PROGRAMMING WITH C & C++

B.A. / B.Com. SECOND YEAR

Semester – 3

Lesson Writers

Sri G. Venugopal Rao
M.Sc., M.Phil.
Lecturer,
Dept. of Computer Science,
J.K.C. College, Guntur.

Smt. M. Nirupama Bhat
MCA., M.Phil.
Lecturer,
Dept. of Computer Science,
J.K.C. College, Guntur.

Smt. M. Sudha Rani
MCA
Lecturer,
Dept. of Computer Science,
J.K.C. College, Guntur.

Sri Y. Venkateswara Rao
MCA
Lecturer,
Dept. of Computer Science,
J.K.C. College, Guntur.

Sri J. Venkata Rao
M. Sc.
Lecturer,
Dept. of Computer Science,
J.K.C. College, Guntur.

Dr.Vasantha Rudramalla
M.Tech.,Ph.D.
Faculty,
Dept. of Computer Science & Eng.
Acharya Nagarjuna University.

Editor

Prof. I. Ramesh Babu , M.E., Ph.D.
Dept. of Computer Science,
Acharya Nagarjuna University.

Director

Dr. NAGARAJU BATTU
MBA, MHRM, LLM, M.Sc.(Psy), M.A.(Soc), M.Ed., M.Phil., Ph.D.
Centre for Distance Education,
Acharya Nagarjuna University,
Nagarjuna Nagar 522 510, GUNTUR.

Ph : 0863 - 2346208, 2346222, 2346259 (Study Material)
Website : www.anucde.info
e-mail : anucdedirector@gmail.com

B.A. / B.Com . SECOND YEAR : Semester - 3

PROGRAMMING WITH C & C++

First Edition : 2023

No. of Copies :

Published by :
Dr. Nagaraju Battu
Director,
Centre for Distance Education,
Acharya Nagarjuna University.

Printed at :

FOREWORD

Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining a ‘A’ Grade from the NAAC in the year 2014, the Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 285 affiliated colleges spread over the two districts of Guntur and Prakasam.

The University has also started the Centre for Distance Education with the aim to bring higher education within reach of all. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even housewives desirous of pursuing higher studies. With the goal of bringing education in the door step of all such people. Acharya Nagarjuna University has started offering B.A, and B, Com courses at the Degree level and M.A, M.Com., L.L.M., courses at the PG level from the academic year 2021-22 on the basis of Semester system.

To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers invited respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.

It is aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn facilitate the country’s progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Coordinators, Editors and Lesson -writers of the Centre who have helped in these endeavours.

Prof. P.Rajasekhar
Vice –Chancellor,
Acharya Nagarjuna University

ACHARYA NAGARJUNA UNIVERSITY - GUNTUR

Structure of B.Com (Computer Applications) Programme under Revised CBCS

Semester-wise Syllabus under CBCS (w.e.f. 2020 – 21 Admitted Batch)

II – Year B.A. / B Com (CA); Semester – II

COURSE 3C: PROGRAMMING WITH C & C++ (with Practical Component)

Model Outcomes :

At the end of the course, the students is expected to DEMONSTRATE the following cognitive abilities (thinking skill) and psychomotor skills.

A. *Remembers and states in a systematic way (Knowledge)*

1. Develop programming skills
2. Declaration of variables and constants use of operators and expressions
3. learn the syntax and semantics of programming language
4. Be familiar with programming environment of C and C++
5. Ability to work with textual information (characters and strings) & arrays

B. *Explains (Understanding)*

6. Understanding a functional hierarchical code organization
7. Understanding a concept of object thinking within the framework of functional model
8. Write program on a computer, edit, compile, debug, correct, recompile and run it

C. *Critically examines, using data and figures (Analysis and Evaluation)*

9. Choose the right data representation formats based on the requirements of the problem
10. Analyze how C++ improves C with object-oriented features
11. Evaluate comparisons and limitations of the various programming constructs and choose correctone for the task in hand.

D. Working in 'Outside Syllabus *Area' under a Co-curricular Activity*(Creativity)
Planning of structure and content, writing, updating and modifying computer programs for user solutions

E. Exploring C programming and Design C++ classes for code reuse (Practical skills***)

SYLLABUS

COURSE 3C: PROGRAMMING WITH C & C++

Unit-I Introduction :

Introduction - Structure of C program – C character set, Tokens: Constants, Variables, Keywords, Identifiers – C data types - C operators (arithmetic, relational, logical, increment and decrement) - Standard I/O in C (scanf, printf) - Conditional Control statements (if and Switch) Statements.

Unit-II Loops And Arrays :

Repetitive statements: While, Do While and For Loops - Use of Break and Continue Statements –**Arrays**: Introduction – Types of arrays, one dimensional arrays - Declaration of one dimensional arrays–Accessing array elements–Storing values in an array –Two Dimensional Arrays Declaration of two dimensional arrays – Accessing array elements– Storing values in 2-D arrays.

Unit- III Strings and Functions :

Strings: Definition, Declaration and Initialization of String Variables - String Handling Functions – **Functions** : Defining Functions - Function Call – passing parameters: Call By Value, Call By Reference.

Unit- IV Classes and Objects :

Introduction to OOP and its basic features - C++ program structure - Classes and objects - Friend Functions- Static Functions –Constructor – Types of constructors – Destructors – Operators.

Unit-V Inheritance :

Inheritance - Types of Inheritance -Types of derivation- Public – Private - Protected Hierarchical Inheritance - Multilevel Inheritance – Multiple Inheritance - Hybrid Inheritance.

References :

(1) Computer Fundamentals and Programming in C by Reema Thareja from Oxford University Press
(2) Mastering C by K R Venugopal and Sudeep R Prasad, McGraw Hill
(3) Let Us C, Yashavant Kanetkar
(4) E. Balagurusamy "Object oriented programming with C++
(5) R.Ravichandran "Programming with C++"
(6) The C++ Programming Language Bjarne Stroustrup

Online Resources :

https://www.tutorialspoint.com/cprogramming/i
ndex.html https://www.learn-c.org/
https://www.programiz.com/c-programming
https://www.w3schools.in/c-tutorial/
https://www.cprogramming.com/tutorial/c-
tutorial.html
https://www.tutorialspoint.com/cplusplus/inde
x.html
https://www.programiz.com/cpp-
programminghttp://www.cplusplus.com/doc/tutorial/ https://www.learn-
cpp.org/
https://www.javatpoint.com/cpp-tutorial

ACHARYA NAGARJUNA UNIVERSITY-GUNTUR

Structure of B.Com (Computer Applications) Programme under Revised CBCS

Semester-wise Syllabus under CBCS (w.e.f. 2020-21 Admitted Batch)

II Year B.A. / B. Com. (CA); Semester- III

COURSE 3C: PROGRAMMING WITH C & C++ Practical Component

1. Write C programs for
   a. Fibonacci Series
   b. Prime number
   c. Palindrome number
   d. Armstrong number.

2. 'C' program for multiplication of two matrices

3. 'C' program to implement string functions

4. 'C' program to swap numbers

5. 'C' program to calculate factorial using recursion

6. 'C++' program to perform addition of two complex numbers using constructor

7. Write a program to find the largest of two given numbers in two different classes using friend function

8. Program to add two matrices using dynamic constructor

9. Implement a class string containing the following functions :
   a. Overload + operator to carry out the concatenation of strings.
   b. Overload == operator to carry out the comparison of strings.

10. Program to implement inheritance.

RECOMMENDED CO-CURRICULAR ACTIVITIES :

(Co-curricular activities shall not promote copying from textbook or from others work and shall encourage self/independent and group learning)

MEASURABLE

1. Assignments (in writing and doing forms on the aspects of syllabus content and outside the syllabus content. Shall be individual and challenging)
2. Student seminars (on topics of the syllabus and related aspects (individual activity)
3. Quiz (on topics where the content can be compiled by smaller aspects and data (Individuals or groups as teams)
4. Field studies (individual observations and recordings as per syllabus content and related areas (Individual or team activity)
5. Study projects (by very small groups of students on selected local real-time problems pertaining to syllabus or related areas. The individual participation and contribution of students shall be ensured (team activity))

General

Group Discussion
Visit to Software Technology parks / industries

RECOMMENDED CONTINUOUS ASSESSMENT METHODS:

Some of the following suggested assessment methodologies could be adopted :

1. The oral and written examinations (Scheduled and surprise tests),
2. Closed-book and open-book tests,
3. Coding exercises,
4. Practical assignments and laboratory reports,
5. Observation of practical skills,
6. Individual and group project reports,
7. Efficient delivery using seminar presentations,
8. Viva voce interviews.
9. Computerized adaptive testing, literature surveys and evaluations,
10. Peers and self-assessment, outputs form individual and collaborative work.

MODEL QUESTION PAPER

B.A. / B.Com. DEGREE EXAMINATION.

Second Year – Third Semester

Part – II : Arts / Commerce

Paper III : PROGRAMMING WITH C AND C++

Time: Three hours **Max. Marks : 70**

SECTION A **(5 x 4 = 20 Marks)**

Answer any **FIVE** of the following questions.

1. Wrote the structure of C Program.
   C ప్రోగ్రాం యొక్క నిర్మాణంను రాయండి.
2. Explain C data types.
   C లోని డేటా టైప్స్ ను వివరించండి.
3. Explain for loop with example.
   ఫర్ లూప్ ను ఉదాహరణతో వివరించండి.
4. Explain accessing array elements with examples.
   శ్రేణి మూలకాలు యాక్సెస్ చేయడానికి ఉదాహరణతో వివరించండి.
5. Explain declaration and initialization of string variables.
   స్ట్రింగ్ వేరియబుల్ మరియు డిక్లరేషన్ ఇనిషియలైజేషన్ గురించి వివరించండి.
6. Write a short note on Functions.
   ఫంక్షన్స్ గురించి ఒక చిన్న గమనికను వ్రాయండి.
7. Explain friend function with an example in C++.
   C++ లోని ఫ్రెండ్ ఫంక్షన్ను ఒక ఉదాహరణతో వివరించండి.
8. Explain destructors in C++.
   C++ లోని డిస్ట్రక్టర్లను వివరించండి.
9. Explain Public and private concepts in C++.
   పబ్లిక్ మరియు ప్రైవేట్ అంశాలను C++ లోనికి వివరించండి.
10. Explain about benefits of inheritance.
    వారసత్వ ప్రయోజాలను గురించి వివరించండి.

SECTION B **(5 x 10 = 50 Marks)**

Answer any **FIVE** of the following questions.

11. Explain operators in C.

    C లోని ఆపరేటర్స్ ను వివరించండి.

12. Explain if and switch statements with examples.

    if మరియు switch statements ను ఉదాహరణతో వివరించండి.

13. Explain array with types of arrays.

    శ్రేణుల రకాలతో శ్రేణినిని వివరించండి.

14. Explain while and do while loops with examples.

    while మరియు do while లూప్ లను ఉదాహరణతో వివరించండి.

15. Explain string handling functions.

    స్ట్రింగ్ హ్యాండ్లింగ్ ఫంక్షన్లను వివరించండి.

16. Briefly explain call by value and call by reference with examples.

    కాల్ బై వేల్యూ మరియు కాల్ బై రిఫరెన్స్ ను క్లుప్తంగా వివరించండి ఉదాహరణతో.

17. Explain structure of C++ program with one example.

    ఒక ఉదాహరణతో C++ ప్రోగ్రాం నిర్మాణమును వివరించండి.

18. Explain constructor with different types.

    వివిధ రకాలతో కంస్ట్రక్టర్ ను వివరించండి.

19. Explain inheritance with various types.

    వివిధ రకాల వారసత్వాన్ని వివరించండి.

20. Write C++ program to implement multilevel inheritance.

    C++ ప్రోగ్రాంతో మల్టీ లెవెల్ ఇన్హెరిటెన్స్ ను అమలు చేయటం వ్రాయండి.

*****

CONTENTS

LESSON – 1

STRUCTURE OF C LANGAUGAE

AIMS AND OBJECTIVES :

The objectives of this lesson are

- To discuss the history and development of C program language levels.
- To be acquainted with the various elements (character set) of C language.
- To know the structure of the C program and some program characteristics of C language.

STRUCTURE OF THE LESSON :

1.1. INTRODUCTION :

Computer will not understand any of the natural languages. So we need a language to communicate with computer. There are programming languages specially developed so that we could pass our data and instructions to the computer to do a specific job.

There are two types of programming languages. These are Low Level Languages and High-Level Languages.

- Low level Languages
- High level languages

1.1.1. Low Level Languages :

The term low level means closeness to the way in which the machine can understand. Low-level languages are further divided into Machine language and Assembly language.

1.1.2. Machine Language :

Machine Language is the only language that can be directly understood by the computer. It does not need any translator program. We also call it machine code and it is written as strings of 1's (one) and 0’s (zero). When this sequence of codes is fed to the computer, it recognizes the codes and converts it into electrical signals. For example, a program instruction may look like this :

1011000111101

It is not easy to learn and write instructions in this form. Because of the complexity very few people can write the programs in this language. It is considered to be the first-generation language.

The only advantage is that program written in machine language run very fast because no translation is required for the CPU.

1.1.3. Assembly Language :

A low-level programming language that is slightly user-friendlier than machine language. The software, which translates a program written in assembly language to machine language, is called an assembler. These are considered to be second-generation languages.

Advantages of assembly language are:

It is easier to understand and write programs, which are nearer to English language. It saves a lot of time for the programmer, easier to correct errors and modify the program instructions whenever necessary.

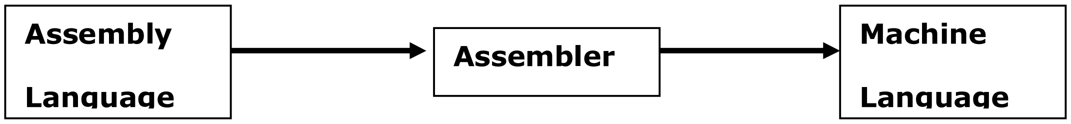
1.1.4. High-level Language :

These are third-generation languages or 3GLs. These are closer to so-called natural languages (we talk). High-level languages are simple languages that use English and mathematical symbols like +, -, %, / etc., for its program construction. Any program written in High-level language has to be converted to machine language for the computer to understand.
Higher-level languages are problem-oriented languages because the instructions are suitable for solving a particular problem. Choice of a particular high-level language depends on the

application/problem. Advantages of High-Level Languages are easy to learn and use because that they are similar to the languages used by us in our day-to-day life.
Translator is a program, which play major role by converting source language to object language. We have translators like assembler, compiler and interpreter.

1.1.5. Assembler :

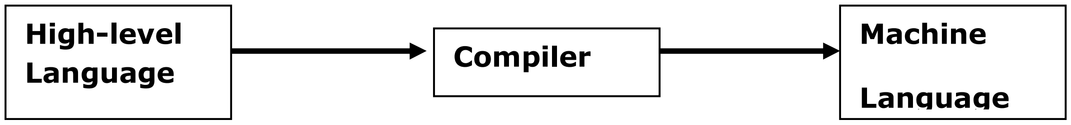
Assembler is a translator or converter, which converts assembly language program (Mnemonics or symbols) to machine understandable code called machine language (0's and 1's).



1.1.6. Compiler :

The programs written by the programmer in high-level language is called source program. Compiler translates the source program to machine language; it is called the object code. Compiler is a program translator like assembler but more sophisticated. It scans the entire program first and then translates it into machine code.

Every high-level language has its own compiler. For example, FORTRAN compiler will not compile source code written in COBOL language.



Interpreter :

An interpreter is another type of program translator used for translating higher-level language into machine language. It takes one statement of high-level language, translates it into machine language and immediately executes it. Translation and execution are carried out for each statement.
It differs from compiler in the following aspects :
Compiler translates the entire source program into machine understandable form at a time and generates an object code, which can be used on repeated execution of the program. With interpreter the program needs to be retranslated every time you want the program to be executed.

1.2. STRUCTURE OF C PROGRAM :

C is a general-purpose programming language. C instructions consist of terms that resemble algebraic expressions, augmented by certain English keywords such as if, else, for, do and while. C is characterized by the ability to write very concise source programs, due in part to the large number of operations included within the language. C compilers are commonly available for computers of all sizes. C programs are highly portable.

C was originally developed in the 1970's by Dennis Ritchie at Bell Telephone Laboratories now AT & T Bell laboratories. It is an outgrowth of two earlier languages, called BCPL and B, which were also developed at Bell Laboratories. It was written originally for programming under an operating system called UNIX, which itself was later rewritten almost entirely in C. The C language is often, described as a "middle-level" language. It permits programs to be written in much the same style as that of most modern high-level languages.

Some important characteristics of C language are.

1.2.1 Integrity :

This refers to the accuracy of the calculations. The integrity of the calculations is an absolute necessity in any computer program.

1.2.2. Clarity :

This refers to the overall readability of a program. If a program is clearly written, it should be possible for another programmer to follow the programming logic.

1.2.3. Simplicity :

Keeping things as simple as possible usually enhances the clarity and accuracy of a program consists with the overall program objectives.

1.2.4. Efficiency :

This is concerned with execution speed and efficient memory utilization.

1.2.5. Modularity :

Many programs can be broken into a series of identifiable sub tasks. It is a good programming practice for the programmer to implement each individual subtask as a separate program module.

```
Documentation section
Linkage section
Define section
Global declaration section
Main()
{
declaration part
executable part
}
subprogram section
{
function-1()
function-2()
.
.
function-n()
}
```

1.2.6. Documentation section :

The documentation section consists of a set of comment lines giving the name and reference of the program. The comment lines should be enclosed /* -----------------*/.

Ex : /* this is my first c program*/

// This is my first c program.

Sometimes this section is optional.

1.2.7. Link section :

The first line of the program. This section provides instructions to the compiler to link functions from the standard input/output library. This is called preprocessor directive.

Ex : #include<stdio.h>

This is called header file, which includes all the input and output functions to the compiler.

#include<math.h>

This includes all the mathematical operations like sqrt, pow, log.

#include<string.h>

This includes all the string functions like is upper, is lower, etc.

1.2.8. Definition section :

This section defines all the symbolic constants.

Syntax: #define symbolic name value

Ex: #define is 1000

#define pa 2000

#define name “sairam”

Note :

Symbolic names are nothing but variable names. There was no blank space in between # and define.

#define statement should not end with semicolon.

1.2.9. Global declaration section :

There are some variables that are used in more than one function such variables are called global variables. These variables are declared in the global declaration section. That is outside of all the functions. They are called global variables.

main function:

Every C program must have one main function called “main”. The execution of C program starts from the main function. In this function we have two parts. Those are declaration part and executable part.

In the declaration part declare all the variables used in the executable part. These two parts must appear in between the opening ({) and closing braces (}).

Subprogram Section :

Subprogram section contains all the user-defined functions that are called in the main function. User defined functions are generally placed immediately after the main function.

Examples :

1. **A Program for printing single statement.**

```
#include<stdio.h>

    main()

        {

            printf (“This is my first program”);

        }
```

Output : This is my first program.

2. **A Program for printing address line by line.**

```
#include<stdio>

     main ()

        {
            printf(“G.VENUGOPAL RAO\n”);
            printf(“LECTURER \n”);
            printf(“ DEPT OF COMPUTERS\n”);
            printf(“ J.K.C.COLLEGE\n”);

        }
```

Output :
```
G.VENUGOPAL RAO
LECTURER
DEPT OF COMPUTERS
J.K.C.COLLEGE
```

1.3. C CHARACTER SET :

Character set means the characters and symbols that can be used in a C program. They are grouped to form the commands, expressions and other tokens for C language. Character set is the combination of characters, digits, special characters and blank spaces.

C language uses :

Alphabets	:	Upper case letters A to Z
		Lower case letters a to z
Digits	:	0 to 9
Special Characters	:	{,}, #, : ,;,&,? Etc.,

1.4. SUMMARY :

In this chapter C has efficient programming characteristics and it explains the history and development of, C language.

1.5. KEY WORDS :

- **Computer :** A computer is an electronic machine that can store and deal with large amounts of information.
- **Computer Program :** It is a sequence or set of instructions in a programming language for a computer to execute.
- **Languages :** A computer language is a formal language used to communicate with a computer.

1.6. SELF ASSESSMENT QUESTIONS :

1. What is a C Program? Discuss the types of programming languages?
2. Explain the structure of C language program?
3. Write about the C Character Set?

1.7. FURTHER READINGS :

1. "Programming with C" by Byron C.Gotttfried.
2. "Let us C" by Yeshavanth kanethkar.
3. "Working with C" by Yeshavanth kanethkar.

LESSON - 2
TOKENS IN C

AIMS AND OBJECTIVES:

The objectives of this lesson are to

- Discuss the C program elements (keywords).
- Explain different C program entities like identifiers, data types and expressions.

STRUCTURE OF THE LESSON:

2.1. CONSTANTS :

A constant is an identifier whose value can never be changed during the program execution. A constant is very similar to variables in the C programming language, but it can hold only a single variable during the execution of a program. It means that once we assign value to the constant, then we can't change it throughout the execution of a program- it stays fixed.

Use of the Constants in C :

A constant is basically a named memory location in a program that holds a single value throughout the execution of that program. It can be of any data type- character, floating-point, string and double, integer, etc. There are various types of constants in C. It has two major categories- primary and secondary constants. Character constants, real constants,

and integer constants, etc., are types of primary constants. Structure, array, pointer, union, etc., are types of secondary constants.
What are Literals in C ?

Literals are referred to as the values that we assign to the variables that remain constant throughout the execution of a program. Generally, we can use both the terms- literal and constants interchangeably. For instance, the expression *"const int = 7;",* is a type of constant expression, while we refer to the value 7 as a constant integer literal.
C has four basic types of constants. They are

1. Integer constant
2. Floating-point constant
3. Character constant
4. Back slash character constant
5. String constant

2.1.1. Integer constant :

It can be an octal integer or a decimal integer or even a hexadecimal integer. We specify a decimal integer value as a direct integer value, while we prefix the octal integer values with 'o'. We also prefix the hexadecimal integer values with '0x'.

The integer constant used in a program can also be of an unsigned type or a long type. We suffix the unsigned constant value with 'u' and we suffix the long integer constant value with 'l'. Also, we suffix the unsigned long integer constant value using 'ul'.

Represenation samples of Integer constant,

55	—>	Decimal Integer Constant
0x5B	—>	Hexa Decimal Integer Constant
O23	—>	Octal Integer Constant
68ul	—>	Unsigned Long Integer Constant
50l	—>	Long Integer Constant
30u	—>	Unsigned Integer Constant

Example : 0 1 743 5280

2.1.2. Floating constant :

This type of constant must contain both the parts- decimal as well as integers. Sometimes, the floating-point constant may also contain the exponential part. In such a case when the floating-point constant gets represented in an exponential form, its value must be suffixed using 'E' or 'e'.

Example 1 :

We represent the floating-point value 3.14 as 3E-14 in its exponent form.

Example 2 : 0. 1. 0.2 825.602 2E-8 0.06E-3

3*105 can be represented as follows :

300000. 3E5 3E+5 3.0E+5 .3E+6 30E4

5.026*10-17 can be represented as follows :

5.026E-17

.5026E-16

2.1.3. Character constant :

The character constants are symbols that are enclosed in one single quotation. The maximum length of a character quotation is of one character only.

