B.Sc –COMPUTER SCIENCE

II YEAR III SEMESTER

OBJECT ORIENTED PROGRAMMING WITH C++

Prepared by

Dr.K.Shanmugavadivu

SEMESTER III

CORE III - OBJECT ORIENTED PROGRAMMING WITH C++

UNIT - I

Object-Oriented Programming: Principles - Benefits of OOP - Application of OOP - Tokens, Expression and Control Structures: Tokens - Keywords - Identifiers and Constants - Data types - Constants - Variables - Operators - Manipulators - Expressions - Control Structure.

UNIT - II

Functions: Prototyping - Call by Reference - Return by Reference - Inline Functions - Default Arguments - const Arguments - Function Overloading - Friend and Virtual Functions, Classes and Objects - Class - Member Functions - Arrays with in a Class - Memory Allocation for Objects - Static data members - Static member functions - Arrays of Objects - Objects as Function Arguments - Friendly Functions - Returning Objects - const Member Functions - Pointers to Members, Constructors and Destructors.

UNIT - III

Operator Overloading and Type Conversions - Inheritance: Extending Classes - Derived Classes - Single Inheritance - Multilevel Inheritance - Multiple Inheritance - Hierarchical Inheritance - Hybrid Inheritance - Virtual Base Classes - Abstract Classes, Pointers, Virtual Functions and Polymorphism: Pointers - Pointers to Objects -this Pointer - Pointers to Derived Classes - Virtual Functions - Pure Virtual Functions

UNIT - IV

Managing I/O Operations: C++ Streams - C++ Stream Classes - Unformatted I/O and Formatted I/O Operations - Managing Output with Manipulators. Working with Files: Classes for File Stream Operations - Opening and Closing a File - Detecting end-of-file - File Pointers and Their Manipulators - Sequential I/O Operations - Updating a File - Error Handling during File Operations - Command Line Arguments

UNIT - V

Templates: Class Templates - Class Templates with Multiple Parameters - Function Templates - Function Templates with Multiple Parameters - Overloading of Template Functions - Member Function Templates - Non-Type Template Arguments. Exception Handling: Basics - Exception Handling Mechanism - Throwing Mechanism - Catching Mechanism - Rethrowing an Exception - Specifying Exceptions

TEXT BOOK

1. E.Balagurusamy, "Object Oriented Programming with C++", 5th Edition, Tate McGraw Hill Publications, 2011.
2.

REFERENCE BOOKS

1. M. T. Somashekara, "Object Oriented programming with C++", 2nd Edition, Prentice Hall of India Learning Limited, 2012.
2. Behrouz A.Forouzan, "A Structured Approach Using C++", 2ndEdition, Cengage Learning, 2003.

UNIT -1

Overview of C language:

1.C language is known as structure oriented language or procedure oriented language
2.Employs top-down programming approach where a problem is viewed as a sequence of tasks to be performed.
3.All program code of c can be executed in C++ but converse many not be possible
4. Function overloading and operator overloading are not possible.
5. Local variables can be declared only at the beginning of the block.
6. Program controls are through jumps and calls to subroutines.
7.Polymorphism, encapsulation and inheritance are not possible.
For solving the problems, the problem is divided into a number of modules. Each module is a subprogram.
8. Data abstraction property is not supported by procedure oriented language.
9. Data in procedure oriented language is open and can be accessed by any function.

Overview of C++ language:

1. C++ can be considered as an incremental version of c language which consists all programming language constructs with newly added features of object oriented programming.
2. c++ is structure(procedure) oriented and object oriented programming language.
3. The file extension of C++ program is ".CPP"
4. Function overloading and operator overloading are possible.
5. Variables can be declared in inline i.e when required 6.In c++ more emphasis is give on data rather than procedures
7. Polymorphism, encapsulation and inheritance are possible.
8. Data abstraction property is supported by c++.
9. Data access is limited. It can be accessed by providing various visibility modes both for data and member functions. there by providing data security by data hiding
10.Dymanic binding is supported by C++
11..It supports all features of c language
12.It can be called as an incremental version of c language

Difference Between Procedure Oriented Programming (POP) & Object Oriented Programming (OOP)

	Procedure Oriented Programming	**Object Oriented Programming**
1	program is divided into small parts called **functions**.	program is divided into parts called **objects**.

2	Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	Importance is given to the data rather than procedures or functions because it works as a **real world**.
3	follows **Top Down approach**.	OOP follows **Bottom Up approach**.
4	It does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
5	Data can move freely from function to function in the system.	objects can move and communicate with each other through member functions.

6	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
7	Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
8	It does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
9	Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
10	Example of Procedure Oriented Programming are : C, VB, FORTRAN, Pascal.	Example of Object Oriented Programming are : C++, JAVA, VB.NET, C#.NET.

Principles(or features) of object oriented programming:

1. Encapsulation
2. Data abstraction
3. Polymorphism
4. Inheritance
5. Dynamic binding
6. Message passing

Encapsulation: Wrapping of data and functions together as a single unit is known as encapsulation. Bydefault data is not accessible to outside world and they are only accessible through the functions which are wrapped in a class. prevention of data direct access by the program is called data hiding or information hiding

Data abstraction :

Abstraction refers to the act of representing essential features without including the

back ground details or explanation. Classes use the concept of abstraction and are defined as a list of attributes such as size, weight, cost and functions to operate on these attributes. They encapsulate all essential properties of the object that are to be created.

The attributes are called as data members as they hold data and the functions which operate on these data are called as member functions.

Class use the concept of data abstraction so they are called **abstract data type** (ADT)

Polymorphism: Polymorphism comes from the Greek words "poly" and "morphism". "poly" meansmany and "morphism" means form i.e.. many forms. Polymorphism means the ability to take more than one form. For example, an operation have different behavior in different instances. The behavior depends
upon the type of the data used in the operation.

Different ways to achieving polymorphism in C++ program:

1)Function overloading 2) Operator overloading

```
#include<iostream>usin
g namespace
std;  int  main()
{int          a=4;
a=a<<2;

      cout<<"a="<<a<<endl;
return 0;

}
```

Inheritance: Inheritance is the process by which one object can acquire the properties of another.

Inheritance is the most promising concept of OOP, which helps realize the goal of constructing software from reusable parts, rather than hand coding every system from scratch. Inheritance not only supports reuse across systems, but also directly facilitates extensibility within a system. Inheritance coupled with polymorphism and dynamic binding minimizes the amount of existing code to be modified while enhancing a system.

When the class child, inherits the class parent, the class child is referred to as derived class (sub class) and the class parent as a base class (super class). In this case, the class child has two parts: a derived part and an incremental part. The derived part is inherited from the class parent. The incremental part is the new code written specifically for the class child.

Dynamic binding:

Binding refers to linking of procedure call to the code to be executed in response to the call. Dynamic binding(or late binding) means the code associated with a given procedure call in not known until the time of call at run time.

Message passing:

An object oriented program consists of set of object that communicate with each other.

Objects communicates with each other by sending and receiving information .

A message for an object is a request for execution of a procedure and there fore invoke the function that is called for an object and generates result

Benefits of object oriented programming (OOPs)

- **Reusability:** In OOP"s programs functions and modules that are written by a user can be reused byother users without any modification.
- **Inheritance**: Through this we can eliminate redundant code and extend the use of existing classes.

- **Data Hiding:** The programmer can hide the data and functions in a class from other classes. Ithelps the programmer to build the secure programs.

Reduced complexity of a problem: The given problem can be viewed as a collection of differentobjects. Each object is responsible for a specific task. The problem is solved by interfacing the objects. This technique reduces the complexity of the program design.

Easy to Maintain and Upgrade: OOP makes it easy to maintain and modify existing code as newobjects can be created with small differences to existing ones. Software complexity can be easily managed.

- **Message Passing:** The technique of message communication between objects makes the interfacewith external systems easier.

  Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.

BASIC STRUCTURE OF C++ LANGUAGE : The program written in C++ language follows this basicstructure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. Documentation section
2. Linking section
3. Definition section
4. Global declaration section & class declarations
5. Member function definition
6. Main function

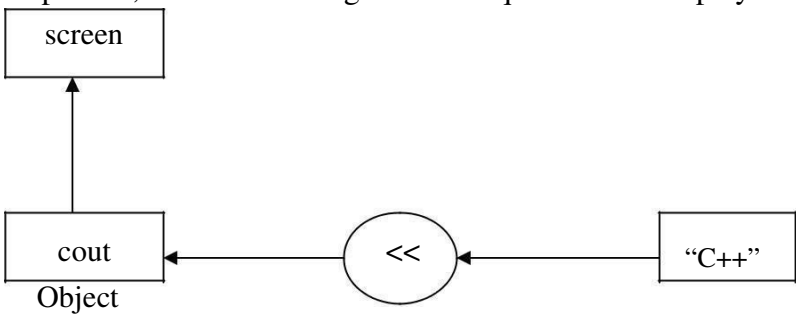
```
section main()
{
 Declaration section
Executable section
}
```

1. **DOCUMENTATION SECTION :** comes first and is used to document the use of logic orreasons in your program. It can be used to write the program's objective, developer and logic details. Thedocumentation is done in C language with /* and */ . Whatever is written between these two are called comments.

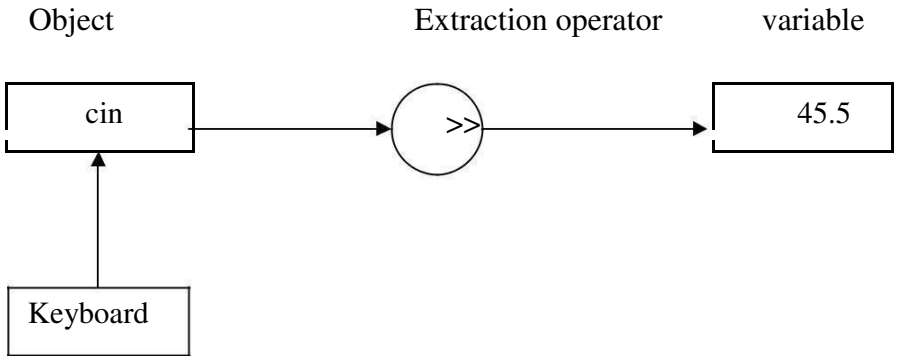
2. **LINKING SECTION :** This section tells the compiler to link the certain occurrences ofkeywords or functions in your program to the header files specified in this section.e.g. #include<iostream>
using namespace std;

- directive causes the preprocessor to add the contents of the iostream file to the program. It contains declarations for cout and cin.

cout is a predefined object that represents the standard output stream. The operator << is an insertion operator, causes the string in double quotes to be displayed on the screen.



The statement cin>>n; is an input statement and causes the program to wait for the user to type in a number. The number keyed is placed on the variable "n". The identifier cin is a predefined object in C++ that corresponds to the standard input stream. The operator >> is known as extraction operator. It extracts the value from the keyboard and assigns it to the value variable on its right.



3. **DEFINITION SECTION :** It is used to declare some constants and assign them some value. e.g. #define MAX 25

Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

4. GLOBAL DECLARATION SECTION : Here the variables and class definations which are usedthrough out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program). **A CLASS** is a collection of data and functions that act or manipulate the data. The data components of a class are called **data members** and function components of a class are called **member functions**

A class ca also termed as a blue print or prototype that defines the variable or functions common to all objects of certain kind. It is a **user defined data type**

e.g. int i; //this declaration is done outside and before main()

SUB PROGRAM OR FUNCTION SECTION : This has all the sub programs or the functions whichour program needs.

```
void display()
{ cout<<"C++ is better that
C";
}
```

SIMPLE „C++" PROGRAM:

```
#include<iostream>
using namespace std;
void display()
{ cout<<"C++ is better that C";
} int
main()
{ display()
        return
        0;
}
```

6.MAIN FUNCTION SECTION : It tells the compiler where to start the execution from main()

{ point from execution starts

} main function has two sections

1. declaration section : In this the variables and their data types are declared.
2. Executable section or instruction section : This has the part of program which actually performs the task we need.

namespace:

namespace is used to define a scope that could hold global identifiers.

ex:-namespace scope for c++ standard library.
A classes ,functions and templates are declared within the namespace named std using namespace std;-->directive can be used.

user defined name space:

syntax for defining name space is

```
namespace namespace_name
{
//declarations of variables.functions,classes etc...
}
```

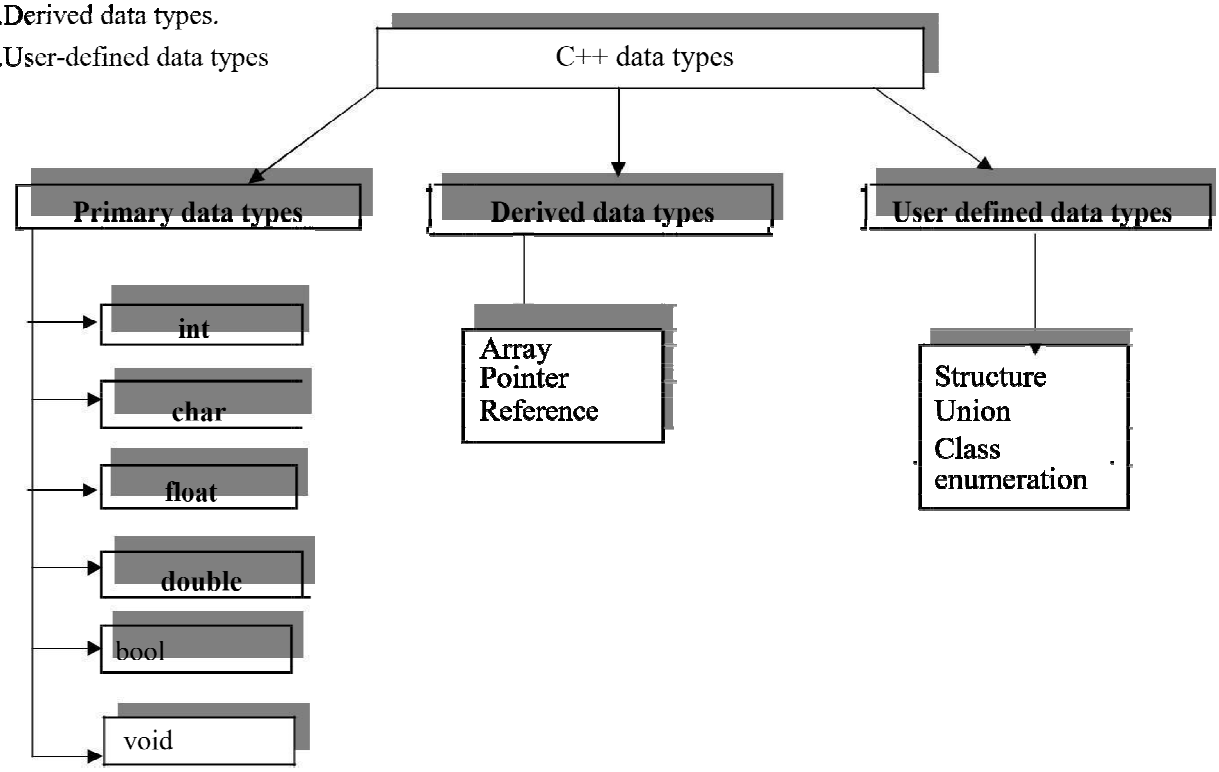
```
#include<iostream>
using namespace std;
namespace sample
{` int m; void
        display(int n)

                { cout<<"in namespace
                N="<<n<<endl;
        }
}

using namespace sample;
int main()
{
int a=5;
m=100;
display(200);
cout<<"M in sample name space:"<<sample::m;
return 0;}
```

