program. Nested loops can be used as per the requirement of the program. Any loop can be used inside any loop. For loop can be used inside while loop, while loop can be used inside do-while loop and goes on. Nested for loops are used to cycle through matrix/tabular data and multidimensional arrays. Since a table is a matrix of rows and columns, more than one loop is required through the row then across the column. These are also used for mathematical functions where you need to do complex evaluations on data. Insert the new loop within the brackets of the first loop to nest a loop.

Types of Nested Loops:

There are three different types of nested loops. They are

- Nested For Loop
- Nested While Loop
- Nested do-while Loop

Nested For Loop:

A for loop inside the other for loop is known as Nested for loop. An inner loop within the body of an outer one. The second pass of the outer loop triggers the inner loop again. This repeats until the outer loop finishes. A break within either the inner or outer loop would interrupt this process.

Syntax:

for (initialization; condition; increment/decrement)

{

statement ();

for (initialization; condition; increment/decrement)

{

```
statement ();
….
}
….
}
```

Example

```
int main ()
{
for (int i=0; i<2; i++)
{
for (int j=0; j<2; j++)
{
printf("%d, %d\n", i, j);
}
}
return 0;
}
```

Output

```
0, 0
0, 1
0, 2
```

```
0, 3
1, 0
1, 1
1, 2
1, 3
```

Nested While Loop:

A while loop inside another while loop is defined as Nested While Loop. In a nested while loop, one iteration of the outer loop is first executed, after which the inner loop is executed. Once the condition of the inner loop is satisfied, the program moves to the next iteration of the outer loop.

Syntax

```
While (condition 1)
{
statement ();
while(condition 2)
{
statement();
….
}
….
}
```

Example

```
int main ()
{
int i=1, j=1;
while (i<=5)
{
j=1;
while (j<=5)
{
printf (“%d”, j);
j++;
}
printf(“\n”);
i++;
}
return 0;
}
```

Output

```
1 2 3 4 5
```

```
1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5
```

Nested do-while loop

A do-while loop inside another do-while loop is known as Nested do-while loop. The outer do-while loop contains the inner do-while loop as a set of statements. Though the test conditions of inner and outer do-while loops are false for the first time. Both the inner and outer statements of do-while loops are executed once irrespective of their test conditions.

Syntax

```
do
{
statement(s);
do
{
statement();
…..
}
while (condition 2)
…..
while (condition 1)
….
}
```

Example

```
int main()
{
int a;
clrscr();
a=1;
do
{
printf("%d\n",a);
a++;
}
while (a<5)
getch ();
return 0;
}
```

Output

```
1

2

3

4

5
```

Check your progress A

Give Short Answers

a. What are the different types of loops?

b. What are the different types of nested loops?

7.6 BREAK AND CONTINUE STATEMENTS

Break Statement

In C language, the break statement is used inside loops and switch case, for loop, while loops and do-while loop cases. When a break statement is executed the loop will be terminated immediately and the program control resumes at the next statement following the loop. When the nested loops are used the execution of innermost loop will be stopped by the break statement.

Syntax

break;

Example

```
int main()
{
int i=10;
while(I<20)
{
printf("value of i :  %d\n", i);
i++;
if(i>15)
{
break;
}
}
return 0;
}
```

Output

```
value of i: 10
value of i: 11
value of i: 12
value of i: 13
value of i: 14
value of i: 15
```

Continue Statement

This is a loop control statement like break statement. This is opposite to that of break statement but it forces to execute the next iteration of the loop instead of terminating the loop. The name itself defines the continue statement as it forces the loop to continue in the next iteration. When the continue statement is executed, the code inside the loop is skipped and the next iteration begins.

Syntax:

Continue;

Example

```
 int main()
{
for (int i=0; i<=6; i++)
{
if (i==3)
{
continue;
}
printf("%d",i);
}
return 0;
}
```

Output

```
0 1 2 4 5 6
```

Check your progress B

Give short answers

a. Define break statement?

b. Define continue statement?

7.7 INFINITE LOOPS

A loop which repeats endlessly and never terminates is known as infinite loop. The infinite Loops is also known as Endless Loop. Due to lack of exit condition the loop executes indefinitely and never terminates. You can intentionally or unintentionally have found infinite loops in the program. In general, infinite loop in a program either produces continuous output or does nothing as the loop is indefinite. It can become a major problem while working with infinite loops as you might not know which loops are running forever. To avoid infinite loops, each loop should be created separately and check whether it completes or not.

Example

```
int i:
for(i=0; i<20. i--)
{
printf(“%d\n”, i);
}
```

In this example, loop will execute until ‘i’ is less than 20. Initial value of ‘I’ is 0. The value of ‘i’ keeos on decreasing with every iteration and the condition *i<20* will always be true. So this loop will never terminate and keep on printing the values of ‘i’ till infinity. So it is an infinite loop. To make it a finite loop, you should use *i++* instead of *i--*.

Check your progress C

Give short Answer

a. Define infinite loops with example?

7.8 SUMMARY

In C, control statements and loops help in the execution and evaluation of the program. Various loops are discussed to execute different scenarios. For loop is used to excuse set of statements in a repeat mode until a particular condition is true. The while loop statement in C allows repeatedly to run the same block of statements until condition is met. if loop in C allows check the condition for true first before performing the necessary action. Another amazing loop case is nested loops where one loop can be nested inside another loop. When there is no proper termination condition, loop will never terminate and keeps on executing forever. Such loops are called as infinite loops.

7.9 KEYWORDS

Loop: A loop allows us to execute a block of statements sevear times.

Nested Loop: One loop can be used inside other loop. Such condition is called as nesting of loops.

Infinite Loop: A loop that goes on executing forever because of no exit condition is called as infinite loop. It never terminates.

Answers to Check Your Progress

Check your progress A

a. For loop, while loop, do-while loop

b. Nested for loop, nested while loop, nested do-while loop

Check your progress B

a. Break statement is executed the loop will be terminated immediately and the program control resumes at the next statement following the loop.

b. This forces the loop to continue the iteration, when the continue statement is executed, the code inside the loop is skipped and the next iteration begins.

Check your progress C

a. A loop which repeats forever is known as infinite loop. This is also known as Endless Loop.

7.10 Self-Assessment Questions

1. Define Loops and different types of loops?
2. Describe Nested loops and different types of nested loops?
3. Describe break, continue statement?
4. Describe infinite loops?

7.11 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**: M.Sc. (Mathematics)**	**Course Code**	**: MAL-645**
Subject	**: Programming in C**		

LESSON-8

ARRAYS

STRUCTURE

8.0 OBJECTIVES

After reading this lesson, you should be able to:

(a) Define the arrays and how they are initialized.

(b) Explain the concept of encryption and decryption.

(c) Define multidimensional arrays.

(d) How a string can be stored in array?

8.1 INTRODUCTION

In various situation, we need to combine similar objects into a group and this group is referred by a common name. For this purpose, C supports arrays. An array is a data structure which contains a set of homogenous elements and these elements can be accessed by their specific position in the group. For example, a list of 50 employees in an organization can be grouped as shown in Fig. 8.1.

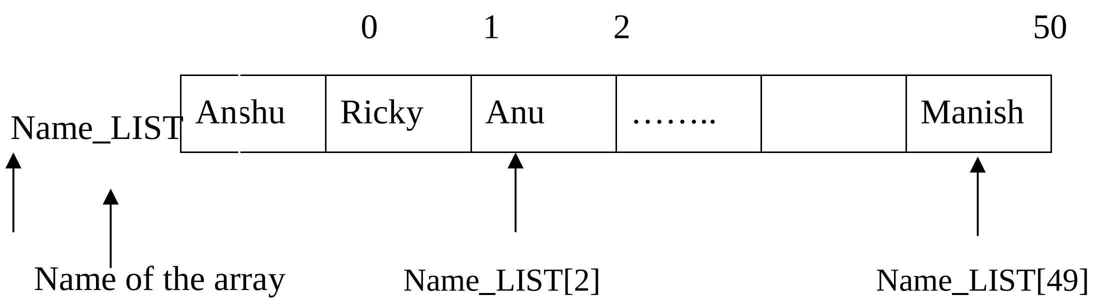


Figure 1 Schematic Representation of an array

Figure 1 shows the list of employees called Name_LIST(Name of an array) which is a set of 0 to 49 memory locations. These memory locations share a common name i.e. Name_LIST. Each element within the array can be designated by an integer known as an index number of subscript. For example the 1st name in the list will be referred to as Name_LIST[0],2nd name in list will be referred to as Name_LIST[1] and so on.