Example 1 :

‘B’

‘5’

‘+’

Some predefined character constants exist in the C programming language, known as escape sequences. Each escape sequence consists of a special functionality of its own, and each of these sequences gets prefixed with a ‘/’ symbol. We use these escape sequences in output functions known as ‘printf()’

Example 2 : 'A', 'X', '3', '$'

2.1.4. Back slash character constants (or) escape sequences :

C supports some back slash character constants that are especially used in output statements. They consist of two characters, which will be enclosed within the single codes. Every back slash character constant will perform its individual task.

S.NO	CONSTANT	MEANING	ASCII VALUE
1	‘\n’	New line	010
2	‘\a’	Bell	007
3	‘\b’	Back space	008
4	‘\r’	Vertical tab	011
5	‘\t’	Horizontal tab	009
6	‘\f’	Form feed	012
7	‘\r’	Carriage return	013
8	‘\o’	Null	000

2.1.5. String Constant :

A string consists of any number of consecutive characters enclosed in double quotations.

The string constants are a collection of various special symbols, digits, characters, and escape sequences that get enclosed in double quotations.

The definition of a string constant occurs in a single line :

“This is Cookie”

We can define this with the use of constant multiple lines as well :

” This\

is\

Cookie”

The definition of the same string constant can also occur using white spaces :

“This” “is” “Cookie”

All the three mentioned above define the very same string constant.

Ex : "green", "Jkc college"

Creation and Use of Constants in C :

We can create constants in the C programming language by using two of the concepts mentioned below :

- By using the ‘#define’ preprocessor
- By using the ‘const’ keyword.

Use of the ‘const’ Keyword :

The ‘const’ keyword is used to create a constant of any given datatype in a program. For creating a constant, we have to prefix the declaration of the variable with the ‘const’ keyword. Here is the general syntax that we follow when using the ‘const’ keyword :

const datatype constantName = value ;

OR

const datatype constantName ;

Let us look at an example to understand this better

const int a = 10 ;

In this case, a is an integer constant that has a fixed value of 10.

The program will run as follows :

```
    #include<stdio.h>
    #include<conio.h>
        void main()
            {
                int q = 9 ;
                const int a = 10 ;
                q = 15 ;
                a = 100 ;                          // creates an error
                printf("q = %d\na = %d", q, a ) ;
            }
```

The program given above creates an error. It is because we are trying to change the value of the constant variable (a = 100).

Use of the '#define' preprocessor :

One can also use the '#define' preprocessor directive to create the constants. And when we create the constants by making use of the preprocessor directive, we must define it in the very beginning of the program. It is because we must write all the preprocessor directives before the global declaration.

Here is the syntax that we must use for creating a constant by making use of the '#define' preprocessor directive :

#define CONSTANTNAME value

Let us look at an example to understand this better,

#define PI 3.14

In this above-mentioned case, PI is a constant, and it has a value of 3.14.

We can run a program for this as follows :

```
    #include<stdio.h>
    #include<conio.h>
    #define PI 3.14
        void main()
            {
                int a, area ;
                printf("Enter the radius of the given circle here : ") ;
                scanf("%d", &a) ;
                area = PI * (a * a) ;
                printf("The area of the circle is = %d", area) ;
            }
```

2.2. VARIABLES :

A variable is an identifier that is used to represent some specified type of information or a variable is an identifier whose value may or may not be changed during the program execution.

```
int a, b, c;

Char d;  a=3; b=4; d = 'a';
```

All variables must be declared before they can appear in executable statements.

```
int a, b, c;
float root1, root2;
char flag;
```

2.3. KEYWORDS :

There are certain reserved words, called keywords that have standard, predefined meanings in C. These keywords are to be used for their intended purpose only in a C program. There are 32 words defined as keywords in C. They are always written in lower case. A complete list is as follows :

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

In addition to this list of keywords, your compiler may define a few more. If it does, they will be listed in the documentation that comes along with your compiler.

2.4. IDENTIFIERS :

Identifiers are names given to various program elements, such as variables, functions and arrays. Identifiers consist of letters and digits, in any order, except that the first character must be a letter. It allows both uppercase and lowercase letters. Upper and lowercase letters are not interchangeable, i.e.; uppercase letter is not equivalent to the corresponding lowercase letter and vice versa. The underscore character (_) can also be included. An underscore is often used in the middle of an identifier.

Valid Identifiers	**Invalid Identifiers**
A, B, Stno, tax_rate etc.,	4th "x"

2.5. C – DATA TYPES :

C supports several different types of data, each of which may be represented differently within the computer's memory. A data type is a C token that tells about the type of data being assigned to an identifier.

C supports a rich set of data types. The category follows :

- Primary data types or Primitive data types
- Secondary data types

While using the above data types, one should intimate the compiler by providing corresponding format specifier or control string.

Primary data types :	**Secondary data types :**
short int	Arrays
int	Functions
long int	Pointers
float	Structures
double	Unions
long double	Enumeration

2.6. SUMMARY :

In this chapter Basics required to write a C program, like character set, keywords, data types. There are three basic data types in C like integer, float and character.

2.7. KEY WORDS :

Tokens : Tokens in C is the most important element to be used in creating a program in C.

Program : C is the combination of both low level (assembly) and high-level programming languages.

Constants : Constant is a value or variable that can't be changed in the program.

Keywords : Keywords are words that have special meaning to the C compiler.

Identifiers : Identifiers" or "symbols" are the names you supply for variables, types, functions, and labels in your program.

Data types : A data type is a collection or grouping of data values, usually specified by a set of possible values, a set of allowed operations on these values, and/or a representation of these values as machine types.

2.8. SELF ASSESSMENT QUESTIONS :

1. What is Token?
2. Explain the constants?

3. Differences between variables and keywords?
4. Write the properties of Identifiers?
5. Explain various data types in C?

2.9. FURTHER READINGS :

1. “Programming with C” by Byron C.Gotttfried.
2. “let us C” BY Yeshavanth kanethkar.
3. “working with C”BY Yeshavanth kanethkar .
4. “Programming *in ANSI* C” by E.Balagurusamy.
5. “Sams Teach Yourself C “ by Tony Zhang.

LESSON - 3

OPERATORS AND STANDARD I / O FUNCTIONS IN C

AIMS AND OBJECTIVES :

The objectives of this lesson is to

- ➢ know various types of operators that can be used in a C program.
- ➢ discuss various types of input and output functions of a C program.

STRUCTURE OF THE LESSON :

3.1. C OPERATORS (ARITHMETIC, RELATIONAL, LOGICAL, INCREMENT AND DECREMENT) :

An operator performs an operation over one or more operands. C supports a rich set of operators. The following are the some of the examples.

3.1.1. Binary Operators :

A binary operator is an operator that performs operations on two operands or values. There are many binary operators in C. Some of them are: +, -,*, /, %, +=,-=,*=,/= etc.,

Arithmetic operators are the regular operators that can be used to perform basic arithmetic operations like addition subtraction, multiplication and division in a program in order to solve a problem.

Operators	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Reminder

Suppose a and b are integer variables such as a=10 b=3

Expression	**Value**
a + b	13
a - b	7
a * b	30
a / b	3
a % b	1

v1, v2 are floating points variables whose values are 12.5 and 2.0

Expression	**Value**
v1 + v2	14.5
v1 - v2	10.5
v1 * v2	25.0
v1 / v2	6.25

The value of an expression can be converted to a different data type if desired. To do so the expression must be preceded by the name of the desired data type, enclosed in parentheses.

(Data type) expression

int a = 10;
int b = 3;
a / b becomes 3
(float)a / (float)b becomes 3.33
int 10 + 15.5 becomes 25

3.1.2. Unary Operator :

A unary operator performs operation on one operand. The unary operators are : ++, - -

Ex:

→ + + a or a+ + is equivalent to a = a + 1

→ - - a or a- - is equivalent to a = a - 1.

Here	+ +a	-	Pre-incrementation
	a+ +	-	post-incrementation
	- -a	-	pre-decrementation
	a- -	-	post-decrementation

- In Pre-incrementation the value is incremented first, then assigned.
- In Post-incrementation, the value is assigned first, then incremented.
- In Pre-decrementation, the value is decremented first, then assigned.
- In Post-decrementation, the value is assigned first, then decremented.

For example:

```
        int i = 5;
      printf ("%d\n",i);      →     5
printf ("%d\n",i++);  →     5
printf ("%d\n",i);      →     6
printf ("%d\n",++i);  →     7
printf ("%d\n",i);      →     7
printf ("%d\n",i--);   →     7
printf ("%d\n",i);      →     6
printf ("%d\n",--i);   →     5
```

3.2. RELATIONAL OPERATORS :

These operators maintain relation with two values and two expressions.

Operators	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

3.3. LOGICAL OPERATORS :

These operators maintain comparisons with two values and two expressions.

Operator	Meaning
&&	AND
||	OR
!	NOT

Assignment Operators :

In C there are different assignment operators available. They are: =, +=, -=, *=, /=, %=.

Syntax : identifier = expression;

Example :

```
a=3;
x=y;
sum=a+b;
    a + =  4;          // a = a+4;
```

```
a - =  4;            // a = a- 4;
a * =  4;            // a = a4;
a /=  4;             // a = a/4;
a %=  4;             // a = a%4;
```

The Conditional Operator or Ternary Operator Pair :

Simple conditions and operations can be carried out with the conditional operator (? :). An expression that makes use of the conditional operator is called conditional expressions.

Conditional expression is written in the form :

Exp1? Exp2: Exp3;

Exp1, Exp2, Exp3 are the general arithmetic expressions.

The operator "?:" works as follows.

Exp1 is evaluated first; if it is true then the Exp 2 is executed and becomes the value of the expression.

If Exp1 is false Exp3 is executed and becomes the value of the expression.

Note :

Only one of the expressions (either exp2 or exp3) is evaluated.

Ex: a=10 and b=15

X=(a>b)?a:b;

In this example X will be assigned a value of b, because b value is big.

This can be achieved using the if- else statement as follows.

```
If(a>b)
X=a;
Else
X=b;
```

Expressions :

An expression represents a single data item, such as a number or a character. It may also consist of a combination of entities interconnected by one or more operators.

```
Ex:         c=a+b
            x=y;
            x<=y
            x==y
            ++i;
```

3.4. STANDARD I/O IN C[(printf(), scanf()] :

3.4.1. printf( ) FUNCTION :

A standard output statement with which the user is able to display or view the result or output of a program is nothing but a **printf()** statement. **printf()** is a C library function. This function can be used to output any combination of numerical values, single characters, and strings.

Syntax : printf (control-string,arg1,arg2,...);

Where control string refers to a string that contains formatting information, and arg1, arg2 are arguments that represent individual output data items.

%c	Single character
%d	Decimal integer
%e	Floating point value
%f	Floating point value
%s	String
%x	Hexa decimal
%o	Octal

However, both **printf()** and **scanf()** are the standard input and output statements, they can be referred as formatted I/O statements or standard I/O.

Example:

```
#include<stdio.h>
main()
    {
        printf("HELLO HOW ARE YOU\n");
        printf("I AM FINE");
    }
```

Output :

```
HELLO HOW ARE YOU
I AM FINE
```

3.4.2. scanf( ) function :

In order to provide input to any program one has to use a standard input statement in C. Another way to input data into the computer is through a standard input device using C library function **scanf()**. This function can be used to enter any combination of numerical values, single characters and strings.

Syntax : scanf(control-string, arg1,arg2,.... argn);

Where control string refers to a string containing certain required formatting information and arg1, arg2...argn are arguments that represent the individual input data items.

%c Single character

%d	Decimal integer
%e	Floating point value
%f	Floating point value
%s	String
%x	Hexa decimal
%o	Octal

Each variable name must be preceded by an ampersand (&). The arguments are actually pointers that indicate where the data items are stored in the computer's memory.

Example :

```
int a;
float b;
char c;
scanf("%d%f%c",&a,&b,&c);
char name[20];
scanf("%s", name);
```

Example :

```
#include<stdio.h>
main()
  {
      int a,b;
      printf("enter the values of a and b\n");
      scanf("%d%d",&a,&b);
      printf("the entered values are\n");
      printf("a is %d \n b is %d", a, b);
  }
```

3.5. SUMMARY :

In this chapter we have learned the some C operators like arithmetic, relational, logical, increment and decrement. WE have leant the some Standard I/O in C functions in C.

3.6. KEY WORDS :

- **Operators :** An operator is a character or characters that determine the action that is to be performed or considered.
- **Synatx :** Syntax is *the grammatical structure of sentences*.

- **Standard I/O :** C programming language libraries that allow input and output in a program.
- **Scanf() :** The scanf() function is a commonly used input function in the C programming language.
- **Prinf() :** The printf() function sends a formatted string to the standard output (the display).

3.7. SELF ASSESSMENT QUESTIONS :

1. What is operator? List the various types of operators in C?S
2. Write the arithmetic operators in C
3. Compare the relational and logical operators in C
4. Describe the increment and decrement operators?
5. What are the commonly used I/O functions in C?
6. Explain scanf() function?
7. Discuss the printf()function?

3.8. FURTHER READINGS :

1. "Programming with C" by Byron C. Gotttfried.
2. "let us C" BY Yeshavanth kanethkar.
3. "working with C"BY Yeshavanth kanethkar .
4. "Programming *in ANSI* C" by E. Balagurusamy.
5. "Sams Teach Yourself C " by Tony Zhang.

LESSON – 4
STATEMENTS IN C

AIMS AND OBJECTIVES :

The main objective of this lesson is to know the effective programming techniques using conditional branching.

STRUCTURE OF THE LESSON :

4.1. CONDITIONAL BRANCHING (OR) DECISION MAKING :

Conditional branching is the most basic control feature of any programming language. It enables a program to make decisions, to decide whether or not to execute a statement or a block of statements based on the value of an expression. The expression may result in either true or false value. Since the value of the expression may vary from one execution to another, this feature allows a program to react dynamically to different data.

C supports various types of conditional branching statements. The following categories illustrate several conditional control structures.

1. Simple if
2. if-else
3. Nested if
4. else-if ladder

4.1.1.DECISION MAKING THROUGH SIMPLE if STATEMENT :

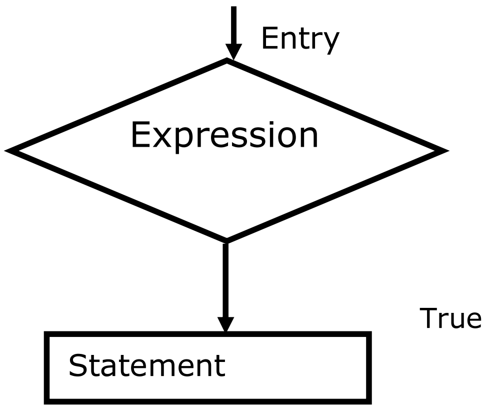
The **simple if** statement is a wonderful decision making statement and is used to control the flow of execution of a single or multiple instructions.

The general form of **“simple if”** follows :

If (condition/expression)

Statement;

In this statement the given condition is tested first and responds accordingly. If the result of expression is true then the given statement is executed. If the result is false the statement cannot be executed.



When multiple statements are to be executed using if control structure then it may be referred as compound if.

Syntax:

```
If (expression)
   {
       statement-block;
   }
statement-x;
```

The statement-block may be a single statement or a group of statements. If the expression is true statement-block will be executed, otherwise the statement-block will be skipped and the execution will jump to the statement-x.

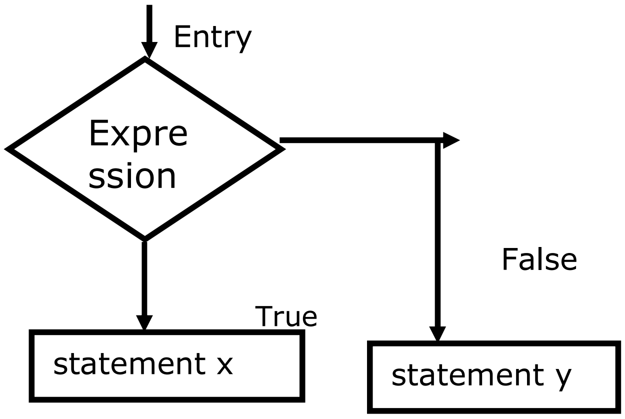
Example:

Program to find biggest of two numbers.

```
#include<stdio.h>
main()
   {
       int a, b;
       printf("\n\t Enter A value              : ");
       scanf("%d", &a);
       printf("\n\t Enter B value              : ");
       scanf("%d", &b);
       if (a>b)
               printf("\n %d is Greater than %d", a, b);
       if (b>a)
               printf("\n %d is Greater than %d", b, a);
   }
```

4.1.2. DECISION MAKING THROUGH if – else STATEMENT :

In **if-else** control statement there exists an extension of the simple if statement. It allows the user to perform another block of statements in case the condition result is false.



Flowchart

syntax :

```
if (expression)
        statement-x;
else
        statement-y ;
```

Here the **expression** is evaluated; if the result of the expression is a true then statement-x is executed otherwise statement-y will be executed.

Example:

Program to check whether given number is even or odd

```
#include<stdio.h>
main()
{
        int n;
        printf("\n Enter a number..:");
        scanf("%d",&n);
        if (n%2==0)
                printf("\n Given  number is even"):
        else
                printf("\n Given  number is odd"):
        getch();
}
```

4.1.3. DECISION MAKING USING nested if :

A **nested if** control structure consists of multiple if statements in one another. Here each if statement consists of subsequent branching statement. Literally a nested if consists of one if statement in another if statement. It is used when multiple conditions are to be evaluated.

Syntax:

```
if(expression)
   {
        if(expression)
```

```
{
      if(expression)
       {


             ---
             ---
```

Here evaluations of expressions or conditions are based on the first condition. If the first condition itself is false, then there is no evaluation of other conditions. At any level of expression the program control may be altered.

Example :

A Program to find the Biggest of 3 numbers using nested if

```
#include<stdio.h>
main()
   {
       int a,b,c,big;
       printf("\n Enter the value of a  :   ");
       scanf("%d",&a);
       printf("\n Enter the value of b  :   ");
       scanf("%d",&b);
       printf("\n Enter the value of c  :   ");
       scanf("%d",&c);
       if (a>b)
             if (a>c)
                   big = a;
             else
                         big = c;
             else
             if (b>c)
                   big = b;
             else
                   big = c;
        printf("\nBiggest of three numbers is:%d”,big);
       }
```

4.1.4. DECISION MAKING THROUGH else-if LADDER :

In **else if ladder** number of conditions are checked depending on the falsity of the previous condition. Literally, too many conditions are evaluated in if else ladder.

Syntax:

```
If <condition1>
{
    ------
}
else if <condition2>
{
    ---- True block 1
}
else
{
    ------ False block
}
```

In this, condition1 is checked and if it is true then its corresponding condition is executed. If the condition is false then next condition is verified. If all the given conditions are false then false block is executed. Only one of all the available blocks gets executed. After the execution of any one of the blocks, control is transferred to next statement after the construct.

Example:

A Program to find biggest of three numbers

```
#include<stdio.h>
main()
  {
       int a,b,c;
       clrscr();
       printf("enter three numbers:");
       scanf("%d%d%d",&a,&b,&c);
       if(a>b)
```

```
                if(a>c)
                        printf(“%d is big”,a);
                else
                        printf(“%d is big”,c);
        else if(b>c)
                printf(“%d is big”,b);
        else
                printf(“%d is big “,c);
    }
```

Example :

A Program to award grade to a student.

```
#include<stdio.h>

main()

    {

            int marks;
            printf("enter the marks\n");
            scanf("%d",&marks);
            if(marks>79)
                        printf(" HONOURS\n");
            else if(marks>59)
                        printf("FIRST");
            else if(marks>49)

                        printf("SECOND");
            else if(marks>39)
                         printf("THIRD");
            else
                        printf("FAIL");
    }
```

4.2. SWITCH STATEMENT :

C provides a special kind of conditional control structure that acts as an alternative to **if else ladder**. When there are more conditions or paths in a program, if-else branching can become more difficult. In such situations switch may act better. The **switch** statement allows the user to specify an unlimited number of execution paths based on the value of a single expression.

Each execution path is referred as a case. However, all the cases should be unique. Each case must be terminated by a '**break**' statement. The '**default'** case is not mandatory.

In a switch statement, there are four different keywords to be used :

- **switch**
- **case**
- **break**
- **default**

Though the switch control structure enables the user to improve clarity of the program, it causes more errors. So, it requires more attention while implementation.

Syntax:

```
switch(expression)
       {
        case  value1:
                           statement;
         break;
         case  value2:
                           statement;
          break;
           :
           :
           :
           default :
                           statement;
       }
```

Among all the cases, only one case can be executed successfully because each case is terminated by a **'break'** statement.

Example :

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

Output : 1 case

```
2 case
3 case
default
```

Example :

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

Output : 1 case

Example :

A Program to accept one integer value and print the related week day.

```
#include<stdio.h>
main()
    {
        int n;
        printf("enter the value of n:\n");
        scanf("%d",&n);
        switch(n)
            {
                case 1:
                      printf("Monday");
                   break;
                case 2:
                      printf("tueseday");
                   break;
                case 3:
                      printf("Wednesday");
                   break;
                case 4:
                      printf("thurseday");
                   break;
                case 5:
                      printf("friday");
                   break;
                case 6:
                      printf("saturday");
                   break;
                case 7:
                      printf("sunday");
                   break;
            }
    }
```

Example :

A Program to accept two integer values and perform arithmetic operation by getting the user input.(1) Addition, 2) Subtraction, 3) Multiplication, 4) Division and 5) Exit).

```
#include<stdio.h>
main()
    {
          int a, b, c, ch;
          clrscr();
          printf("\n\t\t\t Enter two numbers :        ");
          scanf("%d %d", &a, &b);
          printf("Enter your choice:")l
          printf("1)Addition\n2)Subtraction");
          printf("\n3)Multiplication");
          printf("\n4) Division. \n5) Exit").
          scanf("%d"&ch);
          switch (ch)
                  {
                        case 1:
                                 c = a + b;
                             break;
                        case 2:
                                 c = a - b;
                             break;
                        case 3:
                                 c = a * b;
                             break;
                        case 4:
                                 c = a / b;
                             break;
                        default :
                        printf("\n Invalid option  ");
                        exit(0);
          }
       printf("\n\t\t\t Result   :%d",c);
    }
```

4.3. SUMMARY :

In this we have learnt about the conditional looping and unconditional statements. You have also seen conditional statements, if, if-else, nested If and switch. Usage of break, continue, goto and exit statements.

4.4. KEY WORDS :

- **Conditional Control statements :**

  Use if to specify a block of code to be executed, if a specified condition is true.

- **Simple if :**

  If the given condition is true then the statements inside the body of "if" will be executed.

- **If-else :**

  if-else statement is used to perform the operations based on some specific condition. If the given condition is true, then the code inside the if block is executed, otherwise else block code is executed.

- **Nested if :**

  A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement.

- **Else-if ladder** :

  The conditional expressions are evaluated from the top downward. As soon as a true condition is found, the statement associated with it is executed, and the rest of the ladder is bypassed. If non of the conditions is true, then the final else statement will be executed.

