OBJECT ORIENTED PROGRAMMING

UNIT-I

1.1 Why Programming Language?

Developments in software technology continue to be dynamic. New tools and techniques are announced in quick succession. This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry. These rapid advances appear to have created a situation of crisis within the industry. The following issued need to be addressed to face the crisis:

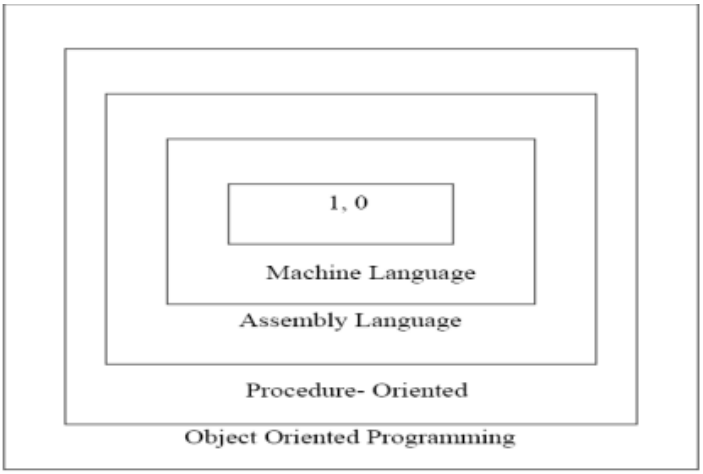
- How to represent real-life entities of problems in system design?
- How to improve software productivity and decrease software cost?
- How to improve the quality of software?
- How to manage time schedules?

Software product should be evaluated carefully for their quality before they are delivered and implemented. Some of the quality issues that must be considered for critical evaluation are:

1. Correctness
2. Maintainability
3. Reusability
4. Openness and interoperability
5. Portability
6. Security
7. Integrity
8. User friendliness

1.2 Software Evaluation

The software evolution has had distinct phases “layers” of growth. These layers were building up one by one over the last five decades as shown in fig., with each layer representing and improvement over the previous one.



Programmers write instructions in various programming languages to perform their computation tasks such as:

(i) Machine level Language
(ii) Assembly level Language
(iii) High level Language

Machine level Language: Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions.

Assembly level Language : An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

High level Language :

High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada , Algol, BASIC, COBOL, C, C++, JAVA, FORTRAN, LISP, Pascal, and Prolog.

Such languages are considered high-level because they are closer to human languages and farther from machine languages. In contrast, assembly languages are considered lowlevel because they are very close to machine languages.

The high-level programming languages are broadly categorized in to two categories:

(1) Procedure oriented programming (POP) language.

(2) Object oriented programming (OOP) language

With the advent of languages such as c, structured programming (POP) became very popular and was the main technique of the 1980‟s. Structured programming was a powerful tool that enabled programmers to write moderately complex programs fairly easily. However, as the programs grew larger, even the structured approach failed to show the desired result in terms of bug-free, easy-to- maintain, and reusable programs.

Object Oriented Programming (OOP) is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts

1.3 Procedure Oriented Programming Language

- In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading, calculation and printing.

- Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions.
- The primary focus is on functions. A typical structure for procedural programming is shown in fig.

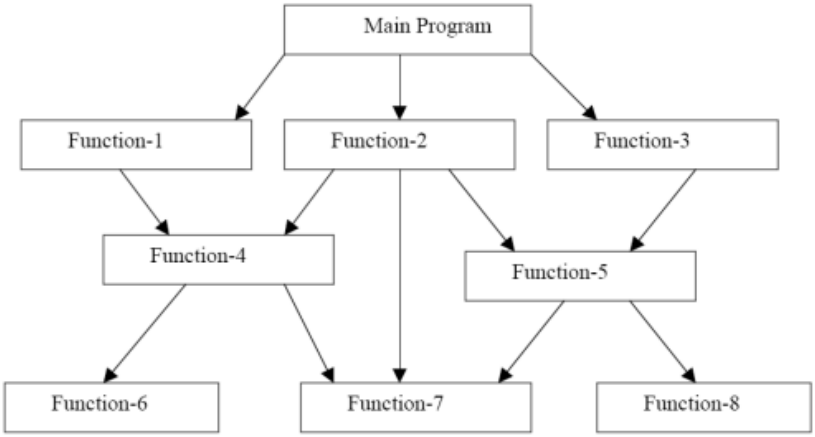


Fig. 1.2 Typical structure of procedural oriented programs

The disadvantage of the procedure oriented programming languages is:

1. Global data access (In a large program it is very difficult to identify what data is used by which function)
2. It does not model real word problem very well
3. No data hiding.

Characteristics of procedure oriented programming:

1. Emphasis is on doing things (algorithm)
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data
4. Data move openly around the system from function to function
5. Function transforms data from one form to another.
6. Employs top-down approach in program design

1.4 Object Oriented Programming Paradigm

- The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach.
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the function that operate on it, and protects it from accidental modification from outside function.
- OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects.
- The organization of data and function in object-oriented programs is shown in fig.
- The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects.

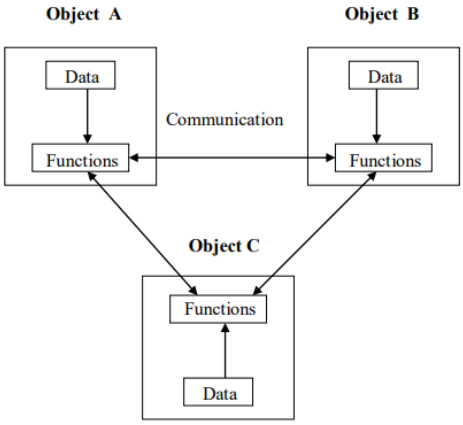


Fig: The organization of data and function in object-oriented programs

Features of the Object Oriented programming

1. Emphasis is on data rather than procedure.
2. Programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure.
5. Data is hidden and can't be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added.
8. Follows bottom-up approach in program design

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.

