C++ Tutorial

In this article, we will be covering one of the most sought after object oriented programming language in the world i.e. C++. A famous saying goes around in the programming fraternity that “If programming in Pascal is like being put in a straightjacket, then programming in C is like playing with knives, and programming in C++ is like juggling chainsaws." C++ is a general-purpose programming language with a bias towards systems programming that supports efficient low-level computation, data abstraction, object-oriented programming, and generic programming.

Bjarne Stroustrup from Bell Laboratories developed C++ after years of hard work since he first started development around 1979. C was precursor to C++ as Bjarne was trying to add object-oriented features and improving memory model used in C. In fact, C++ was first named as “C with Objects”. With addition of new features in the language, Stroustrup named it as C++ in 1983. Remember ++ operator used in C (it is increment operator)! The C++ programming language mimics a memory and computation model that is used in most computers. At its core, lies powerful and flexible mechanism for abstraction that allows the programmer to introduce and use new types of objects matching that of the application code in hand. In essence, just like C, C++ supports a direct low level style of programming that rely on direct manipulation of hardware resources to deliver a high degree of efficiency. It also supports higher-level programming that includes user-defined types to provide a model of data and computation that is closer to how human divides a problem into tasks. This is also called as data abstraction, object-oriented programming, and generic programming.
The article is divided into 23 sections. Section 1 gives an introduction to C++. Section 2 describes C++ Environment Setup. The next section lays down basic C++ syntax. Section 4 describes C++ Comments. Section 5 talks about C++ Data Types. Section 6 describes C++ Variable Types. The next section gives C++ Variable Scope. This is followed by a description of C++ Constants and Literals. Section 9 describes C++ Modifier Types. The next section lays down basics of C++ Storage Classes. Section 11 describes C++ Operators. Section 12 talks about C++ Loop Types. Section 13 describes C++ Decision Making. The next section gives C++ Functions. This is followed by a description of C++ Numbers. Section 16 describes C++ Arrays. The next section gives C++ Strings. This is followed by a description of C++ Pointers. Section 19 describes C++ References. The next section lays down basics of C++ Date & Time. Section 21 describes C++ Basic Input and Output. Section 22 talks about C++ Data Structures. Section 23 concludes this article.

C++ Introduction

Bjarne Stroustrup designed and implemented C++ at AT&T Bell Laboratories in an attempt to improve C language by giving it an object oriented approach. C++was first made commercially available in 1985 while generic programming constructs were added to the language in around 1987-1989. The formal standardization of C++ started in 1990 under guidance of the American National Standards Institute, ANSI, and later under the International Standards Organization, ISO, leading to an international standard in 1998. Thereafter we are having regular revisions to the language as per decided periodicity. The author designed C++ with sole aim to deliver the flexibility and efficiency of C for systems programming together with *Simula’s* facilities for object-oriented programming. C++ is designed to make programming more enjoyable for serious programmers. C++ is designed as a general-purpose programming language that is a better than C language, that supports data

abstraction, object-oriented programming, generic programming and system programming all in one. C++ was primarily aimed at programmers engaged in demanding real-world projects.
C++'s evolution has been driven by real problems. C++ design is devoid of any sterile quest for perfection. Here, every feature must have a reasonably obvious implementation and there is adequate support for each supported style. The aim of C++ was to improve the quality of programs produced by making better design and programming techniques that are simpler to use and are affordable. C++ was designed with a burning desire for a high degree of C compatibility, uncompromising efficiency, object oriented approach and immediate real-world utility.
C++, since start, has been source-and-link compatible with C and except for minor details, C++ is a superset of C. This implies that C++ programmers immediately had a complete language and toolset available. However, backwards compatibility with C leaves C++ with some syntactic and semantic quirks. For example, the C declarator syntax is far from elegant and the rules for implicit conversions among built-in types are chaotic. Programmers that migrate from C to C++ do not appreciate the fact that radical improvements in code quality can only be achieved by making radical changes to programming styles.
C++ finds wide applications in many domains such as those given below:

Both System and application Development C++ has been used in development of almost all the major Operating Systems like Windows, Mac OSX and Linux. Many drivers and middle wares have been written in C++. It has equally contributed at the application side. The core part of many browsers like Mozilla Firefox and Chrome has been written using C++. C++ also has been used in MySQL Database. Latest is C++ is getting used in many machine learning and AI based models (especially for image processing).

Programming Languages Development C++ has been used as a base in development for many new programming languages such as C#, Java, JavaScript, Perl, UNIX's C Shell, PHP and Python, and Verilog etc.

Scientific and numerical programming C++ is heavily used in scientific computing because of fast speed and computational efficiencies.

Games Development C++ being object oriented is extremely fast. This enables programmers to code procedurally for CPU intensive functions of a multi-player gaming engine. C++ also provides greater control over hardware.

Embedded System C++ is being heavily used in developing Medical and Engineering Applications like software for MRI machines, high-end CAD/CAM systems etc.
C++ is a complete object-oriented programming whose framework rests on all four pillars of OOPS i.e. encapsulation, abstraction, inheritance and polymorphism. Standard C++ language consists of three important parts that are given below:

1. The core language it includes all the building blocks including variables, constants, data types, classes, operators etc.
2. The C++ Standard Library it includes complete set of methods to work on streams, files, strings, sockets etc.
3. The Standard Template Library or STL it gives tools set to use advanced data structures such as vectors and templates.

C++ environment Setup

In order to work on C++, you need two software components i.e. a text utility to write your program and a compiler to compile and run your program. Often, these two components are clubbed together in a single application called Integrated Development environment or IDE. Text utility varies from operating system to operating and depends heavily on the programmer's choice and comfort factor. Many people use Vi or Vim (Linux) to write code, old programmers prefers EMACS (Linux), while new breed of programmers move towards atom, notepad, nano, Eclipse etc. The files in which you write your code (set of instructions) in your editor are called source files and for C++ they typically are named with the extension .cpp, .cp, .h, or .c.

There are many C++ compilers available. Notable ones are given below:

1. Apple C++. Xcode
2. Bloodshed Dev-C++
3. Clang C++
4. Cygwin (GNU C++)
5. Mentor Graphics
6. MINGW - "Minimalist GNU for Windows"
7. GNU CC source
8. IBM C++
9. Intel C++
10. Microsoft Visual C++
11. Oracle C++
12. HP C++

The most used C/C++ compiler is GNU. We present the most common way to install a C++ compiler in each of Windows, Linux and Mac environment. In order to compile C++ code on Macbook, you need to install *Xcode* tool set which has got the Clang compiler for C/C++. It is an IDE (Integrated Development Environment) where you can write, compile, and run your programs. Post Xcode installation, you can use a text editor (atom, nano or vim) to write code and the Terminal to compile and run C++ programs. You can download Xcode from the App Store on your Mac. You can also simply install the xcode command line tools if you don't want the whole IDE. Another way to install C++ compiler is to use macOS package managers like MacPorts and Homebrew to install the needed compiler. You can instal gcc using brew install gcc. This command will GNU C++ compiler at its default location in */usr/local/Cellar/gcc/8.2.0/bin* (assuming gcc version installed is 8.2.0).

To compile a source file say helloworld.cpp, you can write

```
XXX:~$ gcc helloworld.cpp.
```

The best way to run C++ programs on windows is to use an C/C++ compliant IDE. Dev-C++ from Bloodshed Software is the recommended IDE for C and C++ programming on Windows. Here are basic steps to install Dev C++ software on your windows machine.

1. Visit website for bloodshed (http://www.bloodshed.net) to download the Dev C++ on your Windows machine.

2. You can choose "Original Dev-C++ 5" Under package Dev-C++ 5.0 (4.9.9.2) with Mingw/GCC 3.4.2 compiler and GDB 5.2.1 debugger (9.0 MB).
3. Run the exe file that you have downloaded in step 2.
4. You will be directed to SourceForge website, and your C++ download will start automatically. Click on save button to save and once it is downloaded, click it to run.
5 Click Ok to few screens and complete installation

This concludes installation process and you can use Dev-C++ to compile your C or C++ programs.

You can install GCC compiler on Ubuntu (it comes as g++ for C++ and gcc for C programs) using the following command.

XXX:~$ sudo apt -get install g++

Then check the version using the following command

XXX:~$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

C++ Syntax

This section lays out basic C++ syntax. The first step to learn any language is to study its rules and regulations which together are called as syntax. Formally, the term "Syntax" means an approved set of pre-defined protocols or rules that we need to follow while working in a programming language. Just like any other programming language, C++ has its own unique syntax.
So let us begin with a simple "Hello World!" program. Open any IDE or text editor and create file HelloWorld.cpp with following content.

```
// Basic header file for giving Input/output function
# include<iostream>
//Standard namespace
using namespace std;
/* Entry point of the code */
int main()
{
      //Print Hello world on console and exit
      cout<<"Hello world!"<<endl;
      return 0;
}
```

Now run this program from command prompt

(base) 10:28:00:~ % g++ hello.cpp
(base) 10:28:03:~ % ./a.out
Hello world!

For this code to run, g++ should be in your PATH environment variable and your present working directory should be the one containing file hello.cpp.

For bigger projects containing many files, you can compile the project by writing a *makefile*.

Let us understand this program line by line. Set line number in your ide (in vim use set nu option) and paste the code here

```
 1 // Basic header file for giving Input/output function
 2 # include<iostream>
 3
 4 //Standard namespace
 5 using namespace std;
 6 /* Entry point of the code */
 7 int main()
 8 {
 9         //Print Hello world on console and exit
10         cout<<"Hello world!"<<endl;
11         return 0;
12 }
13
```

Line 1 is a comment for the reader as to that why we are including the iostream header file. Comments are ignored by the compiler. C++ ignores white space. Whitespace denotes blanks, tabs, newline characters and comments. Whitespace gives logical separation and is used by compiler to identify where one element in a statement ends, and the next element begins.

Line 2 tells compiler to include standard header iostream. Iostream is needed for input output functionality to a stream (console in this case). The compiler will look into the standard path for this. A user can include his own header file say XX.h that is located in same directory by simply writing #include “XX.h”

Line 3 is blank (kept blank for readability)

Line 4 is comment to denote we are using standard namespace

Line 5 tells compiler to use standard namespace in this file. Namespace is collection of names for objects and variables from the standard library. Namespaces allow us to group named entities into small and narrow scope that can be accessed privately for different programmes. This allows organizing the elements of programs into different logical scopes referred to by name. If you do not want to include standard namespace then you have to add std:: prefix before every standard object e.g. std::cout<<”…”;

Line 6 gives comment about main() to tell reader that it is the main entry point of this code. This means it is the first function that is called when the program executes

Line 7 starts definition of main(). Any code for a method that is written in {} gets executed.

Line 8 and 12 are curly brackets that enclose the body of main()

Line 9 gives comment

Line 10 uses *cout* function to print Hello world! On to console and then leave a line

Line 11 gives return statement with return code telling compiler to take control back from this program. Return code 0 denotes successful execution.