#include<iostream>
This directive causes the preprocessor to add content of iostream file to the program.
some old versions of C++ used iostream.h .if complier does not support ANSI (american nation standard institute) C++ then use header file iostream.h

DATA TYPES:

A data type is used to indicate the type of data value stored in a variable. All C compilers support a variety of data types. This variety of data types allows the programmer to select the type appropriate to the needs of the application as well as the machine.
ANSI C supports the following classes of data types:

1.Primary (fundamental) data types.
2.Derived data types.
3.User-defined data types



Primary data types:

1.integer data type
2.character data type
3.float point data type
4.Boolean data type
5.void data type

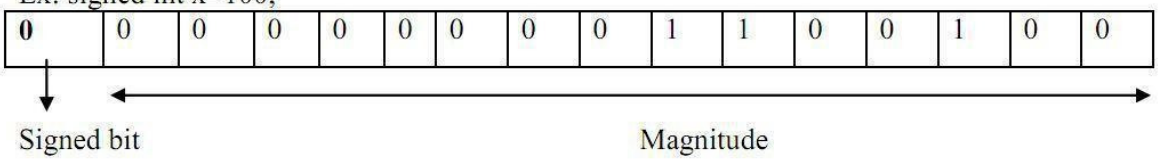
integer data type:-

This data type is used to store whole numbers. These numbers do not contain the decimal part. The size of the integer depends upon the world length of a machine (16-bit or 32-bit). On a 16-bit machine, the range of integer values is - 32,768 to +32,767.integer variables are declared by keyword int. C provides control over range of integer values and storage space occupied by these values through the data types: short int, int, long int in both signed and unsigned forms.

Signed integers: (16-bit machine):

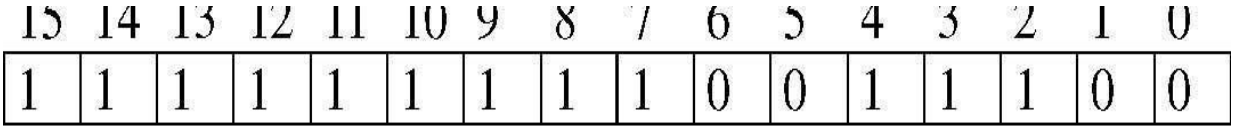
A signed integer uses 1 bit for sign and 15 bits for the magnitude of the number

A signed integer uses 1 bit for sign and 15 bits for the magnitude of the number.
(-2^{15} to $+2^{15}-1$).
Ex: signed int x=100;



MSB(most significant bit) = 100(10) 0000000001100100(2)

Representation of negative number : -100(10)=1111111110011100(2)



1

5 14

13 12

11 10 9 8 7 -1*2 +1*2 +1*2 +1*2 +1*2 +1*2 +1*2 +1*2 +1*2 +

6 5 4 3 2 1 0

0*2 +0*2 +1*2 +1*2 +1*2 +0*2 +0*2

=-32768+16384+8192+4096+2048+1024+512+256+128+0+0+26+8+4+0+0 =-100(10)

NOTE: **Signed bit (MSB BIT)**: 0 represents positive integer, 1 represents negative numbers

Unsigned integers: Unsigned integers use all 16 bits to store the magnitude. Stores numbers does not have anysign & Size qualifier and range of integer data type on a 16-bit and machine are shown in the table:

DATA TYPE	MEMORY REQUIRED OR STORAGE SIZE IN BYTES		RANGE		FORMAT SPECIER
	TURBO C (16 BIT)	GCC/ COMPILERS IN LINUX (32 BIT)	TURBO C (16 BIT)	GCC (32 BIT)	
short int or signed short int	2	2	-32768 To 153276715 (-2 to +2 -1)	-32768 To 32767 15 15 (-2 to +2 -1)	%hd
short int or signed short int	2	2	0 to 65535 (0 to +2 -1)	0 to 65535 (0 to +2 -1)	%hu
signed int			-32768	-2,147,843,648	%d

or int	2	4	To 153276715 (-2 to +2 -1)	to 2,147,843,647 31 31 (-2 to +2-1)	or %i
unsig ned int	2	4	0 to 65535 16 (0 to +2 -1)	0 to 4,294,967,295 32 (0 to2-1)	%u
long int or signed long int	4	4	-2,147,843,648 to 2,147,843,647 31 31 (-2 to +2-1)	-2,147,843,648 to 2,147,843,647 31 31 (-2 to +2-1)	%ld
unsigned long int	4	4	0 to 4,294,967,295 32 (0 to2 -1)	0 to 4,294,967,295 32 (0 to2-1)	%lu
long long int or signed long long int	Not supported	8	--------	- 9223372036854775808 To 9223372036854775807 63 63 (-2 to +2-1)	%Ld

***Character data type*:(char)**

A single character can be defined as a character data type. Character data type occupies one byte of memory for storage of character. The qualifiers **signed** or **unsigned** can be applied on char data type. **char** is the key word used for declaring variables size and range of character data type on 16 bit or 32 bit machine can be shown below

Data type	MEMORY REQUIRED OR STORAGE SIZE (in bytes)	RANGE	FORMAT SPECIER
char or signed char	1	128 to 127(-2 7 to 2 7 -1)	%c
Unsigned signed char	1	0 to 256 (0 to 2 -1)	%c

Floating Point Types:

Floating point number represents a real number with 6 digits precision occupies 4 bytes of memory. Floating point variables are declared by the keyword **float.**

Double floating point data type occupies 8 bytes of memory giving 14 digits of precision. These are also known as double precision numbers. Variables are declared by keyword **double long double** refers to a floating point data type that is often more precise than double precision.

Size and range of floating point data type is shown in the table:

Data type (key word)	Size (memory)	Range	format specifier
Float	32 bits (4 bytes)	3.4E-38 to 3.4E+38	%f
Double	64 bits (8 bytes)	1.7E-308 to 1.7E +308	%lf
long double	80 bits (10 bytes)	3.4E-4932 to 1.1E+4932	%Lf

Boolean data type:-

Boolean or logical data type is a data type, having two values (usually denoted true and false), intended to represent the truth values of logic and Boolean algebra. It is named after George Boole, who first defined an algebraic system of logic in the mid 19th century. The Boolean data type is the primary result of conditional statements, which allow different actions and change control flow depending on whether a programmer -specified Boolean condition evaluates to true or false.
C99 added a Boolean (true/false) type which is defined in the <stdbool.h> header Boolean variable is defined by kkey word **bool;** Ex:

bool b;
where b is a variable which can store true(1) of false (0)

Void type

The void type has no values. This is usually used to specify the return type of functions. The type of the functionsaid to be void when it does not return any value to the calling function. This is also used for declaring general purpose pointer called void pointer.

Derived data types.

Derived datatypes are Arrays , pointer and references are examples for derived data types. **User-defined data types:**

they The data types defined by the user are known as the user-defined data types. They are structure,union,class and enumeration

C++ Tokens

IDENTIFIERS: Identifiers are the names given to various program elements such as variables, functions and arrays. These are user defined names consisting of sequence of letters and digits.

Rules for declaring identifiers:

- The first character must be an alphabet or underscore.
- It must consist of only letters, digits and underscore.
- Identifiers may have any length but only first 31 characters are significant. It must not contain white space or blank space.
- We should not use keywords as identifiers. Upper and lower case letters are different.

Example: ab Ab aB AB are treated differently **Examples of valid identifiers:** a, x, n, num, SUM, fact, grand_total, sum_of_digits, sum1

Examples of *Invalid identifiers*:$amount,³num´, grand-total, sum of digits, 4num.