1.5 Basic Concepts of Object Oriented Programming

These include:

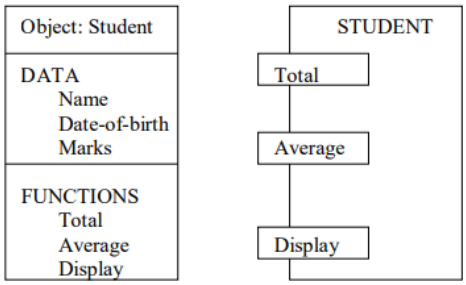
- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

Objects

- Objects are the basic run time entities (real world entity) in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.
- Program objects should be chosen such that they match closely with the real-world objects. When a program is executed, the objects interact by sending messages to one another.
- The term objects means a combination of data and program that represent some real word entity.

For example:
consider an example named Smith; Smith is 25 years old and his salary is 2500. The Smith may be represented as an object. The data part of the object would be (name: Smith, age: 25, salary: 2500). The program part of the object may be collection of programs (retrive of data, change age, change of salary. In the Smith object the name, age and salary are called attributes of the object.

Another Example:



Classes

- A group of objects that share common properties for data part and some program part are collectively called as class.
- In C ++ a class is a new data type that contains member variables and member functions that operate on the variables.
- A class is thus a collection of objects of similar types.

For examples,

Mango, Apple and orange members of class fruit. Classes are user-defined that types and behave like the built-in types of a programming language. The syntax used to create an object is not different than the syntax used to create an integer object in C. If fruit has been defines as a class, than the statement

Fruit Mango;

Will create an object mango belonging to the class fruit.

Data Encapsulation And Data Abstraction:

The wrapping up of data and function into a single unit (called class) is known as **encapsulation**. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding.

Data Abstraction refers to the act of representing essential features without including the back ground details or explanations. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

The attributes are sometime called **data members** because they hold information. The functions that operate on these data are sometimes called methods or **member function**.

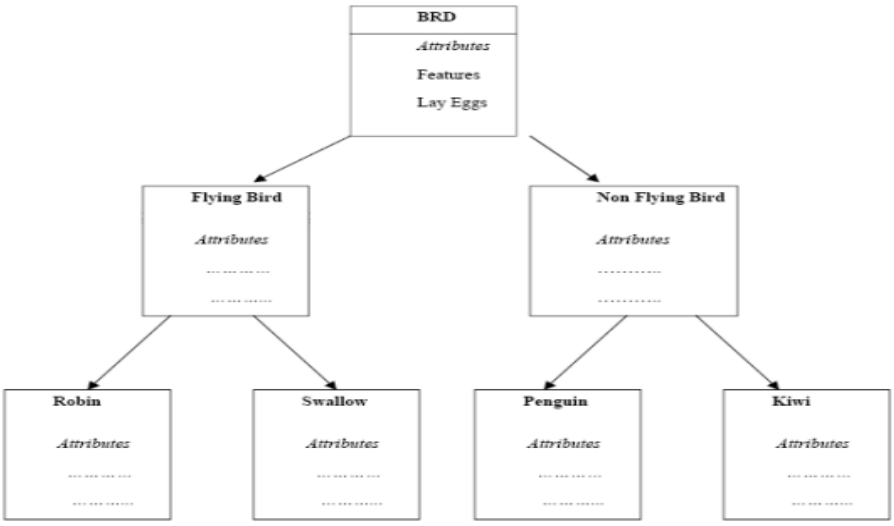
Since the classes use the concept of data abstraction, they are known as **Abstract Data Types (ADT).**

Inheritance:

Inheritance is the process by which objects of one class acquire the properties of another class.

In the concept of inheritance provides the idea of reusability. This mean that we can add additional features to an existing class without modifying it. This is possible by designing a new class will have the combined features of both the classes.

For example, the bird, „robin" is a part of class „flying bird" which is again a part of the class „bird". The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in fig.



Polymorphism: (overloading)

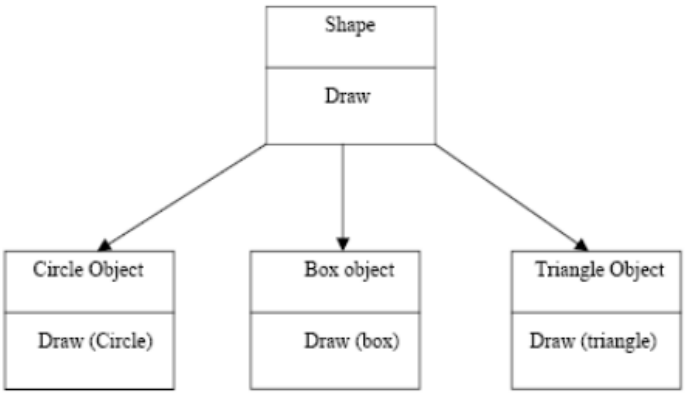
Polymorphism means the ability to take more than one form. An operation may exhibit different instance. The behaviour depends upon the type of data used in the operation.

Overloading may be operator overloading or function overloading.

The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.

It is able to express the operation of addition by a single operater say '+'. When this is possible you use the expression x + y to denote the sum of x and y, for many different types of x and y; integers , float and complex no. You can even define the + operation for two strings to mean the concatenation of the strings.

Fig. illustrates that a single function name can be used to handle different number and different types of argument. This is something similar to a particular word having several different meanings depending upon the context. Using a single function name to perform different type of task is known as function overloading.



Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. **Dynamic binding (late binding)** means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance.

Consider the procedure "draw" in the above fig. by inheritance, every object will have this procedure. Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current reference will be called.

Message Passing :

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

The process of programming in an object-oriented language, involves the following basic steps:

1. Creating classes that define object and their behavior,

2. Creating objects from class definitions, and

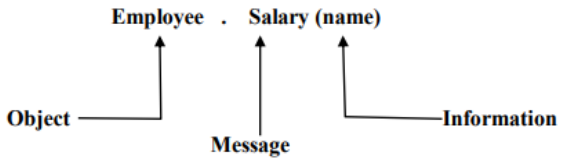
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another.

A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results.

Message passing involves specifying the **name of object, the name of the function (message) and the information** to be sent.

Example:



1.6 Benefits and Drawbacks OF OOP:

The principal advantages are:

1. Through inheritance we can eliminate redundant code and extend the use of existing classes.

2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.

3. This principle of data hiding helps the programmer to build secure programs that can't be invaded by code in other parts of the program.