A typical C++ program has a collection of objects such as class, object, methods, interface, directives, macros, pre-processor statements, variables, global etc. **An Object** is basic entity in C++ and it can be anything such as circle, point, cylinder. An object has attributes and behaviours. Example: A car has properties - colour, name, brand, price etc.; a car object has behaviours (basically methods) - start/stop, drive, brake, service etc. An object is an instance of a class which is a blueprint describing the behaviours/states that object of its type support. A method is basically a behaviour or action. A class can contain many methods. It is in methods where the logics are written, data is manipulated, and all the actions are executed. Every object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

In C++, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity. E.g.

```
int x = 100;
temp += temp +20;
```

A block is a set of logically connected statements that are surrounded by opening and closing braces.

```
{
      x=x*2;
      cout<<x;
}
```

C++ does not recognize the end of the line as a terminator. For this reason, it does not matter where you put a statement in a line. For example block 1 and block 2 are same from compiler point of view.

Block 1

```
x = 10;
y = 29;
mul(x, y);
```

Block 2

```
x = 10; y = 29; mul(x, y);
```

C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. It can start with A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9). However, C++ does not allow special characters such as @, $, and % within identifiers. E.g

```
employeeName, x, 55rajesh etc.
```

Further note that C++ is a case-sensitive programming language. Thus, **`EmployeeRecord`**, **`employeeRecord and employeerecordpower`** are three different identifiers in C++.

There is a set of words called reserved key that cannot be used to name any identifier in user code. E.g . *asm, else, new* are reserved keywords.

C++ Comments

In this section, we will cover about all types of comments used in C++. Comments is often given least importance in coding but they are read much more time than the actual lines of code. They primarily serve to explain C++ code, increase its readability and decrease its cognitive complexity. Apart from giving basic information about the file, functions, author, year, change log etc. they often suggests why current implementation is chosen. It can also be used to prevent execution when testing alternative code.
A well-documented program is often one of the expected deliverable from a programmer. Comments makes a program more readable and easier to find error. Programmers often include tags to create low level documentation of the code. Comments also server an important purpose at code reviews. For e.g. a comment on top of a function giving its space and time complexity will help reviewer appreciate the function's efficiency and see how it impacts the overall module.
In almost all programming languages, a comment is a programmer-readable explanation or annotation to explain the code below. Comments are statements that are ignored by the compiler (they are not executed). In C++, Comments are both singled-lined and multi-lined.

Single-line Comments □ begin with 2 forward slashes (//). Text sandwiched between // and the end of the line is ignored by the compiler (will not be executed).

```
// Output Hello World! to console
cout << "Hello World!";
```

This example uses a single-line comment before a line of code. The backslash is a continuation character and will continue the comment to the following line:

```
// This comment will also comment the following line \
std::cout << "This line will not print" << std::endl;
```

C++ Multi-line Comments □ they start with /* and ends with */. Any text between /* and */ will be ignored by the compiler.

```
/*
 * The code below will print Hello World!
 * to the screen
 */
cout << "Hello World!";
```

As a general rule, each object in a C++ program should have a comment block, be it file, function, constant, macro, pre-processor, class etc. Any statement in your program that is not obvious or complicated should be commented as well. Similarly, any use of external library function call should be annotated. Following is the example of a file header that is usually present at the top of source or header file

```
/*************************************************************
 *
 * Program:       Hello World
 *
 * File:          Hello.cpp
 *
 * Function:      Main Entry point of the program
 *
 * Description:   Prints the words "Hello world" to the
 *                screen
 *
 * Author:        Nikhil Jain [nj] (nikhil.XXX@gmail.com)
 *
 * Year:          2020-2021
 *
 * Environment:   g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0.
 *
 * Notes:         This is an introductory, sample program.
 *
 * Revisions:     1.00  10/1/2020 (nj) First release
 *
 *************************************************************/
```

Comments are also sometimes used to enclose experimental code or trial code that we temporarily want the compiler to ignore. This can be useful in finding errors in the program. If a program does not give the desired result, it might be possible to track which particular statement contains the error by commenting out code. The following comment block is placed above a function definition

The following is comment block for a class

```
/*-------------------------------------------------------
                    <copyright notice>

Determine if input type A is same as type B. Example:

template <typename T, typename U>
```

```
void do_something(const T&, const U&, bool flag);


template <typename T, typename U>
void do_something(const T& t, const U& u)
{
    do_something(t, u, is_same<T,U>::value);
}
------------------------------------------------------------*/
```

The following is good way to annotate a function so that its low level documentation can be created for further reference.

```
/// <summary>Sorts the list to by the given column</summary>
/// <param name="sel_criteria">Column to be sorted (index-1-based)
and sort direction (pos. = ascending)</param>
/// <returns>Documentation of return type</returns>
void CDBTaglist::SetSorting(int sel_criteria) { ... }
```

C++ Data Types

This section gives detailed insights into data structures used in C++. In C++, data types are used for declarations of variables. This tells the compiler the type and size of data associated with variables. For example,

```
int income = 90000;
```

Here income is a variable of type int and can store value in range for 2 bytes or 4 bytes depending upon the machine size. C++ has 7 fundamental system defined datatypes given in following table

Data Type	Meaning	Size (in Bytes)
`int`	Integer	2 or 4
`float`	Floating-point	4
`double`	Double Floating-point	8
`char`	Character	1
`wchar_t`	Wide Character	2
`bool`	Boolean	1
`void`	Empty	0

Let us drill down each data type
`C++ int` int denotes integers. Its size is machine dependent and is usually 4 bytes (value can be between -2147483648 to 2147483647). For e.g.

```
int roll_no = 11;
```

`C++ float and double` — float and double are system's containers to store floating-point numbers which are usually decimals and exponentials. The size of float is generally 4 bytes and the size of double is 8 bytes. For e.g.

```
float percentage = 93.74;
double volumeOfSphere = 251.64534;
double distance = 3E21;
```

`C++ char` — Keyword char is used for storing characters. It can hold 1 byte. C++ characters are enclosed inside single quotes ' '. For e.g.

```
char gender = 'M';
```

`C++ wchar_t` — Wide character wchar_t is char data type with size 2 bytes instead of 1. It is used to represent non-ANSI (Unicode characters) that require more memory to represent them than a single char. IT is usually represented by appending L in in front of the value. C++11 revision also included char16_t and char32_t. For example,

```
wchar_t inputChar = L'□'  // storing Hebrew character;
```

`C++ bool` — The bool data type is used for storuing variables having binary values i.e. True or False or On or OFF. IT finds application in conditional statements and loops. For example,

```
bool condFlag = true;
```

`C++ void` — The void keyword indicates that given object is devoid of data. It means "no value". We use it in functions and pointers. Please note that void cannot be used to declare variables.

We can modify char, int and double data type using 4 modifiers i.e., signed unsigned, short and long. The following table gives list of internal datatypes with modifiers for C++.

DT	Size (Bytes)	Description
`signed int`	4	used for integers (same as int)
`unsigned int`	4	can only store positive integers
`short`	2	used for small integers (range **-32768 to 32767**)
`unsigned short`	2	used for small positive integers (range **0 to 65,535**)
`long`	at least 4	used for large integers (same as long int)
`unsigned long`	4	used for large positive integers or 0 (same as unsigned long int)
`long long`	8	used for very large integers (same as int long long).

`unsigned long long`	8	used for very large positive integers or 0 (same as int long long int)
`long double`	12	used for large floating-point numbers
`signed char`	1	used for characters (guaranteed range **-127 to 127**)
`unsigned char`	1	used for characters (range **0 to 255**)

Some examples are as follows:

```
long bL = 4235632;

long int counter = 556342;
```

long double dPrecise = 134234.56343;

Let us see the sizes for each datatype on a mac machine. Save the following code in a C++ source file say basicSize.cpp

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


int main()
{
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of unsigned char : " << sizeof(unsigned char) <<
endl;
   cout << "Size of signed char : " << sizeof(signed char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of unsigned int : " << sizeof(unsigned int) <<
endl;
   cout << "Size of signed int : " << sizeof(signed int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of unsigned short int : " << sizeof(unsigned short
int) << endl;
   cout << "Size of signed short int : " << sizeof(signed short int)
<< endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of signed ong int : " << sizeof(signed long int) <<
endl;
   cout << "Size of unsigned long int : " << sizeof(unsigned long
int) << endl;
   cout << "Size of long long int : " << sizeof(long long int) <<
endl;
   cout << "Size of unsigned long long int : " << sizeof(unsigned
long long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of long double : " << sizeof(long double) << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

   return 0;
}
```

Now compile and run the program

```
(base) 14:58:34:~ % g++ basicSize.cpp
(base) 14:58:36:~ % ./a.out
Size of char : 1
Size of unsigned char : 1
Size of signed char : 1
Size of int : 4
Size of unsigned int : 4
Size of signed int : 4
Size of short int : 2
Size of unsigned short int : 2
Size of signed short int : 2
Size of long int : 8
Size of signed ong int : 8
Size of unsigned long int : 8
Size of long long int : 8
Size of unsigned long long int : 8
Size of float : 4
Size of double : 8
Size of long double : 16
Size of wchar_t : 4
```

You can create your own data type declarations from existing data types using keyword 'typedef' using the following syntax

```
typedef type yourname;
```

Then you can declare variables using your name type.

```
yourname firstname;
yourname lastname;
```

C++ provides enumeration type to declare a variable that can take any value from a given set. Its syntax is an optional type name and a set of zero or more identifiers that can be used as values of the type. Each enumerated value, called enumerator, is a constant whose type is the enumeration. We use *enum* to create an enumeration using following syntax

```
enum Nameofenum { comma seprated list of values} variableList;
```

Here, the Nameofenum is the enumeration's type name.

For example, the following code defines an enumeration of customer class called customerClass and the variable cClass of type customerClass. Later c is assigned the value "gold".

```
enum customerClass { platinum=1, gold, silver, bronze } cClass;
```

```
cClass = gold;
```

If you do not initialize any value then by default, the value of the first name is 0, the second name has the value 1, and rest of the names will have value incremented by 1 from previous value. However, you can give your own initializer value. For instance, in the below declaration, each value has different initial value.

```
enum customerClass { platinum=10, gold=13, silver=15, bronze=17 }
cClass;
cClass = gold;
```

Apart from system's basic datatypes, advanced Data types are derived from them. They are called derived types or DT. Common DTs are arrays, pointers, function types, structures, etc.

C++ Variable Types

Let us explore more on C++ variable types. A variable is a named storage needed to carry out logic in the program. A variable is something that can be accessed and modified as per need.

Since C++ is a strongly typed language, every variable in C++ has a specific type from which compiler determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

Like every other programming language, C++ also has rules for naming each object including variables. The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive.

We have already seen basic data types and their memory requirements. . C++ also gives advanced datatypes such as Enumeration, Pointer, Array, Reference, Data structures, vectors, maps and Classes. To define a variable, we use the following syntax.