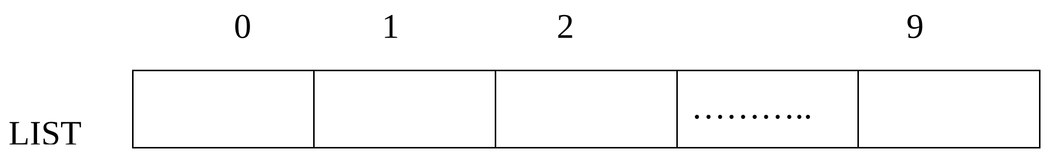
There are two types of arrays namely one dimensional arrays and multi-dimensional arrays. In one dimensional array, each element is specified by a single index or subscript. This type of array is suitable for representing the list of items of similar types. An array which has more than one subscript is known as multidimensional array. This type of array is suitable for table processing or matrix multiplications.

8.2 DECLARING AN ARRAYS

An array can be declared in C just like any other variable. It contains type, array name and size. For example an array called LIST of 10 cities of integer type can be declared as shown below:

int LIST [10];

The above declaration means that LIST is an array of 10 memory locations, each of which is of integer type.



It may be noted that the first element in array has subscript 0 in C language. Examples of valid array declarations are:

(i) int roll-no[50];

(ii) char names[20];

(iii) float sal[50];

Example: Write a program to print the number of entered values into a one dimensional array

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i, number[5];
clrscr();
printf("Enter 5 number \n");
for(i=0;i<5;i++)
scanf("%d", &numbers[i]);
printf("Array elements are \n");
for(i=0;i<5;i++)
printf("%d\n",numbers[i]);
getch();
return 0;
}
```

Output :

```
4
6
```

```
7
3
2
Array elements are
4
6
7
3
2
```

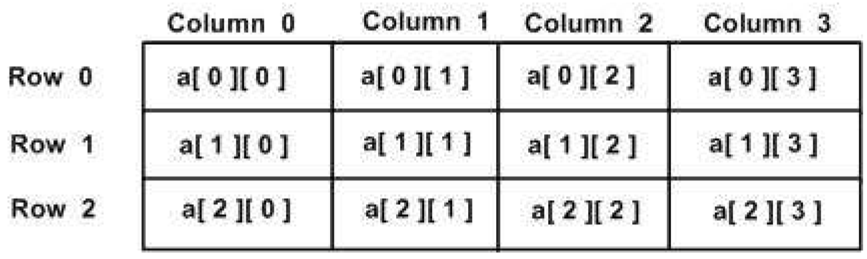
Example: Write a program which reads a list of five numbers and prints the list in reverse order.

```
#include <stdio.h>
main()
 {
int list[10];
int n, i;
printf("enter the number of elements in the list\n");
scanf("%d", &n);
printf("enter the elements\n");
 for (i=0;i<n;i++)
{
printf(" enter elements\n");
scanf("%d", &list[i]);
}
printf("the list in reverse order\n");
printf("%d", list[i]);
}
```

Multidimensional arrays have more than one subscript. C supports arrays of arbitrary dimensions. For example, a two dimensional array of 5 rows and 4 columns of integer type can be declared as given below. Let us assume that the name of the array is mat.

int mat[5][4];

It may be noted that the above declaration is 2-dimensional array which has 5 rows and 4 columns. Another declaration of 2-dimensional ,which contains three rows and four columns can be shown as follows −



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the form a[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

8.3 ARRAYS AND MEMORY

Consider the following array declaration:-

int arr[8];

8	45	67	89	34

What happens in memory when we make this declaration? 16 bytes get immediately reserved in memory, 2 bytes for each of the 8 integer. Since the array is not being initialized, all 8 values present in it would be garbage values. This so happens because the storage class of this array is assumed to be auto. If the storage class is declared to be static then all the array elements would have a default initial value as zero. Whatever, be the initial values, all the array elements would always be present in contiguous memory locations. This arrangement of array element in memory is shown below:

65500 65502 65504 65506 65508

8.4 INITIALIZING ARRAYS

Arrays may be initialized when they are declared, just as any other variables. Place the initialization data in curly {} braces following the equals sign. Note the use of commas in the examples below. An array may be partially initialized, by providing fewer data items than the size of the array. The remaining array elements will be automatically initialized to zero.

If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data. Examples of initializing one dimensional arrays are as shown below:

int i = 5, intArray[6] = { 1, 2, 3, 4, 5, 6 }, k;

float sum = 0.0f, floatArray[100] = { 1.0f, 5.0f, 20.0f };

double piFractions[ ] = { 3.141592654, 1.570796327, 0.785398163 };

Examples of initializing two dimensional arrays are as shown below:

int a[2][3] = { { 5, 6, 7 }, { 10, 20, 30 } };

int b[2][3] = { { 1, 2, 3 }, { 3, 2, 1 } };

Check Your Progress A

Fill in the blanks

1. Arrays are stored in ……….locations.
2. Arrays have……… types elements.
3. Arrays can be of …….dimension or ………

8.5 ENCRYPTION AND DECRYPTION

Encryption is the process of encoding a message or information in such a way that only authorized parties can access it and those who are not authorized cannot. Encryption does not itself prevent interference, but denies the intelligible content to a would-be interceptor. In an encryption scheme, the intended information or message, referred to as plaintext, is encrypted using an encryption algorithm – a cipher text. Decryption is the process of transforming data that has been rendered unreadable through encryption back to its unencrypted form. In decryption, the system extracts and converts the garbled data and transforms it to texts and images that are easily understandable not only by the reader but also by the system. Decryption may be accomplished manually or automatically. It may also be performed with a set of keys or passwords. AES(advanced encryption standard), DES(digital encryption standard),RSA etc. are some examples of encryption and decryption algorithms. We can implement various encryption and decryption methods using c. Here is an example to encrypt and decrypt to password as shown below:

Example: Write a program to encrypt and decrypt a password.

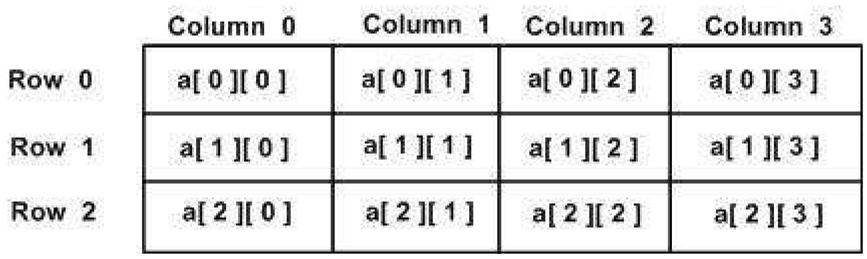
```
#include <stdio.h>
#include <string.h>
void encrypt(char password[],int key)
{
unsigned int i;
for(i=0;i<strlen(password);++i)
{
password[i] = password[i] - key;
}
}
void decrypt(char password[],int key)
{
unsigned int i;
for(i=0;i<strlen(password);++i)
{
 password[i] = password[i] + key;
}
}
int main()
{
char password[20] ;
printf("Enter the password: n ");
scanf("%s",password);
printf("Passwrod     = %sn",password);
encrypt(password,0xFACA);
printf("Encrypted value = %sn",password);
decrypt(password,0xFACA);
printf("Decrypted value = %sn",password);
return 0;
}
```

8.6 MULTIDIMENSIONAL ARRAYS

Multidimensional arrays have more than one subscript. C supports arrays of arbitrary dimensions. For example, a two dimensional array of 5 rows and 4 columns of integer type can be declared as given below. Let us assume that the name of the array is mat.

int mat[5][4];

It may be noted that the above declaration is 2-dimensional array which has 5 rows and 4 columns. Another declaration of 2-dimensional ,which contains three rows and four columns can be shown as follows –



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Figure 2 Multidimensional Array

Thus, every element in the array a is identified by an element name of the form a[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Example: Write a program to add two multidimensional arrays.