4. It is possible to have multiple instances of an object to co-exist without any interference.

5. It is easy to partition the work in a project based on objects.

6. Object-oriented systems can be easily upgraded from small to large systems.

7. Message passing techniques for communication between objects makes the interface description with external systems much simpler.

8. Software complexity can be easily managed.

Disadvantages of OOP

1. The length of the programmes developed using OOP language is much larger than the procedural approach. Since the programme becomes larger in size, it requires more time to be executed that leads to slower execution of the programme.
2. It is applied only when it is required. It is not suitable for all types of problems.
3. Programmers need to have brilliant designing skill and programming skill along with proper planning because using OOP is little bit tricky.
4. OOPs take time to get used to it. The thought process involved in object-oriented programming may not be natural for some people.
5. Everything is treated as object in OOP so before applying it we need to have excellent thinking in terms of objects.

1.7 Object Oriented Language

The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

1. Object-based programming languages, and

2. Object-oriented programming languages.

Object-based programming is the style of programming that primarily supports encapsulation and object identity. Major feature that are required for object based programming are:

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding.

Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding.

Object-oriented programming can therefore be characterized by the following statements:

Object-based features + inheritance + dynamic binding

1.8 Application of OOP

The promising areas of application of OOP include:

- Real-time system

• Simulation and modelling

• Object-oriented data bases

• Hypertext and Hypermedia

• AI and expert systems

• Neural networks and parallel programming

• Decision support and office automation systems

• CAD systems

1.9 Introduction of C++

- C++ is an object-oriented programming language. It was developed by **Bjarne Stroustrup** at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980‟s.
- Stroustrup, an admirer of Simula67 and a strong supporter of C, wanted to combine the best of both the languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++.
- Therefore, C++ is an extension of C with a major addition of the class construct feature of Simula67. Since the class was a major addition to the original C language, Stroustrup initially called the new language "**C with classes‟.**
- However, later in 1983, the name was changed to C++. The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented version of C.

1.10 Applications of C++

- **Operating Systems**: Be it Microsoft Windows or Mac OSX or Linux - all of them are programmed in C++.
- **Libraries**: Many high-level libraries use C++ as the core programming language. For instance, several Machine Learning libraries use C++ in the backend because of its speed.
- **Graphics:** Even the popular games that are heavy on graphics use C++ as the primary programming language.
- **Banking Applications:** One of the most popularly used core-banking system - Infosys Finacle uses C++ as one of the backend programming languages.
- **Cloud/Distributed Systems:** Large organizations that develop cloud storage systems and other distributed systems also use C++ because it connects very well with the hardware and is compatible with a lot of machines.
- **Databases:** Postgres and MySQL - two of the most widely used databases are written in C++ and C, the precursor to C++.
- **Embedded Systems:** Various embedded systems like medical machines, smart watches.
- **Telephone Switches:** Because of the fact that it is one of the fastest programming languages, C++ is widely used in programming telephone switches, routers, and space probes.
- **Compilers:** The compilers of various programming languages use C and C++ as the backend programming language.

1.11 Advantages and Disadvantages of C++ Language

Advantages of C++

- **Speed**: C++ is faster than most other programming languages and it provides excellent concurrency support.
- **Closer to hardware**: C++ is closer to hardware than most other programming languages like Python, etc.
- **Portability**
- **Multi-paradigm:** The term "Paradigm" refers to the style of programming.
- **Memory Management :C**++ gives the programmer the provision of total control over memory management.
- **Scalability:** C++ program is capable of running on a small scale as well as a large scale of data.
- **Compatibility with C:** C++ is pretty much compatible with C. Virtually, every error-free C program is a valid C++ program.

Disadvantages of C++

- **Security vulnerabilities:** Poor coding can make it easy for hacks to gain entry, get control over the application's execution flow.
- **Absence of garbage collector:** there is no garbage collection in C++,
- **Use of pointers:** The pointers C++ use up a lot of memory. Memory corruption can also happen when they use wrong values.
- **No support for built-in thread:** Since threads were a relatively new concept, C++ doesn't support them.
- **Not flexible at all:** The syntax is very strict, and a little mistake can give you a series of errors. It is not user-friendly for the beginner.

1.12 C++ Comments:

C++ introduces a new comment symbol //(double slash). Comments start with a double slash symbol and terminate at the end of line. A comment may start any where in the line and what ever follows till the end of line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multi line comments can be written as follows:

```
// this is an example of
 // c++ program
// thank you
```

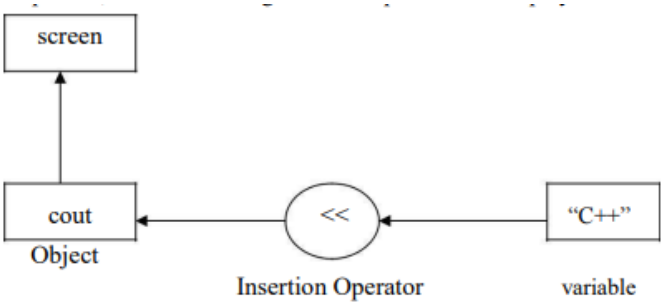
The c comment symbols /* ….*/ are still valid and more suitable for multi line comments.

```
/* this is an example of c++ program */
```

1.13 Output Operator: (Output Stream)

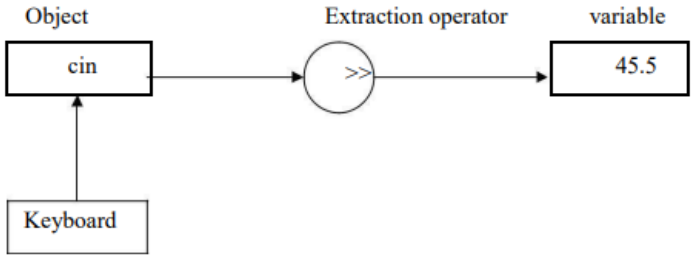
- The **cout** is a predefined object of **ostream** class.
- It is connected with the standard output device, which is usually a display screen.

- The cout is used in conjunction with stream insertion operator (<<) to display the output on a console.
- The statement cout << also called the 'put to' or 'insertion' operator directs the information on its right to the object on its left.