$amount : Special character is not permitted grand-total :
hyphen is not permitted. sum of digits : blank spaces
between the words are not allowed.
4num : should not start with a number (first character must be a letter or underscore

Note: Some compilers of C recognize only the first 8 characters only; because of this they are unableto distinguish identifiers with the words of length more than eight characters.

Variables:A named memory location is called variable.

OR

It is an identifier used to store the value of particular data type in the memory.

Since variable name is identifier we use following rules which are same as of identifier

Rules for declaring Variables names:

- The first character must be an alphabet or underscore.
- It must consist of only letters, digits and underscore.
- Identifiers may have any length but only first 31 characters are significant.
- It must not contain white space or blank space.
- We should not use keywords as identifiers.

Upper and lower case letters are different. Variable names must be unique in the given scope

Ex:int a,b,a;//is in valid
Int a,b;//is valid

Variable declaration: The declaration of variable gives the name for memory location and its size andspecifies the range of value that can be stored in that location.

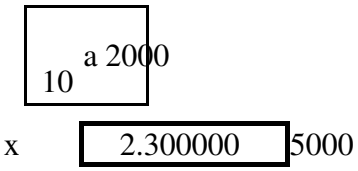
Syntax:

Data type variable name;

Ex:

int a=10;

float x=2.3;



KEYWORDS :

There are certain words, called keywords (reserved words) that have a predefined meaning in „C++“ language. These keywords are only to be used for their intended purpose and not as identifiers. The following table shows the standard „C++“ keywords

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while	class	friend	new	delete
this	public	private	protected	inline	try
throw	catch	template			

CONSTANTS:

Constants refer to values that do not change during the execution of a program.

Constants can be divided into two major categories:

1.*Primary constants*:

a)Numeric constants

- Integer constants.
- Floating-point (real)

constants. b)Character constants

- Single character constants String

constants2.*Secondary constants*:

- Enumeration constants.
- Symbolic constants.
- Arrays, unions, etc.

Rules for declaring constants:

1.Commas and blank spaces are not permitted within the constant.

2.The constant can be preceded by minus (-) signed if required.

3.The value of a constant must be within its minimum bounds of its specified data type.

Integer constants: An integer constant is an integer-valued number. It consists of sequence ofdigits. Integer constants can be written in three different number systems:

1.Decimal integer (base 10).

2.Octal integer (base 8).

3.Hexadecimal (base 16).

***Decimal integer constant*:**It consists of set of digits, 0 to 9.

Valid declaration: 0, 124, -56, + 67, 4567 etc.

Invalid declaration: $245, 2.34, 34 345, 075. 23,345,00. it is also an invalid declaration.

Note: *Embedded spaces, commas, characters, special symbols are not allowed between digits*

They can be preceded by an optional + or ± sign.

Octal integer: It consists of set of digits, 0 to 7.

Ex: 037, 0, 0765, 05557 etc. (valid representation) It is a sequence of digits preceded by 0.

Ex: Invalid representations

0394: digit 9 is not permitted (digits 0 to 7 only)

235: does not begin with 0. (Leading number must be 0).

Hexadecimal integer: It consists of set of digits, 0 to 9 and alphabets A, B, C, D, E, andF. Hexadecimal integer is a sequence of digits preceded by 0x or 0X. We can also use a through f instead of A to F.

Ex: 0X2, 0x9F, 0Xbcd, 0x0, 0x1. (Valid representations)

Ex: Invalid representations: 0af, 0xb3g, 0Xgh.

0af: does not begin with 0x or 0X.

0xb3g, 0Xgh: illegal characters like g, h. (only a to f are allowed)

The magnitude (maximum value) of an integer constant can range from zero to some maximum value that varies from one computer to another.
Typical maximum values for most personal computers are: (16-bit machines)
Decimal integer constant: 32767 (215-1)
Octal integer constant: 077777
Hexadecimal integer constant: 0X7FFF
Note: The largest value that can be stored is machine dependent.

Floating point constants or Real constants : The numbers with fractional parts are called real constants. These are the numbers with base-10 which contains either a decimal part or exponent (or both). **Representation:** These numbers can be represented in either decimal notation or exponentnotation (scientific notation).

Decimal notation: 1234.56, 75.098, 0.0002, -0.00674 (valid notations)

Exponent or scientific notation:

General form: Mantissa e exponent

Mantissa: It is a real number expressed in decimal notation or an integer notation.
Exponent: It is an integer number with an optional plus (+) or minus (-) sign.
E or e: The letter separating the mantissa and decimal part.
Ex: (Valid notations)

$1.23456\mathbf{E+}3$ (1.23456×10^{3})

7.5098 **e+**1 (7.5098×10^{1})

$2\mathbf{E-}4$ (2×10^{-4})

These exponential notations are useful for representing numbers that are either very large or very small. Ex: 0.00000000987 is equivalent to**9.87e-9**

Character constants:-

***Single character constants*:**It is character(or any symbol or digit) enclosed within single quotes.

Ex: „a‟ „1‟ „*‟

Every Character constants have integer values known as **ASCII** values

ASCII:- ASCII stands for American Standard Code for Information Interchange. Pronounced ask-ee, ASCII is acode for representing English characters as numbers, with each letter assigned a number from 0 to 255.Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort.A SCII codes represent text in computers, communications equipment, and other devices that use text. Most modern character-encoding schemes are based on ASCII, though they support many additional characters.
Below is the ASCII character table and this includes descriptions of the first 32 non - printing characters. **String constants or string literal:**

String constant is a *sequence of zero or more characters* enclosed by double quotes.

Example:

“MRCET” “12345” “*)(&%”

Escape Sequences or Backslash Character Constants

C language supports some nonprintable characters, as well as backslash (\ ) which can be expressed as escape sequences. An escape sequence always starts with backslash followed by one or more
special characters.
For example, a new line character is represented "\n" or endl

Escape sequence	Character
'\a'	audible alert
'\b'	back space
'\f'	form feed
'\n'	new line
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	Backslash
'\o'	Null

OPERATORS AND EXPRESSIONS

An ***operator***is a symbol which represents a particular operation that can be performed on data. An ***operand*** is the object on which an operation is performed.

By combining the operators and operands we form an *expression*. An ***expression*** is a sequence of operands and operators that reduces to a single value.

C operators can be classified as

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bit wise operators
8. unary operator
9. Special operators
10. Additional operators in c++

1.ARITHMETIC OPERATORS : All basic arithmetic operators are present in C.

Operator	meaning
+	add
-	subtract
*	multiplication
/	division
%	modulo division(remainder)

An arithmetic operation involving only real operands(or integer operands) is called real arithmetic(or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

/*C program on Integer Arithmetic Expressions*/

```
#include<iostraem.h>
void main()
{ int
a, b;
cout<"Enter any two
integers"; cin>>a>>b;
cout<<"a+b"<< a+b;
cout<<"a-b"<< a-b;
cout<<"a*b"<< a*b;
cout<<"a/b"<< a/b;
cout<<"a%b"<< a%b;
}
```

OUTPUT:

```
a+b=23 a-
b=17
a*b=60
a/b=6
a% b=2
```

2.RELATIONAL OPERATORS : We often compare two quantities and depending on their relationtake certain decisions for that comparison we use relational operators.

operator	meaning
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to is
!=	not equal to

/* C program on relational operators*/

```
#include<iostream.h>
void main()
{ int
a,b;
clrscr(
);
cout<<"Enter a, b
values:"; cin>>a>>b;
cout<<"a>b"<< a>b;
cout<<"a>=b"<< a>=b;
cout<<"a<b"<< a<b;
cout<<"a<=b"<< a<=b;
cout<<"a==b"<< a==b;
cout<<"a!=b"<<
a!=b; }
```

OUTPUT:

Enter a, b values:
5 9 a>b: 0 //false
a<b: 1 //true
a>=a: 1 //true

a<=b: 1 //true
a==b: 0 //false
a!=b: 1 //true

3.LOGICAL OPERATORS:

