INDEX

EXPLORING PROGRAMMING BASICS AND OOPs CONCEPTS

WHAT IS C++?:-

The C++ is a general-purpose programming language that inherits several features of C language. Adding the concept of object oriented programming (OOP) to the C language is the main purpose of C++ programming. C++ is one of the most popular languages employed primarily by application software systems, drivers and client-server applications.

OOP allows programmer to create objects in the code and implement real world entities. These objects have certain properties and methods. While designing C++ modules, the whole world is considered an object. For example a person is considered an object having the properties name, address, city, hobbies, etc.

WHY WE NEED C++:-

C++ is also Middle-level language; it takes benefit of both low-level and high-level languages. It can handle large programs due to versatility. Other than portability, high speed, reusability and rich library.

- As C++ allows create hierarchy-related objects, special object oriented libraries can be built. These libraries can be used later by other programmers.
- C++ is widely used to develop mobile/PC games and other graphic-based applications.
- It eases the evaluation of the toughest calculation.
- It maps real-world problems easily.
- It is convenient to add or implement new feature easily in C++ and hence makes it easy to expand and maintain a program.
- Operating systems, such as Windows XP, can be designed in C++.
- Web browsers, such as Mozilla firefox, chrome can be written in C++.
- Databases, such as MySQL, can be designed in C++.
- Compilers, such as Apple C++, Clang C++ and Bloodshed Dev-C++, use this language.
- C++ is used in scripting medical and engineering applications.

HISTORY OF C++:-

Before C++ there is C language, which was mainly developed as a system programming language by **Dennis Ritchie** at **Bell Telephone Laboratories** in 1972. Its main aim was to produce s minimalistic language which can ensure easy compilation of programs, efficient memory access for program storage and develop efficient code.

C language was efficient and flexible, it became obsolete (popular) in 1973. Later in 1983, the American National Standard Institute (ANSI) established a group to set a formal standard for C. They released the **C89 standard** in 1989, which commonly known as **ANSI C**. this version was adopted by the International Organization for Standardization in 1990 and hence became C90. After several years, a new version of C called **C99** by **ISO** in 1999. This new version contains many features which had been implemented in C++.

1979 (Introduction of 'C with Classes')

1983 (Renamed as C++)

1998 (ANSI-ISO approved standard for C++)

2011(C++11 released)

2014(C++14 released)

C++ programming language was developed by **Bjarne Stroustrup** at **Bell laboratories** in 1979 with name of "C with Classes". To make enhancement to C language, it was renamed as C++ in 1983. It runs on a variety of platforms, such as Windows, Mac OS, and different versions of UNIX. Later, in 1998, the ANSI-**I**nternational **O**rganization for **S**tandardization (ISO) approved a standard **C++ ISO/IEC 14882:1998** for C++. In 2011 and 2014, there were two updates for C++ language, namely, **C++11** and **C++14**, respectively, for providing additional functionality to the language.

C++ is considered as a **middle-level** language as it combination of both High-level and low-level language features. Almost all C programs are C++ programs with minor differences which allow to run the C programs under C++ compiler. As the name itself indicates, it is an extended version of C, i.e. superset of C which is derived from C programming language.

PROGRAMMING BASICS:-

It is essential that we first understand its basic concepts, before learning to new language.

C++ Keyword:-

Keywords are predefined reserved words in C++ library. Which cannot use as identifiers in a program. These keywords are used to perform an internal operation and have special meanings.

Keywords Common in C and C++(32 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

The reserved words specific to C++

Reserved Words in C++(30 keywords)				
Asm	Bool	Catch	Class	Const_cast
Delete	Dynamic_cast	Explicit	False	Friend
Inline	Mutable	Namespace	New	Operator
Private	Public	Protected	Reinterpret_cast	Static_cast
Template	This	Throw	True	Try
Typeid	Typename	Using	Virtual	Wchar_t

While using the standard American Standard Code for Information Interchange (ASCII) character set, it not necessary to use these reserved words. But this character set can be used for programming with characters that lacks in C++ and as a more convenient alternative to few C++ operators.

Additional Reserved Words in C++(11 keywords)			
And	Bitand	Compl	Not_eq
Or_eq	Xor_eq	And_eq	Bitor
Not	Or	Xor	

Reserved Words VS Predefined Identifiers:-

Predefined identifiers are that have special use at some position but also use as variables.

Predefined Identifiers Words in C++	
Cin	Cout
Endl	Include
INT_MIN	INT_MAX
Iomanip	Iostream

Main	MAX_RAND
Npos	NULL
Std	String

Data Types:-

There is a need for different types of variables to store information, they do nothing but just reserve memory to stores value at some location in memory, there is no required to know the excact location of variable. Only two thing to know about variable is:

- Name of the variable
- The data_type of it can store

They are stored in the form of ones and zeros but interpreted in different ways based on the data type they stored.

Data type defines the types of data that a variable can store. On defining the data type of variable, OS assigns it some memory and defines what this reserved memory can store.

Data types are classified into two types:-

- Built-in data types.
- User-defined data types.

Built-in data type:-

Type	Keyword	Description
Integer	Int	Specifies the integer values
Character	Char	Stores character types
Floating Point	Float	Represent single-precision floating point value
Double floating point	Double	Represents double-precision floating point value
Boolean	Bool	Stores boolean values (true or false)
Void	Void	Represents the absence of type

These standard data types can be further modified using modifiers, viz signed, unsigned, short and long.

Group	Types names	Size
Character types	**Char**	One byte. Minimum 8 bits
	Char16_t	Not smaller than char. Minimum 16 bits.
	Char32_t	Not smaller than char16_t. Minimum 32 bits.
	Wchar_t	Can denote the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. Minimum 8 bits.
	Signed **short** int	Not smaller than char. Minimum 16 bits.
	Signed **int**	Not smaller than short. Minimum 16 bits.
	Signed **long** int	Not smaller than int. Minimum 32 bits.
	Signed **long long** int	Not smaller than long. Minimum 64 bits.
Integer types (unsigned)	**Unsigned char**	(same size as their signed counterparts)
	Unsigned short int	
	Unsigned int	
	Unsigned long int	
	Unsigned long long int	

Floating-point types	**Float**	4 bytes
	Double	8 bytes
	Long double	10 bytes
Boolean type	**Bool**	1 byte
Void type	**Void**	No storage
Null pointer	**Decltype(nullptr)**	Minimum 4 bytes

User-defined data types:- user-defined data types are the data types defined by user according to their requirements. The user-defined data types can handle multiple data types using single identifier. But this data types need to define before using them. Structure, union, class and enumeration are user defined data types with array, pointer.

Variables:-

These are named storage that program can manipulate. It is named memory allocated by compiler depend on their data types. The name of a variable can contain letters, digits and underscore character. But that name must begin with either a letter or an underscore. As C++ is case-sensitive, upper and lowercase letters are distinct.

Example:-

Int x; float num_1;

OOPs CONCEPT:-

Object Oriented Programming (OOP) approach is introduced to fix the gaps encountered in procedural language. Focus of this approach is on data need to manipulate rather than logic to manipulate data. OOP is actually a programming paradigm (pattern) that is based on the concept of object which represents data. The basic is keeping data and its function together; keep preventing modification of it by outside function.OOP revolves around objects, whose behavior is described by the Classes (instance [example] of object).

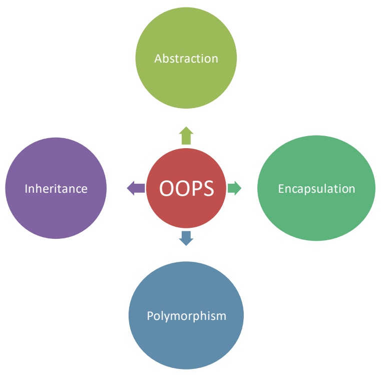
C++ is dependent on objects and classes, and those function on four main concept of OOP as named abstraction, inheritance, encapsulation and polymorphism. **Abstraction** is used to represent only the essential features without showing the internal details, while **encapsulation** is wrapping up of data and function together for perform single operation. **Polymorphism** is the ability to take more than one form; **inheritance** is the method of acquiring the properties of one class into another.

Object:-

It is basic unit of OOP which contains data variable and code (member function) to manipulate it. It represent real world object and qualities of these object are functions.

Object are instances of class and assigned certain space in memory. An object can interact with other objects without knowing their details. Each object has different data variables.

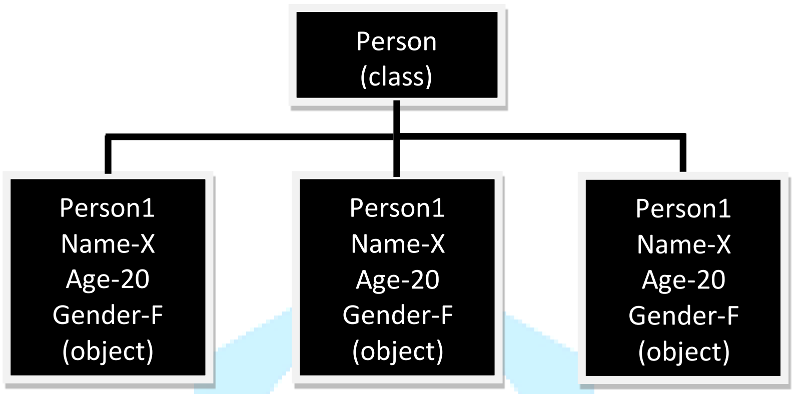
The initialization of objects is through special class member function named **constructor**. Each time object is out of scope, **destructor** (another special class member function) is called for releasing memory hold by object.



Class:-

It is a user-defined data type, which binds the data and its members in a single package. For example consider human is class that contains body parts and performs some actions. Classes are declared with keyword class.

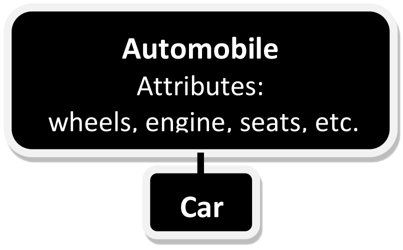
- A class is a user-defined data type.
- Member function of class can be defined inside or outside the defination of the class.
- Classes include data members and member functions.
- Classes in C++ are the base of all the OOPs concept: inheritance, abstraction,encapsulation and polymorphism.
- Multiple objects of a class can be created and these objects have separate copies of data members.



Inheritance:-

It is important concept of OOP concept that allows you to define a new class from existing one, making it easier to create or maintain an application. This helps reusing of funstionality that already defined. This helps in reducing the code size and leads to faster implementation of application.

Existing class from which the new class is inherited is called a **base class** and new class that inherits is called a **derived class**. Derived class can access and use functions of base class. Inheritance supports concept of reusability in OPP, additional features can be added to existing class without altering it.



Consider class automobile, and on it you want to reuse feature of that class in another one named car. Inheritance saves lines of code defining same function in other class.

Data Abstraction:-

It refers to displaying only the required information to the outside world without showing background details. That show essential features of the application and hiding the details.