1.14 Input Operator: (input Stream)

- The **cin** is a predefined object of **istream** class.
- It is connected with the standard input device, which is usually a keyboard.
- The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.
- The operator >> is known as 'get from' or 'extraction' operator.
- It extracts value from the keyboard and assigns it to the variable on its right.
- **Example: cin >> number 1**



1.15 Cascading Of I/O Operator:

cout<<"sum= " << sum << "\n" <<average;

cin >> number1 >> number2;

1.16 Structure of C++ Program

Include Files
Class declaration
Member functions definitions
Main function program

Example:1

```
#include <iostream.h>
#include<conio.h>
void main ()
{
cout << "Hello World!";
 getch();
 }
```

Output:-Hello World!

Example:2 AVERAGE OF TWO NUMBERS

```
 #include<iostream.h>  // include header file
#include<conio.h>
void main()
 {
 float number1, number2,sum, average;
 cin >> number1; // Read Numbers
cin >> number2; // from keyboard
 sum = number1 + number2;
average = sum/2;
cout << ”Sum = “ << sum << “\n”;
 cout << “Average = “ << average << “\n”;
 getch();
 } //end of example
```

The output would be: Enter two numbers: 6.5 7.5

Sum = 14 Average = 7 Program 1.11.1

1.17 Compiling and Linking

- The process of compiling and linking again depends upon the operating system.

Unix AT&T C++

This process of implementation of a C++ program under UNIX is similar to that of a C program. We should use the “cc” (uppercase) command to compile the program. The command ,

CC example.C

Turbo C++ and Borland C++

- Turbo C++ and Borland C++ provide an integrated program development environment under MS DOS.
- They provide a built-in editor and a menu bar includes options such as **File, Edit, Compile and Run**.

1.18 Variables and Constants

Variables:

- A variable is a data name that may be used to store a data values. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution.

Constants:

- Constants is refered to fixed values that do not change during the execution of a program.

Character Set

The collection of characters which are used in C++ programming language are called character set. It consists of

1. Letters : a to z, A to Z

2. Digits : 0 to 9

3. White spaces

- Horizontal Tab \t
- New Line \n

4. Special characters

,	Comma	**&**	Ampersand
.	Period	^	Caret
;	Semicolon	*****	Asterisk
:	Colon	-	Minus Sign
?	Question Mark	+	Plus Sign
'	Aphostrophe	<	Opening Angle (Less than sign)
"	Quotation Marks	>	Closing Angle (Greater than sign)
!	Exclaimation Mark	**(**	Left Parenthesis

|	Vertical Bar	)	Right Parenthesis
/	Slash	[	Left Bracket
\	Backslash	]	Right Bracket
~	Tilde	{	Left Brace
-	Underscore	}	Right Bracket
$	Dollar Sign	#	Number Sign
%	Percentage Sign		

1.19 TOKENS:

Tokens are individual words and punctuation marks in passage of text. In C++, program the smallest individual units are known as C++ Tokens. C++ has Six types of Tokens. The Tokens are shown in figure. C++ programs are written using these tokens and the syntax of the language.

C++ is the superset of C and so most constructs of C are legal in C++ with their meaning and usage unchanged. So tokens, expressions, and data types are similar to that of C.

Following are the C++ tokens : (most of c++ tokens are basically similar to the C tokens)

- Keywords
- Identifiers
- Constants
- Strings
- Operators

Keywords

Keywords/Reserved words :

- It has predefined meaning and cannot be used for any other purpose in a C program.
- C++ has more numbers of keyword than C.
- They are always written in lower case.

Keywords in C			
auto	else	register	union
break	enum	return	unsigned
case	extern	short	void
char	float	signed	Volatile
const	for	sizeof	while
continue	goto	static	.
default	if	struct	.

do	int	switch	.
double	long	typedef	.

There are another reserved words that were not in C, are therefore new to C++, and here they are –

```
dynamic_cast namespace reinterpret_cast try
bool explicit new static_cast typeid
catch false operator template typename
class friend private this using
const_cast inline public throw virtual
delete mutable protected true wchar_t
```

Identifiers

"Identifiers" or "symbols" are the names you supply for variables, types, functions, and labels in your program. Once defined they can be referred at any point within the C program.

Rules for forming identifiers / variables:

- The first character should be an alphabet or underscore.
- Subsequent characters may be any combination of letters, digits and underscores.
- Special characters are not allowed
- Blank spaces are not permitted
- Reserved words cannot be used as identifiers.

Examples of valid identifiers:

- r
- sumOfRoots1and2
- _XYZ
- maxThrowsPerTurn
- TURNS_PER_GAME
- R2D2

Examples of invalid identifiers:

- 2hotToHandle // does not start with a letter
- Net Pay // contains a space
- ALPHA;BETA // contains an invalid character ;

Important points to note:

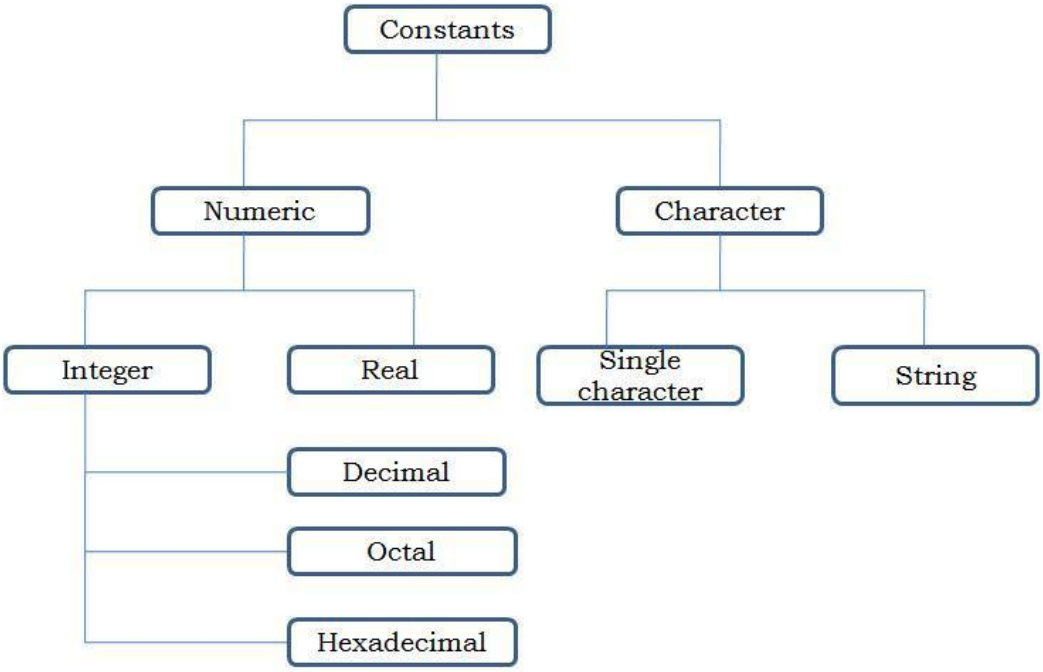
- Spaces are not allowed in an identifier. If you need one which consists of two or more words, use a combination of uppercase and lowercase letters (as in