- **Switch case statements :**

  A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

4.5. SELF ASSESSMENT QUESTIONS :

1. What are the conditional control statements in C?
2. Explain Switch and Case statements with example?
3. Write the break and default statements with example?

4.6. FURTHER READINGS :

1. "Programming with C" by Byron C.Gotttfried.
2. "let us C" BY Yeshavanth kanethkar.
3. "working with C"BY Yeshavanth kanethkar.
4. "Programming *in ANSI* C" by E.Balagurusamy.
5. "Sams Teach Yourself C " by Tony Zhang.
6. "The Spirit of C" BY Henry Mullish.

LESSON - 5

LOOPS IN C

AIMS AND OBJECTIVES :

The aim(s) of this lesson is to learn about a very important feature of the C language called looping. Looping also called iteration, it is used in programming to perform the same set of statements over and over until certain specified conditions are met.

STRUCTURE OF THE LESSON :

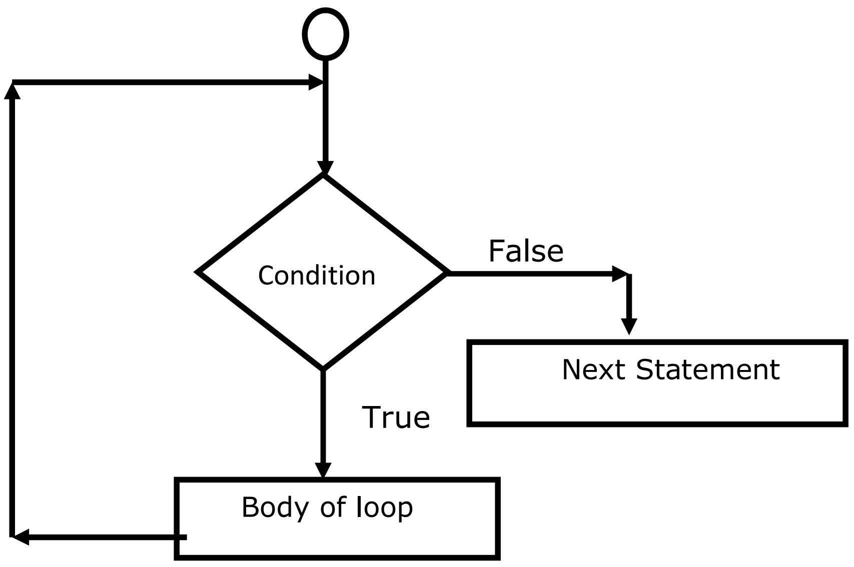
5.1. Repetitive Statements (while, do while and for Loops) :

5.1.1. While Statement :

The **while** control structure executes a single or multiple statements for repeated number of times based on a given condition. It executes the statements as long as the given condition or expression results in a true value. It terminates execution as and when the condition is false.

Syntax :

```
initialization statement;
while(condition)
  {
     …
     Condition reachable statement;
  }
```



Flowchart :

Here the condition is tested every time, it executes the block of statements. The keyword **while** verifies the trueness and falsity of the expression and responds accordingly. If the condition is false for the first time the minimum number of iterations is 0 in while control structure. It requires three statements in order to perform repetitive tasks.
They are

- Initialization statement
- Conditional statement
- Condition reachable statement

If any of the above statements is ignored then the while may not perform well.

Examples on while :

Example 1 :

A program to print natural numbers from 1 to 10 .

```
#include<stdio.h>
main()
     {
          int i;
          i=1;
          while (i<=10)
               {
                    printf("%d\n", i);
                    i++;
               }
     }
```

Example 2 :

A Program to find whether the given number is prime or not.

```
#include<stdio.h>
#include<conio.h>
main()
    {
        int i,n,c;
        clrscr();
        printf("enter n vale");
        scanf("%d",&n);
        i=1;
        c=0;
        while(i<=n)
            {
                if(n%i==0)
                c++;
                i++;
            }
        if(c==2)
                printf("%d is prime",n);
        else
                printf("%d is not prime",n);
        getch();
    }
```

Example 3:

A Program to find large and small numbers with in the given numbers.

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

```
    {
           int num,n,i,lar=0,small=32767;
           clrscr();
           printf("enter the range:\n");
           scanf("%d",&n);
           i=1;
           while(i<=n)
              {
                     printf("enter element\n");
                     scanf("%d",&num);
                     if(num>lar)
                     lar=num;
                     if(num<small)
                     small=num;
                     i=i+1;
               }
            printf("large=%d    small=%d",lar,small);
    }
```

Example 4 :

A Program to find the given number is an automorphic number or not.

```
#include<stdio.h>
#include<conio.h>
main()
    {
          int m,n,nsq,den=1;
          clrscr();
          printf("enter the value of n:\n");
          scanf("%d",&n);
          m=n;
          nsq=n*n;
```

```
        while(m>0)
            {
                    den=den*10;
                    m=m/10;
            }
        if((nsq%den)==n)
              printf("%d is an automarphic",n);
        else
              printf("%d is not an automarphic",n);
    }
```

Example 5 :

A Program to print strong numbers up to some specified rage.

```
#include<stdio.h>
main()
    {
        int j,sum,i,n,fact,d,temp;
        clrscr();
        printf("enter the value of n:\n");
        scanf("%d",&n);
        for(j=1;j<n;j++)
            {
                sum=0;
                temp=j;
                while(temp>0)
                    {
                        d=temp%10;
                        fact=1;
                        for(i=1;i<=d;i++)
                        fact=fact*i;
                        sum=sum+fact;
                         temp=temp/10;
                    }
                if(sum==j)
                    printf("%d is strong number\n",j);
            }
    }
```

5.1.2. do - while Statement :

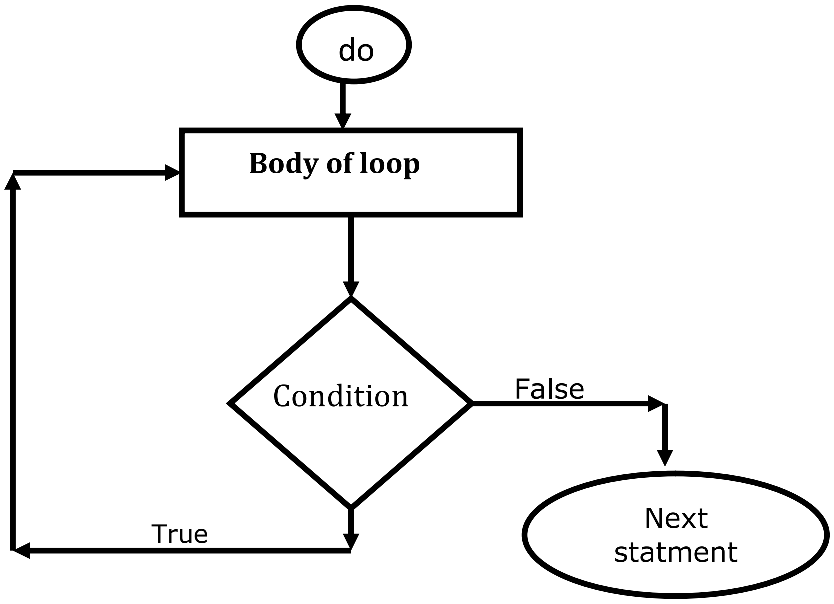
C provides another form of while control structure i.e., **do-while** control structure. In **do-while** control structure the statements in the block get execute first, later on the condition is evaluated. Hence the user can assume that the minimum number of iterations for do while control structure as 1, even if the expression or condition results in false for the first time.

Syntax:

```
Initialization statement;
do

{

     ---------        ----

     Condition reachable statement;

 } while(condition);
```

Flowchart :

Here the statements in the loop will be executed until the given condition becomes false. The while statement should be terminated by a semicolon (;) in do while.



Examples on do-while Statement :

Example 1 :

A Program to find sum of given integers.

```
#include<stdio.h>

#include<conio.h>

main()
```

```
    {
        int n,sum=0,i=0;
        printf("enter any number\n");
        scanf("%d",&n);
        do
            {
                sum=sum+i;
                i=i+1;
            }
        while(i<=n);
         printf("%d",sum);
    }
```

Example 2 :

A Program to find sum of n even numbers.

```
 #include<stdio.h>
 #include<conio.h>
main()
    {
        int n,sum=0,i=0;
        clrscr();
        printf("enter any number\n");
        scanf("%d",&n);
        do
           {
               sum=sum+i;
               i=i+2;
           }
        while(i<=n);
        printf("%d",sum);
    }
```

Example 3 :

A Program to find reverse of the given integer.

```
#include<stdio.h>

#include<conio.h>

main()

   {
        int n,rem=0,k;
        clrscr();
        printf("enter any number\n");
        scanf("%d",&n);
        do
            {
                k=n%10;
                rem=rem*10+k;
                n=n/10;
            }
        while(n>0);
        printf("%d",rem);
   }
```

Example 4 :

A Program to find largest and smallest of given numbers.

```
#include<stdio.h>

#include<conio.h>

main()

   {
        int i,n,num,lar=0,small=32767;
        clrscr();
        printf("enter the range:\n");
        scanf("%d",&n);
        i=1;
        do
           {
                printf("enter an element\n");
                scanf("%d",&num);
                if(num>lar)
                lar=num;
                if(num<small)
                small=num;
                i=i+1;
            }
        while(i<=n);
        printf("large=%d\nsmall=%d",lar,small);
    }
```

Example 5 :

A Program to find lcm and gcd of given numbers.

```
 #include<stdio.h>
 #include<conio.h>
main()
    {
        int m,n,a,rem,lcm;
        clrscr();
        printf("enter two values\n");
        scanf("%d%d",&m,&n);
        a=m*n;
        do
            {
                rem=n%m;
                n=m;
                m=rem;
            }
        while(m>0);
        printf("gcd = %d\n",n);
        lcm=a/n;
        printf("lcm=%d",lcm);
    }
```

Example 6 :

A Program to find NCR.

```
 #include<stdio.h>
 #include<conio.h>
main()
     {
            int n,r,s,i,ncr,fact1,fact2,fact3;
            clrscr();
            printf("enter the value of n and r:\n");
            scanf("%d%d",&n,&r);
            s=(n-r);
            fact1=1;
            i=1;
            do
                  {
                        fact1=fact1*i;
                        i++;
                  }
            while(i<=n);
            printf("fact1  is %d\n",fact1);
             fact2=1;
            i=1;
            do
                  {
                        fact2=fact2*i;
                        i++;
                  }
            while(i<=r);
            printf("fact2 is %d\n",fact2);
            fact3=1;
            i=1;
            do
                  {
                        fact3=fact3*i;
```

```
                i=i+1;
            }
        while(i<=s);
        printf("fact3 is %d\n",fact3);
        ncr=fact1/(fact2*fact3);
        printf("the ncr is  %d",ncr);
    }
```

5.2. for LOOP :

C provides a more flexible form of looping control structure that improves clarity of the code. It is for control structure. Usually, for control statement is used to perform fixed number of iterations. The major difference between **for** and **other looping structures** is the number of iterations. In case of while and do-while the number of iterations are indefinite. The user may not predict the number of iterations. On the other hand for specifies the number of iterations in the statement itself.

Syntax*:*

```
for(initialization; test condition; increment/decrement part)
    {
        Body of the loop;
    }
```

The initialization may contain single or multiple assignment statements. A control variable is involved in this part of statements.

The test condition verifies the validity of the control variable for each iteration.

Increment or decrement part increments or decrements the value of the control variable in order to reach the test condition.

Examples on for :

Example 1 :

A Program to find the given number is perfect number or not.

```
 #include<stdio.h>
main()
    {
        int sum=0,num,i,n;
        clrscr();
        printf("enter the number:\n");
        scanf("%d",&num);
        for(i=1;i<num;i++)
          {
              if(num%i ==0)
              sum=sum+i;
           }
              if(sum==num)
        printf("%d is perfect number",num);
        else
        printf("%d is not perfect number",num);
    }
```

Example 2 :

A Program to print armstrong numbers up to some range.

```
#include<stdio.h>
main()
  {
        int sum=0,rem,k,n,m;
        clrscr();
        printf("enter the number:\n");
        scanf("%d",&n);
        for(m=1;m<=n;m++)
          {
              sum=0;
               k=m;
              while(k>0)
                {
                    rem=k%10;
                    sum=sum+rem*rem*rem;
                    k=k/10;
                }
              if(sum==m)
              printf("%d is the amstrong number\n ",m);
          }
  }
```

Example 3 :

A Program to print largest of n numbers.

```
 #include<stdio.h>
main()
   {
        int num,n,larg,i;
        clrscr();
        printf("enter the total number  of values:\n");
        scanf("%d",&n);
        printf("enter the numbers:\n");
        scanf("%d",&num);
        larg=num;
        for(i=0;i<n-1;i++)
          {
                printf("enter an element:\n");
                scanf("%d",&num);
                if(larg<num)
                larg=num;
          }
        printf("large number is  %d",larg);
   }
```

Example 4 :

A Program to print multiplication table.

```
 #include<stdio.h>
 #include<conio.h>
 #definecolmax 10
 #definerowmax 12
main()
   {
        int row,column,y;
        clrscr();
        printf("\tMULTIPLICATION TABLE \t\n");
        printf("-----------------------------------------\n");
```

```
            for(row=1;row<=rowmax;row++)
            {
                  for(column=1;column<=colmax;column++)
                        {
                              y=row*column;
                              printf("%4d",y);
                        }
                  printf("\n");
            }
            printf("-------------------------------------------\n");
        }
```

Example 5:

A Program to print triangle in this shape.

```
                                        *
                                       * *
                                      * * * *
```

```
 #include<stdio.h>
 #include<conio.h>
main()
   {
        int i,j,n,k;
        clrscr();
        printf("enter n value  :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
           {
              for(k=0;k<n-i;k++)
              printf(" ");
              for(j=0;j<=i;j++)
                 {
                    printf("*");
                    printf(" ");
                 }
              printf("\n");
              }
        getch();
   }
```

5.3. USE OF break AND continue STATEMENTS :

5.3.1. break :

This statement takes control out of the switch statement or loop structure. In other words, a break statement takes the control out of the current block in execution. The control is transferred to the statement that follows the block.

Syntax : break;

5.3.2. Continue Statement :

To skip a part of the body of the loop in execution on certain condition and for the loop to be continued for the next iteration continue statement is used.

Syntax : continue;

5.4. SUMMARY :

This chapter you have learned the following like Looping can be used to perform the same set of statements over and over until specification conditions are met. Looping makes your program concise. There are three statements, for, while and do-while, that are used for looping in C and also some useful information from the break and continue statements.

5.5. KEY WORDS :

- **Loops :**

  Loop is used to execute the block of code several times according to the condition given in the loop.

- **while :**

  The while statement lets you repeat a statement until a specified expression becomes false.

- **do-while :**

  The do-while statement lets you repeat a statement or compound statement until a specified expression becomes false.

- **for loop :**

  The for loop in C language is used to iterate the statements or a part of the program several times.

- **break and continue statements :**

  Break statement stops the entire process of the loop. Continue statement only stops the current iteration of the loop.

5.6. SELF ASSESSMENT QUESTIONS :

1. What is looping? Explain various types of looping in C programming?
2. Explain the various loop constructs in available in C language?
3. How can the do-while loop vary from the while-loop?
4. What are breaking control statements?

5.7. FURTHER READINGS :

1. “Programming with C” by Byron C.Gotttfried
2. “let us C” BY Yeshavanth kanethkar
3. “Working with C”BY Yeshavanth kanethkar
4. “Programming *in ANSI* C” by E.Balagurusamy
5. “Sams Teach Yourself C “ by Tony Zhang
6. “The Spirit of C” BY Henry Mullish

LESSON - 6
A R R A Y S

AIMS AND OBJECTIVES :

The objectives of this lesson are to

- ➢ Explain the powerful data type called array(s).
- ➢ Various types of arrays : one – dimensional, two – dimensional and multi – dimensional arrays.
- ➢ How an array can be declared and initialized.

STRUCTURE OF THE LESSON :

6.1. INTRODUCTION - TYPES OF ARRAYS :

Many applications require the processing of multiple data items, which are having common characteristics. In such situations it is often convenient to store the data items in an **array**.

An **array** is a group of related data items that are stored under a common name (or) an **array** is a collection of identical (similar) type of variables stored continuously in memory. Each item in an **array** is called an element. All elements together referred by a name **array** and are stored in a set of continuous memory slots.

6.1.1. Declaring an Array :

The general form of declaring an **array** is as follows :

Syntax : storage-class data-type array-name[array-size];

Here storage-class refers to the storage class of the array, data-type indicates type of the declared array, array-name indicates the name of the array and array-size indicates how many elements that the array can contain. The storage class is optional. Default values are **automatic** for arrays defined within the function or a block, and **external** for arrays defined outside of a function. Brackets ([ ]) are required in declaring the size of an array. The brackets pair is also called the array subscript operator.

Examples :

```
int x[8];

char text[20];

static float n[10];
```

Where static specifies the storage-class. int, char, float specifies the data types of the arrays whose names are x, text, n. The sizes of the arrays are 8,20 and 10 respectively. Arrays in C language have index starting at **0**. That means x contains the elements from **0 to (n-1).**

6.1.2. Initializing an Array :

With the help of an array, we can initialize each element in an array. For instance, we can initialize the first element in the array of day, which is as follows.

Char day[6]="sunday";

The result of the above declaration is not same because of null character '\0', which is automatically added to the end of the string as follows.

```
day[0]='s'

day[1]='u'

day[2]='n'

day[3]='d'

day[4]='a'

day[5]='y'

day[6]='\0'.
```

The second way to initialize an array is to initialize all elements in the array together. For example, the following declaration initializes an integer array.

int y[5]={100,25,89,23,56};

Here the integers inside the braces are assigned to the corresponding elements of the array i.e., 100 is given to the first element (y[0]),25 is given to the second element (y[1]), and so on.

An example of initializing an array :

```
    #include<stdio.h>
    main()
        {
            int i;
            int x[5]={100,25,89,23,56};
            for(i=0;i<5;i++)
                {
                    printf("x[%d] is initialized with %d\n", i, x[i]);
                }
         }
```

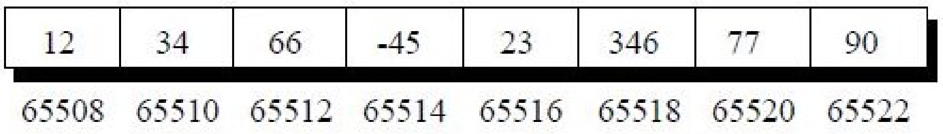
6.1.3. Array Elements in Memory :

When we declare an array in C, they can reserved the memory immediately as per there size.

Example : int arr[8] ;

It can reserved 16 bytes in memory, 2 bytes each for the 8 integers (under Windows/Linux the array would occupy 32 bytes as each integer would occupy 4 bytes). And since the array is not being initialized, all eight 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.

Diagram to show the functionality of the for loop :



6.2. ONE DIMENSIONAL ARRAYS :

A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or one-dimensional array. A single-dimensional array is a group of elements that share a common name and differentiated from one another by their positions within the array.

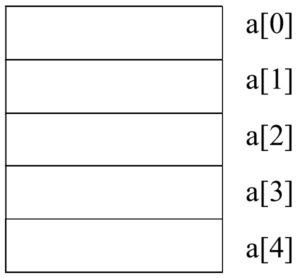
6.2.1.Declaring arrays :

Arrays must be declared before they are used. Every element in the array is manipulated using its index. The starting index of element is 0 and ends with n-1. A list of data items can be given one variable name using one subscript and such a variable is called one-dimensional array. The general form of array declaration is.

Syntax : data-type variable-name[size];

Example : int a[5];

Here index represents five integers, this complete set is called an array. Here, **int** specifies the data-type of the variable, the word **a** specifies the name of the variable and the number 5 specifies how many elements of the type **int** will be in the array. This number is often called the "dimension" of the array. The bracket [] specifies the indication dealing with the arrays. The computer reserves 5 storage locations as shown below.



6.2.2.Initializing :

We can initialize the elements in an array in the same way as the ordinary variables when they are declared. The general from is

Syntax : data-type array-name [size] = {list of values};

Example : int a[5] = { 1, 2, 3, 4, 5 };

In this form the size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. In the above example, compiler allocates space for five elements only.

char name[ ] = { 'a', 'b', 'c', 'd'};

Thus the above statement declares the name array of four characters, initialized with the string "abcd".