Data Encapsulation:-

It is process of binding data and functions that use data into a single unit (called classes). It ensures easy manipulation and at same time keeps them safe from misuse by outsiders. It provide important concept of OOP is **data hiding**. You can achieve encapsulation and data hiding by creating classes in program.

Take example of mobile phone. The external layout is includes display screen and keypad buttons to dial numbers. The outer layout called **abstraction**. Inner implementation of phone include how display and buttons are connected to each other that inner layout implementation is referred to as **encapsulation**.

Overloading:-

This concept also part of polymorphism. It is ability to function perform different tasks. When more than one definition is specified for function within scope, it is **function overloading**. In the same way, an operator specified by a different definition in the same scope is referred as **operator overloading**. As addition operator is produce two numbers sum , if applied on two strings, it will concatenate them produce a third string.

Polymorphism:-

The word poly means 'many' and morphism means 'form/state'. The ability to take more than one form is polymorphism. In technical terms. Polymorphism is capability of using an operator or functionin different ways. A single function or an operator can function in many ways on different instances depending upon the usage. It have two type:-
Static polymorphism (compile time):- which function is to be called is decided at compile-time. It can be achieved trough function overloading and operator overloading.
Dynamic polymorphism (run time) :- which function is to be called is resolved at run-time. It can be achieved trough function overriding.

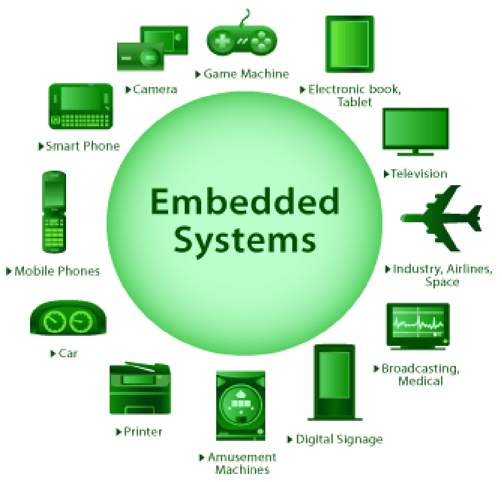
INTRODUCING C++ PROGRAMMING

SCOPE OF C++ PROGRAMMING LANGUAGE:-

C++ is a general-purpose programming language, which is efficient and fast in performing input/output operations on different software applications. It is widely used to create modern games, operating system, browsers, large-scale software infrastructure and in other areas.
C++ used by major companies:-
Adobe:- To develop product such as Photoshop, Illustrator and inDesign.
Amazon:- uses in its e-commerce site.
Autodesk:- uses for Computer-Aided Design.(AutoCAD).
Facebook(Meta):- Most of backend services such as ads, feeds and search are written in C++.

Also object oriented programming is also employed in various sectors and fields. Almost all major companies are using C++ language for developing Pc games, operating system, desktop applications and more.

C++ was developed to resolve the limitations which were encountered in earlier C programming language. Enabling the feature of abstraction, C++ is very reliable when it comes to performance, memory size and specific data structure. It can also communicate with Hardware; therefore it is often used with driver development and embedded systems.



C++ VERSIONS:-

Based upon requirement and latest advancements in technology, several versions of C++ are introduced. With the addition of new features and improved performance in each version, every new C++ version helps in the generation of more efficient programs.

ANSI C++ Committee was founded in 1990 and ISO C++ Committee was founded a year later (1991). Standard Template Library (STL) was implemented in 1992. There is constant addition of several libraries in every new version, starting from the 3rd edition (C++98) in 1998.
Different versions of C++; which are standardized by ISO.

Sr. No	Version	C++ Standard	Years
1	C++98	ISO/IEC 14882:1998	1998
2	C++03	ISO/IEC 14882:2003	2003
3	C++11	ISO/IEC 14882:2011	2011
4	C++14	ISO/IEC 14882:2014	2014
5	C++17	ISO/IEC 14882:2017	2017
6	C++20	ISO/IEC 14882:2020	2020

FEATURES OF C++:-

It the multi paradigm, general purpose, free forms, cross-platform, statically typed programming language, which works on the concept of classes and objects. It can use writing programs in different styles, which can work more efficiently than any other language, such as

FORTRAN, C, Smalltalks any language. Each style of program need to maintain the run-time and should have space efficiency, as C++ supports both.

- It is also called statically typed language because the code will be type checked before it is executed.
- You can maintain and expand C++ program easily. Whenever you need to include a new feature, you can easily add to the existing structure of an object.
- The complex problems will be solved easily as it divides those complex problems into smaller sets by using objects.
- For providing lots of graphic effects to your 3D games, C++ is a scalable language (a language that can work efficiently for smaller as well as complex tasks and can grow as per need).
- It is often used for the development of editors, compilers, databases, communication systems and complex real-life applications system.
- It is static-typed programming language, i.e. it easily handles the errors and bugs before executing the programs and it provides better performance than any other high-level languages such as Java and Python.
- C++ includes numerous compilers that run on different platforms and its standard library used in the code will run on many platforms without any changes.
- C++ is an object oriented and general programming language that provides the following **four pillars**:-
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

USES OF C++:-

The uses of this language are very vast. It is used in almost every organisation directly or indirectly.

- It is widely used in the development of modern games, operating systems, desktop applications, etc, therefore, it is often known as an irreplaceable programming language.
- It is used by programmers in very corner of the world in various application domains.
- Compared to other programming languages, it provides performance that allows writing efficient code with high abstraction level.
- Bigger companies such as Facebook, Amazon and adobe need C++ developers to help in optimizing their work on their products.
- It is also used on popular mobile platforms, such as Android and iOS as well as desktop platforms, such as Windows and Mac OS X.

BENEFITS OF C++:-

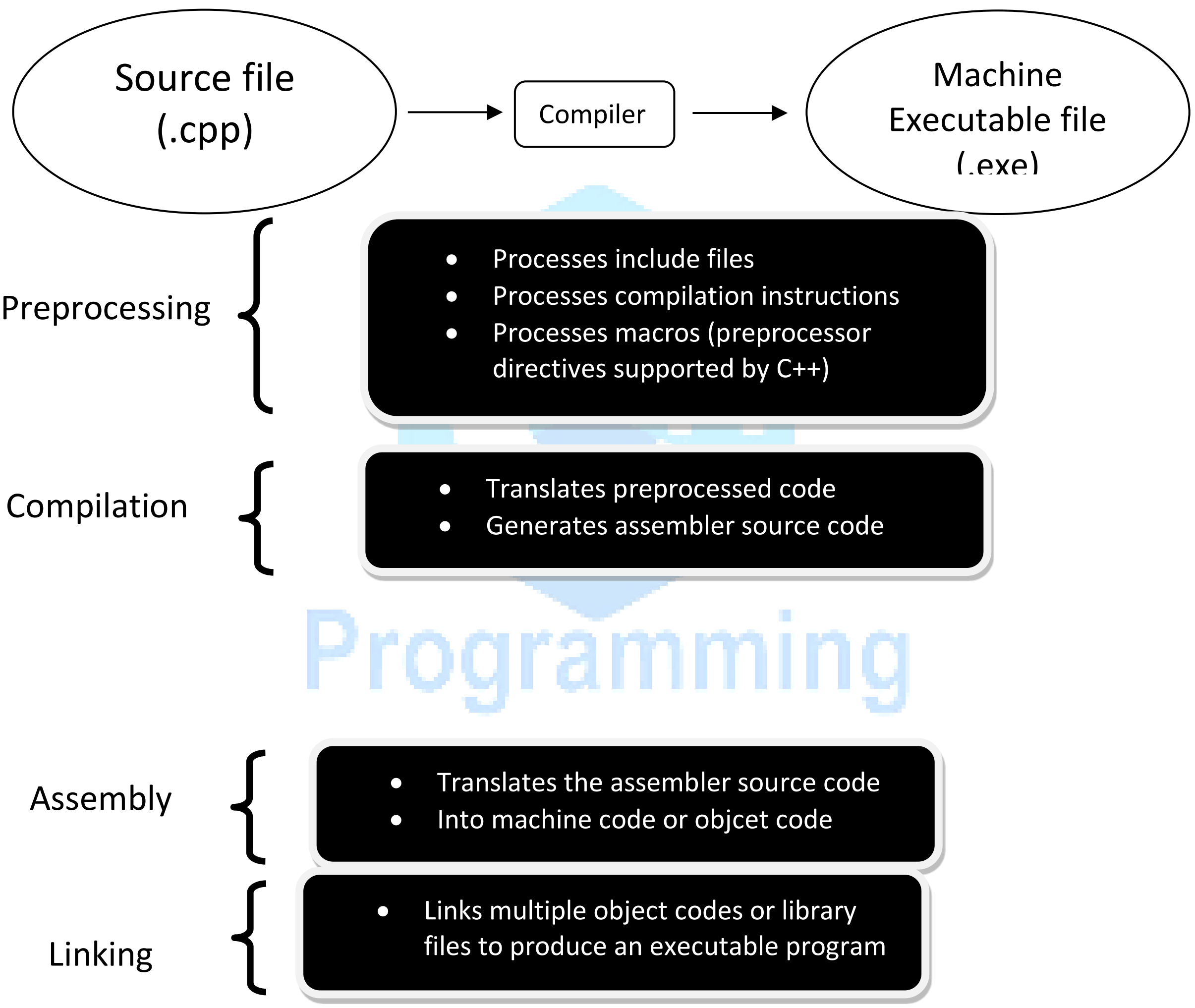
- The objects can be reused as building blocks for other programs.
- By using its code on functions, you will have control over the speed and resource usage.
- It is the most versatile language which can be used in the development of operating systems, database engines, embedded systems, desktop applications, web development areas, etc.
- It can be used as multi-paradigm programming language that provides different types of programming styles to solve every task that fits your use case.

- It is very flexible language that allows static and dynamic type checking either at compile or run time. Most of type checking is static type checking (verifying the program's type safety before execution).
- It is a compiled language that compiles the code directly to machine code.

C++ PROGRAM:-

A C++ program is text that contains lines of code as per C++ programming rules; this file is referred as **source file** with **.cpp** extension.

This source file need to convert into machine code, tool that do it, is **compiler**. The executable generated file after compiling is known as **machine executable program**, with **.exe** extension.


Source file (.cpp)
Compiler
Machine Executable file (.exe)
Preprocessing
Processes include files
Processes compilation instructions
Processes macros (preprocessor directives supported by C++)
Compilation
Translates preprocessed code
Generates assembler source code
Assembly
Translates the assembler source code
Into machine code or objcet code
Linking
Links multiple object codes or library files to produce an executable program

ENVIRONMENT SETUP FOR C++ PROGRAMMING:-

Editor and **Compiler** need to write code and convert into machine code respectively.

Text Editor:-

A tool for write or type program is called as **text editor.** That code can be written by any text editor as notepad, notepad++, sublime editors or epsilon. Depending on OS, and the files can saved with extension as .cpp, .cp etc.

Compiler

A computer program that converts high-level language into machine understandable low-level language is called **compiler**. By default, most of the compilers will use **.cpp** extension if extension not specify.