```
#include <stdlib.h>

#include <stdio.h>

int main( void )

{

int a[ 2 ][ 3 ] = { { 5, 6, 7 }, { 10, 20, 30 } };

int b[ 2 ][ 3 ] = { { 1, 2, 3 }, {  3,  2,  1 } };

int sum[ 2 ][ 3 ], row, column;
/* First the addition */
for( row = 0; row < 2; row++ )
for( column = 0; column < 3; column++ )
sum[ row ][ column ] =
 a[ row ][ column ] + b[ row ][ column ];

/* Then print the results */
printf( "The sum is: \n\n" );
for( row = 0; row < 2; row++ ) {
for( column = 0; column < 3; column++ )
printf( "\t%d",  sum[ row ][ column ] );
printf( '\n' );   /* at end of each row */
}
return 0;
}
```

8.7 SUMMARY

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. Complier does not perform any bound checking on an array. The array variable acts as a pointer to the zeroth element of the array. In one dimensional array, zeroth element is a single value, whereas in two dimensional array, this element is a one dimensional array. Array elements are stored in contiguous memory locations and they can be accessed using pointers.

8.8 KEYWORDS

Array: An array is a set of homogenous elements.

One Dimensional Array: An array whose elements are specified by a single subscript.

Two-Dimension Array: An array whose elements are specified by more than one **Subscript:**

String: A string is a group of characters of any length.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. Continuous
2. Same or homogeneous
3. Single or multidimensional

8.9 SELF ASSESSMENT QUESTIONS

1. Why an array is called a derived data type?
2. Can an array is assigned to another of same type and size?

3. What would be contents of the following array after initialization?

   int A[5]= {4,5,6};

4. Write a program to copy the contents of an array into the another in a reverse array.

5. Write a program to perform the following operations on 2-dimensional arrays

   (a) Addition (b) Multiplication

8.10 SUGGESTED READINGS

A.K. Sharma, Fundamentals of Computers & Programming with C, Dhanpat Rai Publications, New Delhi.

Yashavant P. Kanetkar, Let Us C, Bpb Publications, New Delhi.

Class	**:**	**M.Sc. (Mathematics)**	**Course Code :**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON-9

FUNCTIONS

STRUCTURE

9.0 OBJECTIVES

After reading this lesson, you should be able to:

(a) Define the functions and how they are initialized.

(b) Explain the concept of encryption and decryption.

(c) Define multidimensional arrays.

(d) How a string can be stored in array?

9.1 INTRODUCTION

Sometime we need to perform same set of operations at many different places in a program then the programmers write the same set of instructions as many times as it is required as shown below:

```
scanf(“ enter the values of base and height\n”);
scanf(“%d%d”,&base, &height);
area=1/2*base*height;
scanf(“ enter the values of base and height\n”);
scanf(“%d%d”,&base, &height);
area=1/2*base*height;
```

It would be wasteful of programmer’s time and effort to code the sequence of instructions every time it is required. Therefore a better way to achieve the same effect, however would be to use subprograms. A subprogram is a name given to a set of instructions that can be called by another program or a subprogram.

A function is a block of statements that performs a specific task. Suppose we are building an application in C language and in one of our program, we need to perform a same task more than once. In such case you have two options –

a) Use the same set of statements every time you want to perform the task
b) Create a function to perform that task, and just call it every time you need to perform that task.

Using option (b) is a good practice and a good programmer always uses functions while writing codes in C.

Functions are divided into two categories.

Functions	
Library Functions	User Defined Functions

Figure 1 Types of Functions

Predefined standard library functions – such as puts(), gets(), printf(), scanf() etc – These are the functions which already have a definition in header files (.h files like stdio.h), so we just call them whenever there is a need to use them.

User Defined functions – The functions that we create in a program are known as user defined functions. Syntax of a function as shown below:

return_type function_name (argument list)

{

Set of statements – Block of code

}

return_type: Return type can be of any data type such as int, double, char, void, short etc. Don’t worry you will understand these terms better once you go through the examples below.

function_name: It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it’s name.

argument list: Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A function which is used to add two integer variables, will be having two integer argument.

Block of code: Set of C statements, which will be executed whenever a call will be made to the function.

Functions are used because of following reasons –

a) To improve the readability of code.

b) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.

c) Debugging of the code would be easier if you use functions, as errors are easy to be traced.

d) Reduces the size of the code, duplicate set of statements are replaced by function calls

9.2 PASSING ARGUMENTS

In C, We can pass arguments in two ways:-

Call by Value: In the call by value method, value of the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn't affect actual parameters. It only performs the operation on formal arguments.

Call by Reference: Unlike call by value, in this method, address of actual arguments (or parameters) is passed to the formal parameters, which means any operation performed on formal parameters affects the value of actual parameters.

Function call by value is the default way of calling a function in C programming. Before we discuss function call by value, lets understand the terminologies that we will use while explaining this:

Actual parameters: The parameters that appear in function calls.

Formal parameters: The parameters that appear in function declarations.

Example

```
#include <stdio.h>

int sum(int a, int b)

{

int c=a+b;

return c;

}

int main(

{
```

```
int var1 =10;
int var2 = 20;
int var3 = sum(var1, var2);
printf("%d", var3);
return 0;
}
```

In the above example variable a and b are the formal parameters (or formal arguments). Variable var1 and var2 are the actual arguments (or actual parameters). The actual parameters can also be the values. Like sum(10, 20), here 10 and 20 are actual parameters.

Example of Function call by Value

As mentioned above, in the call by value the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn’t affect actual parameters. Lets take an example to understand this:

```
#include <stdio.h>
int increment(int var)
{
var = var+1;
return var;
}
```

```
int main()
int num1=20;
int num2 = increment(num1);
printf("num1 value is: %d", num1);
printf("\nnum2 value is: %d", num2);
return 0;
}
```

Output

```
num1 value is: 20
num2 value is: 21
```

We passed the variable num1 while calling the method, but since we are calling the function using call by value method, only the value of num1 is copied to the formal parameter var. Thus change made to the var doesn’t reflect in the num1.When we call a function by passing the addresses of actual parameters then this way of calling the function is known as **call by reference**. In call by reference, the operation performed on formal parameters, affects the value of actual parameters because all the operations performed on the value stored in the address of actual parameters. It may sound confusing first but the following example would clear your doubts.

Example of Function call by Reference

```
#include <stdio.h>
```

```
void increment(int  *var)
{
/* Although we are performing the increment on variable var, however the var is a pointer that holds the address of variable num, which means the increment is actually done on the address where value of num is stored. */
*var = *var+1;
}
int main()
{
int num=20;
increment(&num);
printf("Value of num is: %d", num);
return 0;
}
```

Output

```
Value of num is: 21
```

Table 1 Difference Between Call by Value and Call By Reference

Call By Value	Call By Reference
It copies the original value of the argument	It passes address of the arguments
Changes made to the parameter are not reflected in the original parameter.	Changes made to the parameters are reflected in the original parameter.
Actual and formal parameters are created in different memory locations.	Actual and formal arguments are created in same memory.

Check Your Progress A

Fill in the blanks

1. In call by value, we pass the ……….. of argument.
2. In call by reference, we pass the ……… of argument.
3. Library functions are included in C program using……………

9.3 DECLARATION AND CALLS

Lets take an example – Suppose you want to create a function to add two integer variables.Function will add the two numbers so it should have some meaningful name like sum, addition, etc. For example lets take the name addition for this function.

return_type addition(argument list)

This function addition adds two integer variables, which means I need two integer variable as input, lets provide two integer parameters in the function signature. The function signature would be –

return_type addition(int num1, int num2)

The result of the sum of two integers would be integer only. Hence function should return an integer value.

int addition(int num1, int num2);

So you got your function prototype or signature. Now you can implement the logic in C program like this:

How to call a function in C?

Example: Creating a user defined function addition()

```
#include <stdio.h>
int addition(int num1, int num2)
{
int sum;
/* Arguments are used here*/
sum = num1+num2;
return sum;
}
int main()
{
 int var1, var2;
 printf("Enter number 1: ");
 scanf("%d",&var1);
 printf("Enter number 2: ");
 scanf("%d",&var2);
 int res = addition(var1, var2);
 printf ("Output: %d", res);
 return 0;
}
```

Output

```
Enter number 1: 100
Enter number 2: 120
Output: 220
```

9.4 RECURSION

Recursion is a programming technique that allows the programmer to express operations in terms of themselves. In C, this takes the form of a function that calls itself. A useful way to think of recursive functions is to imagine them as a process being performed where one of the instructions is to "repeat the process".