```
type variable_list;
```

```
int medianSalary = 19000;
flaot result;
string name;
```

A variable has two operations i.e. declaration and definition that can either be done one at a time or in a single statement. The declaration tells compiler that there is one variable with the given type and name so that it can proceed for further compilation. A variable declaration is to satisfy compilation requirement, its actual value is needed at run time while linking of the program. A variable declaration is useful in multiple file case wherein you can define your variable in one of the files so that it becomes available at the time of linking of the program. For this, you need to use extern keyword to declare a

variable at any place. Though you can declare a variable multiple times in your C++ program, but it can be defined only once in a file, a function or a block of code.

Let us conclude this section by describing two terms that are heavily used in context of variables in both C and C++ i.e. Lvalue and Rvalue. L-value refers to memory location which identifies an object and is the identity of the variable. L-value can appear on either left hand or right hand side of an assignment operator (=). Expressions that are mapped to modifiable locations are called “modifiable l-values“. A modifiable l-value cannot have an array type, an incomplete type, or a type with the const attribute because all these are not modifiable. Any structure or unions with any members with the const attribute will not be "modifiable l-value". An identifier qualifies for modifiable lvalue if it refers to a memory location with type arithmetic, structure, union, or pointer. For example, if sum is an integer variable, then sum is a modifiable l-value since it can change its value. R-value refers to data value that is stored at some address in memory that can’t be assigned any value. This implies that r-value can only and only appear on right of an assignment operator (=).

C++ Variable Scope

In this section, let us explore variable scope in C++. By scope, we mean an area of a block in a program. It is a region of the program and more or less there are 3 scope levels defined for a variable −

1. Local scope when it is defined inside a function or a block and accessible only in that block.

2. Formal scope when it is defined as a function parameters.

3. Global scope when it is defined outside of all functions.

Let us drill down each variable scope. Local variables are defined inside functions or a block in a function and are local to it. The variable has no scope outside that function or block. Global variables have scope in the entire program. It is usually defined outside of all the functions and usually on top of the program. The global variables will hold their value throughout the life-time of your program. A global variable can be accessed by any function in the file and is available for use for the entire duration of the program after its declaration.

We can use same name for local and global variables but value of local variable inside a function will take precedence. Also note that a local variable is declared, it is not initialized by the system. It is up to the user to initialize it. Global variables are initialized automatically by the system after you declare them.

C++ Constants and Literals

In this section, we will find out more on C++ constant and literals. In C++, const modifier is used to create constants i.e. variables whose value cannot be changed once they are initialized. Generally

constants are written in CAPITAL and usually placed in the top of the program file. Its syntax is as follows:

```
const type variable = value;
const int PI = 3.14;
```

You cannot reset the value of a constant. For e.g. the following code snippet will give error as we are trying to change a constant literal's value.

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


int main()
{
   int i,j;
   ....
   const int SPEED_OF_LIGHT = 300000000;
   ...
   SPEED_OF_LIGHT = 250000000 // Error! Cannot change
SPEED_OF_LIGHT which is a constant.
   ...
}
```

Another way to creare a constant is using the #define preprocessor directive. For e.g.
define SPEED_OF_LIGHT 300000000

C++ literals are data used for representing fixed point values that we can directly use in the code. For example: 100, 3.1421, 'y', 'n' etc.

Please note that a variable can be assign different values but literals can't. There are 6 types of literals in C++.

A. Integer literal An integer is a fixed point numeric literal. It has no fractional or exponential part. There are three types of integer literals used in C++ programming i.e. decimal (base 10), octal (base 8) and hexadecimal (base 16). Please note that octal literal starts with a 0 while hexadecimal literal starts with a 0x. For e.g.

Decimal literal : -1, 0, 100 etc.
Octal literal : 045, 067, 043 etc.
Hexadecimal literal: 0x8f, 0xa2, 0x721 etc.

B. Floating point literal A floating-point literal is different from integar literal as it has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form. Decimal form must have

the decimal point, the exponent, or both and while the exponential form must have the integer part, the fractional part, or both. The signed exponent is denoted by e or E. For e.g.

```
3.14134        // Legal
314159E-5L     // Legal
620E           // Illegal: incomplete exponent
22f           // Illegal: no decimal or exponent
.e755           // Illegal: missing integer or fraction
```

C. Boolean literal C++ has two Boolean literals i.e. true which denotes for success/truth and false which stands for failure/falsehood. A programmer usually uses considers the value of true equal to 1 and value of false equal to 0.

D. Character literal A character literal is created by enclosing a single character inside single quotation marks. For example: 'a', 'm', 'F', '2', '}' etc. Please note that f for literals begins with L, you should store it in wchar_t type of variable as it is a wide character literal. For e.g.

```
'y', 'n', L'fu', '(', ']' etc.
```

E. String literals String literals are proper strings that are enclosed in double quotes (""...."). A string literal can have anything such as plain characters, escape sequences, and universal characters. We can break a long line into multiple lines using string literals and separate them using whitespaces. Here are some examples of string literals that are identical strings.

```
"India is great"
```

```
"India \
is great"
```

```
"India \
is \
great"
```

F. Escape sequence this type of literal is used for some special operation or to denote a special character that cannot be typed using keyboard. For example, newline (enter), tab, question mark, etc. Some common escape sequences are \n to add new line; \t to add tab space; \b to add a backspace etc.

C++ Modifier Types.

Here, we look into C++ modifiers. C++ allows the char, int, and double data types to have modifiers to be added before the type field so as to change the behaviour of base data type. We have already discussed them when describing data types. A modifier is used to change the meaning of the base type as per the need or requirement of the function or program. There are mainly 4 types of data types of modifiers –

A. signed
B. unsigned
C. long
D. short

The modifiers signed, unsigned, long, and short are usually used to modify integer base types. In addition, signed and unsigned can be applied to char, and long can be applied to double. Please note that the modifiers signed and unsigned can also be used as prefix to long or short modifiers. For example, unsigned long int.

In C++, you can use a shorthand notation for declaring unsigned, short, or long integers by using the words unsigned, short, or long (without writing int). It automatically implies int. For e.g. both the below declaration are same from compiler point of view.

```
unsigned counter;
unsigned int counter;
```

Apart from them, there are some additional type qualifiers in C++. const qualifier before a variable make it immutable i.e. const objects cannot be changed by your program during execution. The volatile modifier tells the compiler that a variable's value may be changed in ways not explicitly specified by the program and so it must not store it into memory and instead cache it in its registers. The restrict is a new qualifier that got added in C++ in C99 revision. *restrict* when applied to a pointer pointing to any C++ object, then the only way to access this object is via this pointer.

C++ Storage

This section talks about C++ storage classes used to house variables or functions. In C++, a storage class defines the scope (visibility) and life-time of variables and/or functions used in a program. These specifiers precede the declaration type. There are following storage classes that can be used in a C++ Program.

1. auto this is the default storage for all local variables. For e.g.

```
{
      int x;
      auto int counter;
```

```
}
```

2. register it defines local variables that you want to be stored in a CPU register instead of RAM. We need this for a variable that changes often and needs to be accessed very frequently. But this is an advisory to compiler. If no CPU register is free, then register directive is ignored and the variable is treated as normal local variable. The variable that needs to be stored in a register can be of size at most equal to the register size. Further, we can't have the unary '&' operator applied to it because it has no memory location. For e.g.

```
{
   register int  sum;
}
```

3. static static class tells the compiler to maintain the local variable during the entire life the program instead of creating and destroying it each time it comes into picture and goes out of frame. Therefore, static allows variable to maintain its values between function calls. The static modifier used in front of global variable causes its scope to be restricted to the file in which it is declared. So it becomes file scoped. For C++, a static data member of a class will make all objects of that class share the static data member. Similarly, static member functions of a class are allowed to access only the static data members or other static member functions, they cannot access the non-static data members or member functions of the class.

4. extern this class is used to give a reference of a global variable in order to make it visible to all the program files that forms part of the project. The keyword 'extern' in front of variable declaration tells the compiler that this variable name is defined at a storage location previously. For multiple file case, you define a global variable or function, which will be used in other files also, then extern will be used in another file to give reference of defined variable or function. Point to remember is that extern is used to declare a global variable or function in another file.

5. mutable it applies only and only to class objects. The mutable modifier a member of an object to override const member function i.e. a mutable member can be modified by a const member function.

C++ Operators

Now, it is time to explore C++ operators. In C++, Operators work on variables and values. Most of the operations are mathematical or logical in nature. C++ is rich in built-in operators and provides the following types of operators −

A. Arithmetic Operators They are used to perform arithmetic operations on 2 or more operands. Some common operands are + (addition), - (subtraction) , * (multiplication), / (divison), %(remainder) ,++ (increment), -- (decrement). For e.g

```
i++;
```

```
sum = a + b;
```

B. Relational Operators they are used to determine the relation between variables or values. Here, we compare two values and return the result as either true (1) or false (0). Some common relational operators are as == (check equality), != (checks inequality), > (greater than), < (lesser than), >= (greater than or equal to) and <= (less than or equal to).

For e.g.

```
if (x>THRESHOLD)
{
    …
}
```

C. Logical Operators used to determine the logic between variables or values. Logical operators are && (Logical AND),|| (Logical OR Operator) and ! (Logical NOT operator).

For e.g.

```
if (x<5 && y>10)
{
}
```

D. Bitwise Operators Bitwise operator does not work on numbers but on bits. It is done bit-by-bit operation. The three Bitwise operators are & (bit wise AND), | (Bitwise OR), ^(Bitwise XOR), ~ (binary 1's complement), << (binary left shift operator) and >> (binary left shift operator).

E. Assignment Operators This class of operators are used for assignment. For e.g = (simple assignment), += (add AND assignment), -= (subtract AND assignment), *= (multiply AND assignment) etc.

F. Misc Operators They are a heterogeneous operator set. It includes sizeof (used to find size of a variable), ternary operator condition ? exp1: exp2 (conditional evaluation), , operator, .(dot) and -> (arrow), cast (change type of a variable temporarily) , & (pointer address) and *(pointer access)

The operator precedence and associativity are also important. Precedence dictates priority among execution of different operators while associativity gives the direction in which operator are evaluated

(left to right or right to left). Both of them affects how an expression is evaluated. For example, the multiplication operator has higher precedence than the addition operator, so * will be evaluated before +.

For example :

```
x = 7 * 5 + 3 - 2 * 10; //x will be equal to 18
```

C++ Loop Types

In this section, we will look into various types of loops present in C++. Often in a program, you need to execute a block of code several number of times. For example, taking marks in 5 subjects from user. Here same set of instructions are executed multiple times sequentially. For such cases, C++ provides loop. A loop statement enables sequential execution of statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages.
C++ provides 3 types of loops

A. The while
B. The for
C. The do…while

The "**while"** loop is used when you want a statement to execute continuously till the condition specified is true. The "for" tells us how many times the loop is executed but while loop does not tell us how many times the loop will execute. The syntax for while loop is simple as shown below:

```
while (condition)
{
    //loop statements;
}
```

The **for** loop is the most popular looping construct used in any programming language. The syntax of for loop is as follows

```
for (variable initialization; condition; increment operation)
{
    //loop statements;
}
```

For e.g.

```
for (int i=0; i<10; i++)
{
    ...
}
```

The initialization step allows you to declare and initialize counter variable. In the next step, you can evaluate the variable against a condition and in the final step, you can increment or decrement the value of the variable. The loop will go on till the condition becomes false. Then, the program will exit the loop and continue with the rest of the statements.