Now day many IDE (integrated development environments) available in the market. An IDE is software that provide complete environment (set up) to write, interpret, compile and execute programs. Such as Microsoft Visual studio which contains C++ compiler or download MinGW, GNU Compiler Collection (GCC) compiler, Clang compiler can run both windows and linux Os.

C++ PROGRAM STRUCTURE:-

Program:-

```
#include<iostream>
Using namespace std;
Int main()
{
      Cout<<"Hello World";
       Return 0;
}
```

#include<iostream> :- line with starting # is called directives #include direct compiler to add file iostream in program and "< >" show included file is system in-built file.

Using namespace std: - it indicates that we are using standard namespace otherwise we need to declare namespace is standard or not.

Int main() :- "main" is describe it is main function with "()"(parenthesis) it indicates it is function. And "int" is return type of function. Compiler starts execution of program from this function.

{}:- These curly brackets after main function indicates body of it. Whatever inside those brackets are contents of main function.

Cout<<: - it is an output stream object that display output on screen.

"Hello World": - it is output string inside double quotes that will display on screen

Return 0: - with return it will return 0 to compiler that indicates successful execution of program.

; :- that indicates termination of statements.

WORKING WITH TOKENS, EXPRESSIONS AND CONTROL STRUCTURES IN C++

TOKENS:-

A **token** is the smallest element of C++ program, it represents the basic building blocks of C++ program constituted (made) by one or more characters, that usually separated by a white space.

Types of tokens used by C++:-

- Keyword
- Identifiers
- Constant
- Strings
- Operators

Keywords:-

These are predefined reserved words in the C++ library. Which cannot be used as identifiers in program. These keywords are divided into several groups and used for special purpose or predefined tasks.

Identifiers:-

It is a symbol used to identify a program element in the code. It is fundamental requirement of any language and used to represent many elements.

Identifier used for:-

- Define an object or variable name
- Define class, structure or union name
- Define function or class-member function name
- Define the label name, macro name or parameter name
- Specify the enumerated type name
- Denote member of a class, structure, union or enumeration

Rules for using identifiers:-

- The first character of an identifier must be an alphabet or an underscore, while remaining could be digits or letters.
- Keywords should not be used as identifiers.
- An identifiers should not begin with a digit.
- As C++ is case-sensitive language, the upper-case and lower-case letters are different from each other.
- Except underscore (_), no other special character can be used in an identifier.

Examples:-

Valid identifiers :- World, hi5, _name, fname_lname, _5, 123namehi

Invalid identifiers :- 5, int, fname-lname

Constant:-

It value in programming that cannot be changed during execution. It appears in the program as fixed value.

Define constant as

1. Using **const** keyword
2. Using **#define** pre-processor.

Example:-

Const int myval=50;

Types of Constants:-

- Integer constants
- Character constants
- Floating constants
- Strings constants

Integer constant:-

These integers that do not contain any decimal or fractional part. constant is local to file where it created.

Demical (base 10):- it represents a sequence of digits that should not begin with 0(zero).

Example:- 100, -202,etc.

Octal (base 8):- it represents a sequence of digits that should begin with 0(zero).

Example:- 067, 010,etc.

Hexadecimal (base 16):- it represents sequence of numbers that begin with 0x/0X.

Example:- 0xD, 0XFA,etc.

Character constant:-

It contain one or more characters which should be enclosed within single quotes. Like 'd','1' etc. there are few character which cannot be typed directly from the keyboard. These are called as non-graphic characters, which can be represents using escape sequences. These are single character provide with backslash. For example "\n" for take new line in program.

Eacape sequence	Task	Eacape sequence	Task
\n	New line	\r	Carriage return
\t	Horizontal tab	\f	Form feed
\v	Vertical tab	\a	Alert
\b	backspace	\\	Backslash
\?	Question mark	\0	The null character
\'	Single quote	\ooo	Octal
\"	Double quotes	\xhhh	Hexadecimal

Floating constants:-

It is often known as real constants, can br written in fractional or exponential form. They are number with fractional parts that specify the parameters in integers and real numbers.

A floating constant in fractional form must have at least one digit and a decimal point. It can be positive or negative.

Example:- +521.20, 376.0, -41.62.

Real constant in exponential form includes two parts: **mantissa** it can be real or integer number that before **e** or **E** and after it is **exponent** represents only integer value.
Example:- 52E12, 8.33e29.
There are three types of floating constant:-

- Float -> 1.234e3 or 1.234E3
- Double -> 2.34e4 or 2.34E3
- Long double -> 0.123e5 or 0.123E5

Strings constants:-
It is group of multiple characters within double quotes. Example “good” strings are automatically appended with a special character ‘\0’ for indicates the end of string. Size of string is increased by one character.

Operators:-

These are specified by predefined symbols which carry out certain computations, and the participants of an operation are called operands. An operand may be either a constant or a variable. Based on the number of operands required for the operation operators divided into three parts.
Unary operators:- it operates on single operand. -(-5),+(-5)
Binary operators:- it operates on two operands is said to be a binary operators.
Ternary operators:- it operates on three operands. A typical operator is conditional operator.
Syntax = <condition>?<true-case-code>:<false-case-code>;

Arithmatic operator:-
These are basic and binary operators.

S.No	Operator	Meaning
1	+	Addition
2	-	Subtraction
3	*	Multiplication
4	/	Division
5	%	Modulus

Relational operator:-
These are used for comparing two values which result either **true(1)** or **false(0)**

S.no	Operator	Meaning
1	>	Greater than
2	>=	Greater than or equal to
3	<	Less than
4	<=	Less than or equal to
5	==	Equal to
6	!=	Not equal to

Logical operators:-

It combines two conditions and the result of logical combinations is give one values as **true(1)** or **false(0)**

S.no	Operator	Meaning	Example	Result
1	&&	AND	Val1&&val2	Returns true if both are true
2	||	OR	Val1||val2	Returns true if both or either of val1 or val2 are true
3	!	NOT	!(val1)	Returns opposite value of the given expression

Assignment operator:-

It is specified by = which assign value from right to left side operands.

Operator	Meaning
+=	X+=y is equivalent to x=x+y
-=	X-=y is equivalent to x=x-y
*=	X*=y is equivalent to x=x*y
/=	X/=y is equivalent to x=x/y
%=	X%=y is equivalent to x=x%y
=	X=y will assign value of y to x

Increment and Decrement Operators:-

The increment operator (++) is used to increase value of variable by 1. **Post-increment operator** will return value first then increment variable's value letter. **Pre-increment operator** will increment value then return it.

The decrement operator (--) is used to decrease value of variable by 1. **Post-decrement operator** will return value first then decrement variable's value letter. **Pre- decrement operator** will decrement value then return it.

Bitwise operators:-

These operators works on bits and convert them into number. They do not work on floating point number.

Truth table of bitwise operators					
A	B	A&B	A|B	A^B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	1	1	1	0	0
1	0	0	1	1	0

Operator	Meaning	Function
&	Bitwise AND	Copies a bit if available in both operands.
|	Bitwise OR	Copies a bit if available in any one of the operands.
^	Bitwise XOR	Copies the bit if it is available in one operand but not in both.
~	Bitwise complement	Flips the bit values
<<	Bitwise left shift	The value of the left operand is moved left by the number of bits specified by the right operand
>>	Bitwise right shift	The value of the left operand is moved right by the number of bits specified by the right operand.

Precedence of operators:-

Operators in C++ have precedence for evaluate expression in program

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

EXPRESSIONS:-

It is a sequence of operators and operands, which evaluates the computation to produce result.

Constant expressions:-

The expressions that contain only constant values are known as constant expressions.for example : 5/32+16.

Integral expressions:-

The **integral** expressions represent an integer values as output after performing all types of typecasting.
Example : a, 5+a*b, 35 + int(15.00)

Float expressions:-

It represent a floating-point values as output after performing all types of conversions.
Example : 5.46, a+b, 8+float(3).

Pointer expressions:-

It provide the address values as output. Example &a, & refer address.

Relational expressions:-

It also called **Boolean** expressions which produce a boolean type result as true or false.
Example : a+b>250, a+b==x-y,a<=b+c.

Logical expressions:-

It gives boolean type result after combining two or more relational expressions.
Example : a==8 && b==8, a>b || x<=y

Bitwise expressions:-

It control data at bit level are known as **bitwise** expressions. Example : x&y, x^y.

Control structures:-

Types of control structures:- 1. Conditional or selection 2. Iteraration (loops) 3. Jump statements.

Conditional or selection structures:-

If statement:-

It is control statement which executes its associated set of code only if the condition is true.

Syntax:-

```
If(expression)
{
  Statements.
}
```

If…else statement:-

It is conditional statement which is used for executing a set of codes if condition true otherwise execute “else” part.

Syntax:-

```
If(expression)
{
  Statements;
}
Else
{
  Statements;
}
```

Switch and case statement:-

It is used to select among multiple alternatives by comparing the value of an expression with multiple constants. Constants are listed using “case” label along with a “break” statement to exit from execution if there is not matching condition, then default statement code executed.

Syntax:-

```
Switch(expression)
{
 Case constant 1: statement 1;
             Break;
.
.
  Case constant n: statement 2;
             Break;
  Default : statement;
}
```

Note:- “switch”,”case” ,“break” and “default” are keywords.

Iteration structures:-

These structures repeat a set of statements certain number of times until the given condition becomes true. Repetation stops only when the condition becomes false, the iteration structures can also be called as looping statements.

Types of iteration structures:-

1. For loop
2. While loop
3. Do while loop

For loop:-

It is an iteration statement that executes a set of code repeatedly with the given initial value, condition and increment value.

Syntax:-

```
For(initialization; condition; increment)
{
  Statements;
}
```

While loop:-

It is an iteration statement that includes a set of code which executes only when the condition is true. It is executed only when the condition is true, it is known as "**entry controlled loop**".

Syntax:-

```
While(condition)
{
  Statements;
}
```

Do while loop:-

It is first execute set of code first then checks the condition, in this loop, condition is checked only after executing the loop at least once; so it is known as **Exit controlled loop.**

Synatx:-

```
Do
{
  Statement;
}
While(condition);
```

Jump statements:-

These statements determine the movement of program control from one place to another.

1. Return
2. Goto
3. Break
4. Continue
5. Exit

Return:-

It stops the execution of the code and returns the control to calling function. It may or may not return a value. Non-void functions must return a value.

Syntax:-

Return expression;

Goto:-

It is used as an unconditional branching and can transfer the program control anywhere in anywhere in the function. The goto statement is specified by a label called identifier.

Use of goto statement makes the program logically complex and jumbled.

Syntax:-

```
Goto label;
…  .  .  .  .  .
…  .  .  .  .  .
…  .  .  .  .  .
Label:  .  .  .  .  .
…  .  .  .  .  .
…  .  .  .  .  .
```

Break:-

It exit from switch or a loop immediately.

Syntax:-

Break;

Continue:-

It is used to skip over part of the code within the loop body. **Continue** statement terminates the current iteration of the loop.

Syntax:-

Continue;

Exit:-

It is a built-in function that terminates the program itself. A return code "0" exits program without error, return code "1" indicates the unsuccessful termination of program.