6.2.3. Array Elements in Memory

Consider the following array declaration.

int a[8];

This declaration reserves 16 bytes in memory. Because of each 8 integers would be 2 bytes long. Since the array is not initialized, all the 8 values present in it would be garbage values. Whatever be the initial values, all the array elements would always be present in continues memory location. This arrangement of array elements in memory is shown as follows.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
12	34	66	-45	23	346	77	90
4002	4004	4006	4008	4010	4012	4014	4016

Example 1 :

A Program to accept 5 elements into an array and display them back.

```
#include<stdio.h>
main()
   {
       int a[5],i;
       clrscr();
       for(i=0;i<5;i++)
          {
              printf("\n Enter the value of a[%d] :",i);
              scanf("%d",&a[i]);
          }
       printf("\n");
       for(i=0;i<5;i++)
          {
              printf("\t%d",a[i]);
          }
   }
```

Example 2 :

A Program to find out sum and average of the given elements.

```
#include<stdio.h>
main()
   {
       int a[10],i,n,sum=0,avg;
       printf("enter the range of n:\n");
       scanf("%d",&n);
       printf("enter %d numbers:\n",n);
       for(i=0;i<n;i++)
          {
              scanf("%d",&a[i]);
              sum=sum+a[i];
          }
       avg=sum/n;
       printf("sum=%d\n",sum);
```

```
        printf("average=%d",avg);
    }
```

Example 3 :

A Program to find out smallest and largest of the given elements

```
#include<stdio.h>
main()
   {
     int a[10],i,n,small,lar;
     printf("enter the range of n:");
     scanf("%d",&n);
     printf("enter %d numbers:",n);
     small=32767;
     lar=0;
     for(i=0;i<n;i++)
       {
         scanf("%d",&a[i]);
         if(a[i]<small)
         small=a[i];
         if(a[i]>lar)
         lar=a[i];
       }
     printf("large=%d \n small=%d \n",lar,small);
   }
```

Example 4 :

A Program to arrange elements in ascending order

```
#include<stdio.h>
main()
   {
     int a[10],n,i,j,temp;
     printf("enter the size of array:\n");
     scanf("%d",&n);
     printf("enter array elements\n");
     for(i=0;i<n;i++)
     scanf("%d",&a[i]);
```

```
        for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)

        if(a[i]>a[j])
           {
               temp=a[i];
               a[i]=a[j];
               a[j]=temp;
           }
        printf("elements in ascending order\n");
        for(i=0;i<n;i++)
        printf("%d\n",a[i]);
     }
```

Example 5 :

A Program to implement the linear search

```
#include<stdio.h>
main()
    {
        int a[10],n,i,x,found,pos;
        printf("enter the range:\n");
        scanf("%d",&n);
        printf("enter %d numbers\n",n);
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        printf("enter the searching element");
        scanf("%d",&x);
        for(i=0;i<n;i++)
           {
                if(a[i]==x)
                    {
                        found=1;
                        pos=i;

                    }
```

```
            }
        if(found==1)
            printf("element found in position %d\n",pos);
        else
            printf("element not found\n");
    }
```

Example 1:

A Program to accept 5 elements into an array and display them back.

```
#include<stdio.h>
main()
    {
        int a[5],i;
        clrscr();
        for(i=0;i<5;i++)
            {
                printf("\n Enter the value of a[%d] :",i);
                scanf("%d",&a[i]);
            }
        printf("\n");
        for(i=0;i<5;i++)
            {
                printf("\t%d",a[i]);
            }
    }
```

Example 2:

A Program to find out sum and average of the given elements.

```
#include<stdio.h>
main()
        {
            int a[10],i,n,sum=0,avg;
            printf("enter the range of n:\n");
            scanf("%d",&n);
            printf("enter %d numbers:\n",n);
            for(i=0;i<n;i++)
              {
                  scanf("%d",&a[i]);
                  sum=sum+a[i];
              }
            avg=sum/n;
            printf("sum=%d\n",sum);
            printf("average=%d",avg);
        }
```

Example 3 :

A Program to find out smallest and largest of the given elements

```
#include<stdio.h>
main()
        {
            int a[10],i,n,small,lar;
            printf("enter the range of n:");
            scanf("%d",&n);
            printf("enter %d numbers:",n);
            small=32767;
            lar=0;
            for(i=0;i<n;i++)
              {
                  scanf("%d",&a[i]);
                  if(a[i]<small)
                  small=a[i];
                  if(a[i]>lar)
                  lar=a[i];
              }
            printf("large=%d \n small=%d \n",lar,small);
        }
```

Example 4:

A Program to arrange elements in ascending order

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

```
{
        int a[10],n,i,j,temp;
        printf("enter the size of array:\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
        if(a[i]>a[j])
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
        printf("elements in ascending order\n");
         for(i=0;i<n;i++)
        printf("%d\n",a[i]);
 }
```

Example 5 :

A Program to implement the linear search

```
#include<stdio.h>
main()
        {
                int a[10],n,i,x,found,pos;
                printf("enter the range:\n");
                scanf("%d",&n);
                printf("enter %d numbers\n",n);
                for(i=0;i<n;i++)
                scanf("%d",&a[i]);
                printf("enter the searching element");
                scanf("%d",&x);
                for(i=0;i<n;i++)
                        {
```

```
                        if(a[i]==x)
                                {
                                        found=1;
                                        pos=i;
                                }
                        }
                        if(found==1)
                         printf("element found in position %d\n",pos);
                        else
                        printf("element not found\n");
        }
```

6.3.Two Dimensional Arrays Declaration of two dimensional arrays :

So far we have discussed the array variables that can store a list of values. There will be situations, where a table of values will have to be stored. For example :

1	2	3
4	5	6
7	8	9

C allows us to define such table of items by using two-dimensional arrays. This type of array can be declared as

Syntax : data-type array-name[row-size][column-size];

Example : int a[3][3];

This two dimensional array consists of three rows and three columns.

The first row first element can be accessed by a[0][0], the first row second element can be accessed by a[0][1], similarly the third row last by a[2][2].

6.3.1. Declaring And Initializing :

Like single dimensional arrays the initialization can be done in two-dimensional arrays. The general form of two-dimensional array is as follows.

Syntax : Type array-name[row-size][column-size];

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

This declares an array with five rows and three columns, but only the first three rows are initialized, and only the first element of the second row is initialized.

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

if we do not include the inner brackets in the declaration as follows

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

The result is

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

6.3.2.Array Elements in Memory :

Consider the following array declaration.

int a[3][3];

This declaration reserves 18 bytes in memory. Because each of 9 integers would be 2 bytes long. In two-dimensional arrays the memory doesn't contain rows and columns. In memory whether it is one-dimensional or a two-dimensional array, the array elements are stored in one continuous chain. The arrangement of two dimensional array elements in memory are as shown below.

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]	a[2][0]	a[2][1]	a[2][2]
1234	34	1212	33	1434	80	1312	78	1569
4002	4004	4006	4008	4010	4012	4014	4016	4018

Example 1 :

A Program to convert the elements in a matrix order

```
#include<stdio.h>
main()
        {
                int a[10][10],i,j,n,m;
                printf("enter the size of the matrix:\n");
                scanf("%d%d",&n,&m);
                printf("enter the elements\n");
                for(i=0;i<n;i++)
                        {
                                for(j=0;j<m;j++)
                                scanf("%d",&a[i][j]);
                        }
                printf("the elements are:\n");
                for(i=0;i<n;i++)
                        {
                                for(j=0;j<m;j++)
                                printf("%4d",a[i][j]);
                                printf("\n");
                        }
        }
```

Example 2:

A Program to perform matrix addition

```
#include<stdio.h>
main()
        {
                int a[10][10],b[10][10],c[10][10],i,j,n,m,p,q;
                printf("enter the size of the matrix:\n");
                scanf("%d%d",&n,&m);
                printf("enter the 1 matrix elements\n");
                for(i=0;i<n;i++)
                        {
                                for(j=0;j<m;j++)
                                scanf("%d",&a[i][j]);
                        }
                printf("enter the 2 matrix elements\n");
                for(i=0;i<n;i++)
                        {
                                for(j=0;j<m;j++)
                                scanf("%d",&b[i][j]);
                        }
                printf("sum of two matrices:\n");
                for(i=0;i<n;i++)
                        {
                                for(j=0;j<m;j++)
                                        {
                                                c[i][j]=a[i][j]+b[i][j];
                        printf("%3d",c[i][j]);
                    }
                printf("\n");

            }
    }
```

Example 3 :

A Program to perform matrix multiplication

```
#include<stdio.h>
main()
    {
        int a[10][10],b[10][10],c[10][10],i,j,k,n,m,p,q;
        printf("enter the size of 1 matrix:\n");
        scanf("%d%d",&m,&n);
        printf("enter the size of 2 matrix:\n");
        scanf("%d%d",&p,&q);
        if(n==p)
            {
                printf("enter the 1 matrix elements\n");
                for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                scanf("%d",&a[i][j]);
                printf("enter the 2 matrix elements\n");
                for(i=0;i<p;i++)
                for(j=0;j<q;j++)
                scanf("%d",&b[i][j]);
                printf(" matrix multiplication:\n");
                for(i=0;i<m;i++)
                for(j=0;j<q;j++)
                    {
                        c[i][j]=0;
                        for(k=0;k<n;k++)
                        c[i][j]=c[i][j]+a[i][k]*b[k][j];
                        printf("%3d",c[i][j]);
                        }
                  printf("\n");
            }
      else
            printf("multiplication not possible");
}
```

6.4. MULTI- DIMENSIONAL ARRAYS :

C allows arrays of three (or) more dimensions. The exact limit depends on the compiler. C allows multi-dimensional arrays. The general form of multi-dimensional arrays is as follows.

Syntax : data-type array-name[exp 1][exp 2]….[exp n];

Expressions with multiple subscripts refer to elements of "multidimensional arrays". A multidimensional array is an array whose elements are arrays.

So **Three-dimensional array** is an array with two dimensions.

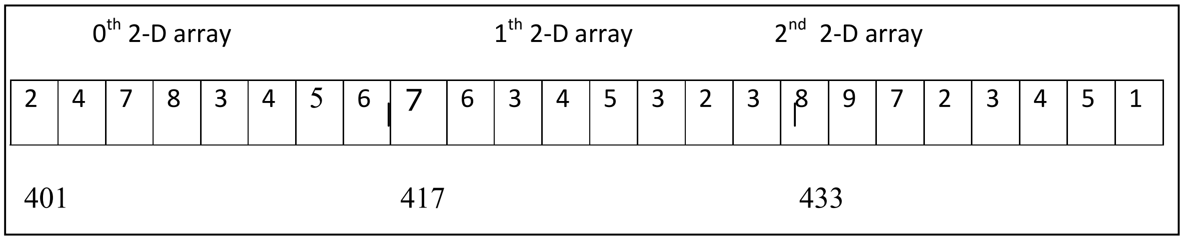
The general form of a three-dimensional array is as follows.

Syntax : data-type array name[exp 1][exp 2][exp 3];

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

A three dimensional array is an array of arrays of arrays. In the above example, the outer array has three elements, each of which is a two dimensional array of four one-dimensional arrays.

In three -dimensional arrays, the array elements are stored in one continuous chain. The arrangement of elements of a three-dimensional array in memory as shown below.



6.5. SUMMARY :

In this Chapter we have learned the An array is a data structure that can be used to store a list of values of same data type. In C, the index to an array starts at 0. We can initialize each individual element of an array after the declaration of the array, or you can place all initial values into a data block surrounded by (and) during the declaration of an array. The memory storage taken by an array is determined by the product of the size of a data type and the dimensions of the array.

6.6. KEY WORDS :

Dimension : A measure of spatial extent, especially width, height, or length.

Array : An array is a variable that can store multiple values.

Character array : A character array is a sequence of characters, just as a numeric array is a sequence of numbers.

Array storage : The Array Store is a store that provides an interface for loading and editing an in-memory array and handling related events.

Memory : Memory in a computer is just a sequential set of "buckets" that can contain numbers, characters, or boolean values

6.7. SELF ASSESSMENT QUESTIONS :

1. What is an array? Explain different types of arrays?
2. Write a program to read and display a two dimensional array?
3. Explain multidimensional arrays with example?

6.8. FURTHER READINGS :

1. Programming with C" by Byron C.Gotttfried.
2. "let us C" BY Yeshavanth kanethkar.
3. "working with C"BY Yeshavanth kanethkar.
4. "Programming in ANSI C" by E.Balagurusamy
5. "Sams Teach Yourself C " by Tony Zhang.
6. "The Spirit of C" BY Henry Mullish.

LESSON - 7
S T R I N G S

AIMS AND OBJECTIVES :

The objectives of this lesson are to know about

- ➢ the concept of string
- ➢ the significance of strings
- ➢ various applications of string functions.

STRUCTURE OF THE LESSON :

7.1. STRING :

A string in C is simply an array of characters. The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

7.2. DECLARATION AND INITIALIZATION OF STRING VARIABLES :

String Declaration :

String is not a basic data type in C programming language. It is just a null terminated array of characters. Hence, the declaration of strings in C is similar to the declaration of arrays.

Like any other variables in C, strings must be declared before their first use in C program.

Syntax of String Declaration : char string_name[SIZE_OF_STRING];

the above declaration, string_name is a valid C identifier which can be used later as a reference to this string. Remember, name of an array also specifies the base address of an array.

SIZE_OF_STRING is an integer value specifying the maximum number of characters which can be stored in this string including terminating null character. We must specify size of string while declaration if we are not initializing it.

Examples of String Declaration

- char address[100];
- char welcome Message[200];

Initialization of Strings :

In C programming language, a string can be initialized at the time of declarations like any other variable in C. If we don't initialize an array then by default it will contain garbage value.

There are various ways to initialize a String Initialization of string using a character array

char name[16] = {'T','e','c','h','C','r','a','s','h','C','o','u','r','s','e','\0'};

or

char name[] = {'T','e','c','h','C','r','a','s','h','C','o','u','r','s','e','\0'};

In the above declaration individual characters are written inside single quotes('') separated by comma to form a list of characters which is wrapped in a pair or curly braces. This list must include a null character as last character of the list.

If we don't specify the size of String then length is calculated automatically by the compiler.

7.3. STRING HANDLING FUNCTIONS :

In C programming language provides a set of pre-defined functions called **string handling functions** to work with string values. Whenever we want to use any string handling function we must include the header file called **string.h**.

The following are the basic string functions. All the string functions are prototyped in: #include <string.h>.

strcmp	Compares one string with another.
strcpy	Copies a string to another string.
strlen	Determines the length of a string.
strlwr	Converts a string to lowercase.
strupr	Converts a string to uppercase.
strcat	Appends one string at the end of another.
strrev	Reverses a string.
strdup	Duplicates a string.

strcmp() Function :

This function compares two strings to find out whether the two strings are same or different. The two strings are compared character by character. If the two strings are identical, function returns a value zero. Otherwise it returns the numerical difference between the ASCII values of the non-matching characters.

Example :

```
#include<stdio.h>
main()
        {
                char string1[ ]=”jerry”;
                char string2[ ]=”ferry”;
                int i,j,k;
                i=strcmp(string1,”jerry”);
                j=strcmp(string1,string2);
                printf(“%d %d %d \n”,i,j,k);
        }
```

Output : 0 4 -98

strcpy() Function :

This function copies the contents of one sting into another. The base address of the source and target strings should be supplied to this function.

Example :

```
#include<stdio.h>
main()
        {
                char string1[20],string2[20];
                printf("\n Enter the two strings :\n ");
                scanf(“%s%s”,string1,string2);
                strcpy(string1,string2);
                printf("\n %s”,string1);
                getch();
        }
```

Input :

Enter the two strings: Sai ram

Output : Sairam

strlen() Function :

This function counts the number of characters present in a string.

Example :

```
#include<stdio.h>
main()
    {
        char s[20],s1[20];
        int n1,n2;
        printf("\n Enter the two words:\n ");
        scanf("%s%s",s1,s2);
        n1=strlen(s1);
        n2=strlen(s2);
        if(n1==n2)
                printf("they are of equal size");
        else
                printf("they are not equal");
        }
```

Input :

Enter the two words : Sai ram

Output :

They are of equal size

strlwr() Function :

- strlwr( ) function converts a given string into lowercase. Syntax for strlwr( ) function is given below.char *strlwr(char *string);
- strlwr( ) function is non standard function which may not available in standard library in C.

Example :

In this program, string "MODIFY This String To LOwer" is converted into lower case using strlwr( ) function and result is displayed as "modify this string to lower".

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

                char str[ ] = "MODIFY This String To LOwer";

                printf("%s\n",strlwr (str));
        return  0;

    }
```

Output :

```
modify this string to lower.
```

strupr() Function :

strupr( ) function converts a given string into uppercase.

Syntax for strupr( ) function is given below.

char *strupr(char *string);

strupr( ) function is non standard function which may not available in standard library in C.

Example :

In this program, string “Modify This String To Upper” is converted into uppercase using strupr( ) function and result is displayed as “MODIFY THIS STRING TO UPPER”.

```
#include<stdio.h>
#include<string.h>
 int main()
    {
            char str[ ] = "Modify This String To Upper";
            printf("%s\n",strupr(str));
            return  0;
    }
```

Output :

```
    MODIFY THIS STRING TO UPPER
```

strcat() Function :

This function concatenates the source string at the end of the target string. It is necessary to place ‘\0’ into the target string, to make its end.

Example :

```
#include<stdio.h>
int main()
    {
        char string1[]=”spot”;

        char string2[]=”buying”;

        strcat(string1,string2);

        printf(“%s”,string1);
    }
```

Output : spotbuying

strrev() Function :

This function reverses the string.

Example :

```
#include<stdio.h>
main()
        {
                char a[20],b[20];
                int i,stringlength=0;
                clrscr();
                printf("\n Enter any string : ");
                gets(a);
                strrev(a);
                printf("\n Reversed string is : ");
                puts(a);
        }
```

Input :

```
        Enter any string :  sairam
```

Output :

```
        Reverse string is : marias
```

strdup() Function :

strdup( ) function in C duplicates the given string.

Syntax for strdup( ) function is given below.

char *strdup(const char *string);

strdup( ) function is non standard function which may not available in standard library in C.

Example :

In this program, string “Raja” is duplicated using strdup( ) function and duplicated string is displayed as output.

```
#include <stdio.h>
#include <string.h>
int main()
        {
                char *p1 = "Raja";
                char *p2;
                p2 = strdup(p1);
```

```
            printf("Duplicated string is : %s", p2);
            return 0;
        }
```

Output :

```
        Duplicated string is : Raja
```

A program to count the number of vowels present in a sentence.

Program Code :

```
    # include<stdio.h>
    # include<conio.h>
    # include<string.h>
     main( )
        {
            char  st[80], ch;
            int  count = 0, i;
            clrscr( );
            printf(" \n Enter the sentence: \n");
            gets(st);
            for( i=0; i<strlen(st); i++)
                switch(st [i ])
                  {
                        case  'A':
                        case  'E':
                        case   'I':
                        case  'O':
                        case  'U':
                        case  'a':
                        case  'e':
                        case  'I':
                        case  'o':
                        case  'u':
                        count ++;
                        break;
                  }
            printf("\n %d vowels are present in the sentence", count);
            getch( );
        }
```

Output :

```
        Enter the sentence : This is a book

        5 vowels are present in the sentence.
```

Note :

- When this program is executed, the user has to enter the sentence.
- Note that gets( ) function is used to read the sentence because the string has white spaces between the words.
- The vowels are counted using a switch statement in a loop.
- Note that a count++ statement is given only once to execute it for the cases in the switch statement.

A program to count no of lines, words and characters in a given text.

Program Code :

```
# include<stdio.h>
# include<string.h>
# include<conio.h>
  main()
        {
                char txt[250], ch, st[30];
                int   ins, wds,  chs,  i;
                printf(" \n Enter the text, type $ st end \n \n");
                i=0;
                while((txt[i++]= getchar( ) ) ! ='$');
                      i--;
                st[ i ] = '\0';
                ins = wds = chs = 0;
                i=0;
                while(txt[ i ]!='$')
                      {
                            switch(txt[ i ])
                                  {
                                        case  ',':
                                        case  '!':
                                        case  '\t':
                                        case  ' ':
                                              {
                                                 wds ++;
                                                 chs ++;
                                                 break;
                                              }
                                        case  '?':
                                        case  '.':
                                              {
                                                 wds ++;
                                                 chs ++;
                                                 break;
                                              }
                                    default:chs ++;
                                    break;
                                  }
                              i++;
                      }
                printf("\n\n no of char (incl.blanks) = %d", chs);
                printf("\n No. of words = %d", wds);
                printf("\n No of lines = %d", ins);
                getch() ;
        }
```

Output :

```
Enter the text, type $ at end
What is a string? How do you initialize it? Explain with example.
With example : $
No of char : (inch. Blanks) = 63
No of words = 12
No of lines =1.
```

7.4. SUMMARY :

A string is a character array with a null character as the terminator at the last element. A string constant is a series of characters by double quotes. It has discussed the concept string. A string is an array of characters. We have briefly introduced the concepts of declaring and initializing the strings. It has described the standard library string functions, which are strlen(), strcpy(), strcmp(), and strcat().

7.5. KEY WORDS :

String : A string is a sequence of characters terminated with a null character '\0'.

String Dimension :

Strings are actually one-dimensional array of characters terminated by a null Character '\0'.

String library functions :

It is used to search whether a substring is present in the main string or not.

7.6. SELF ASSESSMENT QUESTIONS :

1. Define string?
2. Write the Declaration and Initialization of String Variables String?
3. Explain various string functions?

7.7. FURTHER READINGS :

1. "Programming with C" by Byron C.Gotttfried
2. "let us C" BY Yeshavanth kanethkar

3. "working with C"BY Yeshavanth kanethkar
4. *"Programming in ANSI C" by E.Balagurusamy*
5. *"*Sams Teach Yourself C " by Tony Zhang
6. "The Spirit of C" BY Henry Mullish.

LESSON - 8
FUNCTIONS

AIMS AND OBJECTIVES :

The objectives of this lesson is to

- explain the creation and utilization of programmer defined functions.
- discuss the importance of functions.
- how functions can divide the programming task into logically coherent components.
- explain about how a function is defined and called.

STRUCTURE OF THE LESSON :

8.1. FUNCTIONS : DEFINING FUNCTIONS FUNCTION CALL :

Functions :

A function is a self-contained program segment that carries out some specific, well-defined task. It can be referred as a module or procedure or a subtask. A function itself is not a program; rather it extends the ability of a program.

As C is a function-oriented language, it supports a rich set of functions in a sophisticated way. Every C program starts with at least one function called main().There are two different categories in functions, system defined and user defined functions.

8.1.1. System defined functions :

A system-defined function is a subprogram, which is prewritten in the compiler. There is a library in C for system defined functions. For example, printf(), scanf(), clrscr(), getchar(), etc.

8.1.2. User Defined Functions :

Sometimes, the user may require a function, which performs a specific task. In such cases, C permits user-defined functions.

Every function has its own significance and provides a separate scope for variables. A function requires three different types of statements to be specified by the user. They are

- Function prototype declaration
- Function Definition
- Function calling

As C supports top down approach in executing the programs, explicit declaration of the functions are required in a program.

Every function requires definition. Literally the user will not be able to use without defining a function.

Unless the user makes an explicit call to the function, the function declaration and definition have no significance and they perform nothing.

Hence the above three statements are essential for implementing functions.

8.2. ADVANTAGES OF FUNCTIONS :

1. Program debugging is made easy if a C program contains functions.
2. Functions allow a larger task to be subdivided into several smaller tasks, so that they can be managed easily.
3. The length of the source program can be reduced to a maximum extent using functions.
4. The same function can be used for many programs once it is written.
5. Functions may increase program execution speed.
6. Functions improve optimum utilization of memory.
7. Functions are more reliable.

8.3. DEFINING A FUNCTION :

The general structure of a Function :

```
data-type function-name(data-type arg1, data-type arg2….. argn)
        {
                local variable declarations ;
```

```
            body of function
            …………………….
            return(expression);
    }
```

Function name :

Every function requires a name (function name) so that it can be referred or identified by the user. Function names are unique in a program. While specifying names, make sure that the name doesn't posses any space between the characters.

Argument list :

A function may accept any number of values as input from the caller function in order to perform the task assigned to it. Each value is referred as an **argument** or **parameter**. Usually parameters are specified in the brackets '(', ')'.

Example :

```
power(x,n)
quadratic(a,b,c)
```

Argument declaration :
Argument variables must be declared for their types, after the function header and before the opening brace of the function body.

Example :

```
power(x,y)
int x;
float y;
  {
     .
     .
     .
  }
```

return() :

This is to specify the type of the value being returned by the function to its parent function or caller function. The keyword return is not mandatory. It is the last statement of a function. If at all, a function wants to send a value back to the caller, return statement may be kept in use.

return() can be declared in **two** ways :

syntax 1 : return;

It does, not return any value. It acts much as the closing brace of the program.

Example :

```
if(error)
```

```
return;
```

syntax 2 : return(expression);

It returns the value of the expression.

Example :

```
mul(x,y)
int x,y;
{
int p;
p=x*y;
return(p);
}
```

8.4. FUNCTION CALL :

When a function is called, parameters in the called function are bounded to the corresponding arguments supplied by the calling function. Before calling a function, it must be declared with a prototype of its parameters inside the *main().*

- **Functions with no arguments and no return values.**
- **Functions with arguments but no return values.**
- **Functions with arguments and return values.**

Functions with no arguments and no return values :

In this category of function, the caller function does not send any argument to the function and the called function does not return any value.

Example for this type of function is given below. Hear, the *main()* (calling function) calls two functions namely *printline()* and *value()*. Upon call, the program control passes from the calling function, and execution begins from the first executable statement of the called function. The called function is executed until a *return* or closing brace of the function is encountered, at which point the control passes back to the point after the function call. The functions *printline()* and *value()* accepts no values and they also return no value to it.

Example :

```
#include<stdio.h>
main()
   {
                             /* call to printline() that accepts no arguments*/
       printline();
                               /* call to value() that accepts no arguments*/
```

```
          value();
                                                  /*another call to printline()*/
          printline();
 }
                                                  /* function 1 printline()*/
          printline()
{
          int i;
          for(i=0;i<40;i++)
                    {
                              printf("-");
                    }
}
                                                  /* function 2 value() */
value()
{
          int num1,num2;
          int result;
          printf("enter the values of num1 and num2\n");
          scanf("%d%d",&num1,&num2);
          result=num1/num2;
          printf(" the result is %d\n",result);
}
```

Functions with arguments but no return values :

In this category of function, the caller function sends one or more values to called function but in return the called function does not return any value.

Example for this type of function is given below, the function interest() is called from main() and it accepts three arguments. The function interest () is supplied with three arguments namely principal, rate, and period. This function calculates the simple intrest based on the values sent to it, but it does not return the value of simple intrest to the calling function. When the function is called from the calling function, control passes to the function, the result is evaluated, and later displayed.

Example :

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

```
{
                                                    /* function prototyping */
    float intrest(float,float,int);
    float principal,rate;
    int period;
    printf("enter principal, amount  and intrest\n");
    scanf("%f%f" ,&principal,&rate);
    printf("enter the number of years\n");
    scanf("%d",&period);
    intrest(principal,rate,period);
}
float intrest(float p, float r, int n)
  {
    float simple;
    simple = (p*n*r)/100;
    printf("the simple interest for a period of %d years is %f",n,simple);
  }
```

Functions with arguments and return values :

In this category of function, the caller and called functions send values to one another.

Example for this type of function is given below, it illustrates how the function square() accepts the number and manipulates the square of that number. When this function is called from main(), the control passes to the called function, square of the number is evaluated and the result is returned to the environment in which the function is called.

Example :

```
#include<stdio.h>
main()
    {
       int number;
       float result;
                                                    /* function prototyping */
       float square(int);
       printf("enter the number whose square has to be found\n");
       scanf("%d",&number);
       result=square(number);
```

```
        printf("the square root of the number is %f",result);
    }
    float square(int num)
        {
            float res;
            res=num*num;
            return(res);
        }
```

8.5. PASSING PARAMETERS :

Arguments to a function are usually passed in two ways.

1. **Call-by-value**
2. **Call-by-reference**.

8.5.1. Call-by-value :

In call by value, a copy of the variable is made and passed to the function as argument. With this method, changes made to the parameters of the function have no effect on the variables, which called the function, because the changes are made only to the copies.

Example :

A Program for swapping of two integers.