Logical Data: A piece of data is called logical if it conveys the idea of true or false. In C++ we use int data type to represent logical data. If the data value is zero, it is considered as false. If it is non -zero (1 or any integer other than 0) it is considered as true. C++ has three logical operators for combining logical values and creating new logical values:

Truth tables for AND (&&) and OR (||) operators:

X	Y	X&&Y	X||Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Truth table for NOT (!) operator:

X	!X
0	1
1	0

Note:Below program works in compiler that support C99 standards

```
#include<iostream.h>
#include<stdbool.h>
int main()
{ bool
 a,b;
        /*logical and*/
         a=0;b=0;
        cout<<" a&&b "<<
        a&&b<<endl; a=0;b=1;
        cout<<" a&&b "<<
        a&&b<<endl; a=1;b=0;
        cout<<" a&&b "<<
        a&&b<<endl; a=1;b=1;
        cout<<" a&&b "<<
        a&&b<<endl;
         /*logical or*/
         a=0;b=0;
        cout<<" a||b "<< a||b<<endl;
         a=0;b=1;
        cout<<" a||b "<< a||b<<endl;
         a=1;b=0;
        cout<<" a||b "<< a||b<<endl;
         a=1;b=1;
        cout<<" a||b "<< a||b<<endl;
         /*logical not*/
         a=0;
        cout<<" a||b "<< a||b<<endl;
          a=1;
        cout<<" a||b "<< a||b<<endl;
          return 0;
}
```

OUTPUT:

```
0&&0=0
0&&1=0
```

1&&0=0
1&&1=1
0||0=0
0||1=1

1||0=1
1||1=1
!0 =1
!1 =0

4.ASSIGNMENT OPERATOR:

The assignment expression evaluates the operand on the right side of the operator (=) and places its value in the variable on the left.

Note: The left operand in an assignment expression must be a single variable.

There are two forms of assignment:

•Simple assignment

•Compound assignment

Simple assignment :

In algebraic expressions we found these expressions.

Ex: a=5; a=a+1; a=b+1;

Here, the left side operand must be a variable but not a constant. The left side variable must be able to receive a value of the expression. If the left operand cannot receive a value and we assign one to it, we get a compile error.

Compound Assignment:

A compound assignment is a shorthand notation for a simple assignment. It requires that the left operand be repeated as a part of the right expression. Syntax: variable operator+=value

Ex:

A+=1; it is equivalent to A=A+1;

Advantages of using shorthand assignment operator:

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read.
3. The statement is more efficient.

Some of the commonly used shorthand assignment operators are shown in the following table:

Statement with simple assignment operator	Statement with shorthand operator
a=a+1	a+=1
a=a-1	a-=1
a=a*1	a*=1
a=a/1	a/=1
a=a%1	a%=1
a=a*(n+1)	a*=n+1

5.INCREMENT (++) AND DECREMENT (--) OPERATORS:

The operator ++ adds one to its operand where as the operator - - subtracts one from its operand. These operators are unary operators and take the following form:

Operator	Description
++a	Pre-increment
a++	Post-increment
--a	Pre-decrement
a--	Post-decrement

Both the increment and decrement operators may either precede or follow the operand.

Postfix Increment/Decrement :(a++/a--)

In postfix increment (Decrement) the value is incremented (decremented) by one. Thus the a++ has the same effect as

a=a+1; a--has the same effect as a=a-1.

The difference between a++ and a+1 is, if ++ is after the operand, the increment takes place after the expression is evaluated.

The operand in a postfix expression must be a variable.

Ex1:

```
int
a=5;
```

B=a++; Here the value of B is 5. the value of a is 6.

Ex2:

int x=4; y=x--; Here the value of y is 4, x value is 3

Prefix Increment/Decrement (++a/ --a)

In prefix increment (decrement) the effect takes place before the expression that contains the operator is evaluated. It is the reverse of the postfix operation. ++a has the same effect as a=a+1.

--a has the same effect as a=a-1.

Ex: int b=4;

A= ++b;

In this case the value of b would be 5 and A would be 5.

The effects of both postfix and prefix increment is the same: the variable is incremented by 1. But they behave differently when they used in expressions as shown above. The execution of these operators is fast when compared to the equivalent assignment statement.

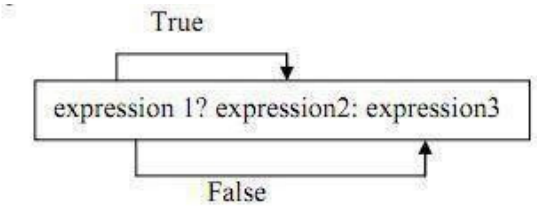
```
#include<iostream.h>
        int main()
{ int
  a=1;
```

```
 int b=5;
++a;
cout<<"a="<<a<<endl;
--b;
cout<<"b="<<b<<endl;
cout<<"a="<<a++<<endl;
cout<<"a="<<a<<endl; cout<<"b="<<b--
<<endl;
cout<<"b="<<b<<end
l; return 0; } a=2 b=4
a=2 a=3 b=4 b=3
```

6.CONDITIONAL OPERATOR OR TERNARY OPERATOR:

A ternary operator requires two operands to operate

Syntax:




```
#include<iostream.h>          void main()
{ int a, b,c; cout<<"Enter a and b values:";
        cin>>a>>b;



         c=a>b?a:b;
        cout<<"largest of a and b is "<<c;
}
Enter a and b values:1 5
largest of a and b is 5
```

7.BIT WISE OPERATORS : C supports special operators known as bit wise operators for manipulationof data at bit level. They are not applied to float or double.

operator	meaning
&	Bitwise AND
^	Bitwise exclusive
<<	OR
>>	left shift
~	right shift
	one's complement

Bitwise AND operator (&)

The bitwise AND operator is a binary operator it requires two integral operands (character or integer). It does a bitwise comparison as shown below:

First Operand Bit	Second Operand Bit	Ope1 & Ope2
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR operator (|)

The bitwise OR operator is a binary operator it requires two integral operands (character or integer). It does a bitwise comparison as shown below:

First Operand Bit	Second Operand Bit	Ope1 | Ope2
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise EXCLUSIVE OR operator (^)

The bitwise EXCLUSIVE OR operator is a binary operator it requires two integral operands (character or integer). It does a bitwise comparison as shown below:

First Operand Bit	Second Operand Bit	Ope1 ^ Ope2
0	0	0
0	1	1
1	0	1
1	1	0

Shift Operators

The shift operators move bits to the right or the left. These are of two types:

. •Bitwise shift right operator

. •Bitwise shift left operator

Bitwise shift right operator

It is a binary operator it requires two integral operands. The first operand is the value to be shifted and the second operand specifies the number of bits to be shifted. When bits are shifted to right,the bits at the right most end are deleted and a zero is inserted at the MSB bit.

```
#include<iostream.h>
void main()
{ int
x,shift;
        cout<<"Enter a number:");
        cin>>x;
        cout<<"enter now many times to right shift: ";
        cin>>shift;
        cout<<"Before Right
Shift:"<<x; x=x>>shift; cout<<"After
right shift:"<<x; }
```

Run1:

Enter a number:8
enter now many times to right shift:1
Before Right Shift:8
After right shift:4

Explanation: The number entered through the keyboard as input is 8 and its corresponding binary number is 1000.

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

After execution of the program the input data x is to be shifted by 2 bits right side. The answer in binary form would be as follows:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

The right shift operator divides the given number by a power of 2. If we shift a binary number two places to the right, we are dividing the given number by 4 (2^2).

Bitwise shift left operator

It is a binary operator it requires two integral operands. The first operand is the value to be shifted and the second operand specifies the number of bits to be shifted left.

When bits are shifting left, the bits at the left most end are deleted.

Ex: int a=2;
a<<=3;

Shift left is the opposite of shift right operator. The left shift operator multiplies the given number by a power of 2. If we shift a binary number three places to the left, we are multiplying the given number by 8 (2^3).