numThrowsThisTurn) or use the underscore to separate the words (as in num_throws_this_turn). We prefer the uppercase/lowercase combination.

- In general, C++ is case-sensitive (an uppercase letter is considered different from the corresponding lowercase letter). Thus r is a different identifier from R. And sum is different from Sum is different from SUM is different from SuM.

Constants

A constant value is the one which does not change during the execution of a program. C++ supports several types of constants.



1. Integer Constant :
An integer constant is a sequence of digits without decimal point. There are 3 types.
*Decimal
*Octal
* Hexadecimal

*** Decimal :**
It consists of a set of digits 0 to 9 preceded by an optional + or - sign. Spaces, commas and non digit characters are not permitted between digits.

Example for valid decimal integer constants are
123
-31
0
562321
+ 78

Some examples for invalid integer constants are
15 750
20,000
Tsh. 1000

*** Octal :**
It consists of any combination of digits from 0 through 7 with zero at the beginning.
Some examples of octal integers are
O26
O
O347
O676

*** Hexadecimal :**
It is preceded by OX or Ox, they may contain alphabets from A to F or a to f. The alphabets A to F refers to 10 to 15 in decimal digits.

Example of valid hexadecimal integers are
OX2
OX8C
OXbcd
Ox

2. Real/Floating point Constant :

- The numbers with decimal point is called real or floating point constant.
- It is formed with the digits 0 to 9 and decimal point.
- The number can be preceded by optional + or - sign.
- The digits before or after the decimal point can be omitted
- No white spaces or special characters are allowed.
- It can be represented with **f** for floating and **l** for double precision floating point numbers.
- It can be expressed in two forms.

*Fractionalform
*Exponential form

Fractional form :
The number is written with decimal point.
Example
0.0026
-0.97
435.29
+487.0

Exponential form :
This is useful for representing very large or very small numbers. The general form is



Example
The number 215.65 can be written in exponential notation as $2.1565e^{2}$
The number -0.0000375 can be written as $-3.752e^{-5}$

Two ways to declare a constant :

- Using const keyword
- Using #define

Using **const** Keyword :
Suppose you have a circle with a fixed radius of 5 units. You can create a symbolic constant like this:
const int radius = 5; const float radius=5.3;

Since radius is declared using the const keyword, statements like: radius = 12; would be illegal.

Using **#define :**
#define allows you to make text substitutions before compiling the program. It should be declared before the main() function.

Here's an example: #define radius 5

#define MAX 10 #define radius 5.3

Before compilation, if the C preprocessor finds MAX as one word in the source code, it replaces it with the number 10. If MAX was part of a string the preprocessor will leave it alone.

3. Character constant :
A Single Character constant represents a single character which is enclosed with in a single quote. All character constants have an equivalent integer value which are called ASCII Values.
Example for character constants are
'5' const char radius= ‘ f ‘; #define radius ‘f’
'x' ‘f ’
4. String constant :
A string constant is a set of characters enclosed in double quotation marks. The characters in a string constant sequence may be an alphabet, numbers, special characters and blank space.
Example of string constants are
"VISHAL" const char name = “ hamisi”; #define name “ hamisi”
"1234"
"God Bless"

Backslash Character Constant/Escape Sequences
Backslash character constants are special characters used in output functions. Although they

contain two characters they represent only one character.

Escape Sequences	
Meaning	**Constant**
Backspace	'\b'
Newline	'\n'
horizontal tab	'\t'
vertical tab	'\v'
single quote	'\' '
double quote	'\" '
question mark	'\?'
Backslash	'\\'
null character	'\0'

1.19 Datatypes

There are many data types in C++ language. A C++ programmer has to use appropriate data type as per his requirement.

C++ language data types can be broadly classified as

1. Primary data type / Built-in Type
2. Derived data type
3. User-defined data type

1. Primary DataTypes

C++ supports five primary data types. The following table gives the basic data types and their range of values

Data type	Keyword	No.of bytes
Character	char	1 byte
Integer	int	2 byte
Single precision real number	float	4 byte
Double precision real number	double	8 byte
Void	void	

- **Character:** To represent single character
- **Integer:** To represent integer numbers
- **Float and Double:** To represent real numbers

- **Void type:** Void data type has no value.

  Two normal uses of **void** are

  (1) to specify the return type of a function when it is not returning any value,
  Eg: void display();
  (2) to indicate an empty argument list to a function.
  Eg: void display (void);

C++ added two more data types:

1. **bool**: For booleans, true or false.

```
bool b = true;
```

2. **wchar_t**: Wide Character. Used for international distribution of programs

2. User defined data types

- structure
- classes (will be discussed in unit-2)
- union
- Enumeration
- typedef

a) typedef:

The typedef feature allows the programmer to define new data type that is equivalent to the existing data types. The user defined data type then can be used to declare variables.

Syntax:

typedef datatype identifier;

Example:

typedef int Mark;

Here Mark is declared as a user defined data type of type int. Then Mark can be used to define variables as

Mark m1,m2;

Now m1 and m2 indirectly refers to the type int.

Example:

```
#include <iostream>
#include<stdio.h>
void main()
{
cout<<"Welcome to DataFlair tutorials!" ;
typedef  int  number; // Use of typedef keyword
 number  n= 80;
cout<<"The number is: "<< n;
getch();
}
```

b) Enumerated type

Enumeration is a user defined data type associated with some values.