```
#include<stdio.h>
main()
    {
        int a=1,b=2;
        swap(a,b);
    }
    swap(int a,int b)
      {
            int temp;
            temp=a;
            a=b;
            b=temp;
            printf("a=%d \t\t b=%d",a,b);
      }
```

8.5.2. Call-By-Reference :

Call-by-reference is a method in which the address of each argument is passed to the function. By this method, the changes made to the parameters of the function will affect the variable, which called the function.

Example:

A Program for swapping of two integers.

```
#include<stdio.h>
main()
   {
       int a=1,b=2;
       swap(&a,&b);
       printf("a=%d \t\t b=%d",a,b);
    }
   swap(int *a,int *b)
       {
           int temp;
           temp = *a;
           *a = *b;
           *b= temp;
       }
```

8.6. SUMMARY :

In this chapter we have learned the A function declaration includes to a function that is defined elsewhere, and specifies what type of arguments and values are passed to and returned from the function as well. A function definition saves the memory space and defines what the function does, as well as the number and type of arguments passed to a function. The return statement used in a function definition returns a single whose type must be matched with the one declared in the function declaration.

8.7. KEY WORDS :

- **Function :** A *function is a block of code which only runs when it is called*
- **Function call :** It is *called inside a program whenever it is required to call a function.*
- **Function categories :** There are two types of function in C programming: Standard library functions. User-defined functions.
- **Function prototype :** A function prototype is simply the declaration of a function that specifies function's name, parameters and return type.

8.8. SELF ASSESSMENT QUESTIONS :

1. Define a function?
2. Explain advantages of functions?
3. Explain different categories of functions?
4. What is meant by call-by-value? Explain with example.
5. What is meant by call-by-reference? Explain with example.

8.9. FURTHER READINGS :

1. "Programming with C" by Byron C.Gotttfried
2. "let us C" BY Yeshavanth kanethkar
3. "working with C"BY Yeshavanth kanethkar
4. *"Programming in ANSI C" by E.Balagurusamy*
5. "Sams Teach Yourself C " by Tony Zhang
6. "The Spirit of C" BY Henry Mullish.

LESSON - 9

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING THROUGH C++

AIMS AND OBJECTIVES :

The objectives of this lesson is to

- Difference between the procedure oriented programming and object oriented paradigm.
- explain the basic concepts of object oriented programming.
- discuss the benefits and applications of object oriented programming.
- explain the structure of a C++ program, writing a sample C++ program, compiling and running of a CPP program.

STRUCTURE OF THE LESSON :

9.1 INTRODUCTION TO OOP AND ITS BASIC FEATURES :

C++ is an object oriented programming language. It was initially named as C with classes. However in 1983, it was renamed as C++. C++ was developed by Bjarne Stroustrup at AT& T Bell laboratories, New Jersey, U.S.A. It is an enhancement of the C programming language with the major addition of class construct feature of Simula67. It was built upon C and hence all standard C features are also available in C++. Thus C++ is the superset of C. Almost all C programs are also C++ programs.

The three important facilities that C++ have are C with classes, function overloading and operator overloading. These features help to create abstract datatype, inheritance from existing datatype and polymorphism. C++ allows the programmer to build programs with clarity, extensibility and ease of maintenance.

9.1.1 Procedure Oriented Programming :

Procedure oriented programming is viewed as a sequence of things to be done, such as reading, calculating, printing etc. A number of functions are written to accomplish the tasks. The primary importance is given to functions and little is given to data that are being used by various functions.

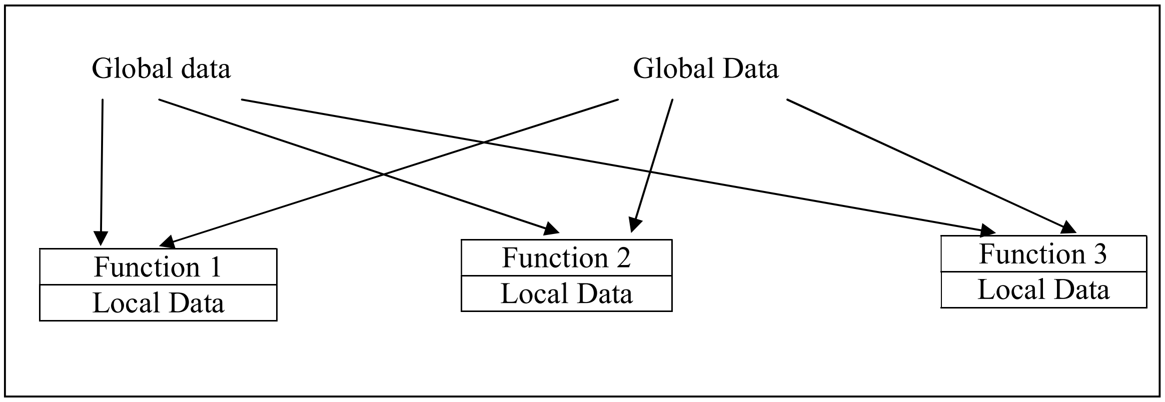
Data is placed as global, so that they may be accessed by all the functions. Each function can have its own data. Global data can be used by any function so that it is difficult to identify what data is used and by which function. Procedure Oriented programming does not model real world problems very well.

Some important Characteristics are :

- Importance is given in doing the algorithms.
- Large programs are divided into smaller programs known as functions Most of the functions share global data.
- Data move over the system from function to function freely.
- Functions transform the data from one form to another.
- Employs top-down approach in program design.

9.1.2 Object Oriented Programming :

Object oriented programming is an approach that provides a way of modularizing the programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules in demand.

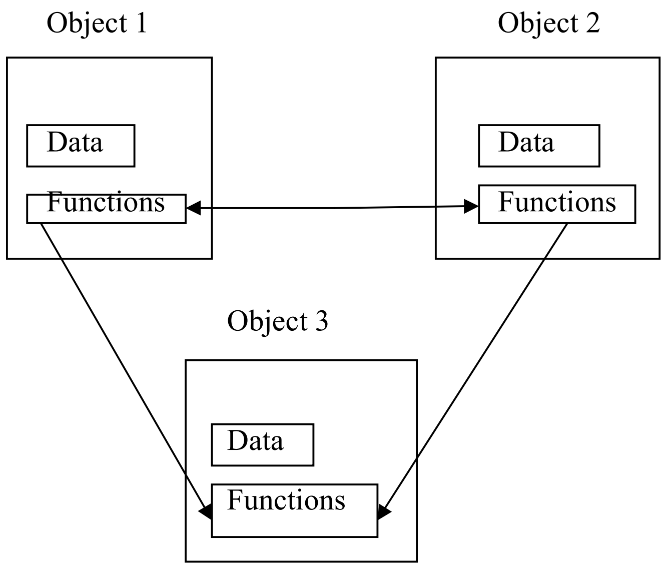


OOPS treats data as a critical element in the program development and does not allow it to freely flow around the system.

It ties data more closely to the function that operates on it and protects it from accidental modifications from outside functions. OOP allows us to decompose the problem into number of entities called objects and build data and function around them. Data of an object can be accessed only by functions associated with that object. Functions of one object can access the functions of another object.

Important Features :

- Importance is given to data then the functions
- Programs are divided into objects
- Data Structures characterize the objects.
- Functions that operate on the data of the object are tied together in the data structure.
- Data is hidden and cannot be accessed by the external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added.
- Follows bottom-up approach is used in the program design



9.2 CONCEPTS OF OOPS :

The major Concepts ofObject Oriented Programming are :

1. Class
2. Object

3. Abstraction
4. Encapsulation
5. Data Hiding
6. Inheritance
7. Reusability
8. Polymorphism
9. Virtual Functions
10. Message passing

1. Class :

Class is an abstract data type (user defined data type) that contains member variables and member functions that operate on data. It starts with the keyword class. A class denotes a group of similar objects.

Eg. :
```
class employee
    {
        int empno;
        char name[25],desg[25];
        float sal;
        public:
        void getdata ();
        void putdata ();
    };
```

2. Object :

An object is an instance of a class. It is a variable that represents data as well as functions required for operating on the data. They interact with private data and functions through public functions.

Eg. : `employee e1, e2;`

In the above example employee is the class name and e1 and e2 are objects of that class.

3. Abstraction :

Abstraction refers to the process of concentrating on the most essential features and ignoring the details. There are two types of abstraction.

i) Procedural Abstraction

ii) Data Abstraction

i) **Procedural Abstraction :** Procedural abstraction refers to the process of using user-defined functions or library functions to perform a certain task, without knowing the inner details. The function should be treated as a black box. The details of the body of the function are hidden from the user.

ii) **Data Abstraction :** Data Abstraction refers to the process of formation of user defined data type from different predefined data types.

Eg : structure, class.

4. **Encapsulation :**

Encapsulation is the process of combining data members and member functions into a single unit as a class in order to hide the internal operations of the class and to support abstraction.

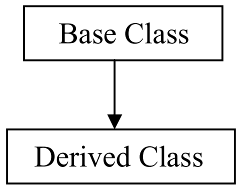
5. **Data Hiding :**

All the data in a class can be restricted from using it by giving some access levels (visibility modes). The three access levels are private, public, protected.

Private data and functions are available to the public functions only. They cannot be accessed by the other part of the program. This process of hiding private data and functions from the other part of the program is called as data hiding.

6. **Inheritance :**

Inheritance is the process of acquiring (getting) the properties of some other class. The class whose properties are being inherited is called as base class and the class which is getting the properties is called as derived class.



7. **Reusability :**

Using the already existing code is called as reusability. This is mostly used in inheritance. The already existing code is inherited to the new class. It saves a lot of time and effort. It also reduces the size of the program.

8. **Polymorphism :**

Polymorphism means the ability to take many forms. Polymorphism allows to take different implementations for same name.

poly	→	many
morphism	→	forms

There are two types of polymorphism, Compile time polymorphism and run time polymorphism. In Compile time polymorphism binding is done at compile time and in runtime polymorphism binding is done at runtime.

Eg.: Function overloading, operator overloading.

Function Overloading : Function overloading is a part of polymorphism. Same function name having different implementations with different number and type of arguments.

Operator Overloading : Operator overloading is a part of polymorphism. Same operator can have different implementations with different data types.

9. Virtual Functions :

Virtual functions are special type of functions which are defined in the base class and are redefined in the derived class. When virtual function is called with a base pointer and derived object then the derived class function will be called. A function can be defined as virtual by placing the keyword virtual for the member function.

10. Message Passing :

An object-oriented program contains a set of objects that communicate with one another. The process of object oriented programming contains the basic steps :

1. Creating classes
2. Creating objects
3. Communication among objects

This communication is done with the help of functions (i.e., passing objects to functions).

9.2.1 Benefits Of OOPS :

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- Programs can be built from the standard working modules that communicate with one another, rather than writing the code from scratch. This leads to saving of development time and higher productivity.
- Principle of data hiding helps the programmer to build secured programs. It is possible to have multiple instances of objects to co-exist without any inheritance.
- Easy to partition the work in project based objects.
- It is possible to map objects in the problem domain to those objects in the program.
- Software complexity can be easily managed.

- Message passing techniques for communication makes the interface descriptions with external systems much simpler.
- The data-centred design approach enables us to capture more details of a model in implementable form.

9.2.2 Applications of OOPS :

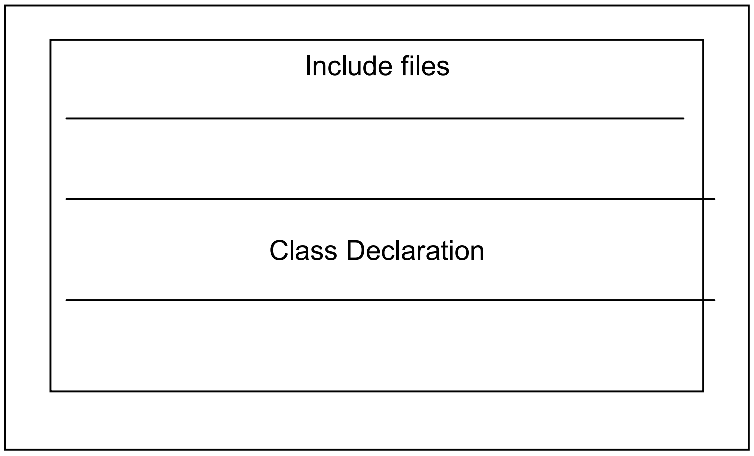
The promising areas for application of OOP includes :

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext. Hypermedia and experttext
- Al and expertsystems
- Neural networks and parallel programming
- Decision support and Automation system
- CIM/CAM/CAD systems

9.3 C++ PROGRAM STRUCTURE :

C++ program contains 4 sections. These may be placed in separate code files and then compiled independently or jointly. A program is commonly organized into 3 separate files. The class declarations are placed in a header file and the definitions of member functions go into another file.

This helps the programmer to separate the abstract specification of the interface (class definition) from the implementation details (member function definition). The main program is placed in a third file which includes the previous two files as well as any other files required.



9.3.1 A Sample C++ Program :

```
//sum of two integers
#include <iostream.h>     //include header file
int  main()
   {
        int x,y,sum;
        cout<<"Enter any 2 numbers: ";
        cin>>x>>y;
        sum = x + y;
        cout<< "The given numbers are ";
        cout<<x;
        cout<<" and ";
        cout<<y;
        cout<<"\n";
        cout<<"Their sum is "<<sum<<"\n";
        return 0;
   }            //End of example
```

Output :

```
Enter any 2 numbers :  4    5
The given numbers are 4 and 5
Their sum is 9
```

In order to make a program understandable, some explanatory notes is included at key places in the program. Such notes are called comments. In C++ the symbols // are used to indicate the start of the comments. The comment starts with a // and terminate at the end of the line. A comment may start any where in the line, and whatever follows till the end of the line is ignored. // is a single line comment.

Example :

```
// This is a
// sample C++
//  program
```

The C comment symbols /*-------*/ is also valid and are suitable for multiline comments. Either or both of the styles can be used in the programs.

Example : /*This is a sample C++ program*/

The program begins with the line :

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

This is called include directive. It tells the compiler where to find information about certain items that are used in your program. iostream is the name of the library that contains the definitions of the routines that handle input from the keyboard and output to the screen. iostream.h is the file that contains the information about the library.

Directives begin with the symbol # at the very start of the line and no space is included between # and include. A C++ program is a collection of functions. The above example contains one function, main(). The program starts with

```
int main()
   {
       and ends with
       return 0;
   }
```

As the return type of the function is integer, 0 is returned here. The lines between the beginning and ending {} are the heart of the program.