One's complen

The bitwise NO
bit, forming the
and vice versa.
Ex: NOT

In C, t he bitwi
Truth table:

Note: One's co
We use this ope

8. SPECIAL OPERATORS

These operators which do not fit in any of the above classification are ,(comma), sizeof, Pointer operators(& and *) and member selection operators (. and ->). The comma operator is used to link related
expressions together.

The SIZEOF operator:

It returns the number of bytes occupied by the operand. The operand may be a variable, a constant (data value) or a data type qualifier.

Ex: int a, c, f, d;

c=sizeof(a);//here c=2,the sizeof operator returns the size of the variable a which is of int type
f=sizeof(long double); //f value is 10 which is the size of the long double qualifier type
d=sizeof(23.345); //d value is 4 which is the size of the float constant value

The sizeof operator is normally used to determine the length of arrays and structures. It is also used to allocate space dynamically to the variable s during execution of a program.

The Comma Operator (,)

The comma operator can be used to link the related express
of expressions is evaluated left to right and the value of the
of the combined expression.

Ex: a=(x=10, y=20, x+y);

First assigns the value 10 to x, then assigns 20 to y and fina

It has the lowest precedence among all the operators.

We use comma operator in loop statements and declaratic
of same type.

9.UNARY OPERATOR: operator which operates on single operand is called unary operator

Operator	Description
+	Unary plus
-	Unary minus
++	Increment
--	Decrement
&	Address
~	Ones complement
Sizeof	Size of operator
Type	Type casting

Operators in c++: All above operators of c language are also valid in c++.New operators introduced inc++ are

Sno	Operator	Symbol
1.	Scope resolution operator	::
2.	Pointer to a member declarator	::*
3.	Pointer to member operator	->*,->
4.	Pointer to member operator	.*
5.	new	Memory allocating operator
6.	delete	Memory release operator

7.	endl	Line feed operator
8.	setw	Field width operator
9.	insertion	<<
10.	Extraction	>>

1.Scope Resolution operator:

Scope:-Visibility or availability of a variable in a program is called as scope. There are two types of scope. i)Local scope ii)Global scope

Local scope: visibility of a variable is local to the function in which it is declared. **Global scope:** visibility of a variable to all functions of aprogram

Scope resolution operator in **"::" .**

This is used to access global variables if same variables are declared as local and global PROGRAM1.2:- #include<iostream.h> int a=5; void main()

```
{ int a=10; cout<<"Local
a="<<a<<endl;
        cout<<"Global a="<<::a<<endl;
}
```

Expected output:

Local a=10

Global a=5

Member Dereferencing operator: -

1.	Pointer to a member declarator	::*
2.	Pointer to member operator	->*,->
3.	Pointer to member operator	.*

Pointer to a member declarator ::*

This operator is used for declaring a pointer to the member of the class #include<iostream.h> class sample

```
{public: int x; };
int main()
{               sample s;       //object
        int sample ::*p;//pointer decleration
         s.*p=10;          //correct
         cout<<s.*p;
}
```

Output:10

2.Pointer to member operator ->*

```
#include<iostream.h>
class sample
        { public:
                        int x;
                void display()
                {
               cout<<"x="<<x<<endl;
```

```
        }
};int
main()
{
        sample s;       //object
        sample *ptr;

        int
        sample::*f=&sample::x;
        s.x=10; ptr=&s;
        cout<<ptr->*f;
         ptr->display();
}
```

3. Pointer to member operator

.*

```
#include<iostream.h>
class sample
        { public: int x;
        }; int
        main()
        { sample s;       //object
                int sample ::*p;//pointer decleration
                s.*p=10;          //correct
                cout<<s.*p;
        }
```

Manipulators:

Manipulators are the operators used to format the data that is to be displayed on screen.The most commonly used manipulators are endl and setw

endl:-it is used in output statement and inserts a line feed. It is similar to new line character (“\n”) ex:

……..

```
cout<<”a=2”<<endl;
cout<”name=sunil”<<endl;
```

…….

Output: a=2 name=sunil setw:- this manipulator allows a specified width for a field that is to be printed on screen

and by default the value printed is right justified.This function is available in header file iomanip.h

```
#include<iostream.h>
#include<iomanip.h>
using namespace std;
int main()
{
int s=123;
cout<<"s="<<setw(10)<<s ;
}
```

output

s= 123

Insertion (<<) and Extraction (>>) operators: the operators are use with output and input objects ex:

```
cout<<”Enter n”;
cin>>n
```

Control statements:-The flow of execution of statements in a program is called as control. Controlstatement is a statement which controls flow of execution of the program. Control statements are classified

into following categories.
1.Sequential control statements
2. conditional control statement
3.Unconditional control statements

(type) expression;
Or
type (expression);

1.Sequential control statements:- statements ensures that the statements) are executed in the same they appear in the program. i.e. By executes the statements in the program order.

Sequential control instructions(ororder in which default system in sequential

2.Conditional control statements
:Statements that are executed when a condition istrue. These statements are divided into three categories. they are

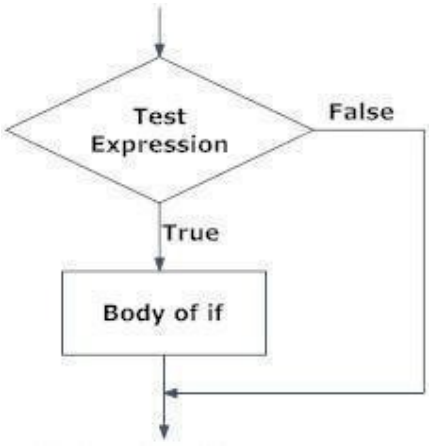
1.Decision making statements
2.Switch case control statement or
3.Loop control statements or repetations

1.Decision making statements:- These statements are used to control the flow of execution of a programby making a decision depending on a condition, hence they are named as decision making statements. Decision making statements are of four types

1.Simple if
2.if else
3.nested if else
4.If else ladder

1.Simple if statement: if the test expression is true then if statement executes statements that
immediately follow if

Syntax:
If(test expression)
{
List of statements;
}

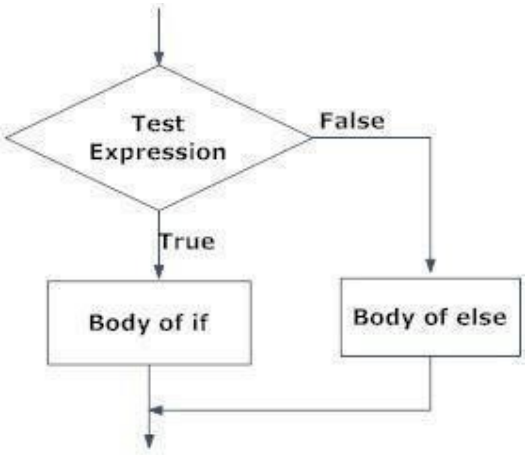


/*largest of two numbers*/
#include<stdio.h> int main()
{ int a,b; cout<<"Enter any two integers:"; cin>>a>>b; if(a>b) cout<<"A is larger than B\n A="<<a;
if(b>a) cout<<"B is larger than A\n
A="<<b; return 0;
}

2. if –else statement:

If test expression is true block of statements following if are executed and if test expression is false then statements in else block are executed if (test expression)

```
{ statement block1;
} else
{ statement block2;
}
```