Syntax:-

Exit(int return_code);

MANAGING INPUT AND OUTPUT DATA

STREAMS AND I/O OPERATIONS:-

In C++, the I/O operations in standard input/output devices are performed by streams, which sequence of bytes that flow inside or outside the programs.

When it flow input device to the main memory, it is referred as an **input operation**. When it flows main memory to output device, it is called an **output operation**.

The **stream** is actually a basic data type or an object used to carry out input/ output operations in C++.
Iostream is a standard library including two basic stream types:-

Istream cin: it is a predefined input stream variable, which is connected to the keyboard by default.

Ostream cout: it is a predefined output stream variable, which is connected to the console by default.

The concept of streams is similar to concept of class, for example **Istream cin** is an input stream, where cin is an object and istream is the type name for the class. So cin is an object of the istream class.

Main highlights:-

- The I/O operations in C++ are type safe, which means input/output operations are already defined for each type. The compiler will generate an error if no I/O operation is defined for a particular type.
- Based on byte streams, the I/O operations are device independent, which means the same set of operations can be utilized on different input/output devices.
- The data (input value) in the input stream comes from the keyboard or any other device.
- The data in the output stream can go to screen or any other output device.
- The initial value from an input device uses input stream as an interface to flow data to a program, which in turn uses output stream as an interface to flow data from the program to an output device.
- The input value uses the istream class to provide input from keyboard and the output value uses the ostream class to display the result on the sreen.

I/O LIBRARY HEADER FILES:-

The standard library of C++ holds files that contain standard functions used by the program. These files are referred to as header files, which define function prototype for library functions. Besides , they also define constants and data types used with library functions.

Header file	Description
<iostream>	It provides the basic I/O stream functionality by using the cin and cout objects.
<iomanip>	It defines parametric manipulators or helper function to control the format of I/O on files.
<fstream>	It provides input/output file streams that handle files and I/O operations on files.
<bitset>	It provides class and function to specify fixed-size bit array.
<exception>	It defines the pre-defined exception classes, which handle unexpected errors.
<complex>	It provides the class to deal with complex numbers and their associated operations.
<algorithm>	It provides functions to deal with standard algorithms.
<functional>	It provides objects and classes to deal with function.
<ios>	It defines classes and functions for the operation of iostreams.
<istream>	It specifies the input streams.
<iterator>	It is used for traversing a sequence.
<memory>	It determines memory allocators.
<locale>	It defines features to make program portable globally.
<map>	It defines classes, operators and functions for map and multimap.
<ostream>	It provides output I/O streams.
<sstream>	It defines the string as source or sink.
<numeric>	It specifies numeric operations.
<new>	It represents memory allocation and de-allocation.
<set>	It represents a container with unique keys.
<queue>	It adds sequences at the head and removes them at the tail.
<stack>	It adds and removes the sequences at the head.
<typeinfo>	It specifies run-time type information.
<string>	It represents a sequence of characters.
<streambuf>	It specifies the buffer classes for iostreams
<stdexcept>	It provides the extra standard exception class.
<utility>	It represents the comparison operators.

STANDARD STREAMS:-

Stream is a sequence of bytes, which acts as an interface to carry out input and output operations. A pre-defined stream provided by the environment of a computer program is referred to as a standard stream.

The standard output stream:-

It refers to the destination stream that receives output from a program. An object of the ostream class which signifies the output on the display screen is the **cout**. It is used with stream insertion operator "<<". Written as : cout<<……..;

It is predefined object, i.e. an instance of ostream class, which is believed to be connected to the standard output device. The << operator is used to display the data items of built-in types, such as integer, float, double, strings and pointer values. A new line at end of the line is provided by using **endl**.

The standard input stream:-

It refers to the source stream that provides data to the program. An example of the istream class which represents an input from the keyboard is **cin**. It used with the **stream extraction operator** as “>>”. Written as : cin>>…….;

It is also posible use the stream extraction operator >> more than once in single statement. Written as : cin>>a>>b;

The cin is a predefined object, i.e. an instance of the istream class, which is believed to be connected to the standard input device.

The standard error stream:-

If you find any error while executing a program, then it will be represented by using the built-in object **cerr**. Which is instance of ostream class. This object is belived to be attached with the standard error device (display screen). The **cerr** object is un-buffered and hence displays the output immediately each time stream is inserted.

It is used with **stream insertion operator** as “<<”.
Written as : cerr<< or cerr<<”error”<<” occurred”; (for more than one insertion operator.)

The standard log stream:-

It controls the error messages passed from buffer to standard error device. You can specify it using built in object **clog** which is instance of ostream class. This object is believed to be attached with the standard error device, which is usually a display screen. The object **clog** is buffered and hence whenever there is an insertion to clog. It would hold its output in a buffer until the buffer is filled or cleared. It is used with stream insertion operator “<<”. Written as : clog<<.

ARRANGING THE SAME DATA SYSTEMATICALLY: ARRAYS

WHAT IS AN ARRAY:-

A series of elements of the same type are known as an **array**. Which can store large amount of data in a structured manner in contiguous memory location.

Consider the array name a with length of 7, it will be named as a[7].

A[index]	Elements or value	Actual address
A[0]	100	1100
A[1]	110	1104
A[2]	115	1108
A[3]	123	1112
A[4]	234	1116
A[5]	245	1120
A[6]	345	1124

Array's features:-

- It is identical in data type, size, shape and purpose.
- It stores elements in contiguous memory locations.
- It is accessed by using an integer index which is referenced by adding an index to a unique identifier.

It available in one or multi dimensional form.

DECLARING ARRAYS:-

Syntax:-

Data_type array_name [size];

Data type:- it specifies data type of each element in array.
Array name:- it is name of array.
Size:- it is integer constant that specifies the number of elements in array will store.
Above declaration is for one dimensional array.
Example:- int a[4];

INITIALISING ARRAYS:-

it is important if values need to be specific. If array is not initialized then it takes an garbage value.
Array can be initialized at declaration time:- Int score[5]={45,50,60,70,80};
If we initialize an array with fewer values than size it will take garbage at remaining index.

ACCESSING ARRAY ELEMENTS:-

An individual array element can be accessed using their index. As int val = a[3];

PASSING ARRAY TO A FUNCTION:-

An entire array cannot pass to function for it needs to provide only array name as an argument to function by declaring function formal parameters by three ways:-

- In form of a pointer:-
  Return_type function_name (data_type *my_parameters)
- In form of an array with size:-
  Return_type function_name (data_type my_parameters[size])
- In form of an array without size:-
  Return_type function_name (data_type my_parameters[])

RETURN ARRAY FROM A FUNCTION:-

In C++, cannot return the complete array as an argument from a function. Though, it is possible to return a pointer to an array by determining the array's name without an index.

A single-dimension array can return from a function by declaring a function that returns a pointer as :-
Int * function_name()

MULTI-DIMENSIONAL ARRAY:-

It can be two or three dimensional array. It store a data in **row and column format** which generally known as matrix. Simply describe a **multi-dimensional array** as **array of array.**
Example:- int a[2][3].

	coulmn 0	coulmn 1	column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]

2d array can be initialized by directly assigning the value while writing the code or by using two for loops. **Example:-** int a[2][3]={1,2,3},{4,5,6};

Initialisation of a three-dimensional array:-
It is an array of arrays of arrays. They use three loops to initialise and show the value of elements. Innermost make 1d array, middle make 2d array and outermost make 3d array.
Example:- int a[i][j][k];

Character array:-

Apart from single and multi dimensional arrays, character arrays are also avilable. A series of characters is known as **character array**. Declaration of it is same as integer or float array.
Syntax:- data_type array_name [size]; **Example:-** char student [10];.
Character array is also called **string**.

CLASSES AND OBJECTS IN C++

CLASSES AND OBJECTS:-

A **class** is specified as a **blueprint** that describes the behaviors/states for an object. It is user-defined data type, which has its own data members and member functions. The member functions are the functions that can access the objects and data members of only that class in which they are defined.

Defining class:-

- A class name should start with an uppercase letter. The first letter of each word should be in uppercase if the name of class comprises more than one word.
- Member functions of the class can be defined inside or outside of the class.
- All the features of OOPs revolve around the classes in C++.
- Objects of a class hold separate copies of data members. We can create as many objects of class as we need.

Main parts of class:-

- **Member variable:-** include data stored in variable.
- **Member function:-** specifies the interaction between the data stored in the variable.
- **Constructor:-** determines how to construct an instance.
- **Destructor:-** specifies how to clean up after an instance.

DEFINING AND DECLARING CLASS:-

It means representing the lists of data members and functions available in a class. During declaration, the type and scope of all the members of the class are also described. Defining class means implementing the functions.

Declaration of a class specifies the class name, object of the class and determines the operations that can be performed on such objects.

Declaration of class is implemented by definition of class that contains actual codes and its execution done by definition. And user depends on class declaration.

A class can be defined with keyword **class** before class name and body of class, which enclosed within a pair of curly braces with semicolon "{….};"define class as follow:-

Syntax:-

```
Class class_name
{
      //declaration statements here
};
```

Example:-

```
Class rectangle
{
   Public:
         Float width, height, area;
         Displayarea();
}; //keyword public specifies that members of class can be accessed from outside class.
```

ACCESSING DATA MEMBERS:-

The scope, within which a data member can be accessed, is referred as accessibility of that data member. The access control of a data member is defined by access specifiers. With defining class we also define data members along with their access specifiers with three types:-

1. Public
2. Private
3. Protected

A class method used to read data members is called an **accessor** and method used to modify data members is called a **mutator** method. Accessor method termed as getter and mutator method termed as setter function. Data members structure in program as:-

```
Class class_name{
Public:
        //code of public members here
Protected:
        //code of protected members here
Private:
        //code of private members here
};
```

Accessing public data members:-

The **public data** members are accessible from anywhere inside or outside class within a program. To access these members needs to use direct member access operator "." with object of class.

Object_name.data_member

Accessing private data members:-

Data members declared as **private** cannot be accessed directly or even viewed from outside the class. With getter and setter functions can retrieve and set the values of the private data members.

Accessing protected data members:-

It is quite similar to a private member, only difference is it can be accessed in child classes which are known as derived classes.

DEFINING MEMBER FUNCTIONS:-

It available in two types namely selector member function it can only read data, and modifier member functions can modify the data. A function with small code can define inside class while others define outside class.

Defining member functions inside the class:-

It defined inside the class are usually small and are by default treated as inline functions. Whenever these functions are called in program, the compiler replaces with the small code of inline function.

Syntax:-

```
Class class_name{
Return_type  function_name(parameters)
{
    Statements;
}
};
```

Defining member functions outside the class:-

The functions with larger code are usually defined outside of class. The member functions can be defined outside the class using the scope resolution operator (::).

Syntax:-

```
Class class1_name{
Return_type function1_name(parameters);
};

Return_type class1_name:: function1_name(arguments)
{
    Statements;
}
```

Constructors:-

It is a special member function of a class which is executed whenever we create new object of class. It contains same name as the class and it does not have any return type. For setting initial values for certain member variables, it is very convenient to use constructors.