```
void recursion()
{
recursion(); /* function calls itself */
}
int main()
{
}
```

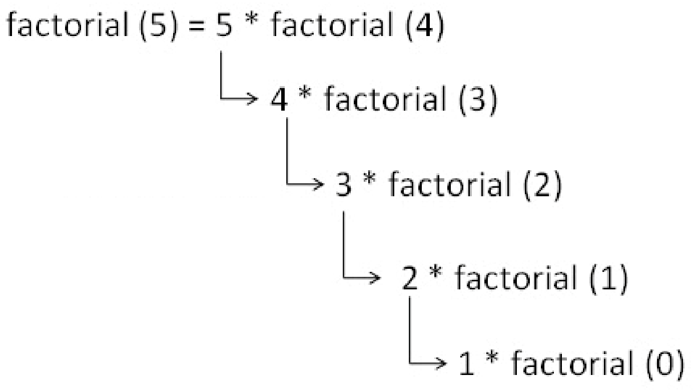


Figure 2 Recursive Function for Factorial

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

The following example calculates the factorial of a given number using a recursive function –

```
#include <stdio.h>
unsigned long long int factorial(unsigned int i)
```

```
 {
 if(i <= 1)
{
return 1;
 }
 return i * factorial(i - 1);
}
int  main()
{
int i = 12;
printf("Factorial of %d is %d\n", i, factorial(i));
return 0;
}
```

Output

```
Factorial of 12 is 479001600
```

9.5 THE MAIN () FUNCTION

All C language programs must have a main() function. It’s the core of every program. It’s required. The main() function doesn’t really have to do anything other than be present inside your C source code. Eventually, it contains instructions that tell the computer to carry out whatever task your program is designed to do. But it's not officially required to do anything.

When the operating system runs a program in C, it passes control of the computer over to that program. This is like the captain of a huge ocean liner handing you the wheel. Aside from any fears that may induce, the key point is that the operating system needs to know where inside your program the control needs to be passed. In the case of a C language program, it's the main() function that the operating system is looking for.

At a minimum, the main() function looks like this:

main()

{

}

Like all C language functions, first comes the function's name, main, then comes a set of parentheses, and finally comes a set of braces, also called curly braces. If your C program contains only this line of code, you can run it. It won't do anything, but that's perfect because the program doesn't tell the computer to do anything. Even so, the operating system found the main() function and was able to pass control to that function — which did nothing but immediately return control right back to the operating system. It's a perfect, flawless program.

9.6 PASSING ARRAYS AS FUNCTION ARGUMENT

As we already know in this type of function call, the actual parameter is copied to the formal parameters.

```
#include <stdio.h>
void disp( char ch)
{
        printf("%c ", ch);
}
int main()
{
char arr[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
for (int x=0; x<10; x++)
{
/* I’m passing each element one by one using subscript*/
disp (arr[x]);
}

 return 0;
}
```

Output

```
a b c d e f g h i j
```

Passing array to function using call by reference

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointeras a parameter to receive the passed address.

```
#include <stdio.h>
void disp( int *num)
{
printf("%d ", *num);
}

int main()
{
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
for (int i=0; i<10; i++)
{
/* Passing addresses of array elements*/
disp (&arr[i]);
}
return 0;
}
```

Output

```
1 2 3 4 5 6 7 8 9 0
```

Check Your Progress B

Fill in the blanks

1. Recursion is calling the ……… function again and again.
2. The default return type of main function is ………..

9.7 SUMMARY

A function is a block of statements that performs a specific task. Functions are used to improve the readability of code, Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch, debugging of the code would be easier if you use functions, as errors are easy to be traced and reduces the size of the code, duplicate set of statements are replaced by function calls. In C language, there are two types of functions namely, **predefined standard library functions** (such as puts(), gets(), printf(), scanf() etc) are the functions which already have a definition in header files ,so we just call them whenever there is a need to use them and **user defined functions** create in a program are known as user defined functions.

9.8 KEYWORDS

Function: A function is a subprogram that can be defined by the user in his program.

Scope: The range of code in a program over which a variable has a meaning is called as scope of the variable.

Formal Parameters: The set of parameters defined in a function are called formal parameters.

Actual Parameters: The set of corresponding parameters sent by the calling function are called actual parameters.

Function Prototype: C allows the declaration of a function in the calling program with the help of a function prototype.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. value
2. address
3. preprocessor directives

Check Your Progress B

1. same
2. int

9.9 SELF ASSESSMENT QUESTIONS

1. Explain in brief local scope.
2. What is meant by a register variable?
3. Explain in brief the functions that return a value.

4. What is meant by a function prototype and why it is needed?

5. What happens to the two parameters used by the following function

   void main(int &x, int y);

6. What happens when a function has more than one return statements?

9.10 SUGGESTED READINGS

A.K. Sharma, Fundamentals of Computers & Programming with C, Dhanpat Rai Publications, New Delhi.

Yashavant P. Kanetkar, Let Us C, Bpb Publications.

E. Balagurusamy, Programming in ANSI C, Tata Mcgraw Hill.

Class	**:**	**M.Sc. (Mathematics)**	**Course Code :**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON-10

STRINGS

STRUCTURE

10.0 OBJECTIVES

After reading this lesson, you should be able to:

(a) Understand how strings are stored in memory

(b) Understand various string functions

10.1 STRINGS

Strings are one-dimensional array of characters. The way integers are stored in an integer array; characters are stored in a character array. Character arrays, which are terminated by a null character '\0' are called as strings. The ‘\0’ looks like two characters but it is actually one. It is different form ‘0’. ASCII value of ‘\0’ is 0 whereas ASCII value of ‘0’ or zero is 48. Elements of arrays are stored in contiguous memory locations. The null character ‘\0’ is important as it is the way to tell where a string ends. If there is no null character, then it is termed as array of characters only and not the string.

Thus, it is necessary to insert the NULL character at the end of the string, while initializing the string. The following declaration and initialization create a string consisting of the word "Hello".

To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "World"

char name[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization, then you can write the above statement as follows:

char name[6] = "Hello";

In this case, null character is added automatically. The only condition is that you need to declare the size of array one larger than the length of string.

The memory representation of the above-defined string in C is:

Index	0	1	2	3	4	5
Variable	H	e	l	l	0	\0
Address	0X20	0X21	0X22	0X23	0X24	0X25

To use strings, you need to include the file string.h from C library. The format specifier for string is %s.

To print the above-mentioned string, Lets see an example-

```
#include<string.h>
#include<stdio.h>
int main ()
{
char name[6] = {'H', 'e', ‘l', 'l', 'o', '\0'};
printf("I am greeting you: %s\n", name );
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

Output

```
I am greeting you: Hello
```

10.2 LIMITATION OF STRINGS

While entering the string using the scanf( ) function, you need to take care that scanf( ) is not capable of receiving multi-word strings. So, names like ‘Mohan

Singh’ would not be accepted. The pressing of space bar would automatically terminate the string. So, %s can be used to read only one word at a time.

Example

```
#include<string.h>
#include<stdio.h>
int main ()
{
char name[16];
printf("Enter your name-”);
scanf(“%s”, name);
printf(“\nYour name is %s”, name);
return 0;
}
```

Output

```
Enter your name- Mohan Singh
Your name is Mohit.
```

In the example above, only Mohan has been accepted as string. Because as soon as we enter the space bar, scanf function terminates the string. C provides two another standard functions gets() and puts() for multiword strings. The function gets() is used to receive multiword strings from the keyboard and puts() is used to print multiword strings on the screen.