/*largest of two numbers*/

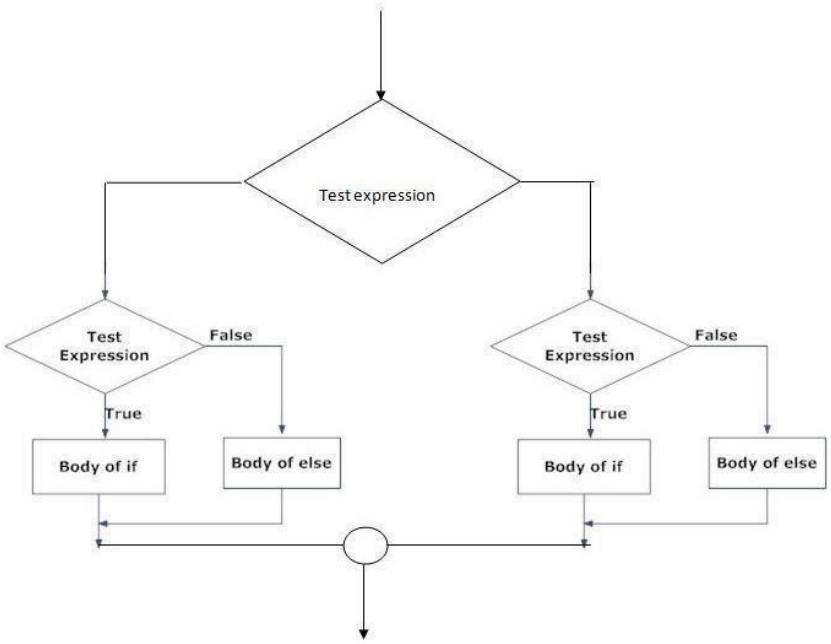
```
#include<iostream.h> int main()
{ int a,b;
cout<<"Enter any two integers:";
cin>>a>>b;

if(a>b) cout<<"A is larger than B\n A="<<a;
else  cout<<"B is larger than A\n A="<<b;

return 0;

}
```

3.Nesting of if-else **statements** It's also possible to nest one if statement inside another. When a series of decisionsare to be made.

If –else statement placed inside another if else statement

Syntax:

```
If(test expression) {If(test expression) {
        //statements
         } else
         { //statements
         }
 } else
{If(test expression) {
         //statements
         }
         else
         { //statements
         }
}
```



/*largest of three numbers*/

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

```
#include<conio.h
> int main()
{ int
a,b,c;
cout<<"Enter a,b,c values:";
cin>>a>>b>>c;
        if(a>b)
        { if(a>c)
                { cout<<"A ia largest among three
                        numbers\n"; cout"A= "<<a;
                }
                els
                e
                { cout<<"C ia largest among three
                        numbers\n"; cout<<"c= "<<c;
                }
        }
        else
        {if(b>c)
                { cout<<"B ia largest among three
                        numbers\n"; cout<<"B="<<b;



                }
                els
                e
                { cout<<"C ia largest among three
                        numbers\n"; cout<<"c="<<c;
                }
        }
getch();
return 0;
}
```

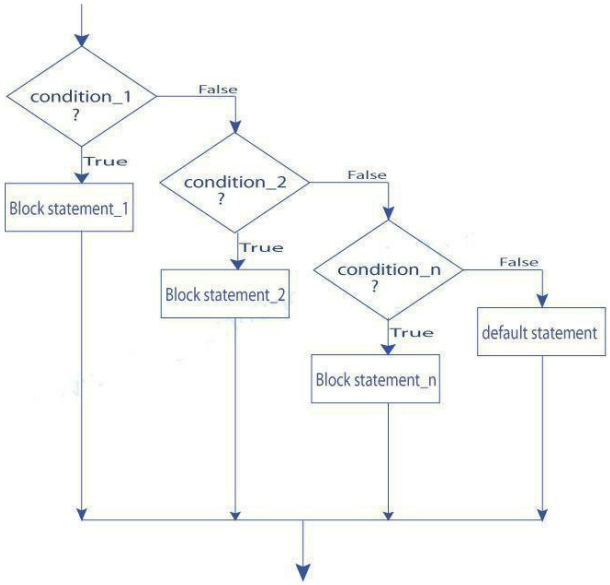
4.if else ladder

```
        if(condition1)
                statement1;
       else if(condition2)
                statement 2;
        else if(condition3)
                 statement n;
         else default statement.
         statement-x;
```

The nesting of if-else depends upon the conditions with which we have to deal.

The condition is evaluated from top to bottom.if a condition is true the statement associated with it is executed. When all the conditions become false then final else part containing default statements will be executed.

```
#include<iostream.h>
        void main()
{ int
per;

cout<<"Enter percentage"; cin>>per;
        if(per>=80) cout<<"Secured
Distinction"<<endl; else if(per>=60)
        cout<<"Secured First
Division"<<endl; else if(per>=50)
cout<<"Secured Second
Division"<<endl; else if(per>=40)
       cout<<"Secured                Third
Division"<<endl; else
cout<<"Fail"<<endl

}
```



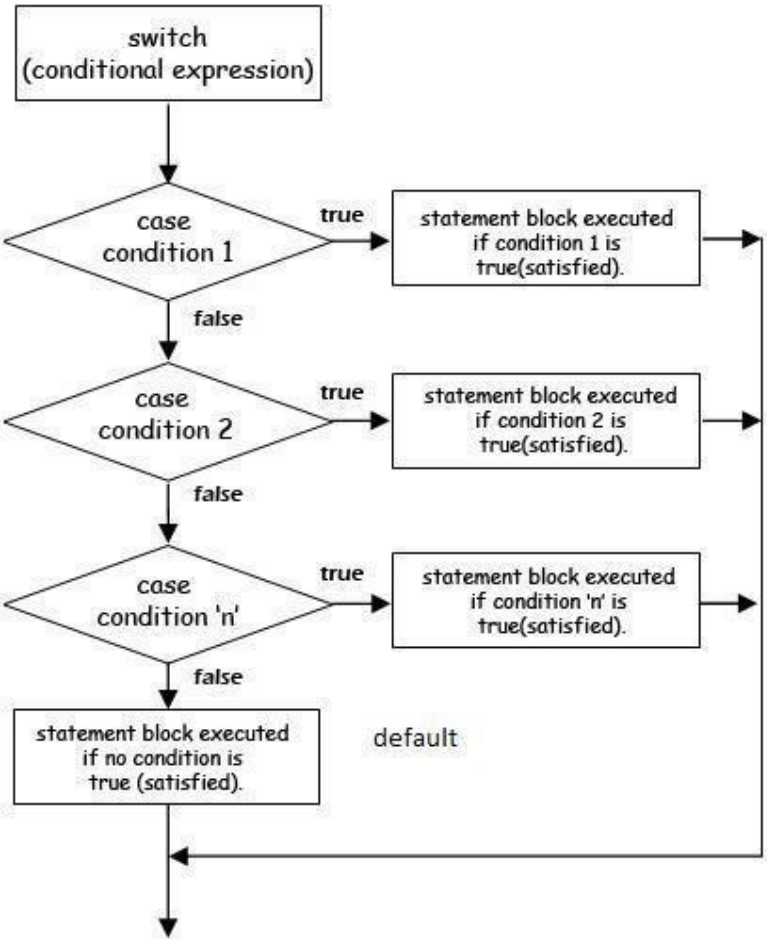
THE SWITCH STATEMENT or MULTIWAY SELECTION :

In addition to two-way selection, most programming languages provide another selection concept known as multiway selection. Multiway selection chooses among several alternatives. C has two different ways to implement multiway selection: the switch statement and else-if construct

If for suppose we have more than one valid choices to choose from then we can use switch statement in place of if statements.