Some important points of constructors:-

- A constructor is called immediately after object creation.
- It has the same name as that of class.
- A constructor is called only once during the lifetime of an object.
- A constructor does not return any value.
- If the user does not specify a constructor, the C++ compiler generates a default constructor which will accept no parameter and will have an empty body.

Use of constructor:-

The main use of a constructor is to initialise data members of an object with user defined values in place of default values. If you do not use constructor, the data members of an object are initialized with 0. Therefore, constructors are used to provide some initial values to the data members of an object.
Constructors have three type:-
Default constructor:- if there is no constructor in program, then the compiler provides a default constructor.
Parameterized constructor:- this constructor takes the parameters.
Copy constructor:- it copies the member values of one object to another object.

NESTED CLASSES:-

A class declared in another class is called a **nested class**. outer class of nested class is called enclosing class. nested class act as a member of outer class and has the same access rights as other members of the enclosing (outer) class. it can be used within the scope of the class in which it is defined. The members of a nested class have no special access to the members of an outer class.
Syntax:-

```
Class outer_class {
    Class inner_class {
         Return_type   function_name (parameters)
          {
               Statements;
           }
    };
};

Int main ()
{
      Outer_class::inner_class   obj_name;
      Obj_name.function_name ();
}
```

LOCAL CLASSES:-

A class declared within a function, is referred to as **local class**. global and static both variable can be accessed using local classes. You need to access global variable using scope resolution operator (: :) in the case where global and class member variable with similar name. there must be no static data member and static functions in local classes. Also the method of local class should not be defined outside the class.
Syntax:-

```
Return_type  functiion_name (arguments)
{
    Class class_name{  ……….  };
}
```

FRIEND FUNCTION:-

A special function that can access the private members of a class as if it is a friend of that class is called a **friend function**. A keyword friend is appended before names of function for specify that is friend of class and not member of class.

Some points about friend function:-

- It can be declared in any access mode.
- It can be defined inside or outside class.
- Scope resolution operator is not used while defining it outside the class because it is not a member function.
- It becomes inline function if it is defined inside the class.
- It can access private data members by using the object name and a member access operator (dot or arrow operator)
- It should be used for limited purpose in a program as it lessens the value of encapsulation.

Syntax:-

```
Class class1_name{
        Private:
                Data_type  var_name;
                Friend  return_type  function_name(class1_name);
};
Return_type  function_name(class1_name obj_name)
{
        Obj_name.var_name=value;
}
Int main()
{
        Class1_name  obj1_name;
        Cout<<function_name(obj1_name);
}
```

IMPLEMENTING OOP CONCEPTS IN C++

Some important features of OOP:-

- It mainly provides importance to data rather than procedure.
- The programs are divided into objects which can be characterised using data structures (array, union, structures, etc).
- The data of an object can be combined together with functions in the data structure.
- There is no possibility of directly accessing the private data by external function as the data is hidden. Through it is possible by using pointers and friend function.
- Objects may communicate with each other trough functions.
- You can easily add new data and functions whenever necessary.
- It follows the bottom-up approach in program designing.
- The complex problems will be solved easily as it divides those complex problems into smaller sets by using objects.

OBJECT:-

It is the fundamental unit of OOP which includes data variables (states) and member functions (behaviours). Objects are instances of class and they can interact with other objects without knowing their details.

Special class function called **constructors** are used for initializing objects. Releasing the memory reserved by the object, another special class member function called **destructor** is called every time the object goes out of scope.

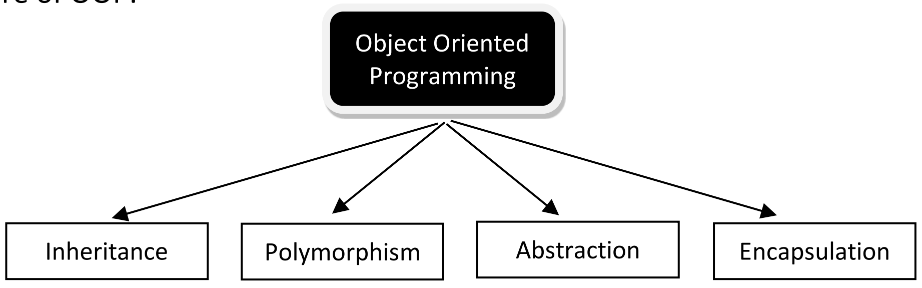
CLASS:-

It is specified as a blueprint that decribes the behavious/states for an object. All the features of OOP revolve around classes in C++. It is a user-defined data type.

The member functions of class can be defined inside or outside the class defination. Classes consists of data members and member functions, which can be accessed depending on the access specifiers. The objects of a class hold separate copies of data members. It is possible to creater as many objects of a class as required.

ARCHITECTURE OF OOP:-

If classes and objects are the base, certainly there would be some pillers to complete the architecture of OOP.



Abstraction:-

It refers to representing the required information to outside without presenting the background details. i.e. hiding the details of an object and show only essential information that a user can understand.

Data abtraction is a programming technique that relies on the separation of interface and implementation.

In case of OOPs, classes offer high level of **data abtraction** by providing public methods to the outside. These methods allow user to do as many manipulations on object data. When defining a class, we specify both data members and member functions in program. But the member functions and built-in data types are hidden while using the object; this is referred as data abstraction. For example we have predefined classes, namely ostream and istream. We can use their objects - cout and cin without seeing the internal code and function specifications.

Access labels enforce abstraction:-

Access labels/ access specifiers can be used to define the access control rules of the class. While define class we can put these labels to class or class members. It is not mandatory to put access label. You can put no label or multiple access lables in the class. Three keyword as public, private and protected used to define access specifiers.

Public access specifier: - by declaring public access specifier with class, all class member will be available to everyone. Other classes can access the public data members and member functions of a class.

Example:-

```
Class myclass{
    public:      //access specifiers
        Int a;
        Void show();
};
```

Private access specifier: - the class members declared with private access specifiers cannot be accessed by anyone outside that class. All the variables and member function are defined as private by default.

Example:-

```
Class myclass{
    public:      //access specifiers
        Int a;
    private:
        Void show();
};
```

Protected access specifier: - class members defined as protected can be accessed by its subclass but cannot be accessed outside the class.

```
Class myclass{
    protected:  //access specifiers
        Int a;
        Void show();
};
```

Every access level is applicable to the member definition following the label. The effect of a particular level is until the next access label is specified or till the time closing brace of the class is encountered.

- All parts of the program can access members defined with a public label. The public members can be used to define the data abstraction view of class members and member functions.
- The code that uses the class cannot use the members defined with a private label in that class. The private sections hide the implementation of code from outer functions.

Encapsulation:-

It is process of binding together the data members and member functions that manipulate the data, thereby keeping the members safe from outside misuse. It is a process of hiding data and functions into a single unit protecting it from outside. Data encapsulation provides an important OOP concept of **data hiding**. C++ supports the properties of encapsulation and data hiding through the user-defined types called **classes**.

Features:-

- Whenever the implementation of the class changes, the interface remains the same.
- It reduces human errors.
- It makes application maintenance easier.
- It improves the understandability of the application.
- It provides enhanced security.

Inheritance:-

It is concept in OOP that allows us to define a new class from an existing class, thereby making it easier to create and maintain a software application. It provide to reuse code functionality and helps in reducing the code size and ensuring faster implementations.

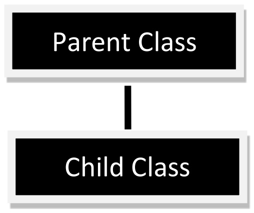
The existing class from which the new class has inherited is called base class, super class or parent class. the new class that inherits the base class is called derived class, sub class or child class. the derived class can access and use all the functions of the base class.

Through inheritance, a programmer is prevented from unnecessary coding.

Types of inheritance:-

- Single inheritance.
- Multiple inheritance.
- Multi-level inheritance.
- Hierarchical inheritance.
- Hybrid(virtual) inheritance.

Single inheritance: only one class can be inherited in single inheritance. This means there will be only one base and one derived class.

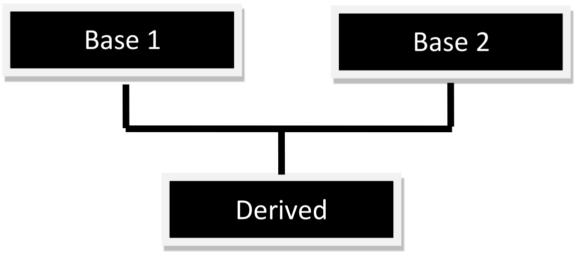


Example:-

```
Class base{
};
Class derived: public base{     //sub class derived from base class.
```

};

Multiple inheritance: when a class is inherited from more than one classes, it is referred to as multiple inheritance. This means there will be one derived class and more than one base class.



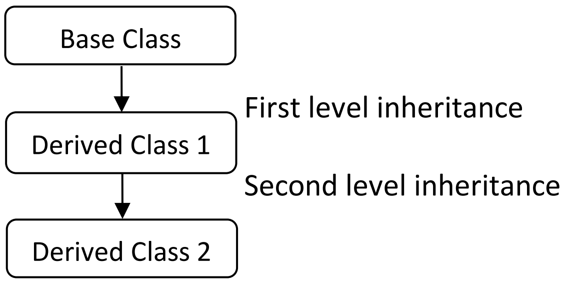
Example:-
```
Class base1{
};
Class base2{
};
Class derived: public base1, public base2{
};
```

Ambiguity resolution in inheritance :-(uncertainty**)**

When functions with same name appear in multiple classes we may face an ambiguity problem in choosing the correct function. This ambiguity can be removed using class resolution operator.

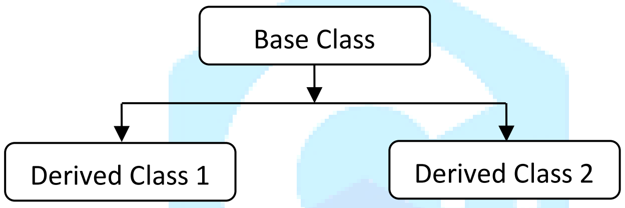
Example:-
```
Class base1{
   Public:
       Void show()
          {
              Cout<<"hello\n";
          }
};
Class base2{
   Public:
       Void show()
          {
              Cout<<"hi\n";
          }
};
Class derived: public base1, public base2{
   Public:
       Void show()
          {
              Base1::show();      //ambiguity resolved.
          }
};
Int main{
    Derived d;
   d.show();
}
```

Multilevel inheritance:- one derived class is inherited from another derived class in multilevel inheritance.



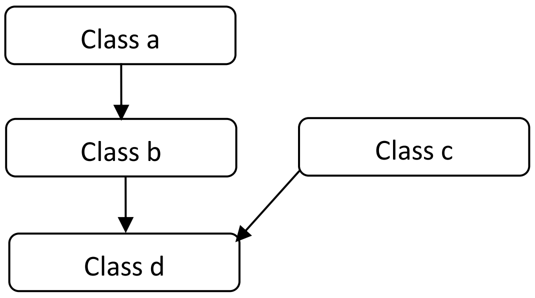
Example:-
```
Class base{
};
Class derived1: public base{
};
Class derived2:public derived1{
};
```

Hierarchical inheritance:- more than one derived class is created from a single class.