The **do while** loop executes at least once because the condition is evaluated only at the bottom of the loop. In the do while loop, the body of the loop will be executed first and then the condition will be evaluated. The syntax of do while loop is :

```
do
 {
 // statement execution;
 }
 while(condition);
```

C++ Decision Making

Let us deep dive into C++ decision making constructs. At every important point of time, one has to take right decisions, be it life or a C++ program. In programming, flow is never completely sequential and we need to make some decisions, evaluate some condition and based on the result, we execute the next block of code or skip it. For example if we need to search an element in a given list then if have to compare each element of list with the given number and if they are equal, we print it and exit else we move to the next element.

Decision-making statements in C++ mark the direction of the flow of program execution. Decision-making statements in C/C++ are as follows:

```
A.    if statement
B.    if..else statements
C.    nested if statements
D.    if-else-if ladder
E.    switch statements
F.    Jump Statements:
a.    break
b.    continue
c.    goto
d.    return
```

if statement is the most basic and most used decision-making statement of the entire set. It evaluates a condition in if statement and if it is true then executes the block else skip that block. Its syntax is as follows

```
if(condition)
{
   // Statements to execute if
   // condition is true
```

```
}
```

For e.g.

```
if(status == true)
{
     ...
}
```

if-else is the next decision making construct. It evaluates a condition in if statement and if it is true then executes the block immediately after if and if the condition is false then it executes the block immediately after else statement. Its syntax is

```
if(condition)
{
     // Statements to execute if
     // condition is true
}
else
{
     // Statements to execute if
     // condition is false

}
```

For e.g.

```
if(status == true)
{
     ...
}
else
{
     ...
}
```

nested if conditional is complex hierarchy of if conditionals. Here one if statement is the target of another if statement. Its syntax is as follows :

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
          // Executes when condition2 is true
```

```
          if (condition3)
          {
               // Executes when condition3 is true
          }
}
```

if-else-if ladder is used when user has to make a cascade of choices. Here the evaluation starts from the top if statement. The moment one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If no condition is true, then the final else statement will be executed. Its syntax is as follows

```
if (condition)
{
     ...
}
else if (condition)
{
    ...
}
.
.
else
{
     ...
}
```

Let us give its practical demonstration. The following program gives grade of student depending upon the marks Copy the following code in a file say *ifelseifLadder.cpp*.

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


int main()
{
     string grade;
    float percentage;


     //take percentage as input from user
    cout<<"Enter your final percentage: ";
    cin>>percentage;

     if (percentage > 90.0)
     {
          grade = "A1";
          cout<<"You have got "<<grade<<"."<<endl;
     }
     else if(percentage > 80.0)
```

```
    {
        grade = "A2";
        cout<<"You have got "<<grade<<"."<<endl;
    }
    else if(percentage > 70.0)
    {
        grade = "B1";
        cout<<"You have got "<<grade<<"."<<endl;
    }
    else if(percentage > 60.0)
    {
        grade = "B2";
        cout<<"You have got "<<grade<<"."<<endl;
    }
    else if(percentage > 50.0)
    {
        grade = "C1";
        cout<<"You have got "<<grade<<"."<<endl;
    }
    else if(percentage > 40.0)
    {
        grade = "C2";
        cout<<"You have got "<<grade<<"."<<endl;
    }
    else
    {
        grade = "F";
        cout<<"You have got "<<grade<<"."<<endl;
    }
}//end main ()
```

Now compile it and run it to see the output.

```
(base) 22:35:42:~ % g++ ifelseifLadder.cpp
(base) 22:36:11:~ % ./a.out
Enter your final percentage: 98
You have got A1.
```

Jump Statements in C/C++ are used to hop on the flow to some other random statement without evaluating any condition. C++ has four types of jump statements that are given below:

```
break
continue
go to
return
```

break statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops, and control returns from the loop immediately to the first statement after the loop. Its syntax is

```
break;
```

continue is just opposite to that of the break statement meaning instead of terminating the loop, it forces to execute the next iteration of the loop. The continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin. Its syntax is

```
continue;
```

return statement marks end of execution of called function and returns the flow of the execution back to the parent function. This statement may or may not need any conditional statements. The return statement may or may not return anything for a void function, but for a non-void function, a return value is returned for sure. Its syntax is

```
return[expression];
```

goto statement is an unconditional jump statement where in code jumps to the satement marked by label. Its syntax is shown below :

```
....
goto label;

...

label:
...
```

Here the first line tells the compiler to go to or jump to the statement marked as a label. The label is a user-defined identifier that marks the target statement. The statement immediately followed after 'label:' is the destination statement.

The ternary operator of ? : --> this operator is used as a single line replace of if else loop. It has the following general form −

```
exp1 ? exp2 : exp3;
```

where exp1, exp2, and Exp3 are expressions. The evaluation pattern is that first exp1 is evaluated. If it is true, then exp2 is evaluated and that becomes the final value. If exp1 is false, then exp3 is

evaluated and its value becomes the final value of the expression. Let us see its application in a small program. Copy the following code in a file say ternaryOpDemo.cpp.

```
#include <iostream>
#include <string>


#define THRESHOLD 6000
using namespace std;


int main()
{
     int salary;


     // take input from users
     cout << "Enter your monthly salary: ";
     cin >> salary;


     // ternary operator checks if
     // marks is greater than 40
     string result = (salary <= THRESHOLD) ? "below Poverty
line" : "above poverty line";


     cout << "You are " << result << "."<<endl;


     return 0;
}//end main ()
```

Now compile and execute the program. You can see we have replaced the if else statement with ternary operator

(base) 22:08:56:~ % g++ ternaryDemo.cpp
(base) 22:08:57:~ % ./a.out
Enter your monthly salary: 5000
You are below Poverty line.

C++ Functions

Let us look at C++ methods or functions. A function or a method is a block of code that gets executed when it is called. Functions are used to perform certain actions, and they are important for modular programming specially reusing code i.e. define the code once, and use it many times. A function is a set of statements that as a whole accomplish a task. Every C++ program has at least one function i.e. main () which is the entry point of the program. A program is said be following modular approach ifs functionality is split into multiple functions. A function is also called as a method or a sub-routine or a procedure etc.

Most of the programmers split their code into separate functions based on logically dividing a specific task into a number of functions. For e.g in order to write a calculator CLI based application, we need functions to present menu, function to add, subtract, multiple, divide and do modulo operation, function to display result etc.

A function declaration or its signature consists of its name, its return type, and set of parameters passed to it. Its syntax is

```
<return_type> functionName (list of formal parameters with type
and name  like <type> name, <type> name ...);
```

For e.g.

```
int addTwoInt( int &a, int &b);
```

A function definition provides the actual body of the function. A complete function thus looks like this

```
return_type function_name( parameter list )
{
   //body of the function
   ...
}
```

A C++ function definition consists of a function header (the declaration part without semicolon) and a function body. Let us explain each part briefly:

Return Type it is the type of object that the functions returns. It may or may not be present. A function may return a value. In case where the function does its work but has nothing to return, you must use the *return_type* as void.

1. Function Name it is the name of the function by which i is called. The function name and the parameter list together constitute the function signature.

2. Parameters or arguments– it is actually a placeholder for compiler to know what kind of object is passed to the function. A function is invoked by passing it a value as an argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters may or may not be present.

3. Function or method body – it is where the entire logic to achieve the designated task resides for the method. It is a group of statements enclosed in {}.

The C++ standard library provides numerous built-in functions that your program can call. For example, function strcat() to concatenate two strings, function memcpy() to copy one memory location to another location and many more functions.

In order to use a function you need to call it or invoke it by supplying it the required arguments and storing its return value in a return_type variable (if the functions returns something). For e.g consider the following simple program to add two integers. Copy the following code in a sile say simpleAdd.cpp

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


/*
 * Formal function signature.
 * Note parameters passed as const refences so
 * that function does not unintentionally modify them
 */
int addInt( const int &n1, const int &n2 );


int main()
{
    int num1=10;
    int num2=20;
    int sum;

    //Invoke the function sum by passing argument nuk1 and num2 and
store output in sum
    sum = addInt(num1, num2);


    //Display the result
    cout<<"Sum of "<<num1<<" and "<<num2<<" is: "<<sum<<endl;
}


int addInt( const int &n1, const int &n2 )
{
    return (n1+n2);
}
```

Now compile and execute the code

```
(base) 9:09:13:~ % g++ simpleAdd.cpp
(base) 9:09:14:~ % ./a.out
Sum of 10 and 20 is: 30
```

We can specify a default value to any parameter in function definition. This value will be used if the corresponding argument is left blank when calling to the function. It can be done by assigning a value to the parameter using assignment operator. If no value is passed for that parameter in function invoking, the default given value is used, but if a value is specified, this default value is ignored and the passed value is used instead. See the declaration of addInt () below for default value

```
int addIntDefault( const int &n1, const int &n2 = 100 );
```

Let us modify the program simpleAdd.cpp by making use of default values. Copy the following code in a file say simpleAddDefault.cpp

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


/*
 * Formal function signature.
 * Note parameter n2 has default value of 100
 */
int addIntDefault( const int &n1, const int &n2 = 100);


int main()
{
    int num1=10;
    //int num2=20;
    int sum;

    //Invoke the function sum by passing argument nuk1 and num2 and
store output in sum
    sum = addIntDefault(num1);


    //Display the result
    cout<<"Sum of "<<num1<<" and 100 is: "<<sum<<endl;
}


int addIntDefault( const int &n1, const int &n2 )
{
    return (n1+n2);
}
```

Now compile the code and execute. Here we invoke addIntDefault() using default value of 100 for n2 and passing 10 for n1 to get a sum of 110.

```
(base) 9:23:56:~ % g++ simpleAddDefault.cpp
(base) 9:23:59:~ % ./a.out
Sum of 10 and 100 is: 110
```

When a program calls a function, program control is transferred from caller function to the called function. A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the caller. Functions are usually performed using Stack data structure.

The formal parameters or formal arguments (variables that we pass to a function when calling it) are like other local variables inside the function and are created upon entry into the function and destroyed upon exit. We can pass arguments to a function using two ways

1. Call by Value here actual value of an argument gets copied into the formal parameter of the function (a new copy is created via deep copy operation). Since, formal parameter is a new variable, changes made to it inside the function have no effect on the actual argument passed from caller function. We can pass the argument either by value or by pointer. for case of pointer, the method copies the address of an argument into the formal parameter i.e makes a new pointer variable but pointing to the same memory location. Inside the called function, the address is used to access the actual argument used in the call and thus any change done on formal parameter can affect the value of original argument in the caller function.