Syntax:

enum identifier {value1, value2, ….., valuen};

Here enum is the keyword. Identifier is the name given to the data type and the values within { } are the values that the enum data type can hold.

- These list of values within { } are called enumerated value or enumerated constant.
- The compiler automatically assigns integer values to the enumerated constants starting from 0. We can also assign integers values explicitly.
- Once defined the enumerated data type can be used to define other variables which can take any one of the enumerated constant associated with that type.

Syntax:

enum identifier {var1, var2,….var n};

Example:

enum day {Sunday,Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, x, num};

enum day week_st, week_end;

week_st = Monday;

week_end = Friday;

Here the compiler assigns the value Sunday =0, Monday= 1,….., Saturday=6

Suppose we define

enum day {Sunday=8,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday};

Then the compiler will assign Sunday=8 , Monday =9, …, Saturday=14.

Example for Enumerated type

```
#include <iostream>
#include<conio.h>
void main()
{
enum colors{red=5, black};
enum suit{heart, diamond=8, spade=3, club};

  cout <<"The value of enum color : "<<red<<","<<black;
  cout <<"\nThe default value of enum suit ";
    cout<< <<heart<<","<<diamond<<","<<spade<<","<<club;
   getch();
}
```

Output

```
The value of enum color : 5,6
The default value of enum suit : 0,8,3,4
```

C) Structure

- Structure in c is a user-defined data type that enables us to store the collection of different data types.
- Each element of a structure is called a member.
- The ,struct keyword is used to define the structure.

Let's see the syntax to define the structure in c.

struct structure_name

{

data_type member1;

data_type member2;

. . .

data_type memeberN;

};

Let's see the example to define a structure for an entity employee in c++.

```
struct employee
{ int id;
char name[20];
 float salary;
} x1,x2;
```

d) Union

Union can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location.

- The union can also be defined as many members, but only one member can contain a value at a particular point in time.
- Union is a user-defined data type, but unlike structures, they share the same memory location

```
union abc
{
int a;
char b;
} var;
```

3. Derived Data Types:

The array, functions, pointers and reference are the derived data type in C++.

a) Array

- An array is defined as the collection of similar type of data items stored at contiguous memory locations. The application of array in C++ is similar to C. the only difference in character array.

  In C,
  char str[3]= “xyz”;
  In C++, the size should be one larger than the number of characters in the string

  Char str[4]=”xyz”

b) Pointer

- is a variable to store the address of another variable
  int a,*p;
  p=&a;
  (will be discussed later in detail)

c) Reference

- It is a variable that behaves as an alias for another variable.
- Reference can be created by simply using an ampersand (&) operator.
- When we create a variable, then it occupies some memory location. therefore, we can access the original variable by using either name of the variable or reference. For example,

  int a=10;

  Now, we create the reference variable of the above variable.

  int &r =a;

The above statement means that 'r' is a reference variable of 'a', i.e., we can use the 'r' variable in place of 'a' variable.

```
1.  #include <iostream.h>
2.  #include<conio.h>
3.  void main()
4.  {
5.  int a=70; // variable initialization
6.  int &b=a;
7.  int &c=a;
8.  cout << "Value of a is :" <<a;
9.  cout << "Value of b is :" <<b;
10. cout << "Value of c is :" <<c;
11. getch();
12. }
```

d) Functins (will be discussed later in detail

1.20 VARIABLES

- A variable is used to hold data within our program.
- A variable represents a location in our computer's memory.
- We can put data into this location and retrieve data out of it.
- Every variable has two parts, **a name and a data type**.
- There are two main types of variables in C: numeric variables that hold only numbers or values, and string variables that hold text, from one to several characters long.
- Before you use a variable, it must be declared.

Declaration of Variables

Declaration :

A variable declaration serves three purposes:

1. It defines the name of the variable.
2. It defines the type of the variable (integer, real, character, etc.).
3. It gives the programmer a description of the variable.

Syntax :

datatype variablename1,..,..,variablenameN;

- where, datatype - valid data type such as int,char etc.,
- variable list - list of variables separated by comma

Example :

int answer; /* the result of our expression */

float d,v;

char x,z,c;

1. The keyword **int** tells C++ that this variable contains an integer value.
2. The variable name is **answer**.
3. The semicolon (**;**) marks the end of the statement.

Assigning values to variables :

The process of giving value to the variables is called assigning values to variables .

Syntax :

datatype variablename=value;

or

datatype variablename;
variablename=value;

Example :

- ✓ int x=10;

or

int x;

x=10;

- ✓ char gender='F';

Dynamic Initialization of Variables

The process of initializing a variable at the moment it is declared at runtime is called dynamic initialization of the variable.

Example:

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a;
 cout<<“Enter Value of a”;
  cin>>a;
int cube = a * a * a;
getch();
}
```

In the above example, the variable cube is initialized at run time using expression a * a * an at the time of its declaration.

Scope of variables

1. Variable can be defined in the block.
2. The blocks are marked using { and } braces.
3. The scope of the variable is in the block where it is declared.

Program :

```
#include <iostream.h>
#include<conio.h>
void main() {
   int a,b;// a and b are integer variable
   cout<<" Enter first number :";
   cin>>a;
   cout<<" Enter the second number:";
   cin>>b;
   int sum;
   sum=a+b;
   cout<<" Sum is : "<<sum <<"\n";
```

```
    getch();
}
```

Output :

TYPES OF VARIABLES

1. Global variable
2. Local variable

1. **Global variables**

A global variable is available to all functions in your program. A local variable is available only to the function in which it's created.

1. Global variables are declared outside of any function.
2. Global variables are typically declared right before the main() function.
3. You can also declare a group of global variables at one time.

Program :

```
1. #include<iostream.h>
2. #include<conio.h>
3. int count =10;        /* Declare a global variable   */
4. void main()
5. {
6. clrscr();
7. cout<<“ global variable value is “<< count;
8. getch();
9. }
```

Output :

global variable value is=10

2. **Local variables :**

A local variable is a variable that is given local scope. Such a variable is accessible only from the function or block in which it is declared.

Program :(Local variable)