Example:-
```
Class base{
};
Class derived1: public base{
};
Class derived2: public base{
};
```

Hybrid inheritance:- when more than type of inheritance are implemented together, it is called hybrid inheritance. It is also called virtual inheritance.

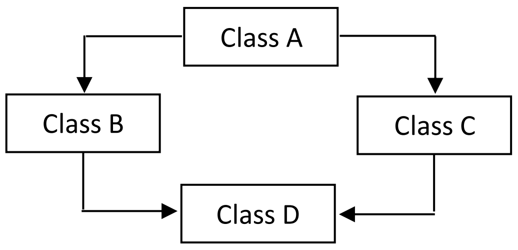


Example:-
```
Class a{
};
Class b:public a{
};
Class c{
};
Class d:public b, public c{
```

};

Virtual base class and the diamond problem:-

There can be situation when the base class is inherited by two derived classes, which are further inherited by a single class.



In the diamond problem, the two B and C classes of a D class have a common class A as base.
In such a problems, all members of A class are inherited twice to D class through class B and C. this may introduce ambiguity (uncertainty), which can be avoided by declaring virtual base class.

We can make a base class virtual by preceding the name of base class with the word **virtual**. It will prevent creation of multiple copies of members of inherited class. Irrespective of number of path available between the virtual base class and derived class, only one copy of the class member is inherited using the concept of virtual base class.

Example:-

```
Class a{
};
Class b: virtual public a{
};
Class c: virtual public a{
};
Class d: public b, public c{
};
```

Polymorphism:-

This word is combination of words "**poly**" which means "**many**" and "**morph**" which means "**forms**", in short, **many forms**. In other words, the ability to take more than one form is refferred to as polymorphism. In technical terms, polymorphism is the capability of using an operator or a function in different ways. A single function or an operator can function in many ways on different instances depending upon the usage.

It can be used in two ways:-

Static (compile time) polymorphism:- it implies an entity existing in different physical forms concurrently i.e. the response to the member function is determined at the time of compilation. The member functions are predefined to behave as per the definition based on the type, number and sequence of arguments. It is also known as static binding or early binding. Static overloading can be implemented using operator overloading and function overloading.

1. **Operator overloading:-** it is technique to define two or more meanings of an existing operator. Example "+" operator use for add two numbers as well as concatenate two strings, "*" used for multiplication as well as "values at operator".

Example:-

```
#include<iostream>
using namespace std;
```

```
class add{
public:
        int a;
        void operator++()               //overload unary operator.
        {
                a=a+1;
        }
        add operator +(add dd)  //overload binary operator.
        {
                add cc;
                cc.a=a+dd.a;
                return cc;
        }
};

int main()
{
        add a,b,c;
        a.a=9;
        b.a=9;
        c=a+b;
        cout<<c.a<<endl;
        ++c;
        cout<<c.a<<endl;
        return 0;
}
```

Function overloading:- this means the same function name exists numerous times in the same class with different types of parameters, different order of parameters or different number of parameter.

Example:-

```
Int add(int a,int b){
        Return a+b;
}
Int add(int a,int b,int c){
        Return a+b+c;
}
double add(double a,double b){
        Return a+b;
}
```

Dynamic (run-time) polymorphism:- it implies an entity changing its form as per the circumstances i.e. choosing the member function during the program execution. When a function exists in multiple forms whose calls are resolved dynamically upon program execution, it is said to exhibit dynamic polymorphism. It is also called dynamic binding or late binding.

While using dynamic polymorphism, it is possible to have same parameter in subclass as in its super classes with the same name.

For example method overriding (defining a method in both parent and child class with similar signature, name or paramters).

Example:-

```
Class base{
 Void play(){
 Cout<<”play game”; }
};
Class der: public base{
Void play(){
Cout<<”play maze”; }
};
```

Virtual function:-

Declaring keyword “**virtual**”, a virtual function is a function which can be defined within a base class and can be redefined in derived class. by making function virtual , we show that static linkage is not required from this function. This means we want the function to be called at runtime for only those objects for which it is called.

Syntax:-

```
Virtual return_type  fucntion_name()
{
…  …
}
```

Example:-

```
#include<iostream>
using namespace std;
class base{
   public:
      virtual void show(){  //virtual fucntion
          cout<<"base class"<<endl;   }
};
class derived:public base{
    public:
       void show(){
          cout<<"derived class"<<endl;    }
};

int main()
{
          base b1;
          derived d1;
          base *ptr;
          ptr=&b1;
          ptr->show();                  //calling base class function
          ptr=&d1;
          ptr->show();                  //calling derived class function
          return 0;
}
```

CONSTRUCTORS AND DESTRUCTORS

CONSTRUCTORS:-

A **constructor** is a special member function of a class which is called automatically whenever you create a new object of a particular class. the name of constructor is exactly the same as name of the class. it does not have any return type. It is used in setting initial values for the data members of class.

Syntax:-

```
Class class_name{
Public:
        Class_name(){       //constructor
…..
}
};
```

These are like normal functions that no need to write any statement to call constructor as normal function. Defining the object of class invokes constructor automatically. Constructors construct the value of data members of class.

They can be defined inside or outside class. the main use of constructor is to initialize the data members with user-defined values instead of default values.

Define constructor outside class:-

```
Class_name :: class_name (parameters)
{
…  …  …
}
```

CHARACTERISTICS OF CONSTRUCTORS:-

Unlike built-data type and derived data-types, data members of class cannot be initialised automatically. Because these are available in different access mode. This prevent access to them, especially in case of private and protected access. Here comes the need of constructor that initialises the data members and efficiently manages the memory.

Characteristics of constructors:-

- Constructors are special member functions of a class which is used to allocate the memory for newly created object and initialise the objects of that class.
- A constructor is called immediately after the object is created.
- It has same name as that of the class.
- A constructor is called only once in the lifetime of an object when the object is created.
- Constructors do not return any value.
- If there is no constructor specified, then a compiler generates a default constructor.
- It is called constructor because it constructs the values of data members of the class.
- Constructors do not have void as return type.
- With the help of constructor, you can create a new object of a particular class and also allow the class to initialise the member variables or allocate storage.

- Constructor results in an error if it not declared as public.
- A constructor can be called explicitly and is not inherited.
- A constructor can take parameters and can be overloaded.
- A constructor cannot be virtual.
- Constructor cannot be a friend function.
- If class contians a constructor, then each object of that class will be initialised before using it.
- The address of a constructor cannot be referenced.

CONSTRUCTOR TYPES:-

This special class functions used for initialising objects, are available in three main types, every one of it has different role.

Types:-

1. **Default constructor**
2. **Parameterised constructor**
3. **Copy constructor**

Default constructor:-

It is the one that can be defined with an empty parameter list. If no constructor is defined in a program, the compiler automatically provides a default constructor that initialises the object's data members with dummy values. The default constructor also reserves the memory for the object. The default constructor also reserves the memory for the object. The default constructor defined by the user can have body but the default constructor defined by the compiler does not have a body.

Syntax:-

```
Class_name(){
… … …
… … …
}
```

Parameterised constructor:-

When parameters are passed to a constructor, they are called parameterised constructors. It takes parameters which provides different values to data members by using values as arguments. It is used in those situations where want the data members of an object to be initialised by the values provided by users at time of object created.

The limitation of default constructor is that the data member of every object are initialised with same values. But parameterised constructor enables you to initialise data members of different objects with different values.

Syntax:-

```
Class_name(argument_list){
… … …
}
```

Copy constructor:-

It copies the member values of one object to a newly created object, i.e. it creates a new object as a copy of an existing object. Copy constructor used for:-

- Initialize one object by using another object of same type
- Copy an object to pass it as an argument to a function
- Copy an object to return it from a function

If the class is not provided with a user-defined constructor, the compiler will automatically supply a copy constructor. The copy constructor defined by compiler do only member-by-member copy operations. Default copy constructors make only shallow copies. The data members are copied irrespective of their types.

Syntax:-

```
Class_name (class_name  &obj)
{
… … …
Class_variable = obj.class_variable
}
```

OVERLOADING CONSTRUCTORS:-

It is similar to overloading function, it will have same name with different number of arguments. Specific constructor will called whose parameters match with arguments. A program with default and parameterized constructor is common example of constructor overloading. Compiler does not provide any default constructor if user defines a constructor explicitly.

Syntax:-

```
Class class_name{
Public:
Class_name(){……}
Class_name(parameters){……}
Class_name(parameters2){……}
};
```

DESTRUCTORS:-

It works opposite to constructor. It is execute whenever class object goes out of scope. When we create an object particular memory is allocated to it. If we will not destroy that object, the memory remains occupied by the unused object. It is also executed whenever the delete expression is applied to a pointer to the object of that class.

Important points on destructors:-

- Destructor name should be exactly the same as the name of the class, but it is preceded by the tilde sign (~).
- Destructor doesn't have any return value.
- It doesn't take any parameters.
- It frees the memory of an object whenever it goes out of scope.
- Destructors cannot be overloaded.
- Destructors can be virtual.

Syntax:-

```
~class_name(){
… … …   }
```

GROUPS OF STATEMENTS: FUNCTIONS

FUNCTIONS:-

A block of code used to perform a set of operation is called a function. Input parameters can also be defined in a function through which arguments can be passed to function also values can be returned from function. Each program contain one main() function and additional functions that have the code to execute related operations in the program.

ADVANTAGES OF FUNCTIONS:-

Functions allows to divide the program in small segments provide several benefits. Typing same set of instructions in a program for one task to be done multiple times can be avoided using functions. Like this there are some advantages of it.

- It can be used to reduce the source program length at various places in a program.
- If you want to modify the code, then you can modify only the function without changing the structure of the program.
- Function allows writing neater(orderly) organization of code and keeps you away from writing the code over and over again.
- It divides the program into logical blocks, thereby making it easy to understand the code.
- Functions can be used by many other programs, which mean a programmer can build on what others have already done instead of building from scratch.
- Functions can avoid writing the same type of code for multiple times.
- It provides top-down modular programming i.e. the main function decides the task to be done in a program, which is further divided into smaller piece of code or function to perform various task of the program.
- You can easily test the individual functions.

DEFINING A FUNCTION:-

By defining a function, we declare the name, return type, parameters and the body of a function. A function definition indicates the program of which instructions or a set of code is to be executed and how and when it should be executed. Therefore it is necessary to define a function before using it in program.

Syntax:-

```
Return_type  function_name(parameters list)          //function header.
{
// body of the function.
}
```

Return-type: - it specifies the type of the data that the function returns, which can be int, char or any other class object. If it does not return any value, the return type is void.

Function-name: - it refers to the actual name of the function. The combination of function name and parameter list defines a function signature.

Parameters: - is followed by function name, which holds the values of arguments passed when a function is invoked. It is often called an actual parameters or argument. The parameter list refers to

the type, order and number of parameters of a function. Parameters are optional, so function may or may not contain parameter list.
Function-body: - it contains a collection of statements that specifies what function does.