Another limitation of a string function is you cannot assign the value of one string to another string. Whereas, value of one character can be assigned to another.

The following code will produce en error-

Char str1[] = “Hello”;

Str1 = “world”;   / error */*

Once defined a string, it cannot be initialized to another values.

Check Your Progress A

1. A string is terminated by ……….
2. While counting the length of string the null character is ………..
3. Function scanf can accept …… strings.
4. Function gets can accept …… strings.

10.3 STRING FUNCTIONS

C language supports a wide range of functions to manipulate the strings. C library has large set of useful string handling functions. The header file string.h is included in a C program to use these functions. Some of these functions are discussed here.

a. strlen(string)

This function counts the number of characters in a string and thus gives us the length of string. We pass the base address of the string to strlen( ) and it returns the length of string. It keeps on counting till it gets a null character and doesn’t count the null character.

Example

```
#include <stdio.h>

#include <string.h>

int main( )

{

char a[15]=”Apple”;

char b[15]={‘A’, ‘p’, ‘p’, ‘l’, ‘e’};

printf(“Length of string a=%d\n”, strlen(a));

printf(“Length of string b=%d\n”, strlen(b));

return 0;

}
```

Output

```
Length of string a =5

Length of string b=5
```

b. strcpy(string2, string1)

This function copies the contents of one string into another. The base address of target and source strings are supplied to the function. It copies the string pointed by source_str to target_str.

Example

```
#include <stdio.h>
```

```
#include <string.h>

int main( )

{

char source[15]=”Apple”;

char target[15];

strcpy(target, source);

printf(“Source string =%s\n”, source);

printf(“Target string =%s\n”, target);

return 0;

}
```

Output

```
Source String = Apple

Target String = Apple
```

c. **strcat(string2, string1)**

This function appends the source string at the end of target string. A string, string2 is appended at the end of another string, string1. The target string should have enough space allocated to hold the source string.

```
#include <stdio.h>

#include <string.h>

int main( )

{
```

```
char source[15]= “Hello”;

char taregt[15]= “Friends”;

strcat(target, source);

printf(“Source string =%s\n”, source);

printf(“Target string =%s\n”, target);

return 0;

}
```

Output

```
Source String = Hello

Target String = HelloFriends
```

d. strcmp(string2, string1)

This function compares two strings two find whether they are same or not. The strings are compared character by character until there is a mismatch or end of one of the string, whichever occurs first. The function returns a zero value if the two strings are identical. If they are not same, the difference is shown by the ASCII values of first non-matching character. The function of this string can be shown by below example.

Example

```
#include <stdio.h>

#include <string.h>

int main( )

{
```

```
char string1[15]= “Fruits”;

char string2[15]= “Friends”;

int a,b;

a=strcmp(string1,“Fruits”);          /* Fruits is compared to Fruits */

b=strcmp(string1, string2) ;         /* Fruits is compared to Friends */

printf(“%d\n”, a);

printf(“%d”,b);

return 0;

}
```

Output

```
0

12
```

There are some other string functions also, which are briefly described in the table-

Table 1 String Functions in C

Function	**Use**
strlen	Length of string
strlwr	Converts string to lower case
strupr	Converts string to upper case
strcat	Appends one string at the end of another

strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strcmpi	Compares two strings, ignoring the case
strnicmp	Compares first n characters of two strings, ignoring the case
strdup	Duplicates a string
strchr	Find first occurrence of a given string in a string
strrchr	Find last occurrence of a given string in another string
strstr	Find first occurrence of a given string in another string
strset	Set all characters of a given string to a given character
strnset	Set first n characters of a string to a given character
strrev	Reverses a string

Check Your Progress B

Give short answer for the following-

1. What is the function of strlen function?
2. Differentiate between strcmp and strcmpi.
3. Which string function is used to reverse the characters of a string?

4. Which string function is used to compare two strings without matching the case?

10.4 ARRAY OF STRINGS

Array of strings is two-dimensional array of characters. The declaration of two-dimensional array of characters is-

Char array_name[row][col];

Here, array_name is the name of the array to store the strings, ‘row’ is the number of strings to be stored and ‘col’ contains the number of columns reserved for each string.

Example

Char months[12][10];

Here, months is the array to store the name of the twelve months and 10 is the columns reserved for each month name. You can initialize this array in the following way-

Char months[12][4]= {“jan”, “feb”, “mar”, “apr”, “may”, “jun”, “jul”, “aug”, “sep”, “oct”, “nov”, “dec”};

You can display the contents of array of strings with the following syntax-

```
for(i=0; i<=11, i++)
        printf(“\n %s”, months[i]);
```

To read the strings from keyboard, the following syntax can be used-

```
for(i=0; i<=11, i++)
```

scanf("\n %s", months[i]);

10.5 SUMMARY

String is a collection of characters where the last character is null. As it is an array, all the characters are stored in contiguous memory locations. There are various operations that can be performed on the string. Various built in functions have ben defined in C library. To use these functions, the header file string.h should be included in a C program. Array of strings can also be defined the way same like a two-dimensional array.

10.6 KEYWORDS

String: A string is an array of characters terminated by a Null character, '\0'.

String Functions: There are various operations that can be performed o the string to manipulate them with the help of inbuilt string functions of string.h header file.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. null character '\0'
2. not counted
3. single-word
4. multi-word

Check Your Progress B

1. The function strlen is used to count the characters in a string.
2. The function strcmp is used to compare two strings without ignoring the case while, strcmpi is used to compare two strings with ignoring the case.
3. strrev( )
4. strcmpi( )

10.7 SELF-ASSESSMENT QUESTIONS

1. Define String.
2. Discuss following string functions-
   strlen( )
   strcat( )
   strcmp( )
   strcpy( )
3. What do you mean by array of strings.
4. Write limitation of a string.

10.8 SUGGESTED READINGS

1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, India
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill
3. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

Class	**:**	**M.Sc. (Mathematics)**	**Course Code :**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON-11

POINTERS

STRUCTURE

11.0 OBJECTIVES

After reading this lesson, you should be able to:

(a) Define pointers.

(b) Define memory bleeding.

(c) Define pointer to pointer.

(d) What are the advantages and disadvantages of pointers?

11.1 INTRODUCTION

A pointer is a variable that stores the address of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable. Pointers are more powerful features of 'C'. The beginners of 'C' find pointers hard to understand and manipulate. Let us assume that y is such a variable. Now, the address of variable x can be assigned to y by the statement: y=&x.

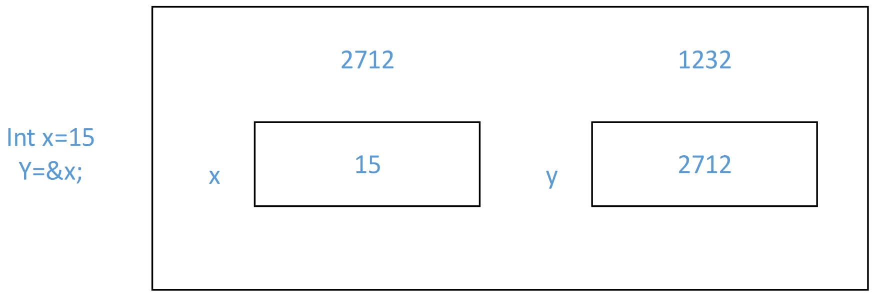


Figure 1 Pointer in Memory

From the above illustration given above, it is clear that y is the variable that contains the address of another variable x, whereas, the address of y itself is 1232. A pointer variable y can be declared in C as:

*int *y;*

The above declaration means that y is a pointer to a variable of type int. Consider the following program segment:

```
# include<stdio.h>
main()
{
int x=15;    int *y;
y=&x;
printf("\n Value of x=%d", x);
printf("\n Address of x=%u", &x);
printf("\n Value of x=%d", y);
printf("\n Address of x=%u", y);
printf("\n Address of y=%u", &y);
}
```

Output

```
Value of x=15
Address of x=2712
Value of x=15
Address of x=2712
Address of y=1232
```

11.2 POINTER ARITHMETIC

A pointer in C is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value.