```
1. #include<iostream.h>
2. #include<conio.h>
3. void main()
4. {
5. int i ;// local variable
6. i=5;
7. cout<<"local variable value of i in main\n"<<i;
8.     getch();
```

9. }

Output :

Local variable value of i in main 5

1.21 operator

An operator is a symbol which represents some operation that can be performed on data.

Operands :

The data items that the operator acts upon are called operands.

- For example, in the expression y + x 4+7
- x and y are operands and + is an operator.

Types of operators : (discussed in C- available in supplementary notes)

1. Arithmetic operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators

Additional / Special operators in c++

1. Insertion Operator (<<)
2. Extraction Operator (>>)
3. Scope Resolution Operator (::)
4. Member Dereferencing Operator
5. Memory Management Operator (new, delete)
6. Manipulators in C++
7. Type cast operator

1. Output Operator (Insertion Operator <<)
    - To display single line or statement on console IO window “cout<<” function is used.

Example:

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

2. Input Operator (Extraction Operator >>)

- To read a value or any input from the user using console IO , input operator "cin<<" is available from the c++ library <iostream.h>.

Example:

```
1. cin >> number1;

2. cin >> valueX >> valueY;
```

3. Scope Resolution Operator (::)

- Visibility or availability of a variable in a program is called as scope. There are two types of scope in c++ programs.

1. Local scope
2. 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 a program. Scope resolution operator in "::" is used to access global variables if same variables are declared as local and global.

Syntax:

```
:: variable–name;
```

Example:

```
#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

4.Member Dereferencing Operator

- In computer programming, a dereference operator, also known as an indirection operator, operates on a pointer variable.
- It returns the location value in memory pointed to by the variable's value.

In C++ Programming we have three different dereferencing operators.

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

5. Memory Management Operator

C Language uses *malloc* and *calloc* functions to allocate memory dynamically at run time. Similarly it uses the functions *Free( )* to free dynamically allocated memory.

C++ also support those functions and it also defines two unary **operators *new* and *delete*** that perform the task of allocating and freeing the memory in a better and easier way.

1. *new* operator

- *new* operator allocates sufficient memory to hold data of objects and it returns address of the allocated memory from the heap.
-

Syntax:

```
pointer-variable = new data-type;
```

Example:

```
int *p =new int;
```

2. *delete* operator

- The memory that your pointer points to because of the *new* operator can also be "deallocated," not destroyed but rather, freed up from your pointer.
- The *delete* operator is used in front of a pointer and frees up the address in memory to which the pointer is pointing.

```
delete pointer-variable;

delete [size] pointer-variable;
```

Example:

```
delete p;

delete q;

delete []array;
```

6.Manipulators

- Manipulators are operator that are used to format the data display.
- The most commonly manipulators are *endl* and *setw*.

To use various manipulators in C++ program <*iomanip.h*> header file is compulsory to include.

List of Manipulators

1. endl
2. setw

***endl* – endline manipulator**

- The *endl manipulator*, when used in an output statement, causes a line feed to be insert. Just like a "\n" new line feed character.

Example: Suppose values of m=20, n=25.75 and p=125 then,

```
cout<<"m="<<m<<endl;

cout<<"n="<<n<<endl;

cout<<"p="<<p<<endl;

Output:

m=20

n=25.75

p=125
```

***setw* – setwidth manipulator**

- *setw manipulator* allows a specified width for a field that is to be printed on screen and by default the value printed is right justified.
- The output will always contain at least the number of digits specified as the field width

```
cout << setw(6);
```

Example 2:

```
cout<<setw(10)<<123.4<< endl

<<"1234567890 << endl;

Output:

      123.40          // Field width 10

1234567890
```

7. **Type Cast operator**

- Type casting refers to the conversion of one data type to another in a program.

Syntax of the type casting

(type) expression;

type: It represents the user-defined data that converts the given expression.

expression: It represents the constant value, variable, or an expression whose data type is converted.

For example, we have a floating pointing number is 5.534, and to convert an integer value, the statement as:

```
int num;
num = (int) 5.534; // cast into int data type
cout << num;
```

When the above statements are executed, the floating-point value will be cast into an integer data type using the cast () operator. And the float value is assigned to an integer num that truncates the decimal portion and displays only 5 as the integer value.

Program to demonstrate the use of the type casting in C++

A simple program to cast one type variable into another type using the type casting in the C++ programming language.

```
#include <iostream>
#include<conio.h>
void main ()
{
    // declaration of the variables
    int a, b;
    float res;
    a = 21;
    b = 5;
    cout << " Implicit Type Casting: " << endl;
    cout << " Result: " << a / b << endl; // it loses some information

    cout << " \n Explicit Type Casting: " << endl;
    // use cast () operator to convert int data to float
    res = (float) 21 / 5;
    cout << " The value of float variable (res): " << res << endl;

    getch();
}
```

Output:

```
Implicit Type Casting:
Result: 4

Explicit Type Casting:
The value of float variable (res): 4.2
```

1.21 C++ Expression

C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language.

An expression can be of following types:

- Constant expressions
- Integral expressions
- Float expressions
- Pointer expressions
- Relational expressions
- Logical expressions
- Special assignment expressions

Constant expressions

A constant expression is an expression that consists of only constant values.

Expression containing constant	Constant value
x = (2/3) * 4	(2/3) * 4

Integral Expressions

An integer expression is an expression that produces the integer value as output after performing all the explicit and implicit conversions.

1. (x * y) -5
2. x + **int**(9.0)
3. where x and y are the integers.

Float Expressions

A float expression is an expression that produces floating-point value as output after performing all the explicit and implicit conversions.

1. x+y
2. (x/10) + y

3. 34.5
4. x+**float**(10)

Pointer Expressions

A pointer expression is an expression that produces address value as an output.

The following are the examples of pointer expression:

1. &x
2. ptr
3. ptr++
4. ptr-

Relational Expressions

A relational expression is an expression that produces a value of type bool, which can be either true or false. It is also known as a boolean expression.

An example

1. a>b
2. a-b >= x-y
3. a+b>80

Logical Expressions

A logical expression is an expression that combines two or more relational expressions and produces a bool type value. The following are some examples of logical expressions:

1. a>b && x>y
2. a>10 || b==5

Special Assignment Expressions

Special assignment expressions are the expressions which can be further classified depending upon the value assigned to the variable.