```
int x,y,sum;
```

This line is called variable declaration. The variable declaration tells the computer that x,y and sum are the name of the three variables used in the program. int word tells the computer the numbers named by these variables are integers.

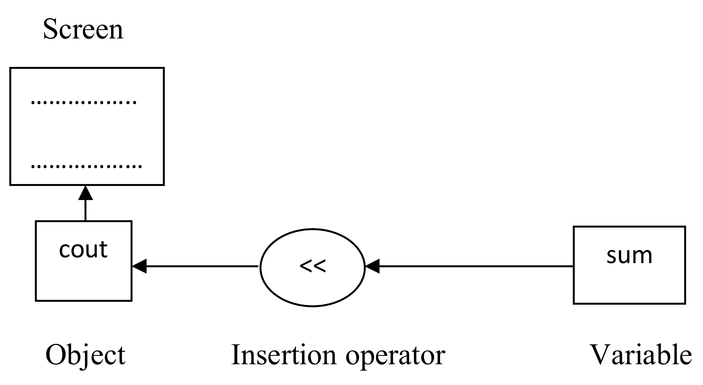
The remaining lines are the instructions that tell the computer to do the corresponding work. These instructions are called executable statements or statements. Every statement should end with a semicolon.

Most of the statements begin with the word cout or cin. These statements are the input and output statements. The << and >> arrows are the operators which tell the direction in which the data is moving.

The operator << is called the insertion operator or put to operator. It inserts (or sends) the contents of the variable on its right to the objects on its left. Cout is a predefined object that represents the standard output stream in C++. Here the standard o/p stream represents the screen. << operator can be overloaded.

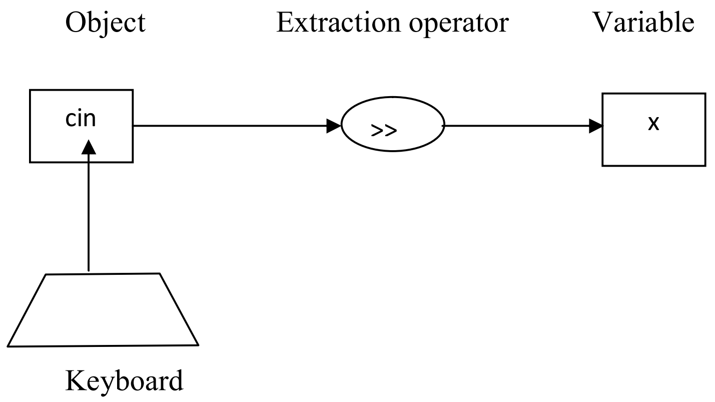
Ex: cout<<"Enter two numbers";

cout<<sum;



The operator >> is known as extraction or get from operator. It extracts or takes the value from the keyboard and assigns it to the variable on its right. >> operator can also be overloaded.

Eg.: cin>>x ;



Cascading of I/O operators :

The multiple use of << in one statement is called cascading. This is known as cascading of output operator.

Eg.: cout<<"Their sum is"<<sum<<"\n";

This statement sends the string "Their sum is" to cout and then sends the value of sum, then the newline.

Similarly, >> operator can be cascaded. This is known as cascading of input operator.

Eg.: cin>>x>>y;

sum = x+y;

cout<<

This is the computational statement. The values of x and y are summed up with + operator and the value is stored in the variable sum.

cout<<"Their sum is "<<sum<<"\n";

"\n" contained at the end of the output statement tells the computer to start a newline after writing the text.

9.3.2 Compiling and Running :

C++ program is typed in using a text editor. There are different text editors. Turbo C++ provides a built-in editor and a menu bar including the options such as File, Edit, Compile and Run. The source file is created and saved under the File Option and can be edited under Edit option. The program is compiled under Compile Option and Run using Run option.

Compilation of the program will produce a machine-language translation of the source code, called the object code. The object code must be linked (combined) with the object code for routines (input and output routines) that are already written. Run option executes the program. If there are no errors in the program, then compiling, linking and running will go smoothly. However, errors may occur which has to be rectified and executed.

9.3.3 Testing and Debugging :

A mistake in the program is usually called a bug, and the process of eliminating bugs is called debugging. There are three kinds of programming errors. They are

- Syntax errors
- Runtime errors and
- Logical errors

The errors that are caused due to the violation of syntax (grammar rules) of the programming language are called syntax errors. E.g.: Omitting semicolon at the end of the statement. These errors can be found during compilation.

There are certain kinds of errors that the computer system can detect only when the program is run. These are run-time errors. Eg:If a computer attempts to divide a number by zero.

There are certain kinds of errors, which cannot be identified during compilation. The program is run successfully but the output is wrong. This is due to a mistake in the logic of the program. These are known as logical errors.

Eg.: By mistake, using + instead of * during addition of two numbers.

9.3.4 Applications of C++ :

- C++ is a versatile language for handling very large programs.
- C++ is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real life application systems.
- Since C++ allows us to create hierarchy-related objects. We can build special object oriented libraries which can be used later by many programmers.
- C++ programs are easily maintainable and understandable.

9.4. SUMMARY :

- Procedure oriented programming follows a Top Down approach where the problem is viewed as sequence of tasks. Functions are used to implement it.
- To overcome the drawbacks, such as free movement of data around the program and

as it is difficult to model real world problems, object oriented programming is introduced.

- Object oriented programming follows a Bottom Up programming approach and it does not allow data to move freely.
- The different concepts of OOPS like class, object, encapsulation, abstraction, inheritance, polymorphism, dynamic binding, message passing are briefly discussed.
- Advantages and applications of OOPS are discussed.
- The structure of a C++ program, writing a sample program, compiling, debugging and running of the programs are discussed.
- The applications of C++ are covered.

9.5 KEY WORDS :

- **Object :** An Object is an instance of a Class.
- **Class :** A Class is a fundamental block of a program that has its own set of methods and variables.
- **Data abstraction :** Data abstraction is the reduction of a particular body of data to a simplified representation of the whole.
- **Encapsulation :** In general, encapsulation is a process of wrapping similar code in one place.
- **Inheritance :** Inheritance is a method through which one class inherits the properties from its parent class.
- **Polymorphism :** The same entity (function or object) behaves differently in different scenarios.
- **Dynamic binding :** If a body of the method is bonded to a method call at runtime then it is called dynamic binding or late binding.

9.6 SELF ASSESSMENT QUESTIONS :

1) Define Procedure oriented programming?
2) Define object oriented programming?
3) What is the difference between object oriented programming and procedure oriented programming?
4) Write the concepts of object oriented programming?

9.7 FURTHER READINGS :

1. Object-oriented programming with C++ by E. Bala Gurusamy
2. Problem solving with C++ by Walter Savitch
3. Mastering C++ by K.R.Venugopal, Rajkumar Buyya, T.Ravi Shankar.

LESSON - 10
CLASSES AND OBJECTS

AIMS AND OBJECTIVES :

This objectives of this lesson are to

- Explain the concept of classes and objects in C++.
- Define a class and member functions in two different ways in C++ program.
- Know how the memory allocation takes place for objects.

STRUCTURE OF THE LESSON :

10.1 INTRODUCTION TO CLASSES AND OBJECTS :

A class is a container that consists of member variables and member functions. Which operate on data. C++ enables you to organize data in the form of templates, which contain attributes and behaviours defined in a C++ program.

A class is based on the encapsulation concept, which ensures that a class hides different attributes and behaviours from the outside world. Figure 10.1 shows that the ABC class contains X, Y and Z attributes and the Start and End behaviours.

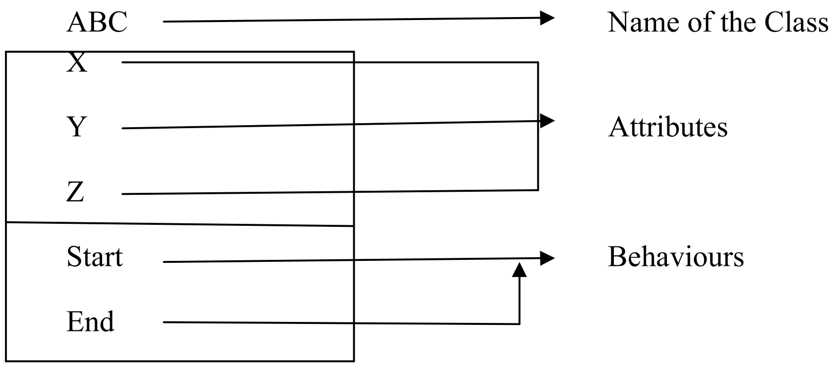


Figure 10.1 Concept of Class

A class defined in a C++ program consists of the following concepts :

- Class Name : It is used to identify a class and its scope within a C++ program.
- Attribute: It used to store data or values that are provided by a user.
- Behaviour: It is used to perform operations that manipulate the data contained in a class.

For example, there is a class, auto, defined in a program that describes its attributes and the behaviours that interact with its attributes. Figure 10.2 shows the structure of the auto class.

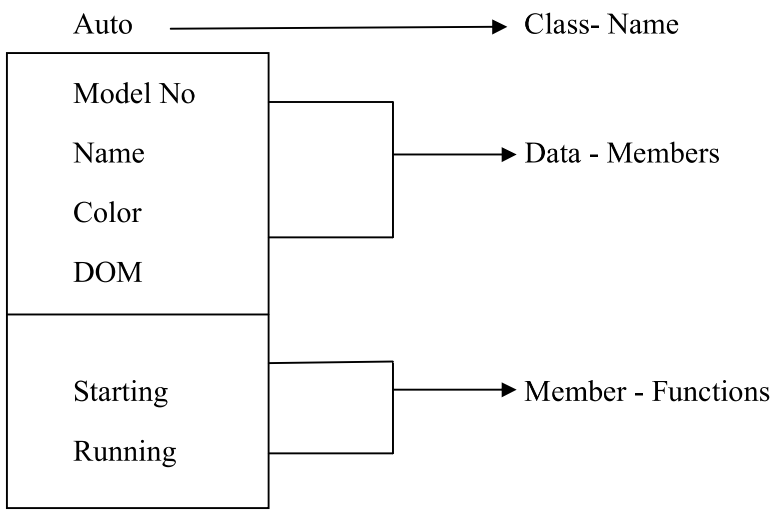


Figure 10.2 The Auto Class

C++ also provides a feature that enables you to use a class that is defined in some other program. This feature helps you develop better and efficient software programs, In C++, an object, which is also called an instance of a class, contains a copy of data defined in the class. Figure 10.3 shows the objects such as chair, table and sofa for the furniture class.

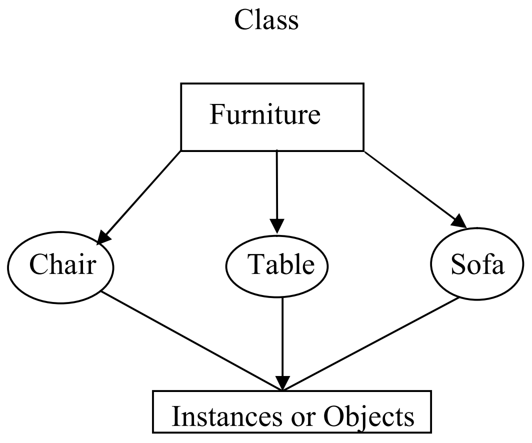


Figure 10.3 Classification of Objects

In the furniture class, which contains objects such as chair, table and sof variables for each object need not be declare separately. Each object shares member variables such as name, colour, type and no. of legs defined with other objects of the class. Figure 10.4 shows how an object shares data-members.

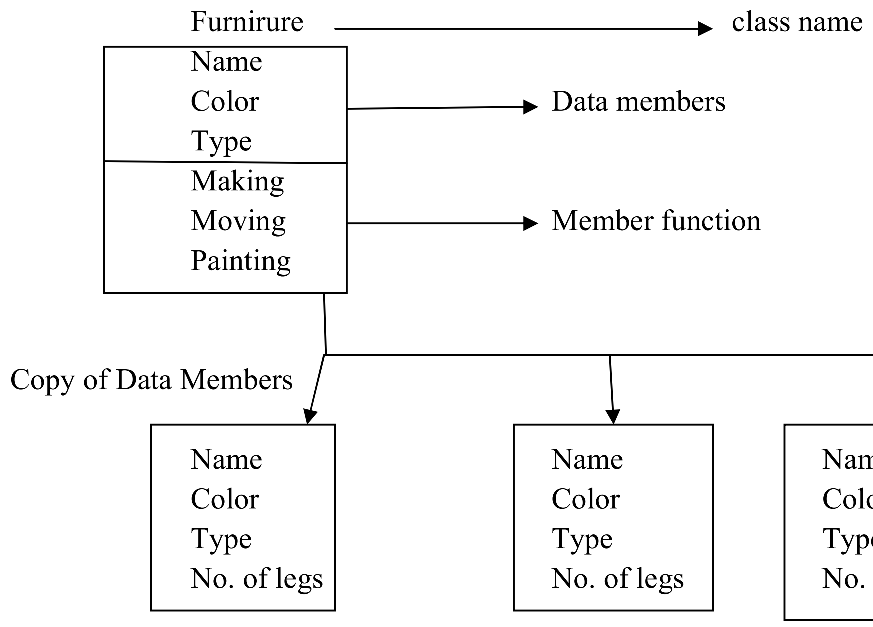


Figure 10.4 Example of Class

10.2 DEFINING A CLASS :

In C++, developers can define a class by specifying data-members and member-functions according to the requirements. The following program shows how you define a class in C++.

```
#include <iostream.h>

#include <conio.h>

class test

    {

      private:

      int a,b,c;

      public:

                            //Member Function for accepting values from the user.

       void enterdata()

             {

                   cout<<endl<<"Enter two integer values";
```

```
                    cin>>a>>b;
                                        //Member Function for displaying values.
                }
            void printdata()
                {
                    c=a+b;
                    cout<<endl<<"The addition of two numbers is:="<<C;
                }
        };
        void main()
            {
                clrscr();
                test t; // declaration of object.
                t.enterdata();
                t.printdata();
                getch();
            }
```

In the above program a test class is defined that contains three integer variables a. b and e. These variables are called the member variables of the class test. The class test also contains two member functions, which are enterdata() and printdata(). The enterdata() member function accepts two integer values from the user as input and stores them in integer variables a and b. The class also contains another member function printdata() that is used to compute the sum of the integer values, which have been provided as input by a user and print the result as output. Figure 10.5 shows the output of the above program.

```
Enter two integer values     12     16

The addition of two numbers is  =   28
```

10. 3 SAMPLE C++ PROGRAM USING CLASS :

The main component of a C++ program is class, which allows you to combine data and functions related to different real-world objects to form a single entity. Consider the following C++ program to understand the use of class for maintaining records of students.

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

#include <iomanip.h>

class student

    {

        int stdroll;

        float percent;

        public:

        void getdata ();

        void display ()

            {

                cout<< student Rollno: <<stdroll<<endl;

                cout<< Student Percent: <<setprecision(2) <<percent;

            }

    };

void student :: getdata ( )

    {

        cout<<"Enter student rollno:=";

        cin>>stdroll;

        cout<<"Enter student percentage:=";

        cin>>percent;

    }

void main()

    {

        clrscr();

        student std;

        std.get.data():

        std.display();

        getch();

     }
```

In the above program code, the class student contains two private me variables and two public member functions. The two private member variables of class student are stdroll and percent that store the student rollno and percentage achieved by a student. Figure 10.7 shows the output of the above program.

```
Enter student rollno       =  121

Enter student percentage =  657.897

student Rollno   :  121

Student Percent  :  657.9
```

10.4 FRIEND FUNCTIONS :

Friend functions allow you to access private and protected members of a class from outside the class. To do this, you have to declare functions as friends in a class. You can create an object of the class in the friend function to access the private and protected members of the class.

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

class integer

    {

        int a,b;

        public;

        friend integer enter_val();

        void print_val()

            {

                cout<<"Value of nl is "<<<<endl;

                cout<<"Value of n2 is "<<b;

            }

     }

integer enter_val()

    {
        integer n;
        cout<<"enter value for a and b;";

        cin>>n.a>>n.b;

        return n;

    }
```

```
void main()
    {
        clrscr();
        integer nl-enter_val();
        n1.print_val();
        getch();
    }
```

The above program shows the output of the friend function as below.

```
enter value for a and b : 2
4
Value of n1 is 2
Value of n2 is 4
```

10.5 STATIC FUNCTIONS :

It is used to access the static attributes and provides direct accessibility of attributes by class-name because in OOPS concept you can access behaviours of a class by the object of the class.

10.6 SUMMARY :

In this chapter you learned about classes and objects. Classes are the user-defined data types that contain member variables and member functions, which operate on data, and you studied a sample C++ program using class.

10.7 KEY WORDS :

- **Class :** A Class is a fundamental block of a program that has its own set of methods and variables.
- **Variables :** Variables are the containers for storing data values.
- **Simple program :** C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- **Friend function :** A friend function in C++ is defined as a function that can access private, protected and public members of a class.
- **Static function :** It is a member function that is used to access only static data members. It cannot access non-static data members not even call non-static member functions.

10.8 SELF ASSESSMENTQUESTIONS :

1. How will you define a class and object in C++.with an example program?
2. Write a properties on class and object?
3. Explain simple C++ program structure?
4. Discuss about friend function and static function.

10.9 FURTHER READINGS :

1. Schildt, H., The Complete Reference C++,. New Delhi: Tata McGraw-Hill, 2005.
2. Balaguruswamy, E., Object-oriented Programming in C++. New Delhi: Tata McGraw-Hill, 2006.

CHAPTER - 11

CONSTRUCTORS, DESTRUCTORS AND OPERATORS

AIMS AND OBJECTIVES :

The objectives of this lesson are to learn about the usage of

- constructors,
- dynamic initialization of objects,
- copy constructors,
- dynamic constructors and
- destructors.

STRUCTURE OF THE LESSON :

11.1 CONSTRUCTORS :

Constructors in C++ are special member functions of a class and have the same name as the name of the class. Constructors are functions, which are invoked whenever a new instance of a class is created. They are often used to initialize the member variables of a class. Constructors may also have parameters. Such constructors are called parameterized constructors. If a constructor has parameters, then the parameter list should be matched to the definition of the constructor. There can be many overloaded constructors inside a class. However the constructors must have different sets of parameters.

11.1.1 PROPERTIES OF CONSTRUCTORS :

A constructor, defined in a class, has the following properties:

- It has the same name as the name of the class.
- It does not return any values.
- Constructors are basically functions, which are invoked first when a class is initialized.
- Constructors are used to initialize an object, when it is created.
- Constructors cannot be invoked explicitly as if they were regular member functions. These are only executed when a new object of a class is created.
- Constructors are invoked automatically when the object is created.
- Each object of the class that has a constructor should be initialized before it is used.
- Constructors can be defined as private or public. The private constructorsare accessible only to member functions, but the public constructors are available to all functions.
- Constructors cannot be inherited, though a derived class can invoke aconstructor of a base class.
- You can overload constructors in C++. Overloaded constructors differ inthe number and type of arguments that they can accept.
- It can be used explicitly to create new objects of its class type.
- Constructors cannot be virtual.

11.1.2 CREATING A CONSTRUCTOR :

A constructor is created with the same name as that of the class. A constructor is declared in the class definition and can be defined in the class or outside the class definition. The following code shows the syntax for creating a constructor in a class.

```
Class class _name

    {

        private:

                                // private member

        public:

        class_name()                  // constructor

            {

                            // constructor initialization

            }

                            //other class member

    };
```

The above code shows two types of member variables, public and private. Private member variables are accessible only in the class but the public construct class_name() is accessible outside the class.

11.1.3 DEFINING A CONSTRUCTOR :

You can define a constructor in a class or outside a class. When you constructor within a class then both the definition and the declaration of the constructor is specified. To define a constructor outside a class, you can use the following syntax : define

```
class_name::constructor()

        {

                // Constructor definition

        }
```

Consider a class with the name X. You can define a constructor with the name X which is same as the name of the class. To define a constructor with the name X outside the class definition, you can use the scope resolution operator represented by the following code shows how to define a constructor X outside the class definition.

```
X::X()
    {
                    //Constructor definition

    }
```

11.1.4 PARAMETERIZED CONSTRUCTORS :

A constructor may or may not have parameters. If a constructor has parameters then it is called parameterized constructor, else it is called a default constructor with no parameters. A default construction in a class can be defined as given below :

```
class class_name

        {

                class_name();       // Default constructor

                { }

        };
```

A parameterized constructor with three parameters in a class can be defined in two ways. The first way to define a parameterized constructor is as follows :

```
class class_name
        {
                class_name(float, int, char)        // Parameterized constructor
                { }
        };
Another way to define a parameterized constructor is as follows :
class class_name
        {
                class name (float x, int y, char a)
                { }                                 //Parameterized constructor
                { }
        };
```

Consider the following program to understand parameterized constructor in a class

```
#include <iostream>

clase paracons

{

     int a, b;

     public:

     paracons (int i, Int j)

        {                              //constructor definition in a class

              a=i;

              b=j;

        }

        void show()
              {

                   ccont<<a<< " "<<b;

              }
};
int main()
   {

        Paracons obj (3,4);

        obj.show();

        return 0;
   }

int main()
   {

        paracons obj (3, 4); ob).show();
        return 0;

   }
```

The paracons class creates a parameterized constructor with two integer parameters. To invoke the paracons constructor in the main function, you need to pass two integer parameters. Figure below shows the output of the above program.

```
3 4
```

11.1.5 MULTIPLE CONSTRUCTORS IN A CLASS :

A class can contain multiple constructors but all the constructors must have different parameters. When you invoke a particular constructor, the parameters passed to the constructor must be in the same format as the parameter specified in the declaration of the constructor that is to be invoked. The following program shows how to use multiple constructors in C++.

```
#include <iostream>
#include <cstdlib>

 using namespace std:

class mulcons

     {

          double a ;

          public:

          mulcons (int x) {

          a=x;

     }

mulcons (double x)

     {

         a = x;

        double showa ()

            {
                return a;
            }
     };
int main()

     {
            mulcons   obj1= 2;
            mulcons obj2=123.123;
            cout << "obj1 “: << obj1.showa() << end1;
            cout << "obj2 “: << obj2.showa() << end1;
            return 0;

     }
```

The mulcons class creates two constructors with integer and float parameters. You must create two objects of the mulcons class to invoke the constructors.The obj1 object invokes the showa() function to display the data from the constructor having the integer parameter. The obj2 object displays the data from constructor having the float parameter. Figure below shows the output of the program.

```
Obj1    :    2

Obj2     :    123.123
```

11.1.6 CONSTRUCTORS WITH DEFAULT ARGUMENTS :

Default arguments in a constructor are formal parameters with values defined at the time of declaration. Values for a constructor are specified when invoking object does not supply an actual parameter value. Actual parameter override the default values, when an object specifies the actual values to invoke a constructor. The following program code shows how to assign a default parameter to constructors.

```
#include <iostream.h>

class defitarg

    {

        Privaces ;

        int e;

        int b;

        public:

                    // Default Parameters given to al and a2 are 0 and 1 respectively.

    defltargfint al-0, int a2=1)

        {
                a=al;

                b=a2;

                                                // Displays the Effect of using Default Constructors.

                void display()
                     {
                          cout<<<<<<b<<endl;
                     }
           };
int main()
   {

        deflcarg a(1,2); // Here a1=1 a2=2

        defltarg b;// Here al=0 a2=1

        defltarg c(3); //Here a1=3 a2=1

        a.display(); // 1-2

        h.display(); // 0-1

        c.display(); // 3-1

        return 0;

   }
```

In the above program, the defltarg class creates a constructor with two parameters, which have default values as zero and one. The defltarg class has three objects as a, b and c. The object a initializes the constructor with two parameter values, which overwrite the default values. The object b initializes the constructor with no parameters. The object c initializes the constructor with one argument, which provides the value to the first parameter of the constructor. Figure below shows the output of the above program.

```
1-2

0-1

3-1
```

11.2 DYNAMIC INITIALIZATION OF OBJECTS :

Dynamic initialization of objects means initializing objects at run time. A constructor helps initialize objects at run time when an object is created. For example, consider the Fruits class, which has various constructors with different parameters. The constructors are initialized at run time in the main function. Dynamic initialization of object refers to initializing the objects at a run time i.e., the initial value of an object is provided during run time. It can be achieved by using constructors and by passing parameters to the constructors. This comes in really handy when there are multiple constructors of the same class with different inputs.

11.2.1 Dynamic Constructor :

The constructor used for allocating the memory at runtime is known as the dynamic constructor. The memory is allocated at runtime using a new operator and similarly, memory is deallocated at runtime using the delete operator.

11.2.2 Dynamic Allocation :

1. In the below example, new is used to dynamically initialize the variable in default constructor and memory is allocated on the heap.
2. The objects of the class g calls the function and it displays the value of dynamically allocated variable i.e ptr.

Below is the program for dynamic initialization of object using new operator :

```
// C++ program for dynamic allocation
        #include <iostream>
        using namespace std;
         class gs
        {
                int* ptr;
                 public:

                                    // Default constructor

                gs()

                {

                                    // Dynamically initializing ptr using new

                        ptr = new int;

                        *ptr = 10;

                }

                                // Function to display the value of ptr
                void display()
                {
                        cout << *ptr << endl;
                }
        };
                                                        // Driver Code
        int main()
        {
                gs obj1;
                                                        // Function Call
                obj1.display();
                 return 0;
        }
```

Output : 10

11.2.3 Dynamic Deallocation :

In the below code, delete is used to dynamically free the memory.

The contents of obj1 are overwritten in the object obj2 using assignment operator, then obj1 is deallocated by using delete operator.

Below is the code for dynamic deallocation of the memory using delete operator.

```
// C++ program to dynamically deallocating the memory
    #include <iostream>
    using namespace std;
    class gs
    {
        int* ptr;
```

```
          public :
                                              // Default constructor
          gs()
          {
                     ptr = new int;
                    *ptr = 10;
           }
                                             // Function to display the value
          void display()
          {
                     cout << "Value: " << *ptr
                    << endl;
           }
};
                                              // Driver Code
int main()
{
                                              // Dynamically allocating memory
                                              // using new operator
          gs* obj1 = new gs();
          gs* obj2 = new gs();
                                             // Assigning obj1 to obj2
          obj2 = obj1;
                                             // Function Call
          obj1->display();
          obj2->display();
                                             // Dynamically deleting the memory
                                              // allocated to obj1
```

```
        delete obj1;
        return 0;
    }
```

Output :

```
Value : 10

Value : 10
```

In the below program is demonstrating dynamic initialization of objects and calculating bank deposit :

```
// C++ program to illustrate the dynamic  initialization as memory is allocated to the object

    #include <iostream>

    using namespace std;

    class bank

    {

        int principal;

        int years;

        float interest;

        float returnvalue;

        public:

                                    // Default constructor

        bank() {}

                                    // Parameterized constructor to calculate
        interest(float)

        bank(int p, int y, float i)

        {

            principal = p;

            years = y;

            interest = i/100;
```

```
                returnvalue = principal;

                 cout << "\nDeposited amount (float):";

                                    // Finding the interest amount

                for (int i = 0; i < years; i++)

                {

                            returnvalue = returnvalue * (1 + interest);

                }

        }

                    // Parameterized constructor to calculate interest(integer)

        bank(int p, int y, int i)

        {

                 principal = p;

                 years = y;

                interest = float(i)/100;

                returnvalue = principal;

                cout << "\nDeposited amount"

                 << " (integer):";

                                            // Find the interest amount

                for (int i = 0; i < years; i++)

                {

                           returnvalue = returnvalue * (1 + interest);

                 }

        }

                                                // Display function

void display(void)
```

```
        {
                cout << returnvalue
                 << endl;
        }
};
                                                    // Driver Code
int main()
{
                                                    // Variable initialization
        int p = 200;
        int y = 2;
        int I = 5;
        float i = 2.25;
                                                      // Object is created with
                                                     // float parameters
        bank b1(p, y, i);
                                                      // Function Call with object of class
        b1.display();
                                                // Object is created with integer parameters
        bank b2(p, y, I);
                                                // Function Call with object of class
         b2.display();
        return 0;
}
```

Output :

```
Deposited amount (float):209.101
Deposited amount (integer):220.5
```

11.3 DESTRUCTORS :

Destructors are functions, which are invoked whenever an object of a class is destroyed. Destructors in C++ also have the same name as the name of the class in which they are defined except that they are preceded by a operator ~. The destructors are invoked when the object of a class goes out of its defined scope. It is not necessary to declare a destructor inside a class. If a destructor is not declared in a class then the compiler will automatically create a default destruct in the class. If a destructor is declared as private, then the class cannot be instantiated. A destructor defined in a class has the following properties :

- Destructors have the same name as class name except that they are preceded by the operator ~.
- Destructors cannot have arguments.
- Destructors do not return values.
- Destruction of objects takes place when the object leaves its scope of definition or is explicitly destroyed.
- Destructors also obey the usual access rules as other member functions.
- Destructors are not parameterized. They do not hold any arguments and cannot be overloaded.
- Destructors cannot be declared const, volatile, const volatile or static.
- Destructors can be declared as virtual or pure virtual.
- You cannot access the address of a destructor.

Consider the following program that shows how to define a destructor in the example_destructor class.