There are four arithmetic operators that can be used on pointers: ++, --, +, and -

To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer −

```
ptr++
```

After the above operation, the ptr will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location. This operation will move the pointer to the next memory location without impacting the actual value at the memory location. If ptr points to a character whose address is 1000, then the above operation will point to the location 1001 because the next character will be available at 1001.

Incrementing a Pointer

We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer. The following program increments the variable pointer to access each succeeding element of the array –

```
#include <stdio.h>

const int MAX = 3;

int main ()

 {

int  var[] = {10, 100, 200};
int  i, *ptr;

/* let us have array address in pointer */

ptr = var;

for ( i = 0; i < MAX; i++) {

printf("Address of var[%d] = %x\n", i, ptr );

printf("Value of var[%d] = %d\n", i, *ptr );

/* move to the next location */

ptr++;

}

return 0;

}
```

Output

```
Address of var[0] = bf882b30

Value of var[0] = 10

Address of var[1] = bf882b34

Value of var[1] = 100

Address of var[2] = bf882b38

Value of var[2] = 200
```

Decrementing a Pointer

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below –

```
#include <stdio.h>

const int MAX = 3;

int main ()

{

int  var[] = {10, 100, 200};

int  i, *ptr;

/* let us have array address in pointer */

ptr = &var[MAX-1];

for ( i = MAX; i > 0; i--) {

printf("Address of var[%d] = %x\n", i-1, ptr );

printf("Value of var[%d] = %d\n", i-1, *ptr );

/* move to the previous location */

ptr--;

}

return 0;

}
```

Output

```
Address of var[2] = bfedbcd8

Value of var[2] = 200

Address of var[1] = bfedbcd4

Value of var[1] = 100

Address of var[0] = bfedbcd0

Value of var[0] = 10
```

11.3 ACCESSING ARRAY ELEMENT USING POINTERS

Pointers and arrays are very closely related to each other. In C the name of an array is a pointer that contains the base address of the array. Infact, it is a pointer to the first element of the array. Consider the array declaration given below:

```
int list={20,30,35,36,39};
```

This array will be stored in the contiguous locations of the main memory. Consider the following program segment-

```
printf("\n Address of Zeroth element of arraylist= ",list);

printf("\n Value of Zeroth element of arraylist= ",list);
```

Output

```
Address of Zeroth element of arraylist= 1001
Value of Zeroth element of arraylist= 20
```

We could have achieved the same output also by the following program segment.

```
printf("\n Address of Zeroth element of arraylist=",  &list[0]);
printf("\n Value of Zeroth element of arraylist=",  list[0]);
```

Both the above given approaches are equivalent because of the following equivalent relations:

```
list= &list[0]
*list=list[0]
```

The first approach is known as pointer method and second as array indexing method.

Check Your Progress A

Give short answers

1. Define pointers.
2. Which arithmetic operations can be performed on pointers?
3. What are the ways to access the elements of arrays?
4. Which type of values are stored in pointers?

11.4 PASSING POINTERS AS FUNCTION ARGUMENTS

C programming allows passing a pointer to a function. To do so, simply declare the function parameter as a pointer type.

Following is a simple example where we pass an unsigned long pointer to a function and change the value inside the function which reflects back in the calling function –

```
#include <stdio.h>

#include <time.h>

 void getSeconds(unsigned long *par);

int main () {

unsigned long sec;

getSeconds( &sec );

/* print the actual value */

printf("Number of seconds: %ld\n", sec );

return 0;

}

void getSeconds(unsigned long *par) {

/* get the current number of seconds */

*par = time( NULL );

return;

}
```

Output

```
Number of seconds :1294450468
```

The function, which can accept a pointer, can also accept an array as shown in the following example-

```
#include <stdio.h>
/* function declaration */
double getAverage(int *arr, int size);
int main () {
/* an int array with 5 elements */
int balance[5] = {1000, 2, 3, 17, 50};
double avg;
/* pass pointer to the array as an argument */
avg = getAverage( balance, 5 ) ;
/* output the returned value  */
printf("Average value is: %f\n", avg );
return 0;
}
double getAverage(int *arr, int size) {
int  i, sum = 0;
double avg;
for (i = 0; i < size; ++i) {
sum += arr[i];
}
avg = (double)sum / size;
return avg;
}
```

Output

```
Average value is: 214.40000
```

11.5 ARRAYS OF POINTERS

Before we understand the concept of arrays of pointers, let us consider the following example, which uses an array of 3 integers –

```
#include <stdio.h>
const int MAX = 3;
int main () {
int  var[] = {10, 100, 200};
int i;
for (i = 0; i < MAX; i++) {
printf("Value of var[%d] = %d\n", i, var[i] );
}
return 0;
}
```

Output

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer −

```
int *ptr[MAX];
```

It declares ptr as an array of MAX integer pointers. Thus, each element in ptr, holds a pointer to an int value. The following example uses three integers, which are stored in an array of pointers, as follows −

```
#include <stdio.h>
const int MAX = 3;
int main () {
int  var[] = {10, 100, 200};
int i, *ptr[MAX];
for ( i = 0; i < MAX; i++) {
ptr[i] = &var[i]; /* assign the address of integer. */
}
for ( i = 0; i < MAX; i++) {
printf("Value of var[%d] = %d\n", i, *ptr[i] );
}
return 0;
}
```

Output

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

11.6 COMPLEX DECLARATION

You can have a pointer point to another pointer that points to the target value. The situation is called pointers to pointers or multiple indirection. These have a complex declaration. In case of multiple indirection, value of a normal pointer is the address of the object that contains the desired value and in case of pointer to pointer , the first pointer contains the address of the second pointer, which points to the object that contains the desired value. At most, 12 levels of indirection can be used in C for pointers.

To access the target value indirectly pointed to by a pointer to a pointer i.e. 2 level of indirection, you must apply the asterisk operator twice, as in this example.

```
#include <stdio.h>

int main(void)

{

int x,*p,**q;

x=10;,p=&x;

q=&p;

printf(“%d”,**q);

return 0;

}
```

Here P is declared as a pointer to an integer and q as a pointer to a pointer to an integer. The call to printf() prints the number 10 on the screen.

Check Your Progress B

Fill in the blanks

1. On decrementing a pointer, its value get decreased by the number of bytes of its ………
2. The maximum levels of indirection that can be used in C for pointers are……..

3. Pointer points to another pointer is called as……

11.7 SUMMARY

C was developed when computers were much less powerful than they are today and being very efficient with speed and memory usage was often not just desirable but vital. The raw ability to work with particular memory locations was obviously a useful option to have. A few tasks these days, such as programming microcontrollers, still need this. However most modern programmers do not need such fine control and the complications of using pointers make programs less clear to understand and add to the ways in which they can be go wrong. So why are pointers still used so much in C & its successor, C++. Pointers look complicated because it is not obvious what they are doing and they can be combined many levels deep but normally they are just being used to bodge in several of C's missing features.

It can get very confusing if these different uses of pointer are combined. For example, even if one is just passing an alterable array of strings to a subroutine, one has to work with a pointer to pointers to pointers to characters with the pointers being used in 3 different ways at once, none of them being the raw memory manipulation pointers were essentially designed for!

11.8 KEYWORDS

Pointer: A pointer is a variable that contains the address of another variable.

Dangling pointers: A dangling pointer is that pointer which has been allocated but does not point to any entity.

The & operator: When a variable x is declared in a program, a storage location in main memory is made available by the compiler.

The * operator: The ‘*’ is an indirection operator or **‘ value at address operator’**.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. A pointer is a variable that contains the address of another variable.
2. There are four arithmetic operators that can be used on pointers: ++, --, +, and -
3. Pointer method and array indexing method.
4. Pointers store address of another variables.

Check Your Progress B

1. Data type
2. 12
3. Indirection

11.9 SELF ASSESSMENT QUESTIONS

1. What is meant by pointer?
2. Are the expression (*p)++ and ++*P equivalent?
3. “Pointers always contain Integers” Comment.

4. Differentiate between an uninitialized pointer and a NULL pointer.

11.10 SUGGESTED READINGS

A.K. Sharma, Fundamentals of Computers & Programming with C, Dhanpat Rai Publications, New Delhi.

Yashavant P. Kanetkar, Let Us C, Bpb Publications.