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

}




```
/*program to simulate a simple calculator */
 #include<iostream.
h> int main() {
float a,b; char opr;

cout<<"Enter number1 operator number2
: ";       cin>>a>>oper>>b; switch(opr)
{
case '+':
          cout<<"Sum : "<<(a + b)<<endl;
           break;
case '-': cout<<"Difference : "<<(a -b)<<endl;
          break;
case '*': cout<<”Product : "<<a * b<<endl;
          break;
case '/': cout<<”Quotient :"<<(a / b)<<endl;
          break;
```

```
default: cout<<”Invalid Operation!"<<endl;
}
return
0;
}
```

Loop control statements or repetitions:

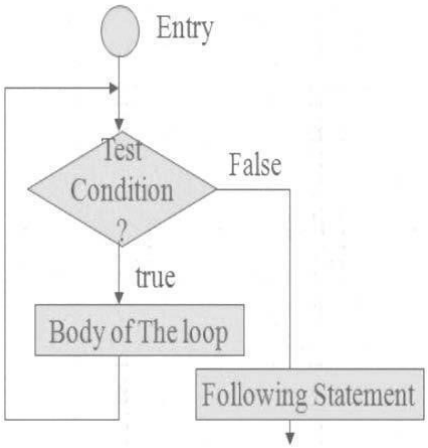
A block or group of statements executed repeatedly until some condition is satisfied is called Loop.

The group of statements enclosed within curly brace is called block or compound statement.
We have two types of looping structures.
One in which condition is tested before entering the statement block called entry control.
The other in which condition is checked at exit called exit controlled loop.
Loop statements can be divided into three categories as given below
1.while loop statement
2.do while loop statement
3.for loop statement

1.WHILE STATEMENT :

```
While(test condition)
{
        body of the loop
}
```

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if is true body is executed once again. This goes on until test condition becomes false.




```
c program to find sum of n natural numbers */
#include<iostream.h> int main() {
int i = 1,sum = 0,n; cout<<"Enter N"<<end; cin>>n;
        while(i<=n)
        { sum =
        sum + i; i =
        i + 1;
        }


        cout<<"Sum of first"<<n<"natural numbers
        is:"<<sum<<endl; return 0;
}
```

2.DO WHILE STATEMENT :

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

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

```
/*c program to find sum of n natural numbers */
#include<stdio.
h> int main() {
int i = 1,sum = 0,n;
        cout<<"Enter
        N"<<endl; cin>>n

        do{
        sum = sum +
```

do-while – (Exit controlled)

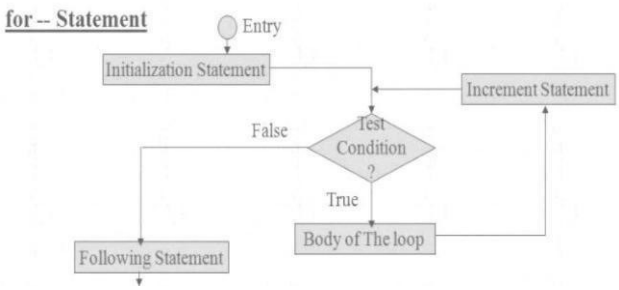
```
        i; i = i + 1;
        } while(i<=n);
        cout<<"Sum of first"<< n<<" natural numbers
        is:"<<sum; return 0;
}
```

Note: if test condition is false. before the loop is being executed then While loop executes **zero** number of times where as do--while executes **one** time

3.FOR LOOP : **It is also an entry control loop that provides a more concise structure**

Syntax:

```
        for(initialization; test expression; increment/decrement)
        { statements;
        }
```



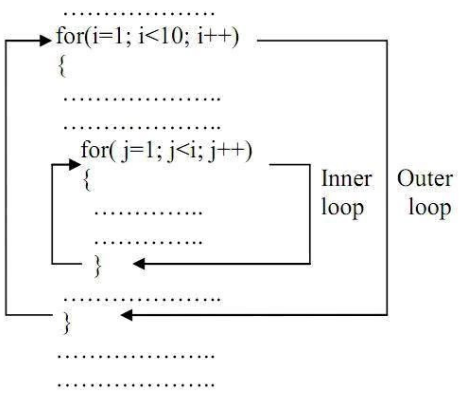
For statement is divided into three expressions each is separated by semi colon;

1.initilization expression is used to initialize variables 2.test expression is responsible of continuing the loop. If it is true, then the program control flow goes inside the loop and executes the block of statements associated with it .If test expression is false loop terminates

3.increment/decrement expression consists of increment or decrement operator This process continues until test condition satisfies.

```
/*c program to find sum of n natural numbers */
#include<stdio.h>


int main()
{
int i ,sum = 0,n;
        cout<<"Enter N";
        cin>>n;

        for(i=1;i<=n;i++)
         { sum = sum + i;
         }

         cout<<"Sum of first"<<n<<" natural numbers
         is:%d"<<sum; return 0;
}
```

Nested loops: Writing one loop control statement within another loop control statement is called nested loopstatement



Ex:

```
for(i=1;i<=10;i++)
for(j=1;j<=10;j++)
cout<<i<<j;
```

```
/*program to print prime numbers upto a given number*/
#include<stdio.h>
#include<conio.h>
void main()
{ int
  n,i,fact,j;
  clrscr();
cout<<"enter the number:";
  cin>>n
  for(i=1;i<=n;i++)
  {fact=0;
        //THIS LOOP WILL CHECK A NO TO BE PRIME NO. OR
        NOT. for(j=1;j<=i;j++)
        { if(i%j==0)
              fact++;
        }
        if(fact==2
        )

              cout<<i<<"\t";
   }
getch( );
}
```

Output:

```
Enter the number : 5
            2 3 5
```

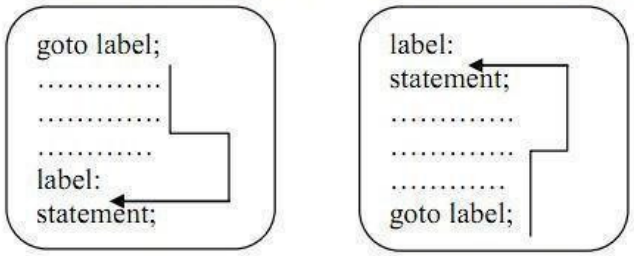
Unconditional control statements:

Statements that transfers control from on part of the program to another part unconditionally Different unconditional statements are

1)goto
2)break
3)continue

1.goto :- **goto statement**is used for unconditional branching or transfer of the program execution tothe labeled statement.

The goto statement to branch unconditionally from one point to another in the program. The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by colon (;). The label is placed immediately before the statement where the control is to be transferred. The general form of goto is shown below:



Forward Jump Backward Jump

The label: can be anywhere in the program either before or after the goto label; statement.

If the label: is placed after the goto label;, some statements will be skipped and jump is known as a Forward Jump.

If the label: is placed before the goto label; a loop will be formed some statements will be executed repeatedly. Such a jump is known as a Backward Jump.

```
/*c program to find sum of n natural numbers */
#include<stdio.
h> int main() {
int i ,sum = 0,n;
        cout<<"Enter
        N"; cin>>n;
        i=1; L1:
                 sum = sum + i;
                i++; if(i<=n)
                goto L1;



        cout<<"Sum of first "<<n<" natural numbers
        is"<<sum; return 0;
}
```

break:-when a break statement is encountered within a loop ,loop is immediatelyexited and the program continues with the statements immediately following loop

```
/*c program to find sum of n natural numbers */
#include<stdio.h>
int main()
{
int i ,sum = 0,n;
        cout<<"Enter
        N"; cin>>n;
        i=1; L1:
                sum = sum + i;
               i++;
               if(i>n)
               break;
                goto L1;

       cout<<"Sum of first"<<n<<"natural numbers is:
        "<<sum; return 0;
}
```

Continue: It is used to continue the iteration of the loop statement by skipping the statementsafter continue statement. It causes the control to go directly to the test condition and then to continue the loop.

```
/*c program to find sum of n positive numbers read from keyboard*/
#include<stdio.h>
int main()
{
int i ,sum = 0,n,number;
        cout<<Enter N";
        cin>>n;
        for(i=1;i<=n;i++)
         { cout<<"Enter a number:";
                cin>>number;
                if(number<0) continue;
                sum = sum + number;
         }
         cout<<"Sum of"<<n<<" numbers is:"<<sum;
        return 0;
}
```