- **Chained Assignment**

Chained assignment expression is an expression in which the same value is assigned to more than one variable by using single statement.

For example:

1. a=b=20
2. or
3. (a=b) = 20

- **Embedded Assignment Expression**

An embedded assignment expression is an assignment expression in which assignment expression is enclosed within another assignment expression.

Let's understand through an example.

```
1.  #include <iostream>
2.  #include<conio.h>
3.  void main()
4.  {
5.   int a;  // variable declaration
6.   int b;  // variable declaration
7.   a=10+(b=90);  // embedded assignment expression
8.   cout <<"Values of 'a' is " <<a;
9.   getch();
10. }
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we applied embedded assignment expression (a=10+(b=90)).

Output

```
Values of 'a' is 100
```

- **Compound Assignment**

A compound assignment expression is an expression which is a combination of an assignment operator and binary operator.

For example,

```
1.  a+=10;
```

1.22 Control Statements in C++:

In C++, **Control Statements** are usually jumped from one part of the C++ code to another depending on whether a particular condition is satisfied or not.

Types of Control Statements in C++:

1. simple if Statement

Simple if statement is used to execute or skip a part of the statement based on the condition

Syntax:

if (condition)

statement;

or

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

Example: To find the given no is even or not using simple if statement

```
1.  #include <iostream.h>
2.  #include<conio.h>
3.  void main ()
4.   {
5.     int num = 10;
6.           if (num % 2 == 0)
7.           {
8.              cout<<"It is an even number";
9.           }
10.    getch();
11. }
```

Output:

```
It is even number
```

2. if-else statement:

if-else statement is used to execute a statement block. It has the following syntax :

```
if(expression is true)
{
action 1;
}
else
{
action 2;
}
```

Example: To find the given no is even or not using simple if-else statement

```
1.  #include <iostream.h>
2.  #include<conio.h>
3.  void main ()
4.   {
5.     int num = 11;
6.           if (num % 2 == 0)
7.           {
8.              cout<<"It is an even number";
9.           }
10.          else
```

```
11.          {
12.              cout<<"It is an odd number";
13.          }
14.   getch();
15. }
```

Output:

```
It is odd number
```

3. The else..if ladder Statement

- The else..if ladder is used for multiple decision making. In this construct each else is associated with another if statement.
- The default else is optional.
- The conditions are evaluated from top to bottom one by one.

Syntax:

```
if(condition1)
{
      Statement block– 1;
}
else if(condition2)
{
       Statement block-2;
}
 else if(condition3)
{
        Statement block-3;
}
……
 else if(condition-n)
{
       Statement block-n;
}
else
{
       Default statement block;
}
```

Example: To find the grade of given number using else if ladder

```
1. #include <iostream>
2. #include<conio.h>
3. void main ()
4. {
```

```
5.      int num;
6.      cout<<"Enter a number to check grade:";
7.      cin>>num;
8.
9.          if(num >= 0 && num < 40){
10.             cout<<"Fail";
11.         }
12.         else if (num >= 40 && num < 60)
13.         {
14.             cout<<"C Grade";
15.         }
16.         else if (num >= 60 && num < 70)
17.         {
18.             cout<<"B Grade";
19.         }
20.         else if (num >= 70)
21.         {
22.             cout<<"A Grade";
23.         }
24.         else
25.         {
26.             cout<<"wrong number";
27.         }
28.   }
```

Output:

```
Enter a number to check grade:66
C Grade
```

Output:

```
Enter a number to check grade:-2
wrong number
```

4. Nested if Statement

If an if..else statement appears as a statement in another if or else block then it is called nested if..else.

Syntax:

```
if (condition1)
{
        if (condition2)
        {
            Statement block-1;
        }
        else
```

```
        {
            Statement block-2;
        }
}
else
{
    Statement block-3;
}
Next statement
```

5. switch statement:

It is a multiple branching statement where based on a condition, the control is transferred to one of the many possible points. It has the following syntax :

Syntax:

```
switch(expression)
{
case value1:
//statement1;
break;
case value2:
//statement2;
break;
...
default:
// default statement;
break;
}
```

Example:

```
#include <iostream.h>
#include <conio.h>
void main()
{
int s; int a,b,c;
cout<<"\n Enter any value of s";
```

```
cin>>s;
cout<<"\n Enter the Two numbers:";

cin>>a>>b;
switch (s)
{
case 1:
c=a+b;
cout<< addition " <<c;
break;
case 2:
c=a - b;
cout<<"\n subtraction" << c;
break;
case 3:
c=a*b;
cout<<"\n multiplication  "<< c;
break;
default:
cout<<"\n select the correct value only";
break;
}
getch();

}
```

6. for loop:

for is an entry controlled loop. It is used when an action is to be repeated for a predetermined time.

It has the following syntax :

```
for(initial value; test condition; increment or decrement)
{
action 1;
```

}

Example: Programm to print 1 to 10 using for loop

```
1. #include <iostream.h>
2. #include<conio.h>
3. void main() {
4.         for(int i=1;i<=10;i++) {
5.           cout<<i <<"\n";
6.         }
7. cout<" end of the program";
8. getch();
9.    }
```

7. **while loop:**

It is also a loop control structure but it's an entry-controlled loop.
It has the following syntax:

```
while (condition is true)
{
action 1;
}
action 2;
```

Example: program to find sum of positive integers

```
#include <iostream.h>
#include<conio.h>
void main() {
    int number;
    int sum = 0;

    // take input from the user
    cout << "Enter a number: ";
    cin >> number;

    while (number >= 0)
     {
     sum += number;
     cout << "Enter a number: ";
        cin >> number;
      }
        cout << "\nThe sum is " << sum << endl;
```

```
    getch(); }
```

Output

```
Enter a number: 6
Enter a number: 12
Enter a number: 7
Enter a number: 0
Enter a number: -2

The sum is 25
```

8. do-while loop:

do-while is an exit controlled loop based on a condition, the control is transferred back to a particular point in the program. It has the following syntax :

```
do
{
action;
}
while(condition is true);
action 2;
```

Example: Rewrite the above program using do…while loop.