E. Balagurusamy, Programming in ANSI C, Tata Mcgraw Hill.

Class	**:**	**M.Sc. (Mathematics)**	**Course Code :**	**MAL-645**
Subject	**:**	**Programming in C**		

LESSON-12

INTRODUCTION TO C++

STRUCTURE

12.0 OBJECTIVES

After reading this lesson, you should be able to:

(a) Understand basic concepts of OOP

(b) Have a idea about C++

(c) Understand the concept of objects and class

12.1 INTRODUCTION

C++ is an objected-oriented language. Bjarne Stroustrup from AT&T Laboratories developed it in 1980. It is an extension of the C programming language, which means that all of the C library functions can be used in a C++ application. Class construct features have been added from Simula67. Since, the feature of classes was the major addition to original C language, C ++ is also called as ‘C with classes’. This name came from the increment operator ‘++’ as the language C++ is an incremented version of C.

C++ is superset of C. So, all the C programs are also C++ programs. In 1997, the ANSI/ISO standard committee standardized various changes and added new features to it. The important features added were classes, inheritance, functions, overloading and operator overloading. They enable creation of abstract data types, inheritance of existing data types and polymorphism support which makes C++ a truly object-oriented language.

12.2 EVOLUTION OF C++

In this section, we see how C++ has evolved over the years and introduce some of the language's features. Since C++ is an object-oriented programming language, it is important to understand the concepts of object-oriented programming. As C was the fastest procedural language, Bjarne Stroustrup decided to add classes, function argument type checking and conversion, and other features to it. Later, virtual functions and operator overloading were added to the language, and the language was renamed to C++ from "C with Classes". After a few refinements, the language became available to the public in 1985. Templates and exception handling were added to C++ in 1989. The Standard Template Library (STL) was developed in 1994. The object-oriented features allow building large programs with clarity, extensibility, and ease of maintenance. The addition of new features to C transformed it from top-down structured approach to bottom-up object-oriented approach.

12.3 FEATURES OF C++

C++ is an object-oriented programming (OOP) language. It offers all the advantages of OOP by allowing the developer to create user-defined data types for modeling the real world situations. The real power lies in the feature of classes. C++ is a versatile language that can handle large programs well. Virtually, it is suitable for any programming task including the development of editors, compilers, communication systems, databases and complex systems. Some of the features of C++ are:

1) C++ allows creating hierarchy-related objects, which improves reusability.
2) C++ maps the real world problem whereas the C part gives ability to get close to machine-level details.
3) The programs are easily maintainable and expandable. A new feature can be added easily.

Check Your Progress A

1. C++ was developed by…….
2. C++ is a combination of…….and ………..
3. Earlier name of C++ was ……..

12.4 OBJECT-ORIENTED PROGRAMMING

There are two programming paradigms:

- Procedure-Oriented
- Object-Oriented

Object-oriented programming paradigm is a model which is organized around 'objects' rather than actions and 'data' rather than logic. The major factor in the invention of object-oriented approach is to remove the flaws of procedural approach. In OOP, data is termed as critical element and is not allowed to move freely in the program. Data is tied closely to the functions which can operate it and is protected from any accidental modification by other functions. The problem is decomposed into number of entities called as objects. The data of an object can only be accessed by the functions associated with it. Some of the important features of OOP are-

- Emphasis is on data rather than procedure.
- Programs are divided into objects.
- Data structures are designed to characterize the objects.
- Data cannot be accessed by the external functions.
- Objects can communicate with each other through functions.

- Bottom-up approach is followed in the program design.
- Functions and data are tied together in a data structure.

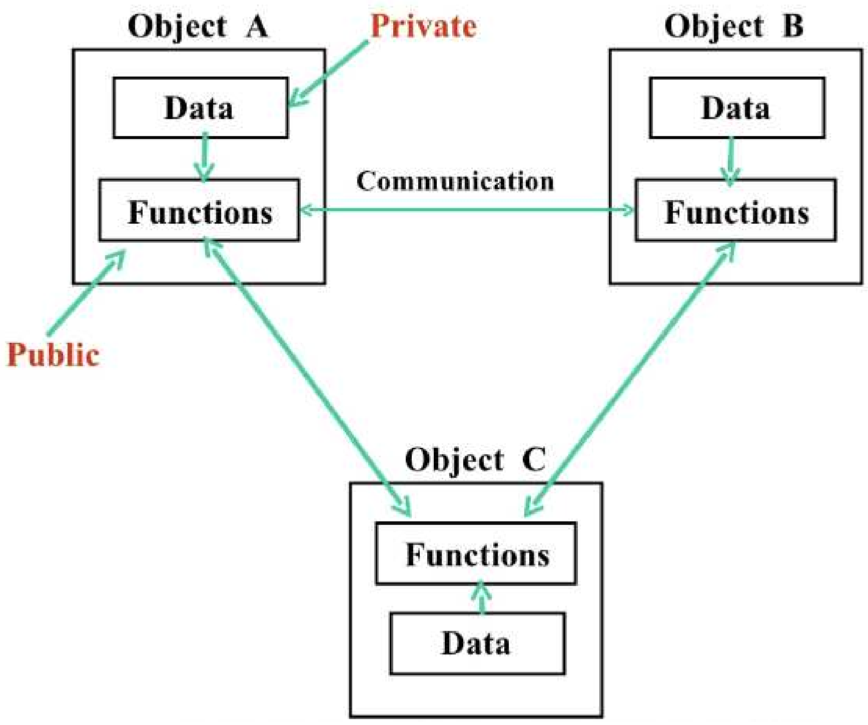


Figure 1 Organization of OOP

The comparison of two programming paradigms is as follows:

Table 1 Comparison of Procedure-Oriented and Object-Oriented Programming

Procedure-Oriented Programming	**Object-Oriented Programming**
Top Down/Bottom Up Design	Identify objects to be modeled
Structured programming	Concentrate on what an object does
Centered around an algorithm	Hide how an object performs its tasks

Identify the tasks; how something is done	Identify an object’s behavior and attributes

12.5 BASIC CONCEPTS OF OOP

Some of the basic concepts of object-oriented programming are-

Class

Class is a collection of objects of similar type. It is the building block that leads to Object Oriented programming. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. A Class is a user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class. A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

Objects

An Object is an instance or variables of the type Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Data Encapsulation

Data encapsulation wraps up the data and functions into a single unit called as class. It is the most important feature of the class. It separates the implementation from the interface. User access to data is only allowed through a defined interface. Data is not accessible to the outside world. Only the functions, which are inside the class, can access the data and thus provide the interface. The insulation of the data from direct access by the program is called as data hiding or information hiding. Encapsulation makes it possible for objects to be treated like 'black boxes'.

Data Abstraction

Abstraction is the ability of representing the essential features without including the background details. Knowledge of the implementation is unnecessary and therefore hidden. Data abstraction defines a data type by its functionality as opposed to its implementation. For example, classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight, cost etc. and functions are defined to operate on them. The attributes are also called as data members as they hold the information. The functions that operate on these data are called methods or member functions.

Inheritance

Inheritance is a means for defining a new class as an extension of a previously defined class. It is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. A ***derived*** class inherits all attributes and behavior of a ***base*** class, i.e., it provides access to all data members and member functions of the base class, and allows additional members and member functions to be added if necessary.

The base class and derived class have an “is a” relationship. For example,

- Baseball (a derived class) **is a** Sport (a base class)
- Pontiac (a derived class) **is a** Car (a base class)

Inheritance provides the idea of reusability. That is, additional features can be added to an already existing class without modifying it by deriving a new class from the existing one.