2. Call by Reference here no deep copy is done but shallow copy is done. It copies the reference of an argument into the formal parameter.

Here actual and formal arguments share the same address space. Thus any change done in called function will get reflected in the caller function.

In this section, let us ponder over numbers in C++. We referent fixed point numbers using in and floating point numbers using float and double depending upon their length and precision needed. C++ uses the + operator for both addition and string concatenation. This is possible due to polymorphism and operator overloading.
The C++ numerics library provides common mathematical functions and types, as well as optimized numeric arrays and support for random number generation. For instance, the header <numbers> part of numerics library provides several mathematical constants, such as std::numbers::pi or std::numbers::sqrt2.
Let us write a sample program to check if a given number is palindrome or not. A palindrome number is a number which when read from reverse side is same as the original number. For e.g 56744765.

Copy the following code in a file say palindrome.cpp.

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


int main()
{
    int i,num,r,s=0;
```

```
    cout << " ------------------------------------------------
-----------------\n";
    cout << "\n\n Program to check whether a given number is
palindrome or not: \n";
    cout << " ------------------------------------------------
-----------------\n";
    cout << "Input a number: ";
    cin>>num;
    for(i=num;i>0; )
    {
        r=i % 10;
        s=s*10+r;
        i=i/10;
    }
    if(s==num)
    {
        cout<<" "<<num<<" is a Palindrome Number."<<endl;
    }
    else
    {
        cout<<" "<<num<<" is a not Palindrome Number."<<endl;
    }
}
```

Now compile and execute the program

```
(base) 9:38:19:~ % g++ palindrome.cpp
(base) 9:38:25:~ % ./a.out
 ----------------------------------------------------------------

 Program to check whether a given number is palindrome or not:
 ----------------------------------------------------------------
Input a number: 12344321
 12344321 is a Palindrome Number.
```

C++ Arrays

An array in any programming language is a collection of data items of same type that are stored at contiguous memory locations and elements can be accessed at random using integer indices within the array. The stored data can be of any type such as primitive data types (int, float, double, char, etc o or derive datatypes such as the structures, pointers etc.).
We need array to avoid declaring individual variables, such as number0, number1, ..., and number99 and instead club them in a single array variable. All arrays have contiguous memory locations with the lowest address corresponds to the first element and the highest address to the last element.

We can declare C ++ array in C++ by specifying the type of the elements and the number of elements required by it.

```
type arrayName [ arraySize ];
```

This is called a 1-D array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type (generic or derived). For example, following line declares a 20-element array called numRecords of type int

```
int numRecords[20];
```

C++ array elements are initialized one at a time or in a single statement as shown below. Please note that number of initializers must not exceed the array capacity.

```
//Array initialization one at a time
for (i=0;i<6;i++)
{
      numRecords[i] = (i+1);
}


//Array initialization in one go
int numRecords[6] = {1,2,3,4,5,6};
```

You can access any array by indexing in the array name with proper index value. This is done by placing the index of the element within square brackets after the name of the array. For example :

```
int currentRecord = numRecords[2];
```

The above statement will take 2nd element from the array and assign the value to currentRecord variable.

Few Advantages of an Array in C++ are as follows:

1. It provides Random access of elements using array index.
2. The program code print decreases as we use less line of code to creates a single array of multiple elements.
3. It gives easy access to all the elements.
4. Let us write a simple code to show all operations on arrays.
5. Array traversal becomes easy using for loop.
6. Sorting and searching becomes easy

However like any other data structure, it also has its share of disadvantages. Array is a non-flexible data structure meaning only fixed number of elements can be entered that is decided at the time of declaration. Unlike a LinkedList, an array size cannot be dynamically adjusted. This makes insertion and deletion of elements costly. If we declare an array of size 20, then the array will contain elements from index 0 to 19. But if we try to access the element at index 10 or more than 10, it will result in Undefined Behaviour.

Multidimensional arrays can be described as "arrays of arrays". For example, a bidimensional array can be imagined as a two-dimensional table made of elements, all of them of a same uniform data type. For e.g.

```
// 2D array to store elements of 3X3 matrix
int mat[3][3];
To access 2nd element from 2nd row, we can write
x =mat[1][1];
```

Arrays can be passed as parameters to function calls by reference. This means that the function can directly access and change the contents of the passed array. While passing a two-dimensional array as a formal parameter, we can omit the size of the first dimension, but must specify the size of second. For example:

```
void displayArray(int Mat[][3],int numRows, int numCols);
```

We can call this function by providing a two dimensional array as follows:

```
int main()
{
      ...
    int arr[4][3];
    ....
    display (arr, 4, 3);
}
```

C++ Strings

Strings are used for storing text characters. A string variable contains one or more characters inside double quotes. C++ uses C style character strings as well as has its own string class type.

The C-style character string has been taken from the C language. This string is nothing but a 1-D array of characters terminated by a null character '\0'. For e.g.

```
char vowels[6] = {'a', 'e', 'i', 'o', 'u', '\0'};
```

The last character in vowels will be '\0' that gets placed by compiler automatically when it will initialize vowels array. So you string would have one space extra in the end. Let us see what happens when we try to make vowels array with 5 characters instead of 6. Save the following cope in a file say checkString.cpp

```
#include <iostream>

using namespace std;

int main () {
```

```
  char vowels[5] = {'a', 'e', 'i', 'o', 'u'};


  cout << "Vowel list: ";
  cout << vowels << endl;


  return 0;
}
```

Now compile and run it.

```
(base) 15:30:13:~ % g++ checkString.cpp
(base) 15:30:19:~ % ./a.out
Vowel list: aeiou
```

C++ supports a wide range of functions that manipulate null-terminated strings. For e.g. strcpy (copy one string to another) , strcat (one string to the end of other), strelen (find length of the string) strcmp (compare two string for equality) etc. The following code shows application of some of these functions.

Copy the following code into a file say CstringFunctions.cpp

```
// Header for cin/cout streamhandling
#include <iostream>
// Header for C type null terminated string functions
#include <cstring>


using namespace std;


int main ()
{


   char strA[10] = "Go";
   char strB[10] = "Corona";
   char strC[10];
   char strFinal[20];
   int  len ;


   // copy strA into strC
   strcpy( strC, strA);
   cout << "strcpy( strC, strA) gives strC as " << strC << endl;


   //Compare two strings
   if (strcmp(strA, strC) == 0)
       cout<<"strA and strC are same"<<endl;
   else
       cout<<"strA and strC are not same"<<endl;
```

```
   // concatenates strFinal with strA and strB and strC with space
in between
   strcat(strFinal, strA);
   strcat(strFinal, " ");
   strcat(strFinal, strB);
   strcat(strFinal, " ");
   strcat(strFinal, strC);


   cout << "strcat operartion on strFinal, strA, strB, str C gives
strFinal as " << strFinal << endl;


   // total length of str1 after concatenation
   len = strlen(strFinal);
   cout << "strlen(strFinal) : " << len << endl;


   return 0;
}
```

Now compile and run it

```
(base) 15:51:26:~ % g++ cStringFunctions.cpp
(base) 15:51:30:~ % ./a.out
strcpy( strC, strA) gives strC as Go
strA and strC are same
strcat operartion on strFinal, strA, strB, str C gives strFinal as Go Corona Go
strlen(strFinal) : 12
```

Let us move to C++ string class that is new addition. To use strings, you must include an additional header file in the source code, the <string> library. string class provides standard container of bytes, with additional features specifically designed to operate with strings of single-byte characters. The string class is an instantiation of the *basic_string* class template that uses char (i.e., bytes) as its character type. It uses its default *char_traits* and allocator types. We can perform many operations on strings such as concatenation, comparison, conversion etc. Let us implement the same program using C++ string. Class and its method. Copy the following code into a file say stringFunctions.cpp

```
// Header for cin/cout streamhandling
#include <iostream>
// Header for C type null terminated string functions
#include <string>


using namespace std;


int main ()
{
```

```
    string strA = "Go";
    string strB = "Corona";
    string strC;
    string strFinal;
    int  len;


    // copy strA into strC
    strC = strA;
    cout << "After copying from strA, strC is " << strC << endl;


    //Compare two strings
    if (strA == strC)
        cout<<"strA and strC are same"<<endl;
    else
        cout<<"strA and strC are not same"<<endl;


    // concatenates strFinal with strA and strB and strC with space
in between
    strFinal = strA + " " + strB + " " + strC;


    cout << "After appending strA, strB and str C, strFinal becomes "
<< strFinal << endl;


    // total length of str1 after concatenation
    len = strFinal.size();
    cout << "strFinal.size() gives len of strFinal as " << len <<
endl;


    return 0;
}
```

Now compile and run the code. You will get the same output

```
(base) 16:01:53:~ % g++ stringFunctions.cpp
(base) 16:01:54:~ % ./a.out
After copying from strA, strC is Go
strA and strC are same
After appending strA, strB and str C, strFinal becomes Go Corona Go
strFinal.size() gives len of strFinal as 12
```

C++ Pointers

C++ is an extension of C programming model. Memory is a sequence of words or bytes that is accessed by an address that is indexed by integers. Modern machines support direct function call stack and input-output operations require special machine instructions or access to ‘‘memory’’ locations with peculiar semantics. Any higher-level language will find using these facilities pretty messy and machine-architecture-specific. C is by far the most successful language providing the programmer with a

programming model that closely matches the machine model. C provides language-level and machine-architecture-independent notions that directly map to the key hardware notions: characters for using bytes, integers for using words, pointers for using the addressing mechanisms, functions for program abstraction, and an absence of constraining language features so that the programmer can manipulate the inevitable messy hardware-specific details.

Before moving to C++ pointers, lets understand pointers in C language. A C array is simply a sequence of memory locations accessed by integer indices. For example:

```
int num [10]; // an array of 10 ints
num[3] = 11; // assign 11 to num[3]
int x = num[3]; // read value in num[3] and assign to x
```

The subscript notation [] is used both in declarations to indicate an array and in expressions referring to elements of an array. A C pointer is a variable that stored the memory address. For example:

```
int *p; // p is a pointer to an int
p = &num[3]; // assign the address of num[3] to p
*p = 14 ; // write 14 to num[3] via pointer p
int ptrVal = *p ; // read from num[3] through p and assign to ptrVal
```

The pointer dereference (‘‘points to’’) notation * is used both in declarations to indicate a pointer and in expressions referring to the element pointed to. C++ built on C's idea of expressions, control structures, and functions. Many C++ tasks such as providing functions as call backs, are performed more easily with pointers. Things like dynamic memory allocation, cannot be performed without them. Every variable is a memory location and every memory location has its address defined that can be accessed using ampersand (&) operator which denotes an address in memory.

A pointer in short is a variable holding memory address of another variable. It can be declared as type *variablename;

```
int *ptr; //pointer to an integer variable
float *fp; //pointer to a floating point number
char *pch; //pointer to a character variable
```

We assign memory address of a variable to a pointer using & operator and access the value at memory location stored in a pointer by * operator. The following code shows working of pointers. Save the code in a file say pointerDemo.cpp

```
#include <iostream>

using namespace std;


int main () {
   int  count = 20;   // actual variable declaration.
   int  *iptr;        // pointer variable
```

```
   iptr = &count;       // store address of var in pointer variable


   cout << "Value of counter variable: ";
   cout << count << endl;


   // print the address stored in ip pointer variable
   cout << "Address stored in iptr variable: ";
   cout << iptr << endl;


   // access the value at the address available in pointer
   cout << "Value of *iptr variable: ";
   cout << *iptr << endl;


   return 0;
}
```

Now run compile and run the program
(base) 22:48:10:~ % g++ pointerDemo.cpp
(base) 22:48:12:~ % ./a.out
Value of counter variable: 20
Address stored in iptr variable: 0x7ffee14227a8
Value of *iptr variable: 20

You see the value of counter can be accessed by variable and pointer both and address of counter variable is a hexadecimal 0x7ffee14227a8. It is always a good idea to initialize a pointer variable with NULL. Null pointer is a constant with a value of zero defined in several standard libraries.

Now we see how we pass pointers to a function. Save the following code in a file say pointerAsArg.cpp.