```
#include <iostream>
 using namespace std;
class destrexam
        {
                int a, b;
                 public:
                destrexam(int i, int j)
                        {                               //constructor definition in a class
                        a=i;
                        b=j:
                    }
                ~destrexam ()
                    {}
                void show()
                    {
                        cout << a << " " << b;
                    }
            };
int main()
    {
                destexam obj (3, 4);
                obj.show();
                return 0;
    }
```

In the above program, the destrexam class contains a destructor. The compiler in the program invokes the destructor implicitly. The output of above code is **3 4**

11.4 OPERATORS :

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provides the following types of operators:

Arithmetic Operators

Relational Operators

Logical Operators

Bitwise Operators

Assignment Operators

Misc Operators

In this will examine the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

Arithmetic Operators :

There are following arithmetic operators supported by C++ language :

Assume variable A holds 10 and variable B holds 20, then :

Operator	**Description**	**Example**
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Relational Operators :

There are following relational operators supported by C++ language Assume variable A holds 10 and variable B holds 20, then:

Operator	**Description**	**Example**
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	*A* == *B* is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then 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 condition becomes true.	*A* > *B* is true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	*A* >= *B* is not true

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	*A* < *B* is not true.
<=	Checks if the value of left operand isless than or equal to the value of right operand, if yes then condition becomes true.	*A* > *B* is not true

Logical Operators :

There are following logical operators supported by C++ language Assume variable A holds 1 and variable B holds 0, then:

Operator	**Description**	**Example**
&&	Called Logical AND operator.	A&&B is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	*A* | | *B* is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	A&&B is true.

Bitwise Operators :

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows :

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

Assume if A = 60; and B = 13; now in binary format they will be as follows :

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The Bitwise operators supported by C++ language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then :

Operator	**Description**	**Example**
&	Binary AND Operator copies a bit to the result if it exists in both operands.	A & B will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	*A* | *B* will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	*A* ^ B will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	*A* will give -61 which is 1100 0011 in 2's complement form due to a signed binary number
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators :

There are following assignment operators supported by C++ language :

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to 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 Bitwise AND assignment operator	C >>= 2 is same as C = C >>2 &= C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator	C |=2 is same as C = C | 2

Misc Operators :

There are few other operators supported by C++ Language.

Operator	**Description**
sizeof	sizeof operator returns the size of a variable. For example, sizeof*a*, where a is integer, will return4.
Condition ? X : Y	Conditional operator. If Condition is true ? then it returns value X : otherwise value Y
,	Comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
. *dot* and -> *arrow*	Member operators are used to reference individual members of classes, structures, and unions.
Cast	Casting operators convert one data type to another. For example, int2.2000 would return 2.
&	Pointer operator & returns the address of a variable. For example &a; will give actual address of the variable.
*	Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var.

Operators Precedence in C++ :

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator :

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	[] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - *type** & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

11.5 SUMMARY :

In this chapter we have learned the special method that is automatically called when an object of a class is created and the operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

11.6 KEY WORDS :

Constructors : In class-based, object-oriented programming, a constructor is a special type of function called to create an object.

Parameterized constructors : Constructors are special function named after the class and without a return type, and are used to construct objects. Constructors, like function, can take input parameters. Constructors are used to initialize objects.

Operators : In computer programming, operators are constructs defined within programming languages which behave generally like functions, but which differ syntactically or semantically.

11.7 SELF ASSESSMENT QUESTIONS :

1. What is a Constructor? Discuss the properties of constructors?
2. Explain the constructors with default arguments?
3. Write about the Operators?
4. Discuss the types of Operators?

11.8 FURTHER READINGS :

1. Schildt, H., The Complete Reference C++,. New Delhi: Tata McGraw-Hill, 2005.
2. Balaguruswamy, E., Object-oriented Programming in C++. New Delhi: Tata McGraw-Hill, 2006.

LESSON – 12
INHERITANCE

AIMS AND OBJECTIVES:

The objectives of this lesson are to learn about

- the concept of inheritance
- how a class can be derived from the base class with desired properties / characteristics
- how the visibility modes control the members of a base class that are inherited into a derived class with the access – specifier(s) etc.

STRUCTURE OF THE LESSON:

12.1 INTRODUCTION TO INHERITANCE :

The main feature of object-oriented programming is the reusability of code that can be achieved through inheritance. You can reuse an existing class to create a new class without modifying it. Inheritance is a process of creating new classes from the existing one. The existing class is a vital component of inheritance. You can add new features to an existing class when creating a new class called the derived class. The existing class is known as the base class. The derived class can use all the data members and member functions of the base class, if appropriate access specifiers, discussed later in the unit, are given. For instance, take an example of an automobile class. The classes car, bus, truck and scooter are derived classes of the automobile base class. All these classes share the properties and functions of the automobile class

12.1.1 Advantages and Disadvantages of Inheritance :

In object – oriented programming languages, inheritance is used to extend the functionality of abstract data types or classes. The advantages of inheritance are given below.

The most important advantage of inheritance is the reusability of code. The existing class code remains the same and is used to create new derived classes.

Software development time is reduced because of the reusability of code. One base class can be used by a number of derived classes.

The derived classes use the properties of the base class. So, the object of the derived class becomes more powerful than that of the base class.

Following are the disadvantages of inheritance.

Inheritance is used to simplify a complicated project: sometimes inappropriate use of inheritance makes a project more complicated. Calling the member functions of a base class through derived class objects causes more compiler overheads.

Sometimes the memory is not properly used in the class hierarchy, as lot of data elements remain unused.

12.1.2 Use of Access Specifier in Inheritance :

Three access specifiers, public, private and protected are used in C++. The keyword protected is used in inheritance. The protected data members and functions behave like public specifiers for derived classes and like private specifiers for other classes. The following program shows the use of the keyword protected :

```
// program to use protected access specifier

        #include<iostream.h>
        #include<conio.h>
        class emp
            {
              protected:
                 int empno;
                 char name[20];
            } ;
```

```
class emp_details : emp
     {
        Public:
           void add_exp()
                {
                     cout<<"Enter Empno:";
                     cin>>empno:
                     cout<<"Enter name:";
                     cin>>name;
                }
           void display()
                {
                     cout<<"Empnos*<<empno;
                     cout<<"\nName="<<name;
                }
     } ;
void main()
     {
           emp_detailsd;
           clrscr();
           d.add_emp();
           d.display();
           getch();
      }
```

The above program shows that the class Emp is able to access data members of the class emp details because they are declared using the protected keyword. If you change the protected keyword with private, the program gives errors. The below figure shows output of the above program.

```
Enter Empno:23
Enter name:Ajay
Empno=23
Name=Ajay
```

12.1.3 Access Specifiers while Deriving a New Class :

A class can also be inherited in three ways from the base class, public, protected and private. There are three ways in which the scope of the data members can be used in a derived class. The following code shown the first way.

```
Class A
     {
           //members of class A
     };
```

```
Class B : public A
    {
        // members of class B
    };
```

The following code shows the second way.

```
Class A
    {
        // members of class A
     };
Class B : protected A
     {
        // members of class B
    };
```

The following code shows the third way.

```
Class A
    {
        // members of class A
    };
Class B; private A
    {
        // members of class B
    };
```

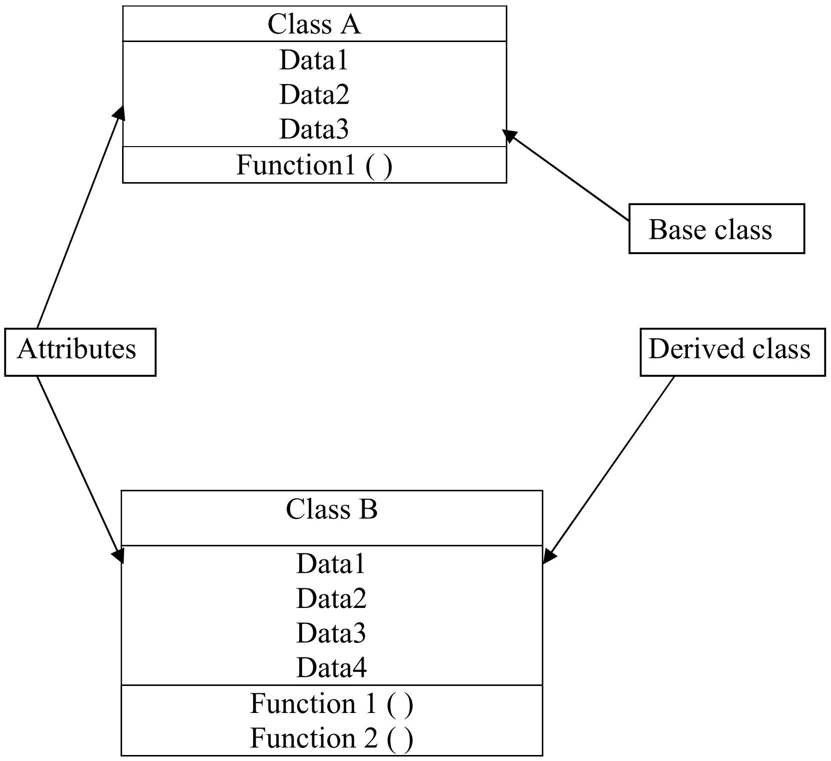
12.2 TYPES OF INHERITANCE :

Different types of inheritance are available in C++ based of the number of base classes and nesting of derivations, such as single inheritance, multilevel inheritance, multiple inheritance, hierarchical, hybrid and multipath inheritance.

12.2.1 Single Inheritance:

When only one class is used to derive a new class, the inheritance between the base class and the derived class is called single inheritance . In this case, the derived class is not used as a base class.

Figure below shows an example of single inheritance.



In the above figure, the B class uses the data members and member functions from the base class, A in addition to its own data members, Data4 and function, Funciton2 ( ).

The following program shows implementations of single inheritance.

```
// program to use the protected data from base class
#include<iostream.h>
#include<conio.h>
#include<sring.h>
Class person
    {
        Char name [20];
        Int age;
        Public:
            Void add _ person (char *n, int a)
              {
                  Strcpy(name,n);
                  Age=a;
              }
```

```
                    Void print_person ( )
                       {
                              Cout<< " Name is "<<name;
                              Cout<<"\nAge is "<<age;
                       }
        };
    Class employee:person
        {
              Int empno;
              Public :
                    Void add_emp( )
                       {
                              Char n [20];
                              Int a;
                              Cout<<"Enter empno:";
                              Cin>>empno;
                                Cout<<"Enter name:";
                                Cin>>n;
                                Cout<<"Enter age:";
                                Cin>>a;
                                Add_person(n,a);
                       }
                    Void print_emp( )
                       {
                                Cout<<"Empno is "<<empno<<end1;
                                Print_person( );
                       };
        };
    Void main ( )
          {
                       Employee e;
                       Clrscr ( );
                       e.add_emp ( );
                       e.print_emp ( );
                       getch( );
          }
```

The below figure shows output of the above program.

```
Enter empno:34
Enter name:Ajay
Enter age :22
Empno is 34
Name is Ajay
Age is 22
```

12.2.2 MULTILEVEL INHERITANCE :

In multilevel inheritance, you derive a class from a derived class. For example, a class B is inherited from a class A, and a class C is inherited from the class B.

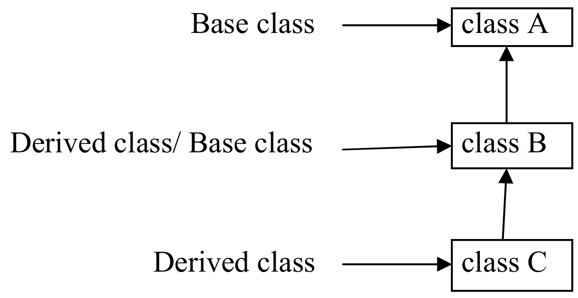


Figure the Multilevel Inheritance

The class B acts as both, a derived class and a base class. It is a derived class for class A and basw class for class C. The class C can access members of class B as well as class A. The following program shows an example of multilevel inheritance.

```
// program to represent the multilevel inheritance
        #include<iostream.h>
        #include<conio.h>
        Class Auto
                {
                        Protected:
                        Float mileage;
                        Int speed;
                };
        Class Fourwheelers: public Auto
                {
                        Protected:
                        Char color [10];
                };
```

```
Class car: public Fourwheelers
        {
                Protected:
                Char carNo[10];
                Char model[10];
                Public:
                Void input ( )
                   {
                        cout<<"Enter the model of a Car ";
                        cin>>model;          //Inherits the attribute of the scooter class
                        cout<<"Enter the color of a car";
                        cin>>color;          //inherits the attributes of the scooter class
                        cout<<"Enter the car number ";
                        cin>>carNo;
                        cout<<"Enter the mileage of a car ";
                        cin>>mileage;        //inherits the attribute of the vehicle class
                        cout<<"enter the speed of a car ";
                        cin>>speed;          //inherits the attributes of the vehicle class
                   }
                Void display ( )
                    {
                        cout<<end1;
                        cout<<"car details are :"<<end1;
                        cout<<"model : "<<model<<end1;
                        cout<<"color : "<<color<<end1;
                        cout<<"number : "<<carNo<<end1;
                        cout<<"mileage : "<<mileage<<" km p 1"<<end1;
                        cout<<"speed : "<<speed<<"km p h "<<end1;
                    }
        };
Void main( )
        {
               Car obj;
               Obj.input( );
```

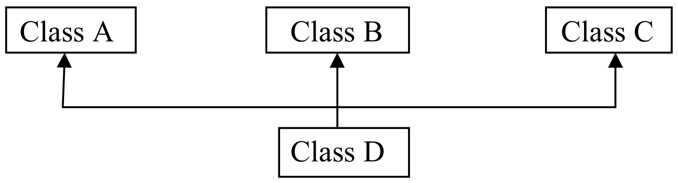
```
                Obj.display( );
                getch ( );
        }
```

In the above program, Auto is the top base class, form which a class Four wheelers has been derived. The class Scooter has been derived from the Four wheelers class. In the main function, the object of scooter class is able to access members from the Auto and Four Wheelers Classes. The below figure shows output of the above program.

```
Enter the model of a Car: Civic
Enter the color of a Car: White
Enter the Car number: DL3C-6030
Enter the mileage of a Car: 12
Enter the speed of a Car: 200

Car Details are :
Model : Civic
Color : White
Number:DL3C-6030
Mileage:12 km p 1
Speed: 200 km p h
```

12.2.3 MULTIPLE INHERITACE :

The process where you derive one class from two or more base classes is called multiple inheritance. The class derived this way inherits the properties of all base classes. For example, a child inherits propertied from both his mother as well as his father. Figure below shows multiple inheritance.



The class D has been derived from class A, Class B and class C. Therefore, it can access members of all the three base classes. The syntax to implement multiple inheritance is

```
Class derived: public Base1, Base2
    {
            // class Body
    }
```

For instance, consider the example of a seaplane that comes in different categories, an air vehicle and a water vehicle. The following program shows how to implement multiple inheritance.

```
//program to inherit the properties of multiple base classes in a derived class
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
Class addition
        {
                protected:
                        int add(int a, int b)
                           {
                                return(a+b);
                           }
        };
Class multiplication
        {
                protected:
                        int mul(int a, intb)
                           {
                                return (a*b);
                           }
        };
Class number :addition, multiplication
        {
                Public:
                        Void solve ( );
        };
Void number : : solve( )
        {
                Int a,b,v,I;
                do
                  {
                      cout<<"\n\n1\tAdd\n2\multiply\n3\tquit";
                      cout<<"\nEnterur choice:";
                      cin>>c;
                      switch (c)
```

```
                        {
                            case 1:
                                  cout<<"Enter two numbers:";
                                  cin>>a>>b;
                                  cout<<"Addition of "<<a<<" and
                                   "<<b<<" is "<<add(a,b);
                                    break;
                            case 2;
                                    count<<"Enter two numbers:";
                                    cin>>a>>b;
                                    cout<<"multiplication of "<<a<<"and
                                    "<<b<<" is "<<mul(a,b);
                                    break;
                            case 3:
                                    exit(0);
                            default:
                            cout<<"Invalid choice";
                            break;
                        }
                }while(c!=3);
        }
    Void main( )
        {
                Number n;
                clrscr ( ) ;
                n.solve( );
        }
```

The above program shows that two base classes have been defined, One and Two. The class Three has been derived from both base classes. The protected data members of both classes have been accessed in derived class, Three.

The below figure shows output of the above program.

```
MAIN MENU
1    Add
2    Multiply
3    Quit
Enter ur choice: 1
Enter two numbers: 3
4
Addition of 3 and 4 is: 7

MAIN MENU
1    Add
2    Multiply
3    Quit
Enter ur choice: 2
Enter two numbers: 3
4
Multiplcation of 3 and 4 is : 12

MAIN MENU
1     Add
2     Multiply
3     Quit
Enter ur choice : 3
```

12.3 Derived class declaration :

The derived class extends its features by inheriting the properties of another class, called base class and adding features of its own. The declaration of derived class specifies its relationship with the base class in addition to its own features.

The syntax for declaring a derived class is :

```
class derived class:[visibility mode Or Access specifier] baseclass
    {
            //Derived class members (member functions and variables)
    };
```

In the above syntax **class** is a keyword, **derived class** is name of the new class. "**:**" is used as a separator between the derived class name and the access speciifier. The visibility mode tells the way in which the base class is inherited (private, protected or public).The visibility mode is optional and the default mode is private. “base class” is the name of the class from which the properties are being inherited.

```
e.g.:   class student : private / public person
               ↓              ↓              ↘
derived class name     Access specifier        base class
          {
          ………  };
```

12.3.1 Visibility Modes : There are three types of visibility modes. They are :

i) Private
ii) Protected
iii) Public

These are used for specifying the way in which the properties of the base class are inherited.

Private : If the access specifier “private”, is used to inherit the properties of base class, then

i) The private data of the base class cannot be inherited but can be accessed through the inherited member.
ii) The protected data of the base class is inherited as the private data. It cannot be used in the main function. But, it can be accessed using the base class or the derived class member function.
iii) The public data of the base class is inherited as private data in the derived class. They are inaccessible to the objects of the derived class. It can be accessed by the member functions of the derivedclass.

Eg.:
```
class base
    {
        private:
            intx; readx();
        protected:
            inty; ready();
        public:
            intz; readz();
    }

classder:privatebase
    {
        private:
            int w;
        public:
        void read( );
        void display( ):
    };
```

In the above example, variable x cannot be inherited, but both y and z are inherited as private variables in the derived class. They can be accessed through the functions read() and display(). They cannot be used directly by the main().

Protected : When the access specifier "protected" is used to inherit base class properties,

- The private data of the base class cannot be inherited but can be accessed through the inherited members.
- Protected member in the base class are inherited as protected data in the derived class.
- The public data in the base class is inherited as protected data of the derived class.

```
class base
    {
        private:
            int x; readx();
        protected:
            int y; ready();
        public:
          int z; readz();
    }
class der: protected base
    {
        private:
          int w;
        public:
          void read( );
          void display( );
    };
```

In the above example, 'x' cannot be inherited. y and z are inherited as protected members and thus can be used in read() and display() they can be further inherited but they can not be accessed from the ,main

Public : If the access specifier "public" is used to inherit the properties of base class, then

- The private data of the base class cannot be inherited as member of derived class but can be accessed through the inherited member functions.

- The protected member of the base class is the member of the derived class.
- The public member of the base class is the public member in the derived class.

```
class base
      {
          private:
             int x; readx();
          protected:
             int y; ready();
          public:
             int z; readz();
       }
class der: public base
       {
          private:
             int w;
          public:
             void read( );
             void display( );
       };
```

In the above example 'x' cannot be inherited into the derived class, 'y' is inherited as a protected member of the derived class and 'z' can be accessed from the main itself.

Visibility Of Inherited Members:

Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

12.5 SUMMARY :

It explained to you the concept of inheritance .C++ allows you to inherit the data members and member functions of the base class to reuse the members for performing complex tasks in a program. Inheritance is of the following types. Then the visibility mode (private, public or protected) in the definition of the derived class specifies whether the features of the base class are privately derived, publicly derived or protected derived.

12.5 KEY WORDS :

Single Inheritance : The inheritance in which a derived class is inherited from the only one base class.

Multiple Inheritances : Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes.

Multilevel Inheritance : In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class.

Hierarchical Inheritance : Hierarchical Inheritance in C++ refers to the type of inheritance that has a hierarchical structure of classes.

Hybrid Inheritance : Hybrid Inheritance in C++ is the process by which a sub class follows multiple types of inheritance while deriving properties from the base or super class

Private : Members cannot be accessed (or viewed) from outside the class.

Protected : Members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

Public : Members are accessible from outside the class.

12.6 SELF ASSESSMENT QUESTIONS :

1. What is Inheritance
2. Write the Use of Access Specifier in Inheritance
3. Explain types of Inheritance
4. Types of visibility modes
5. Differences between Private, Protected & Public

12.7 FURTHER READINGS :

1. Schildt, H., The Complete Reference C++,. New Delhi: Tata McGraw-Hill, 2005.
2. Balaguruswamy, E., Object-oriented Programming in C++. New Delhi: Tata McGraw-Hill, 2006

LESSON - 13

HIERARCHICAL INHERITANCE

AIMS AND OBJECTIVES :

The objectives of this lesson are to learn about

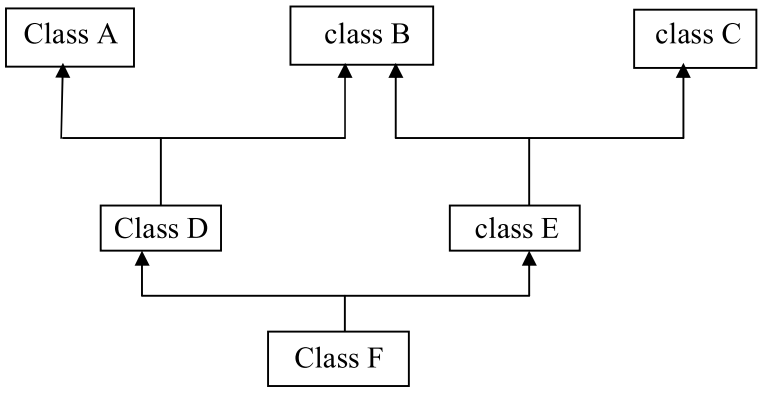
- ➢ what is the hierarchy of derived classes in CPP
- ➢ how the hierarchy of the derived class can be decided in CPP
- ➢ the implementation of hierarchy in CPP etc.

STRUCTURE OF THE LESSON :

13.1 HIERARCHICAL INHERITANCE :

In this type of inheritance, one or more derived classes can be created using one or more base classes. Hierarchical inheritance shows top-down style through splitting a complex class into many subclasses.

The below figure is an example of hierarchical inheritance.



Hierarchical Inheritance

The classes A, B and C are the base classes and the class D derives from the base classes A and B. The class E derives from the base classes. B and C. The class F derives from the base classes, D and E. The figure illustrates the hierarchical relationship between classes. The following program shows the hierarchical relationship between classes.