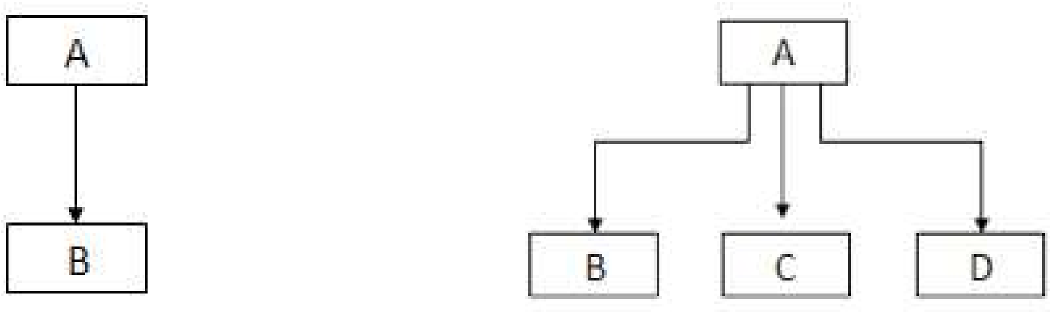


Figure 2 Simple and Hierarchical Inheritance

Polymorphism

Polymorphism is a Greek term, means the ability to take more than one form. It is the ability of different objects to respond differently to virtually the same function. An operation may exhibit different behavior in different instances and this behavior depends upon the type of data used in the operation. For example, a base class provides a function to print the current contents of an object. Through inheritance, a derived class can use the same function without explicitly defining its own. However, if the derived class must print the contents of an object differently than the base class, it can override the base class's function definition with its own definition. In order to invoke polymorphism, the function's return type and parameter list must be identical. Otherwise, the compiler ignores polymorphism. Another example is addition operation. If it is performed on numbers, it adds the two number. But when it is performed on the strings, a third string is produced which is the concatenation of the two strings. This is called as operator overloading. When a function is used to exhibit different forms, it is called as function overloading.

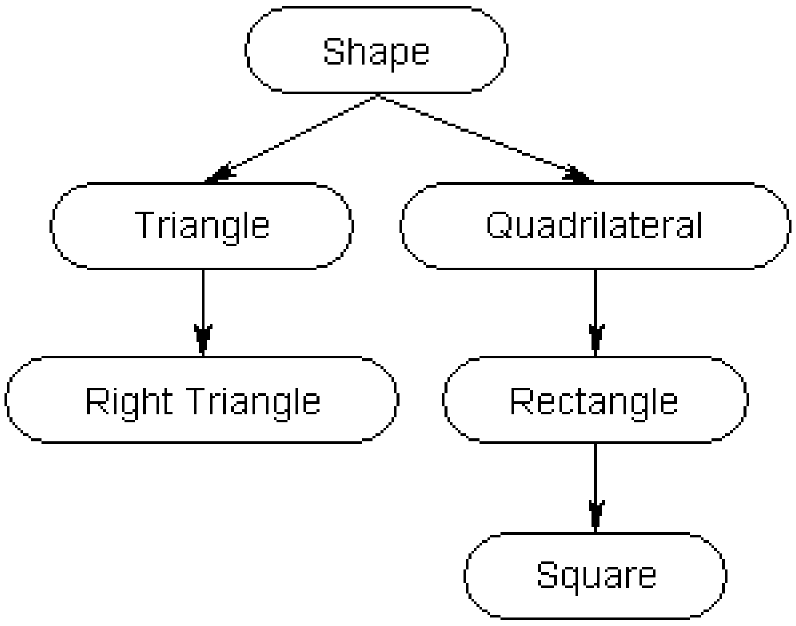


Figure 3 Polymorphism Through Function

12.6 STRUCTURE OF A C++ PROGRAM

A simple C++ program consists of four sections as described in the figure.

INCLUDE FILES
CLASS DECLARATION
MEMBER FUNCTION DEFINITIONS
MAIN FUNCTION PROGRAM

Figure 4 Structure of a C++ Program

Creating the Source File

C++ programs can be created in any text editor like C. for example, vi text editor, edlin etc. Some systems also provide an integrated environment for developing and editing the programs. The file should have the extension .cc or .cpp or .cxx depending upon the editor.

```
#include <iostream.h>
using namespace std;
int main( )
{
cout<<”Hello, World!”
return 0;
}
```

Output

```
Hello, World!
```

The Header File

The C++ language defines several headers, which contain information that is either necessary or useful to your program. The header file iostream is included at the beginning of all the programs that use input and output statements. The directive iostream is included in the program which causes the preprocessor to add the contents of the iostream file. It contains declarations for cout and cin, which are output and input operator respectively.

The operator << is calked insertion or output operator. It inserts the contents of the variable on its right to the object on its left.

cout<< string;

The operator >> is called extraction or input operator. It extracts the value from keyboard and assigns it to the variable on its right. It causes the program to wait for the user to type in a number. The identifier cin is a predefined object in C++ that corresponds to the standard input stream keyboard.

Namespace

Namespace is a concept of ANSI C++ standard committee. It defines scopes for the identifiers used in a program. We need to include the following directive in the program to use namespace-

Using namespace std;

Here, std is the namespace where ANSI C++ standard libraries are defined.

Program Features

C++ program is a collection of functions. Every C++ program must have a main function. The execution begins from main( ). In C++, main( ) returns an integer type of value to the operating system. So, every main function in C++ should end with a return (0) statement and the return type for main is explicitly specified as int. The following code will run with an error warning-

```
main()
{
…….
}
```

The correct syntax should be-

```
int main()
{
…….
return 0;
}
```

Statements are terminated with semicolons.

Classes in C++

A class is user defined data type, also called as abstract data type. It encapsulates a data type and any operations on it. A class is also an extension of a C structure, which is a collection of one or more variables defined under a single name. The biggest difference between the two is the default access to data members and member functions. By default, data members and member functions in a class are **private**, where they are **public** in a structure. An **abstract class** is one that contains at least one pure virtual member function.

A basic C++ class as well as a structure usually contains the following elements:

- Constructor(s) – creates an object.
- Destructor – destroys an object.
- Data members – object attributes.
- Member functions (methods) – operations on the attributes.

C++ Identifiers and Keywords

C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, mango and Mango are two different identifiers in C++.

Here are some examples of acceptable identifiers –

A_123   zara   a123   _avc

Keywords are the reserved words that cannot be used as constant or variable or any other identifier names. Some of the keywords are- asm, else, new, this, auto, enum, operator, throw, true, try etc.

Comments

C++ introduces a new comment symbol – double slash - //. Comments start with double slash and terminate with end of the line. There is no closing symbol. It is a single line comment. For multiline comments, the symbol should be used in every line.

Example 1:

//This is a single line comment.

Example 2:

// This is a multiline comment. We have to use the symbol of double slash in

// starting of every line as there is no ending symbol.

Compile and Execution of Program

The process of compiling and linking depends upon the operating system. Turbo C++ and Borland C++, provides an integrated environment. They provide a built-in editor and a menu bar which includes options such as File, Edit, compile and Run. The file can be created and saved under the File option. We can edit it under Edit option. Compilation is done under the Compile option. Then, execution is performed under the Run option.

Some of the popular compilers for C++ are-

Borland C++

Microsoft Visual C++

(UNIX) g++

Watcom C++

Metrowerks C++ (Mac)

Check Your Progress B

1. Which functions are defined in iostream file.
2. What is the return type of main function in C++?
3. Which new symbol is added in C++ for comments?

12.7 SUMMARY

C++ is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features. It is superset of C, that means, a C program is also an C++ program. C++ is being highly used to write device drivers and other software that rely on direct manipulation of hardware under real time constraints. C++ is widely used for teaching and research because it is clean enough for successful teaching of basic concepts.

12.8 KEYWORDS

Class: A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.

Methods: A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

Enacapsulation: wrapping up of data and functions into a single unit is called as encapsulation.

Abstraction: Abstraction refers to the act of representing the essential features without including the background details or explanations.

ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress A

1. Bjarne Stroustrup
2. C and Simula67
3. C with classes

Check Your Progress B

1. input and output
2. int
3. ‘//’ (double slash symbol)

12.9 SELF ASSESSMENT QUESTIONS

1. What are the differences between C and C++?
2. What do you understand by Object-oriented programming?

3. Define encapsulation and abstraction.
4. Define Inheritance and Polymorphism.
5. What are the features of C++?
6. Define Namespace.
7. Define iostream header file.

12.10 SUGGESTED READINGS

E. Balagurusamy, Object-Oriented Programming with C++, Tata Mcgraw Hill.

Chuck Allison, C & C++ Code Capsules

Margaret Ellis and Bjarne Stroustrup, The Annotated C++ Reference Manual