```
#include <iostream>


using namespace std;


int add( int *ptr1, int *ptr2);


int main ()
{
   int num1, num2;
   cout << "Enter first number: ";
   cin >> num1;
   cout << "Enter second number: ";
   cin >> num2;
   int *ptr1=NULL, *ptr2=NULL;
   ptr1 = &num1;
```

```
   ptr2 = &num2;


   int result;
   result = add(ptr1, ptr2);


   // print the sum of num1 and num2
   cout << num1 <<" + "<<num2<<" = "<<result;
   cout << endl;


   return 0;
}


int add( int *ptr1, int *ptr2)
{
   return (*ptr1 + *ptr2);
}
```

Now compile the code and run it. This code takes two integers as input from user, pass them as pointers ta local function that adds them and return their sum.

```
(base) 22:58:15:~ % g++ pointerAsArg.cpp
(base) 22:58:20:~ % ./a.out
Enter first number: 12
Enter second number: 13
12 + 13 = 25
```

C++ References

A reference variable is a special feature in C++. It is an alias i.e. another name assigned to an already existing variable. After you assign a reference to a variable, either the variable name or the reference name may be used to refer to the variable. It is created with the & operator. Its syntax is

```
type variablename;
....
type &referencename = variablename;
```

For e.g.

```
string welcomeStr = "Welcome to Jio";  // Welcome string of an
app
string &alias2WelcomeStr = welcomeStr;    // reference to
Welcome string
```

References serve 4 prime purpose in C++.

1. You can change the passed parameters in a functio call. If a function receives a reference to a variable, it can modify the value of the variable and it will ger reflected in the parent function. For example, see the following code to swap the values of a and b. The swap function modifies a and b since they are passed as references

   Save the following code in a file say swap.cpp.

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

void swap (int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int a, b;
    cout<<"Enter a: ";
    cin>>a;
    cout<<"Enter b: ";
    cin>>b;
    cout << "Before swap, a: " << a << " and b: " << b<<endl;
    swap( a, b );
    cout << "After swap, a: " << a << " and b: " << b<<endl;
    return 0;
}
```

   Now compile and run the code to see values of a and b change by calling swap() function.

   (base) 23:30:59:~ % g++ swap.cpp
   (base) 23:31:02:~ % ./a.out
   Enter a: 10
   Enter b: 20
   Before swap, a: 10 and b: 20
   After swap, a: 20 and b: 10

2. Reference help us avoid making a copy of large structures: If we have to pass a large object as an argument to a function and that function will not be changing this object, then passing it in any way will cause a new copy to be created which waste memory, storage and cpu. By using references, we can avoid this wastage. See the following code snippet

```
struct StudentRecord
{
   int rollNumber;
   string name;
   string Homeaddress;
   string AadharNumber;
   .....
};

// If we remove & in below function, a new
// copy of the student object is created.
// We use const to avoid accidental updates
// in the function as the purpose of the function
// is to Display s only.
void Display(const Student &s)
{
    cout<<"Student record: "<<endl;
    cout << s.name << "  " << s.Homeaddress << "  " <<
s.rollNumber<<"  " << s.AadharNumber;
}
```

3. We can use references in a loop to modify all elements in one go. See the following code snippet and copy it in a file say referenceLoop.cpp

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<float> result{ 10.1, 20.2, 30.3, 40.4, 50.5 };

    // We can modify all elements in
    // one go if we use reference
    for (float &x : result)
        x = x + 5.05;

    // Print each element
    for (float x : result)
       cout << x << " ";

    cout<<endl;
    return 0;
}
```

Now compile and execute it. You will get all 5 elements of vector changed.
(base) 23:51:28:~ % g++ referenceLoop.cpp
(base) 23:55:06:~ % ./a.out
15.15 25.25 35.35 45.45 55.55

4. We can use references in each loop to avoid creating a copy of individual objects. This is handy when objects are large.

Most of the young programmers get confused between references and pointers. There are 3 major differences between references and pointers. One, we have a NULL pointer but there is no NULL reference. A reference must always be associated with a previously declared variable. Second, reference is use once object i.e. once you assign a reference to an object, you cannot reassign it to another object. Pointers, on the other hand, can be changed to point to another object at any pint of time. Third, reference must be initialized at its creation time whereas pointers can be initialized at any time.

C++ Date & Time

There is no separate date type in C++ standard library. It inherits the structs and functions for date and time manipulation from C. We need to include ctime header file to access date and time related functions and structures. There are four time-related types: clock_t, time_t, size_t, and tm. The types - clock_t, size_t and time_t are capable of representing the system time and date as some sort of integer.

In C++ 11 you can use std::chrono::system_clock::now() to get current system date and time. Let us see this in an example. Copy this code in a file called computeTime.cpp. This code finds how much time system actually takes to calculate square of first 100 natural numbers.

```
#include <iostream>
#include <chrono>
#include <ctime>
#include <time.h>
#include <string.h>


int main()
{
    int i;
    int sq;
    struct tm tm;
    auto start = std::chrono::system_clock::now();
    for (i=1; i<=100; i++)
      sq = i * i;
    auto end = std::chrono::system_clock::now();


    std::chrono::duration<double> elapsed_seconds = end-start;
    std::time_t end_time = std::chrono::system_clock::to_time_t(end);


    localtime_r(&end_time, &tm);
    char CharLocalTimeofUTCTime[30];
    strftime(CharLocalTimeofUTCTime, 30, "%Y-%m-%dT%H:%M:%SZ", &tm);
    //std::string strLocalTimeofUTCTime(CharLocalTimeofUTCTime);
    std::cout << "finished computation at " << std::ctime(&end_time)
```

```
                << "elapsed time: " << elapsed_seconds.count() << "s\n";
}
```

Now compile and execute this code.

```
(base) 0:28:05:~ % ./a.out
finished computation at Sat Sep 25 00:28:06 2021
elapsed time: 1e-06s
```

The tm structure is very important while working with date and time both C and C++. Most of the time related functions makes use of tm structure. In the previus code snippet, we have used tm strucuture to convert time from UTC timezone to IST time zone.

C++ Input and Output

In this section, we will see look into the Input/output capabilities in C++. The C++ standard libraries provide rich I/O capabilities (Input output). This section will touch upon very basic I/O operations that are commonly used in C++ programming.

C++ input output is associated with streams. Streams are sequences of bytes that flow in some direction. For input, bytes flow from a device like a keyboard, a disk drive, or a network connection etc. to main memory and for output, bytes flow from main memory to a device like a display screen, a printer, a disk drive, or a network connection, etc. Some common headers used for i/o are given below

1. <iostream> it provides cout, cin and cerr objects corresponding to standard output stream, standard input stream and standard error stream, respectively.
2. <iomanip> it provides methods to format I/O, such as setprecision and setw.
3. <fstream> it provides methods to manage file processing.

Let us discuss standard i/o objects such as cout, cin, cerr and clog. The object cout is an instance of ostream class defined in iostream. The cout object is represent standard output device i.e. console. The cout is used in conjunction with the stream insertion operator (<<) to display any object on to screen. For e.g. consider the code snippet below. It will output "Welcome to C++" on screen" when executed.

```
#include <iostream>
#include <string>

int main()
{
   string str = "Welcome to C++";
```

```
    cout << "Str is " << str << endl;
}
```

cin in an object of stream class. The cin object represents the standard input device i.e. the keyboard. The cin is used in conjunction with the stream extraction operator (>>) to take any input from user via keyboard. For e.g

```
int x;
cin>>x; //this line will take an integer as user input and store
it in x
```

cerr is an instance of ostream class. The cerr object is connected to standard error device which in most cases is console. The object cerr is un-buffered and each stream insertion to cerr flushes its output immediately instead of storing it in a buffer. For e.g. the following code will display the error string when this program is executed.

```
#include <iostream>
#include <string>


int main()
{
    string errorStr = "Generic error occured";

    cout << "Gener error string is " << errorStr << endl;
}
```

clog is an instance of ostream class. The clog object is connected to the standard error device i.e. the console. Please note that clog is buffered internally with each insertion to clog causes its output to be kept in a buffer until the buffer is filled or until the buffer is flushed.

C++ Data Structures

We use Data structure in C and C++ to store heterogeneous data i.e. data of different kind. Consider all data associated with a book kept in a library. The librarian will associate following attributes to a book

```
name
author
publisher
year
ISBN number
subject
type
```

We can defined all these in a data structure as shown below and also use typedef to create an alias for it:

```
enum Type {paperback, hardcopy, ebook};
struct bookRecord
{
  string name;
  string author;
  string publisher
  int year;
  string ISBN_Number;
  string subject;
  Type t;
};

typedef struct bookRecord BOOK;
```

The formal syntax to define a structure in C++ is

```
struct [structure tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

You can access any member of a structure by member access operator (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You can pass a structure as a function argument in same way as you pass any other variable or pointer.

You can define pointers to structures in very similar way as you define pointer to any other variable as follows. For e.g.

```
BOOK bookSc
BOOK *bk;
```

You can assign address of a struct to pointer of its kind as shown below:

```
bk = &BookSc;

to access any field in BookSc, we can now use -> opertor on
pointer
bk->name; //To get access to name of BookSc
```

You can create alias to any data type in C++ by using keyword typedef. We have already used it above to create alias for Book Record.