FUNCTION DECLARATION:-

Function declaration or prototype informs compiler about existence of a function, the name, and return type and how to call the function. The actual body can define separately. Function declaration differs from function definition in form that it has no body. While defining a function in source file and calling it in another file, function declaration is required.

The compiler compares the function with its prototype whenever a function is called. It also checks whether the parameters are matching or not and show an error if they do not match.
Syntax:-

```
Return_type  function_name(parameter list);         //the parameter's data type must be required.
```

CALLING FUNCTION:-

To execute the statements defined in the function body, you will have to call or invoke the function. When you call a function in a program, the control is transferred to the called function. It performs the specified task, executes the return statements and returns the program control back to its main function.

By using the required parameters along with the function name, you can call a function and store the returned result, if the function returns a value.
Syntax: - function_name(arguments);

FUNCTION ARGUMENTS:-

If arguments are to used by function, the variables accepting the values of the arguments must be declared by the function. These variables are called **formal parameters** of the function. These parameters are written in function header and they are local to function. They get active when function called and inactive when call is over.

The parameters provided with function call are **actual parameters**. These parameters are written in parenthesis when a function called.
Example:-

```
Return_type  function_name(formal parameters)
{
  //statements
}

Function_name(actual parameters)   //function called.
```

Formal parameters are passed in function/method definition. When the method/function is called, actual parameters are passed.
Parameter passing mechanism:-

- Call by value, reference, pointers

Call by value:-

In this method, arguments, the actual parameters to the function. These stored in formal parameters of function. This method changes made to formal parameters does not affect on actual parameters value means original values remain unchanged.

Example:-

```
Void func_name(int a){
 // statements
}

Func_name(value);// calling function.
```

Call by reference:-

In this method, reference of the actual parameters passed to function, which stored reference of actual to formal parameters. It uses the reference of actual parameters inside the function. The changes made in formal parameters affect to actual.

Example:-

```
Void func_name(int &a){
 // statements
}

Func_name(value);// calling function.
```

Call by pointer:-

In this method, address of actual parameters is copy into the formal parameters. It uses the address to access the actual parameters used to call function. In call by pointer, the pointers of actual parameters are passed to the called function. Therefore, function parameters should also be declared as pointer types.

Example:-

```
Void func_name(int *a){
 // statements
}

Func_name(&value);// calling function.
```

Benefits of calling the function using reference over pointers:-

- References cannot be re-assigned while pointers can be reassigned. The assignment must be done at the time of initialization.
- References cannot be null while pointers can be.

RECURSION:-

When function calls itself, it is called as recursion. It can be done by defining a function having a statement that calls the same function in which it is defined. It can be considered similar to an infinite loop. But as every loop has some condition to exit from the recursive function has a **base case** to return the control to the calling function.

Example:-

```
Void func_name(int  a){
 // statements
Func_name(a)
}

Func_name(value);// calling function.
```

IMPLEMENTING STRUCTURES AND UNIONS

Structure:-

It is a collection of variables of different data elements; it is different types of variables under the same name. Data elements will have different types and lengths. Structures encapsulate data members that have different data types. By defining a structure we keep information of particular subject under single unit.

Syntax:-

```
Struct  structure_name
{
    Data_type  member_name1;
    Data_type  member_name2;
…
…
    Data_type  member_nameN;
} object_names;
```

Struct:-it is keyword for declaring a structure.
Structure_name:- it is the structure tag, name of structure type.
Member_name:- it specifies the name of the data member of any type, such as pointers, arrays.
Object_name:- it represents the valid identifiers of objects of structure type.

Defining a structure:-

The **struct** keyword specifies the definition of structure, with structure name. in curly braces member of it specified. After it mentioning the objects or structure variables at end of structure, it is not mandatory.

Declaring a structure:-

After defined structure, structure variable can define as:-

```
        Structure_name  structure_variable;
```

On creating the structure, no memory is allocated. After define the structure variable, required memory will be allocate by compiler.

Initializing a structure:-

It is similar to the initialization of variable of other data types. While initialization, it is important to initialize their value in same order in which they declared in structure.

Syntax:-

```
Structure_name  structure_variable={
                                        Value for member1,
                                        Value for member2,
                                        …
                                        Value for memberN
                                       };
```

ACCESSING THE MEMBERS OF A STRUCTURE:-

Access all the members of a structure by using **dot (.)** operator. To access any data field, place the structure variable name, then dot (.) and then member of structure that want to access.
Syntax:-

```
    Structure_variable.member_name;
```

UNION:-

A union is user-defined data type containing all the members that are stored in same memory address. They use same memory as other object from the list of objects. For holding more than one variable, it uses only a single memory location. Member of union can only access one at time i.e. a union can contain no more than one object from its members list at a given point of time.

Unions and structures are similar; each member of structure has separate location, while union store value of various data types at a single location. When need to store only one value or element, which is one of the several types, using union can stored all values or elements at the same spot. A structure can be used when all the elements or values stored at separate memory location are to be used at once.

The size of the union is equal to the largest (in term of size) member of union. By modifying one member, all the members of a union get affected because they share the same memory space.
Syntax:-

```
Union  structure_name
{
   Data_type  member_name1;
   Data_type  member_name2;
…
…
   Data_type  member_nameN;
} object_names;
```

Union:-it is keyword for declaring a union.
Union_name:- it is the Union tag, name of union type. It should be valid identifier.
Member_name:- it specifies the name of the data member of any type, such as pointers, arrays.
Object_name:- it represents the valid identifiers of objects of union type.

Defining a union:-

Capable of holding only one of its data members at a time, a union is special class type whose way of defining is similar to structure. Keyword union specifies the variables of union type.

Declaring a union:-

Declaring the union variable as:-

```
    Union_name  union_variable;
```

No memory allocated when you create a union. The compiler allocates the required memory only when the union variable is defined.

Initializing a union:-

Union can hold only one member data at time. It is not possible to initialize all member of union together. The union is allocated the memory that the biggest member of union can occupy.
Initializing union by using variable:-
Union_name union_variable;
Union_variable.member1=value;
Union_variable.member2=value;
Union_variable.memberN=value;

Accessing the members of union:-

It is same as accessing structure members by using **dot(.)** operator, which is between union variable and member name.
Union_variable.member_name;

DIFFERENCE BETWEEN STRUCTURE AND UNION:-

Structure and union work more or less in similar way but the main difference that distinguishes them is their storage capacity. The memory space of structure variable is equal to the total size of all members of structure, while memory space of union is equal to size of the largest member.

Structure	Union
It is declared with the keyword “struct”.	It is declared with the keyword “union”.
The memory required to store a variable is the total the size of all members.	The memory required to store a variable is equal to that required by its largest member.
All members can be accessed using dot (.) operator.	You can access only one member at a time.
Each item has its own memory location.	Each item is in a shared memory location.
You can initialize several members at once.	You can initialize only the first member of a union.
Altering the value of a member will not affect other members.	Altering the value of a member will affect other members.

POINTING TO A LOCATION: POINTERS

POINTER:-

Pointer is a variable which stores the address of another variable. It is powerful feature that allows accessing the address of a variable and manipulating its content. Being a powerful programming tool, pointers help to managing the memory efficiently by allowing direct manipulation of the memory. It also helps in enhancing the efficiency and performance of program and facilitates handling of unlimited amounts of data.
Consider,
Var=5;
Address of var=0x22fe4c;
Ptr=address of var.
Address of ptr=0x22fe40;

Declaring a pointer:-

Each variable is assigned a location in the computer's memory and the value of the variable is actually stored at the assigned location.
Reference operator :- (&) gives the address of a variable. It is also called "address of operator" and returns the address of a variable after it.
Dereference operator :- (*) is used to access the value stored at some memory address. It is also called "value at operator" or "indirection operator".
Before start working with pointers, it is important to declare it. It is declared like normal variable with a difference that it is preceded by an *(asterisk) sign. It is declared as:-
Data _type *variable_name;
Data type = base type of pointer.
Variable_name = name of the pointer variable
Since pointers point to a specific address in memory occupied by a variable, they can be used to directly manipulate data stored at that location.

CONSTANT POINTERS VS POINTER TO CONSTANT:-

In some program we may not want to change a value or pointer holding address of the value. For it we can make value or pointer a constant. A const keyword can be placed in two ways within a pointer variable declaration as:-
Pointer to constant: - it declares a pointer to a constant value. The pointer cannot change the value it is pointing to. The constant value is unchangeable but the pointer may be changed to point to different constant value.
Syntax:-
Const <type of pointer> * <name of pointer>;
Constant pointers: - it declares a constant pointer to a value. Here, the address stored by the pointer cannot be changed. The location stored in constant pointer is unchangeable but you can change the value through this pointer.
Syntax:-
<Type of pointer> * const <name of pointer>

BENEFITS OF POINTERS:-

- They access the memory allocated to a variable dynamically.
- You can handle any type of data structure by using pointer.
- Pointers return more than one value to the function.
- Pointers handle arrays and data tables in an efficient ways, and return multiple values from a function through function arguments.
- You can easily search/sort large amount of data.
- You can allocate and de-allocate memory at runtime.
- Pointers are fast in terms of program execution and can handle different types of data structures.
- You can pass the information back and forth between the calling function and the called function.
- They access the address of objects.
- You can resize dynamically allocated memory blocks by using pointers.
- With the help of pointers, the available data can be accessed outside a function.

DRAWBACKS OF POINTERS:-

- Sometimes, using pointers lead to memory leaks.
- You may receive a garbage value if you use a pointer to read an incorrect memory location.
- Pointer issues are hard to debug.

POINTER CONCEPTS:-

Null pointer:-

If a value is not pointing at anything, then it is called a **null value**, and a pointer holding a null value is called a **null pointer**.
Int *ptr=NULL;

Pointer arithmetic:-

A pointer holds an address, which is an integer type value. Therefore, you can perform integer operations on pointers just like numeric values. Four arithmetic operations can perform on pointers:-

Incrementing a pointer using unary operators ++:-

The **++ptr** increment the pointer **ptr** to next element of the array. Assume pointer ptr is integer pointer pointing to location 100 then after ++ptr it will point to 104 is the size of integer is 4 bytes.

Decrementing a pointer using unary operator --:-

The **--ptr** decrement the pointer **ptr** to previous element of the array. Assume pointer ptr is integer pointer pointing to location 104 then after --ptr it will point to 100 is the size of integer is 4 bytes.

Pointer addition:-

(Pointer) + (int) = (pointer)
Ptr + num moves the pointer ptr forward the num element in array.

Pointer subtraction:-
(Pointer)- (pointer) = (int)
Ptr1 – ptr2 returns the number of array elements between the two pointers.