```
// hierarchial inheritance
#include <iostream>
using namespace std;

class A                                              //single base class
        {
                public:
               int x, y;
               void getdata()
                       {
                               cout << "\nEnter value of x and y:\n"; cin >> x >> y;
                       }
        };
class B : public A                                   //B is derived from class base
        {
               public:
               void product()
                       {
                                cout << "\nProduct= " << x * y;
                       }
        };
class C : public A                                   //C is also derived from class base
        {
                public:
               void sum()
                       {
                                 cout << "\nSum= " << x + y;
                       }
        };
int main()
        {
               B obj1;                                       //object of derived class B
               C obj2;                                       //object of derived class C
                obj1.getdata();
                obj1.product();
                obj2.getdata();
                 obj2.sum();
               return 0;
        }                                                    //end of program
```

Output :

```
Enter value of x and y :
2
3
Product= 6
Enter value of x and y :
2
3
Sum= 5
```

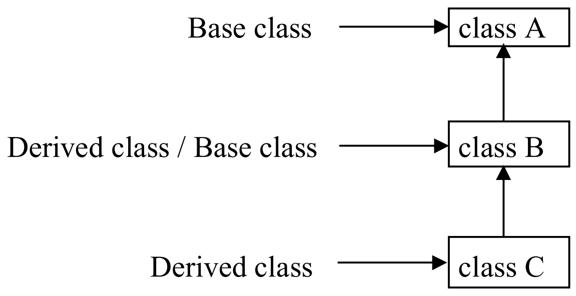
In the above example, there is only one base class A from which two class B and C are derived.

Both derived class have their own members as well as base class members.

The product is calculated in the derived class B, whereas, the sum is calculated in the derived class C but both use the values of x and y from the base class.

13.2 MULTILEVEL INHERITANCE :

In multilevel inheritance, you derive a class from a derived class. For example, a class B is inherited from a class A, and a class C is inherited from the class B.



Multilevel Inheritance

The class B acts as both, a derived class and a base class. It is a derived class for class A and basw class for class C. The class C can access members of class B as well as class A. The following program shows an example of multilevel inheritance.

```
// program to represent the multilevel inheritance
#include<iostream.h>
#include<conio.h>
Class Auto
        {
                Protected:
                Float mileage;
                Int speed;
        };
```

```
Class Fourwheelers: public Auto
        {
                Protected:
                Char color [10];
        };
Class car: public Fourwheelers
        {
                Protected:
                Char carNo[10];
                Char model[10];
                Public:
                Void input ( )
                    {
                        cout<<"Enter the model of a Car ";
                        Cin>>model; //Inherits the attribute of the scooter class
                        cout<<"Enter the color of a car";
                        cin>>color; //inherits the attributes of the scooter class
                        cout<<"Enter the car number ";
                        cin>>carNo;
                        cout<<"Enter the mileage of a car ";
                        cin>>mileage; //inherits the attribute of the vehicle class
                        cout<<"enter the speed of a car ";
                        cin>>speed; //inherits the attributes of the vehicle class
                    }
                Void display ( )
                    {
                        Cout<<end1;
                        Cout<<"car details are :"<<end1;
                        Cout<<"model : "<<model<<end1;
                        Cout<<"color : "<<color<<end1;
                        Cout<<"number : "<<carNo<<end1;
                        Cout<<"mileage : "<<mileage<<" km p 1"<<end1;
                        Cout<<"speed : "<<speed<<"km p h "<<end1;
                    }
        };
```

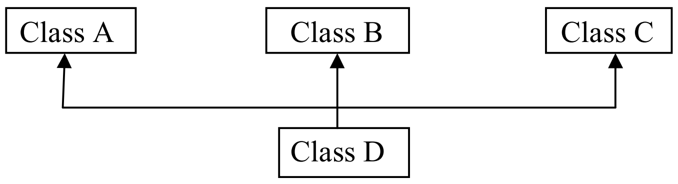
```
Void main( )
        {
                Car obj;
                Obj.input( );
                Obj.display( );
                getch( );
        }
```

In the above program, Auto is the top base class, form which a class Four wheelers has been derived. The class Scooter has been derived from the Four wheelers class. In the main function, the object of scooter class is able to access members from the Auto and Four Wheelers Classes. The below figure shows output of the above program.

```
Enter the model of a Car: Civic
Enter the color of a Car: White
Enter the Car number: DL3C-6030
Enter the mileage of a Car: 12
Enter the speed of a Car: 200
Car Details are :
Model : Civic
Color : White
Number:DL3C-6030
Mileage:12 km p 1
Speed: 200 km p h
```

13.3 MULTIPLE INHERITACE :

The process where you derive one class from two or more base classes is called multiple inheritance. The class derived this way inherits the properties of all base classes. For example, a child inherits propertied from both his mother as well as his father. The below figure shows multiple inheritance.



The class D has been derived from class A, Class B and class C. Therefore, it can access members of all the three base classes. The syntax to implement multiple inheritance is:

```
Class derived: public Base1, Base2
        {
                        // class Body
        }
```

For instance, consider the example of a seaplane that comes in different categories, an air vehicle and a water vehicle. The following program shows how to implement multiple inheritance.

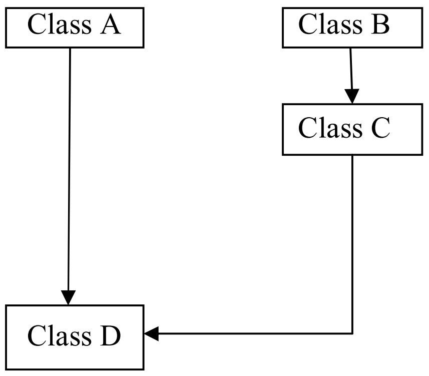
```
//program to inherit the properties of multiple base classes in a derived class
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
Class addition
    {
        protected:
                int add(int a, int b)
                        {
                                return(a+b);
                        }
    };
Class multiplication
    {
        Protected:
                int mul(int a, intb)
                        {
                                Return (a*b);
                        }
    };
Class number :addition, multiplication
    {
        Public:
                Void solve ( );
    };
Void number : : solve( )
    {
        Int a,b,v,I;
        Do
            {
                Cout<<"\n\n1\tAdd\n2\multiply\n3\tquit";
                Cout<<"\nEnterur choice:";
                Cin>>c;
                Switch (c)
                    {
                        Case 1:
                                Cout<<"Enter two numbers:";
                                Cin>>a>>b;
                                Cout<<"Addition of "<<a<<" and"<<b<<" is
                                          "<<add(a,b);
                                Break;
                        Case 2;
                                Count<<"Enter two numbers:";
                                Cin>>a>>b;
                                Cout<<"multiplication of "<<a<<"and
                                          "<<b<<" is "<<mul(a,b);
                                Break;
                        Case 3:
                                Exit(0);
                        Default:
                                Cout<<"Invalid choice";
                                Break;
                    }
            }while(c!=3);
    }
Void main( )
    {
        Number n;
        clrscr( ) ;
        n.solve( );
    }
```

The above program shows that two base classes have been defined, One and Two. The class Three has been derived from both base classes. The protected data members of both classes have been accessed in derived class,Three. Figure shows the output of the above program.

```
MAIN MENU
1    Add
2    Multiply
3    Quit
Enter ur choice: 1
Enter two numbers: 3
4
Addition of 3 and 4 is: 7

MAIN MENU
1    Add
2    Multiply
3    Quit
Enter ur choice: 2
Enter two numbers: 3
4
Multiplcation of 3 and 4 is : 12

MAIN MENU
1     Add
2     Multiply
3     Quit
Enter ur choice : 3
```

13.4 HYBRID INHERITANCE :

When you derive anew class from one or more types of inheritance, the process is called hybrid inheritance. The below figure shows an example of hybrid inheritance.



Hybrid Inheritance

Here, class D is an example of hybrid inheritance because it has been derived from two classes A and C. Class A is a simple class and class C is a derived class of class B. The following program shows hybrid inheritance.

```
// hybrid inheritance
#include <iostream>
using namespace std;

class A
        {
                public:
                int x;
        };
class B : public A
        {
                public:
                B()                 //constructor to initialize x in base class A
                   {
                        x = 10;
                   }
        };
class C
        {
                public:
                int y;
                C()                 //constructor to initialize y
                   {
                        y = 4;
                   }
        };
class D : public B, public C        //D is derived from class B and class C
        {
                public:
                void sum()
                    {
                        cout << "Sum= " << x + y;
                    }
        };
int main()
    {
        D obj1;                     //object of derived class D
        obj1.sum();
        return 0;
    }                               //end of program
```

Output

```
Sum= 14
```

13.5 SUMMARY :

In this chapter we have learned the Inheritance concepts and also in hierarchical inheritance, more than one derived class is created from a single base class.

13.6 KEY WORDS :

Inheritance : Inheritance is a feature or a process in which, new classes are created from the existing classes.

Super class : The parent class whose properties are inherited by another class.

Sub class : The class that inherits properties from another class.

Hierarchical inheritance :

Hierarchical Inheritance in C++ refers to the type of inheritance that has a hierarchical structure of classes. And also a single base class can have multiple derived classes, and other subclasses can further inherit these derived classes, forming a hierarchy of classes.

13.7. SELF ASSESSMENT QUESTIONS :

1. Explain the types f inheritance?
2. Compare the hierarchical inheritance and multilevel inheritance?
3. Discuss the multiple inheritance?
4. Write about hybrid inheritance?

13.8. FURTHER READINGS :

1. Schildt, H., The Complete Reference C++,. New Delhi: Tata McGraw-Hill, 2005.
2. Balaguruswamy, E., Object-oriented Programming in C++. New Delhi: Tata McGraw – Hill, 2006.

LAB MANUAL

310BCO21 – COURSE 3C : PROGRAMMING WITH C & C++ Practical Component

1. Write C programs for
    a. Fibonacci Series
    b. Prime number
    c. Palindrome number
    d. Armstrong number.

2.' C' program for multiplication of two matrices

3.'C' program to implement string functions

4.' C' program to swap numbers

5.'C' program to calculate factorial using recursion

6.'C++' program to perform addition of two complex numbers using constructor

7. Write a program to find the largest of two given numbers in two different classes using friend function

8. Program to add two matrices using dynamic constructor

9. Implement a class string containing the following functions:
    a) Overload + operator to carry out the concatenation of strings.
    b) Overload = =operator to carry out the comparison of strings.

10. Program to implement inheritance.

<u>Programs Code :</u>

1. Write C programs to execute the following

i) Fibonacci Series ii) Prime Number iii) Palindrome Number iv) Armstrong Number

i) FIBONACCI SERIES

Program :

```
#include <stdio.h>

int main()

{

        int i, n;

                                                        // initialize first and second terms

        int t1 = 0, t2 = 1;

                                                        // initialize the next term (3rd term)

        int nextTerm = t1 + t2;

                                                        // get number of terms from user

        printf("Enter the number of terms: ");

        scanf("%d", &n);

                                                        // print the first two terms t1 and t2

         printf("Fibonacci Series: %d, %d, ", t1, t2);

                                                        // print 3rd to nth terms

          for (i = 3; i <= n; ++i)

              {
                printf("%d, ", nextTerm);

                t1 = t2;

                 t2 = nextTerm;

                 nextTerm = t1 + t2;

              }

    return 0;

}
```

OUTPUT :

```
Enter the number of terms: 10

Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

ii) PRIME NUMBER

Program :

```
#include <stdio.h>

int main()
{
      int n, i, flag = 0;
      printf("Enter a positive integer: ");

      scanf("%d", &n);

                                                        // 0 and 1 are not prime numbers

                                                  // change flag to 1 for non-prime number

       if (n == 0 || n == 1)

       flag = 1;

        for (i = 2; i <= n / 2; ++i)

          {
                                                  // if n is divisible by i, then n is not prime
                                                  // change flag to 1 for non-prime number

               if (n % i == 0)

                  {

                      flag = 1;

                      break;

                  }

          }

                                                            // flag is 0 for prime numbers
```

```
    if (flag == 0)

        printf("%d is a prime number.", n);

    else

        printf("%d is not a prime number.", n);

     return 0;

}
```

OUTPUT :

```
Enter a positive integer : 17

17  is a prime number
```

iii) PALINDROME NUMBER

Program :

```
#include<stdio.h>

int main()

{
       int n,r,sum=0,temp;
       printf("enter the number=");

       scanf("%d",&n);

       temp=n;

       while(n>0)
          {
               r=n%10;
               sum=(sum*10)+r;
               n=n/10;
          }
        if(temp==sum)
              printf("palindrome  number ");

        else

              printf("not a palindrome number");

        return 0;
}
```

OUTPUT :

```
Enter the number=1001
palindrome number
```

iv) ARMSTRONG NUMBER

Program :

```
#include <stdio.h>

int main()
{
        int num, originalNum, remainder, result = 0;
        printf("Enter a three digit integer: ");

        scanf("%d", &num);

        originalNum = num;

        while (originalNum != 0)
            {
                                                        // remainder contains the last digit
              remainder = originalNum % 10;

              result += remainder * remainder * remainder;

                                                // removing last digit from the orignal number

             originalNum /= 10;

            }

           if (result == num)
                 printf("%d is an Armstrong number.", num);
           else
                 printf("%d is not an Armstrong number.", num);
           return 0;
}
```

OUTPUT :

```
Enter a three-digit integer : 153

153 is an Armstrong number.

Enter a three-digit integer : 123

123 is not an Armstrong number.
```

2. Write a C program to execute multiplication of two matrices

Program :

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int mat1[3][3], mat2[3][3], mat3[3][3], sum=0, i, j, k;
    printf("Enter first 2*2 matrix element: ");
    for(i=0; i<2; i++)
       {
          for(j=0; j<2; j++)
          scanf("%d", &mat1[i][j]);
       }
    printf("Enter second 2*2 matrix element: ");
    for(i=0; i<2; i++)
       {
          for(j=0; j<2; j++)
          scanf("%d", &mat2[i][j]);
       }
     printf("\nMultiplying two matrices...");
     for(i=0; i<2; i++)
       {
           for(j=0; j<2; j++)
              {
                  sum=0;
                 for(k=0; k<2; k++)
                 sum = sum + mat1[i][k] * mat2[k][j];
                 mat3[i][j] = sum;
              }
       }
     printf("\nMultiplication result of the two given Matrix is: \n");
```

```
      for(i=0; i<2; i++)
         {
            for(j=0; j<2; j++)
            printf("%d\t", mat3[i][j]);
            printf("\n");
         }
  getch();
  return 0;
}
```

OUTPUT :

```
Enter first 2*2 matrix element: 1

2

3

4

Enter second 2*2 matrix element: 1

2

3

4

Multiplying two matrices...

Multiplication result of the two given Matrix is:

7     10

15     22
```

3. Write a C program to implement string functions

Program :

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[10],b[10];
    int ch,len;
    printf("enter str1 ");
    scanf("%s",a);
    printf("enter str2 ");
    scanf("%s",b);
   while(1)
      {
        printf("\n choose ur option");
        printf("\n 1.length\n 2.compare\n 3.copy\n 4.concat\n");
        printf("enter ur choice: ");
        scanf("%d",&ch);
        switch(ch)
           {
             case 1: len=strlen(a);
```

```
                printf("length is %d\n",len);

                break;

                case 2:if(strcmp(a,b)==0)

                       {

                            printf("both strings are equal\n");

                       }

        else

            if(strcmp(a,b)>0)

                printf("%s is greater than %s\n",a,b);

            else

                printf("%s is greater than %s\n",b,a);

         break;

           case 3: printf(" str1 %s\n",a);

                printf("str2 %s\n",b);

               strcpy(a,b);

               printf("after copy strings are\n");

               printf(" str1 %s\n",a);

               printf("str2 %s\n",b);

         break;

           case 4:printf(" str1 %s\n",a);

               printf("str2 %s\n",b);
```

```
                strcat(a,b);

                printf(" str1 %s\n",a);

       break;

  default:

  return 0;

            }

        }

return 0;

}
```

OUTPUT :

```
enter str1 hello

enter str2 hai


 choose ur option

 1.length

 2.compare

 3.copy

 4.concat

enter ur choice: 1

length is 5


 choose ur option

 1.length

 2.compare
 3.copy
 4.concat
enter ur choice: 2
hello is greater than hai

 choose ur option
 1.length
 2.compare
 3.copy
 4.concat
enter ur choice: 3
 str1 hello
str2 hai
after copy strings are
 str1 hai
str2 hai

 choose ur option
 1.length
 2.compare
 3.copy
 4.concat
enter ur choice: 4
 str1 hai
str2 hai
 str1 haihai
 choose ur option

 1.length

 2.compare

 3.copy

 4.concat

enter ur choice: 5
```

4. Write a C program to swap two numbers

Program :

```
// C program to swap two variables

#include <stdio.h>

int main()

{

    int x, y;

    printf("Enter Value of x ");

    scanf("%d", &x);

    printf("\nEnter Value of y ");

    scanf("%d", &y);

    int temp = x;

    x = y;

    y = temp;

    printf("\nAfter Swapping: x = %d, y = %d", x, y);

    return 0;

}
```

OUTPUT :

```
Enter Value of x 6

Enter Value of y 12

After Swapping: x = 12, y = 6
```

5. Write a C program to calculate factorial using recursion

Program :

```
#include<stdio.h>

long int multiplyNumbers(int n);

int main()

{

        int n;

       printf("Enter a positive integer number: ");

       scanf("%d",&n);

       printf("Factorial of %d = %ld", n, multiplyNumbers(n));

       return 0;

}

long int multiplyNumbers(int n)

 {

        if (n>=1)

           return n*multiplyNumbers(n-1);

        else

         return 1;

}
```

OUTPUT :

```
Enter a positive integer: 6
Factorial of 6 = 720
```

6. Write a C++ program to perform addition of two complex numbers using constructor

Program :

```
#include<bits/stdc++.h>

using namespace std;

                                                        // User Defined Complex class

class Complex

{

                                    // Declaring variables public: int real, imaginary;

                                     // Constructor to accept real and imaginary part

   Complex(int tempReal = 0, int tempImaginary = 0)

     {

        real = tempReal;

        imaginary = tempImaginary;

     }

                                                        // Defining addComp() method

                                                      // for adding two complex number

   Complex addComp(Complex C1, Complex C2)

      {

                                                        // Creating temporary variable

         Complex temp;

                                                  // Adding real part of complex numbers
```

```
            temp.real = C1.real + C2.real;

                                                        // Adding Imaginary part of

                                                                // complex numbers

            temp.imaginary = (C1.imaginary + C2.imaginary);

                                                                // Returning the sum

            return temp;

        }

};

// Driver code

int main()

{

                                                            // First Complex number

    Complex C1(3, 2);

                                                    // printing first complex number

    cout << "Complex number 1 : " <<

    C1.real << " + i" <<

    C1.imaginary << endl;

                                                        // Second Complex number

    Complex C2(9, 5);

                                                // Printing second complex number

    cout << "Complex number 2 : " <<
    C2.real << " + i" <<

    C2.imaginary << endl;

                                                  // For Storing the sum

    Complex C3;

                                                  // Calling addComp() method

    C3 = C3.addComp(C1, C2);

                                                  // Printing the sum

    cout << "Sum of complex number : " <<

    C3.real << " + i" <<

    C3.imaginary;

}
```

OUTPUT :

```
Complex number 1 : 3 + i2

Complex number 2 : 9 + i5

Sum of complex number : 12 + i7
```

7. Write a program to find the largest of two given numbers in two different classes using friend function

Program :

```
#include<iostream>
using namespace std;
class Test
{
    private:
    int x, y;
    public:
   void input()
      {
          cout << "Enter two numbers:";
          cin >> x>>y;
       }
     friend void find(Test t);
};
void find(Test t)
{
    if (t.x > t.y)
       {
```

```
                cout << "Largest is:" << t.x;
            }
        else
            {
                cout << "Largest is:" << t.y;
             }
}
int main()
{
        Test t;
         t.input();
         find(t);
         return 0;
}
```

OUTPUT :

```
Enter two numbers:20 30
Largest is:30
```

8. Program to add two matrices using dynamic constructor

Program :

```
#include<iostream>

using namespace std;

class Dc

{

       int num1;

       int num2;

       int *ptr;

       public:

                                                    // default constructor (here, it is dynamic constructor also)

      Dc()

        {

             num1 = 0;

             num2 = 0;

             ptr = new int;

         }

                                                                      //dynamic constructor with parameters

     Dc(int x, int y, int z)

        {

              num1 = x;
```

```
            num2 = y;
            ptr = new int;
            *ptr = z;
        }
    void display()
        {
            cout << num1 << " " << num2 << " " << *ptr;
        }
};
int main()
{
      Dc obj1;
      Dc obj2(3, 5, 11);
      obj1.display();
      cout << endl;
      obj2.display();
}
```

OUTPUT :

```
0 0 0
3 5 11
```

9. Implement a class string containing the following functions:

a . Overload + operator to carry out the concatenation of strings.

b. Overload = =operator to carry out the comparison of strings.

Program :

```
#include<iostream>

#include<string.h>

using namespace std;

class String

{
    public:
    char str[20];

    public:

    void accept_string()

      {

        cout<<"\n Enter String              :   ";

        cin>>str;

      }

    void display_string()

      {

        cout<<str;

      }
    String operator+(String x)                              //Concatenating String
        {
            String s;

            strcat(str,x.str);

            strcpy(s.str,str);

            return s;
        }
};
int main()
{
    String str1, str2, str3;
    str1.accept_string();

    str2.accept_string();

    cout<<"\n ---------------------------------------------";

    cout<<"\n\n First String is          :  ";

    str1.display_string();                                  //Displaying First String

    cout<<"\n\n Second String is         :  ";

    str2.display_string();  //Displaying Second String

    cout<<"\n ---------------------------------------------";

    str3=str1+str2;                        //String is concatenated. Overloaded '+' operator

    cout<<"\n\n Concatenated String is   :  ";

    str3.display_string();

    return 0;
}
```

OUTPUT :

```
Enter String            :   computer

Enter String            :   science

---------------------------------------------

First String is         :  computer

Second String is        :  science

---------------------------------------------

Concatenated String is    :  computerscience
```

b. Overload = =operator to carry out the comparison of strings.

Program :

```
#include<iostream>

#include<stdio.h>

#include<string.h>

using namespace std;

class String

{
    char str[20];
    public:

      void getdata()

         {
              gets(str);
          }
```

```
        int operator ==(String s)
            {
                if(!strcmp(str,s.str))
                return 1;

                return 0;
             }
};
int main()
{
        String s1,s2;
        cout<<"Enter first string :: ";

        s1.getdata();

        cout<<"\nEnter second string :: ";

        s2.getdata();

        if(s1==s2)

            {
                cout<<"\nStrigs are Equal\n";
             }
        else

             {

                cout<<"\nStrings are Not Equal\n";

             }

          return 0;
}
```

OUTPUT :

```
Enter first string :: hello

Enter second string :: hello

Strigs are Equal

Enter first string :: hello

Enter second string :: hai

Strigs are not Equal
```

10. Write a C++ Program to demonstrate implementation of inheritance.

Program :

```
 #include <bits/stdc++.h>
using namespace std;
                                                                    // Base class
class Parent
{
     public:
     int id_p;
};
                                          // Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
     public:
     int id_c;
};
 // main function
int main()
{
            Child obj1;
            // An object of class child has all data members and member functions of class parent
            obj1.id_p = 91;
            obj1.id_c = 17;
            cout << "Parent id is: " << obj1.id_p << '\n';
            cout << "Child id is: " << obj1.id_c << '\n';
             return 0;
}
```

OUTPUT :

```
Parent id is: 91
Child id is: 17
```