Let us combine all these concepts in a program for practical understanding. Save the following code in a file say demoStruct.cpp.

```
#include <iostream>
#include <string>

using namespace std;


enum BookType {paperback, hardcopy, ebook} type;


struct bookRecord
{
    string title;
    string author;
    string publisher;
    int year;
    string ISBN_Number;
    string subject;
    BookType type;
};


typedef struct bookRecord BOOK;


void displayBook(const BOOK *bookptr );


int main()
{
   BOOK Book1;        // Declare Book1 of type Book
   BOOK Book2;        // Declare Book2 of type Book

   // Create Book 1 record
   Book1.title =  "5G Overview";
   Book1.author =  "William Stalling";
   Book1.subject = "Wireless";
   Book1.publisher = "Pearson";
   Book1.ISBN_Number = "1234567";
   Book1.year = 2021;
   Book1.type = hardcopy;


   // Create Book 2 record
   Book2.title = "Let us C";
   Book2.author = "Yashwant Kanitkar";
   Book2.subject = "Programming";
   Book2.publisher = "BPB";
   Book2.ISBN_Number = "3434343";
```

```
   Book2.year = 2020;
   Book2.type = paperback;

   // Print Book1 info, passing address of structure
   displayBook( &Book1 );


   // Print Book2 info, passing address of structure
   displayBook( &Book2 );


   return 0;
}//end main()


// This function accept pointer to structure as parameter.
void displayBook(const BOOK *book )
{
   cout << "Book title : " << book->title <<endl;
   cout << "Book author : " << book->author <<endl;
   cout << "Book subject : " << book->subject <<endl;
   cout << "Book publisher : " << book->publisher <<endl;
   cout << "Book ISBN number : " << book->ISBN_Number <<endl;
   switch (book->type)
   {
      case paperback:
          cout << "Book Type : paperback"<<endl;
          break;
      case hardcopy:
          cout << "Book Type : hardcopy"<<endl;
          break;
      case ebook:
          cout << "Book Type : ebook"<<endl;
          break;
   }
   cout<<endl;
}///end printBook()
```

Now compile and run the code. You can see it displays two book records we created in main().

```
(base) 9:09:39:~ % g++ structDemo.cpp
(base) 9:09:41:~ % ./a.out
Book title : 5G Overview
Book author : William Stalling
Book subject : Wireless
Book publisher : Pearson
Book ISBN number : 1234567
Book Type : hardcopy


Book title : Let us C
Book author : Yashwant Kanitkar
Book subject : Programming
Book publisher : BPB
```

Book ISBN number : 3434343
Book Type : paperback

C++ classes and objects :-

Class is a data type that is user defined which holds its own data member and member function. In other words we can say class is collection of data member and member function which can accessed and use by creating object of that class.
A class represents group of similar objects. It is also a way to bind data describing an entity and its associated function together.
Syntax:-

```
class classname
{


};
```

Classes are needed to represent real world entities. To describe a real world entity we need two different things. One is characteristics and the other is function.

Class definition :-

```
class classname{
    private:
    …
    …
    protected:
    …
    …
    public:
    …
    …
    };
    By default class members are private.
```

Class Method's definition :-
Member functions are often defined in two places:

i. Outside class definition

Syntax:

```
class-name::function-name
```

ii. Inside class definition

Example:-

```
class area
{
public:
    int getarea( )
    {
```

```
        return side*side;
    }
};
```

Object :-
An object is an instance of a class. Whenever class is defined, no memory is allocated but when object is initialized memory is allocated of that class.
Once a class has been created then we can create variable of that type (class type) by using following syntax which is named object.
Syntax:-

```
class_name variable_name;
```

We can create any number of objects belonging to that class by declaring one object in one statement. This statement are written in main( ) function.
The objects can also be defined by putting their name directly after the closing brace of the class.

C++ Inheritance :-

It is a procedure of inheriting properties and behaviours of existing class into a new class.
The prime advantage of inheritance is code reusability.
In the concept of inheritance, the existing class is called as the base class and the new class which is inherited is called as the derived class.
The idea of inheritance achieves the IS - A relationship. For example mammal IS- A animal and cat IS- A mammal hence cat IS- A animal as well and so on.
We can save our time and money by reusing classes. And also it is faster development and easy to extend. It is capable of showing the inheritance relationship and its transitive nature which provides closeness along with actual world problems.

Need of Inheritance :-
1. It is used to represent real world relationship.
2. It provides reusability.
3. It supports transitivity.

There are five types of inheritance :-
1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

Single Inheritance :- In single inheritance a subclass inherits from one base class.

Syntax :-

```
class A
{
…
};
class B : public A
{
…
};
```

Multilevel Inheritance :- It is transitive in nature. In this type of inheritance we'll deriving a class from already derived class.

Syntax :-

```
class A
{
…
};
class B : public A
{
…
};
class C : public B
{
…
};
```

Multiple Inheritance :- In multiple inheritance a subclass inherits from multiple base classes.

Syntax :-

```
class A1
{
…
};
class A2
{
…
};
class B : public A1, public A2
{
…
};
```

Hierarchical Inheritance :- In hierarchical inheritance many sub class inherits from single base class.

Syntax :-

```
class A
```

```
{
…
};
class B1 : public A
{
…
};
class B2 : public A
{
…
};
```

Hybrid Inheritance :- Hybrid inheritance is the combination of two or more forms of inheritance.

Syntax :-

```
class A
{
…
};
class B : public A
{
…
};
class C
{
…
};
class D : public B, public C
{
…
};
```

C++ Overloading :-

In C++ overloading occurs when an equivalent operator or function name is used with different signatures.

Both operators and functions can be overloaded. Various definitions must be distinguished by their signatures (otherwise which to call is ambiguous).

- Note that signature is the operator or a function name and the ordered list of its argument types.
- E.g. sum(int, long) and sum(long, int) have different signatures.
- E.g. add(const Base &) and add(const Derived &) have different signatures even if Derived is -a Base.

- Most specific match is employed to select which one to call.

Tips on overloading :-
1. Use virtual overriding when you wish to substitute different subtypes polymorphically.
2. Use overloading when you wish to provide related interfaces to similar abstractions.
3. Use different names when the abstraction differ.

Operator Overloading :-
To assign more than one operation on a same operator known as operator overloading.
To achieve operator overloading we have to write a special function known as operator( ).
Syntax :-

```
return_type operator op (arg list)
{
    Body;
}
```

You can write operator( ) in two ways -
1. Class function
2. Friend function

Following is the list of operators that can't be overloaded-
1. .
2. ::
3. ?:
4. sizeof( )

There are mainly two sorts of operator overloading :
1. **Unary Operator Overloading** :- An operator which contain only one operand is called unary operator overloading.
2. **Binary Operator Overloading** :- An operator which contain two operands is called binary operator overloading.

Function Overloading :-
Function overloading is a feature of C++ where we have more than one function with the same name and different types of parameters then it is known as function overloading.

C++ Polymorphism :-

Polymorphism is derived from two Greek words i.e. poly and morphs and poly means 'many' , morphs means 'forms'.
The technique of representing one form in multiple forms is called as polymorphism. Here one form represent original form or original method always consist in base class and multiple forms represents overridden method which consist in derived classes.
We can take an example to understand this -
One person have different different behaviors like suppose if you are in class room that time you behave like a student but when you are in market at that time you behave like a customer.

Basically there are two sorts of polymorphism :

1. **Static polymorphism** :- Static polymorphism is also known as early binding and compile time polymorphism. In static polymorphism memory are going to be allocated at compile time.

The overloaded member functions are selected for calling by matching arguments both type and number. This information is understood to the compiler at the compile time and is able to pick the appropriate function for a particular call at the compile time itself.

2. **Dynamic polymorphism** :- Dynamic polymorphism is also known as late binding and run time polymorphism. In dynamic polymorphism memory are going to be allocated at run time.

Virtual function :- When we use an equivalent function name in both the base and derived classes, the function in base class is stated as virtual using the keyword virtual preceding its normal declaration.

When a function is formed virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer rather than type of the pointer.

C++ Abstraction :-

Abstraction in C++ isn't new concept it is oops concept which is employed to hide background detail and show only essential features. Data abstraction may be a procedure of representing the essential features without including implementation details.

The main advantage of abstraction is it reduces code complexity.

Abstract class :- Abstract class is a class which contains a minimum of one Pure Virtual function in it.

Some of the characteristics of abstract class are :

- Abstract class can't be instantiated but pointers and references of abstract class type are often created.
- Abstract class may have normal functions and variables in conjunction with a pure virtual function.
- Abstract classes are usually used for up - casting in order that its derived classes can use its interface.
- The classes that inheriting an abstract class should carry out all pure virtual functions alternatively they're getting to become abstract too.

C++ Encapsulation :-

- A language tool for restricting access to a number of the object's components.

- Encapsulation is a type of process of combining data and functions into one unit which is understood as class.
- Data is merely accessible through the functions present inside the class.
- The process of hiding the implementation details and providing restrictive access results in the concept of abstract data type.
- Data encapsulation led to the relevant concept of data hiding.

Why encapsulation ?

- It gives us secure and consistence results. It means that it gives the user access to a limited data and keeps our valuable data which can change our program or increase the possibilities of mistakes hidden from the user.
- Using encapsulation we can create a function that if a program wanted to vary the length of the rectangle for instance then that the object would appropriately update its area and perimeter without being inconsistent.
- The ideal is to stay as many of the details of each class hidden from all other classes as possible.

Syntax :-
The syntax of encapsulation is extremely easy . For encapsulation we've to declare a class and then we've to mark or tell in our program which data we want to show and which we want to hide.

```
class A
{
public:
double getVolume (void)
{
return length*breadth*height;
}
private:
double length;
double breadth;
double height;
};
```

Advantages and achievements of encapsulation :-

- Makes maintenance of application easier.
- Improves the understandability of the application.
- Enhanced security.

C++ Interfaces :-

Interfaces are almost like abstract classes but differ in their functionality.
In interfaces none of the functions are implemented which means interfaces defines functions without body.
Interfaces are syntactically almost like classes but they lack instance variables and their methods are declared with no body. But it can have final variables which must be initialized with values.
Once it is defined then any number of classes can implement an interface. A single class can implement any number of interfaces or more than one interfaces.
If we are implementing an interface in a class we must implement all the methods defined within the interface as well as a class can also implement its own methods.
Interfaces add most of the functionality that is required for several applications which might normally resort to using multiple inheritance in C++.
The main purpose of interfaces are reusability and extensibility.

C++ Advanced

C++ files and streams :-

- So long we are using iostream standard library which gives us cin and cout methods for reading from standard input and writing to plain output respectively.
- To read and write from a file we need another standard C++ library called fstream which characterizes three new data types:
  - Ofstream
  - Ifstream
  - Fstream

FSTREAM :-

1. Ofstream :- This type of data represents the output file stream and is utilized to create files and to write down information to files.
2. Ifstream :- This type of data basically represents the input file stream and is employed to read information from files.
3. Fstream :- It addresses the file stream generally and has the ability of both ofstream and ifstream which infers it can create files then write information to files and read information from files.

Note:- To perform file processing in C++ main header files <iostream> and <fstream> must be included in your C++ source file.