Array of pointers:-

The concept of arrays is similar to that of pointers. Arrays work very much like pointers to their first elements. An array variable holds the address of its first element just as a pointer holds the address of the variable it is pointing to as:-
Int arr[5];
Int *ptr;
Ptr=arr;
Ptr and arr are equivalent and have similar properties. The main difference is ptr can be assigned a different address but arr cannot assign anything else. It will always have integer type of 5 elements.
Also, we can declare an array of pointers in the same way we declare array of user-defined / built-in data type. An array of pointers is a collection of pointer that represents collection of addresses. Every element of an array is a pointer holding the address of variable of same data type.
Declare an array of pointers as:-
Data_type *array_name [size];

Pointer to pointer:-

It is exactly a pointer that holds the address of another pointer. It forms the chain of pointers holding the address of variable. After specify a pointer to a pointer, the first pointer stores the address of the second pointer, which in turn stores the address of the location that contains the actual value.
Declare pointer to pointer as:-
Int var=45;
Int ptr=&var;
Int **ptr2=&ptr;

Passing pointers to functions:-

One of the uses of pointer variable is that it can pass as argument to functions as we pass references to functions. For this, there is need to declare pointer type variables as function arguments.
Syntax:-
Return_type function_name(data_type *var1, data_type *var2);

Return pointers from functions:-

It is similar as return an array from a function. For this declaration of function that returns a pointer is needed as:-
Int *funcptr()
{
.
.
.
}

POINTERS TO CLASS MEMBERS:-

Same ways as pointers to variables and methods, It is possible to have pointers to member variables and class member functions as:-

```
Class demo {
 Public:
    Int x;
};
Int main()
{
Demo ob;
Demo *p;
P=&ob;
Cout<<ob.x;
Cout<<p->x;
}
```

Use of pointer name with arrow -> symbol is for accessing the member functions as well as data members.

FILE MANAGEMENT IN C++

WHY AND HOW TO USE FILE HANDLING? :-

A bunch of bytes kept on some storage device is called a file. File handling concept can be used for various reasons as:-

- You can store the data of a program in a file that can be read later.
- File handling is often used for permanent storage of data in file.
- The transfer of information from one to another computer can be easily done.

Steps for achieve file handling:-

- Naming file.
- Opening file.
- Reading data from file.
- Writing data into file.
- Closing file.

A set of classes defines the file handling method in the file I/O system. Classes from fstream header file manage the disk file. Therefore it is important to include that header file in program while working with files.

Classes in fstream.h:-

S.no.	Class	description
1	Ofstream	it represents the **output file stream class** and provides the output operations. It derives the function write() and put() from ostream class of iostream.
2	Ifstream	it represents the **input file stream class** and provides the input operations. It derives get(),getline() and read() functions from the istream class of iostream.
3	Fstream	it represents the **input-output file stream class** and provides the input/output operations. It derives all the functions from istream and ostream classes of iostream.
4	Filebuf	it is used to set the file buffers to read and write. It provides the close() and open() member functions.
5	fstreambase	The fstream, ifstream and ofstream classes are inherited from the fstreambase class. the open() and close() functions are also contained in this class.

LIST OF FILE HANDLING FUNCTIONS:-

File handling functions:-

S.No	Function	Description
1	open()	It create a file.
2	close()	It closes a file.
3	getline()	It use to read string.
4	eof()	It return true on end of file.

Opening file:-

To read or write a file, you need to open the file first. You can use either the **ofstream** or **fstream** object to open file for writing, whereas the **ifstream** object is used to open a file for reading.

The **open()** function is member of the fstream, ifstream and ofstream objects which used for opening a file.

Syntax:-

```
File_stream_class  stream_object;
Stream_object.open("file name");
```

For appending information in file it can use as:-

```
Stream_object.open("file name", ios::app);
```

Ios::app is use for append file.

Closing a file:-

As soon as program terminates, all the files associated with that program are closed automatically, thereby, clearing all streams and releasing all the allocated memory. Though everything close automatically, it is suggested to always close all the files before terminating the program.

The close function is used for closing a file. It is a member of fstream, ifstream and ofstream objects. Close function does not take argument.

Syntax:-

```
Stream_object.close();
```

Read and write data from/to a file:-

Write information to a file using the stream insertion operator (<<) in similar way as use this operator with cout to display information on screen. **Ostream** or **fstream** object can use instead of the **cout** object.

Stream extraction operator (>>) use along with the cin object to take input from the user. Similarly this operator can use along with **ifstream** or **fstream** object to read information from file.

Example:-

Writing file:-

```
#include<iostream>
#include<fstream>                    //file class
using namespace std;
int main()
{
        string line;
        ofstream write;                    //file object
        write.open("hello.txt");  //open file
        while(true)
        {
                getline(cin,line);
                if(line=="EXITFILE$$")
```

```
                        {
                                break;
                         }
                         write<<line<<endl;            //write in file
         }
         write.close();         //closing file
         return 0;
}
```

Append in file:-

```
write.open("hello.txt",ios::app);       //add in existing file
```

Reading file:-

```
#include<iostream>
#include<fstream>                    //file class
using namespace std;
int main()
{
         string line;
         ifstream read;                         //file object
         read.open("hello.txt");   //open file
         while(read.eof()==0)
         {
               getline(read,line);   //reading from file
               cout<<line<<endl;   //print line
         }
         read.close();                  //close file
         return 0;
}
```

TEMPLATES IN C++

TEMPLATE:-

It is similar to a blueprint that creates a generic class or a function that reduces code duplication and includes independent writing code of any particular type. It is base of generic programming, which involves ode reuse and enables to write a code that behaves in same way for any type of data. It does not depend on the data type it deals with.

Syntax:-

Template<parameter list> declaration

Template is a keyword, parameter list is template parameters declaration is declaration of class and function.

Template<class my_type> class my_class {……};

My_type is identifiers that will help to define variables in class.

FUNCTION TEMPLATE:-

The special functions that specify a group of functions which can work with generic types are referred to as **function templates**. Function template can altered to one or more data types or classes without repeating code.

Syntax:-

```
Template <class type> return_type  function_name(parameter list)
{
        // code here
}
```

Example:-

```
#include<iostream>
using namespace std;
template<class T>

void add(T a,T b)
{
        cout<<"addition "<<a+b<<endl;
}

int main()
{
        add(45,67);
        add(4.5,6.7);
        return 0;
}
```

CLASS TEMPLATE:-

We can define a class template in same way as function template. A class template contains members that use template parameters as types and are useful for defining container classes.

Syntax:-

```
Template <class type> class class_name{
// code here
}
```

Example:-

```
#include<iostream>
using namespace std;
template <class T>

class square{
        T num;
        public:
        void set(T n)
        {
                num=n;
        }
        void get()
        {
                cout<<"square of "<<num<<" is "<<num*num<<endl;
        }
};

int main()
{
        square<float>s1;
        s1.set(4.56);
        s1.get();
        square<int>s2;
        s2.set(4.56);
        s2.get();
        return 0;
}
```

TEMPLATE PARAMETERS:-

A template is an entity defining a group of classes, functions or variables. One or more template parameters can be passed in parameter list.

Syntax:-

Template <parameter- list> declaration

Type template parameters:-

These are the most common type of template parameters.

Syntax:-
Typename name
Or
Class name

Example:-
```
#include<iostream>
using namespace std;
template<class T,class A>

void add(T n1,A n2)
{
        cout<<"addition "<<n1+n2<<endl;
}

main()
{
        add(5.4,67);
        return 0;
}
```

Non-type template parameters:-

it include values rather than the type they can be ant values such as compile time constant. The non-type template parameters can include built-in types along with generic data type.

Syntax:-
Template<class T, int m>

Example:-
```
#include<iostream>
using namespace std;
template<class T,int m>

class array{
        T array[m];
        public:
        void set()
        {
                for(int i=0;i<m;i++)
                {
                        int a;
                        cout<<"enter value for array "<<i<<" ";
                        cin>>a;
                        array[i]=a;
                }
        }
        void get()
        {
                        for(int i=0;i<m;i++)
                        {
                                cout<<"array "<<i<<" value "<<array[i]<<endl;
                         }
        }
};

int main()
{
        array<int,5>a1;
        a1.set();
        a1.get();
}
```

TEMPLATE SPECIALISATION:-

Having a special behavior for specific data type is termed as template specialization. For defining a different implementation for a template when you pass a particular type as a template parameter, you can define that template's specialization.

Example:-

```
#include<iostream>
using namespace std;
template<class T>

inline T add(T a,T b)
{
        T add=a+b;
        return add;
}
template<>
string add<string> (string m,string n)
{
        return (m+" "+n);
};
int main()
{
        int g=9,h=1,d;
        string c;
        d=add(g,h);
        c=add<string>("hello","world");
        cout<<d<<endl<<c<<endl;
        return 0;
}
```

HANDLING EXCEPTIONS IN C++

EXCEPTION HANDLING BASICS:-

An **exception** is a situation where a program produces an unexpected error during the program execution. It is important to have mechanism that can deal with these exceptions. It is a technique that separates error handling code from the main code of program and thereby makes it easier for the user to read and write code. It transfers the control from one point of a program to another.
An exception can occur because of following condition.

- An attempt to divide by zero
- Accessing an array outside of bounds.
- Running out of memory.
- Running out of disk space.

Keywords for handling exceptions:-
Try: - it is block of program code where an exception is detected. As soon as the exception is detected, try transfer control to catch block by throwing the exception.
Catch: - it determines the type of exception and provide block of code which will execute when an exception is thrown. There can be multiple catch blocks. It is responsible for taking corrective actions for exceptions.
Throw: - it is used to throw an exception when a problem occurs and can be called from anywhere in program. The point where throw is executed is referred to as **throw point**.
Syntax:-

```
Try{
        //protected code;
}catch(exception_name excp1)
{
        //catch block;
} catch(exception_name excp2)
.
.
.
{
        //catch block;
}catch(exception_name excpN)
{
        //catch block;
}
```

Exception handling features:-

- The exception handling is designed to handle the occurrence of exceptions, which changes the flow of program execution.
- It is newly created feature added to the American National Standards Institute (ANSI) C++.
- C++ detects error conditions at any point in program (the throw point) and then transfers the exception to another point in the program, i.e. to exception handler for error processing.

- The exception handling process allows functions to detect error conditions and then defer handling of those error conditions to a direct or an indirect caller of those functions.
- It allows separating of normal processing from error processing.
- Using exception handling mechanism improves the structure, organization and reusability of our software.

Exception handling mechanism:-

It built upon three keywords, namely try, catch and throw. The follow given steps to carryout exception handling mechanism:-

- The **try** block contains the code which may generate exceptions.
- Whenever an exception is found, the throw statement in the try block should be used to throw it.
- A **catch** block catches the exception and handles it appropriately.
- The catch block that catches an exception must follow the try block that throws the exception.

CATCH EXCEPTIONS:-

The role of a **catch** block is to catch the exception thrown by the try block. Therefore, it must follow immediately after the **try** block that throws the exception. You can mention what type of exception needs to be caught by declaring the type of exception. Declaration of exception is enclosed within the round brackets.

Syntax:-

```
Try{
        //protected code
}
Catch(ExceptionName excp)
{
        //code for handling exceptionName exception
}
```

exceptionName is a type of exception and the catch catches the exception of exceptionName type.

THROWING AN EXCEPTION:-

When exception is raised in a program, it can be thrown to an exception handler by using the **throw** statement. It determines the type of exception and throws it to the appropriate catch block that can handle it. The throw statement should be declared inside the try block, not outside it.