OPENING A FILE :-

- A file should be opened before you will read from it or write to it. Each of two the ofstream or fstream object could even be utilized to open a file for writing however the ifstream object is used to open a file for reading purpose only.
- This is the standard syntax for open ( ) function :
  void open (const char *filename, ios::openmode mode);
- The first argument decides the name and site of the file to be opened and then the second argument of the open ( ) member function characterizes the mode wherein the file should be opened.
- You can join two or more of those values by ORing them together. For instance suppose you need to open a file in write mode and want to truncate it in case it already exists. And following will be the syntax :
  - ofstream outfile;
    outfile.open ("file.dat", ios::out | ios::trunc);
  - fstream afile;
    afile.open ("file.dat", ios::out | ios::in);

CLOSING A FILE :-

- When a C++ program closes it consequently closes flushes all the streams then freeing all the allocated memory and shut every one of the accessible files. But it is consistently an incredible practice that a programmer should close all the opened files before program termination.
- This is the standard syntax for close ( ) function
  void close ( );

WRITING TO A FILE :-

Although doing C++ programming you will write information to a file from your program using the stream insertion operator (<<) similarly as you employ that operator to output information to the screen. The main difference is that you simply use an ofstream or fstream object rather than the cout object.

READING FROM A FILE :-

You read information from a file into your program utilizing the stream extraction operator (>>) similarly as you use that operator to input information from your keyboard. The main difference is that you use an ifstream or fstream object rather than the cin object.

C++ Exception Handling :-

- The process of changing system error messages into user friendly error message is termed as exception handling. This is one of the amazing feature of C++ to handle run time error and maintain normal flow of C++ application.
- An exception is an event which happens during the execution of a program that spoil the traditional flow of the programs instructions.
- Exceptions are runtime deviation that a program encounters during execution. It is a circumstance where a program has an unusual condition and therefore the section of code containing it can't handle the problem /matter. Exception involves situations like division by zero or accessing an array outside its bound or running out of memory then so on.

In order to handle these exceptions the exception handling mechanism is used which identifies and affect such condition.

Exception handling system consists of following parts :

- Find the problem (Hit the exception)
- Inform about its occurrence (Throw the exception)
- Receive error information (Catch the exception)
- Take proper action (Handle the exception)

Why exception handling ?

1. Segment of error handling code from normal code : In traditional error handling codes there are consistently if else conditions to deal with errors. These conditions and the code to handle errors get involved with the normal flow. This makes the code less comprehensible and maintainable. The code for error handling becomes separate from the normal flow with try catch blocks.
2. Functions or Methods can deal with any exceptions they pick : A function can throw various exceptions but may decided to handle a number of them. The other exceptions which are thrown but not caught can be managed by caller. But if the caller picks to not catch them then the exceptions are handled by caller of the caller. In C++ a function can define the exceptions that it throws using the throw keyword. The caller of this function should deal with the exception during a number of way (either by specifying it again or getting it).
3. Grouping of error types : In C++ both basic types and objects are often thrown as an exception. We can make an order of exception objects and group exceptions in namespaces or classes then categorize them according to their types.

Generally we use three keywords for handling the exception in C++ language :

1. try
2. catch
3. throw

Try :-

- Try block consists of the code that might create exception. The exceptions are thrown from inside the try block.
- try keyword represents a block of code that can throw an exception.
- The try block groups atleast one program statements with one or more catch clauses.

Throw :-

- Throw keyword is used to throw an exception come across inside try block. Later the exception is thrown the control is transferred to catch block.
- Raising of an exception is done by “throw” keyword.

Catch :-

- Catch block catches the exception which is thrown by throw statement from try block. Then the exception are handled inside catch block.
- The catch block usually defines the action to be taken when an exception occur.

Syntax of exception handling :-

```
try
{
     statements;
     …
     throw exception;
}
catch (type argument)
{
     statements;
     …
}
```

Multiple Catch Exception :-

Multiple catch exception statements are used when a user wish to handle different exceptions in a different manner. For this a programmer must include catch statements with different declaration.

Synatx:-

```
try{
body of try block
}
catch (type1 argument1){
statements;
…
}
catch (type2 argument2){
statements;
```

```
…
}
…
…
catch (typeN argumentN){
statements;
…
}
```

Catch all exceptions :-
Sometimes it may not be feasible to design a separate catch block for each kind of exception. In such cases we can use a single catch statement that catches all types of exceptions.

C++ Dynamic Memory :-

- Every program needs memory to be allocated to it for many purposes.
- When you write a program you mention what storage it needs by declaring variables and instances of classes e.g.
  int a,b,c;
  float nums[100];
  Circle myCircle (2.0,3,3); etc..
- Then when the program executes it can utilize this memory.
- It is impossible for variables or objects to be added during the execution of a program.

Programs and Memory :-

- In order to execute a program it must be loaded into Random Access Memory i.e. RAM.
- A certain amount of RAM is allocated as the fixed storage for your program and this is where the global variables are stored.
- There is a dynamic region of memory called the stack is where the local variables are stored.
- And this storage is fixed at compile time.

Remember that global variables hold their value for the lifetime of the program while local variables only hold their value for the lifetime of the function they are declared in.

- However, there are times when it is mandatory for a program to make use of variable amounts of storage.
- For example you may want to write a program that reads in a list of numbers (until 999 is entered) and wish to finds their average.
- You may wish to store the numbers in an array but of what size?
- Each time the program is run the user may enter any number of numbers maybe 5, 50 or 5000 or any other.

- One solution is to declare an array of the utmost size that it could ever be.

Memory is divided in certain segments like code segment , data segment and stack segment. In addition to these the operating system gives and extra amount of memory called heap.

Dynamic variable :-

- Stored in heap and outside the stack and data segments.
- It may change size during runtime or may disappear for good.
- If no explicit allocation of memory is done means it never exists.
- To create the dynamic variable new keyword is used.
- It is always a best practice to initialize dynamic variables.

Deleting Dynamic Variable :-

- delete operator is employed to delete the memory allocated for the dynamic variable in heap.
- delete dp;

It will delete 8 bytes of the memory allocated for the double variable.

Note that the dp pointer will not be erased as it is created in code or data segment and may be point some other dynamically allocated memory location.

C++ Namespaces :-

- If a program uses classes and functions written by different programmers then it is going to be that the same name is used for different things.
- Namespaces help us to deal with this problem.
- A namespace may be a declarative region that maintains a scope to the identifiers (the names of types, functions, variables etc) inside it.
- It is employed to arrange code into logical groups and to protect name collisions which can occur especially when our code base includes multiple libraries.
- Namespace gives us a class like modularization without class like semantics.
- It obliviates the use of File Level Scoping of C (file) static.

Syntax:

```
namespace Name_Space_Name
{
    Some_Code
```

```
    int x=3;
}
```

There are some features of namespace :

1. Nested namespace
2. using namespace
3. Global namespace
4. Standard library std namespace
5. Namespaces are open

C++ Templates :-

Templates in C++ programming allows function or class to figure on more than one data type directly without writing different codes for various data types. Templates are generally used in larger programs for the objective of code reusability and flexibility of program. The concept of templates are often utilized in two different ways:

- Function templates
- Class templates
- The variable template has also been added in C++ 11.

Templates allow programmer to make a common or standard class or function that can be used for a variety of data types. The parameters used during its definition is of generic type and may get replaced later by actual parameters. This is referred to as the concept of generic programming. The main advantage of applying a template is that the reuse of same algorithm for various data types and hence saving time from writing similar codes.
For example consider a situation where we've to sort a list of students consistent with their roll number and their percentage. Since roll number is of integer type and percentage is of float type then we require to write down separate sorting algorithm for this problem. But through template we can define a generic data type for sorting which can get replaced later by integer and float data type.

Function Templates :-

- A function templates work in identical manner as function but with one key difference.
- A single function template can work on different types at once but different functions are required to perform identical task on different data types.
- If you would like to perform identical operations on two or more types of data then you can use function overloading. But better approach would be to use function templates because you'll perform this task by writing less code and code is simpler to maintain.
- A generic function that represents many functions performing same task but on different data types is understood as function template.
- For instance a function to add two integer and float numbers requires two functions. One function accept integer types and therefore the other accept float types as parameters

even though the functionality is the same. Using a function template a single function are often used to perform both additions. It avoids irrelevant repetition of code for doing same task on various data types.

Why use Function Templates :-

- These templates are instantiated at compile time with the source code.
- Templates are used lower code than overloaded C++ functions.
- Templates are type safe.
- Templates allow user defined specialization.
- Templates allow non type parameters.

Class Templates :-

A class template is a common class that can represent various similar classes operating on data of different types and it is just like function templates. Once a class template is specified then we may create an object of that class by utilizing a specific basic or user defined data types to replace the generic data types used during class definition.
Syntax :

```
template <class T1, class T2,..>
class classname
{
attributes;
methods;
};
```

Template Instantiation :-

- When the compiler provokes a class or function or static data members from a template then it is referred to as template instantiation.
- A class generated from a class template is known as a generated class.
- A function generated from a function template is known as generated function.
- A static data member generated from a static data member template is known as generated static data member.
- The compiler generates a class or function or static data members from a template when it sees an implicit instantiation or an explicit instantiation of the template.

C++ Preprocessor :-

- A preprocessor directive is nothing but a program.

- It makes C++ highly portable i.e the program, written in one machine can be transferred to another.
- During compilation it forms a duplicate image of the original program.
- User can create their own preprocessor directive usually the pre - processor can be placed before the main function but as required can be placed after the main function.
- These are the special instructions for the compiler. This should not be terminated by semicolon.

There are many sorts of pre- processor :
1. # inclusion
2. # macro declaration
3. # error generation
4. # pre defined names
5. # conditional compiler
6. # programs

Include :- To add the given header files into the program file.

Header File :- It is a programming file which contains the definitions and declaration of the library functions.
It contains the declaration of given predefined function. Also user can create their own header file.

C++ Signal Handling :-

Signals are the interrupts that hand over to a process by the operating system which can terminate a program prematurely. You can create interrupts by pressing Ctrl+C on a UNIX or LINUX or Mac OS X and Windows system.
There are the signals which may not be captured by the program but there is a following list of signals which you can catch in your program and may take appropriate actions based on the signal. These signals are described in C++ header file <csignal>.
There are some of the signals in C++ :-
1. SIGABRT :- Unexpected termination of the program such as a call to abort.
2. SIGFPE :- An invalid arithmetic operation for instance divide by zero or an operation resulting in overflow.
3. SIGILL :- Detection of an illegal information.
4. SIGINT :- Receipt of an associated attention signal.
5. SIGSEGV :- An invalid access to storage.
6. SIGTERM :- A termination request gone to the program.

C++ Multithreading :-

You have the main thread suppose but there are going to be some other threads working concurrently in the same process.

- thread object is used by the C++ standard library to form the thread management tasks easier
  - Launching threads
  - Checking if they are finished
  - Keeping an eye on them
  - The library has many functionalities
- Every C++ program has a minimum of one thread
  - A thread running main ( ) started by the C++ runtime.
    - We will call it the main thread
    - You do not need to do something special to run the main thread, when you run the program it automatically starts
  - Main thread can spawn additional threads that have another function as the entry point
    - Starting function of the thread is the entry point
  - All these threads run concurrently with one another
    - via a schedule organized by the OS; the programmer cannot directly control this schedule

Why multithreading ?

- We require concurrency and multitasking in some cases.
- A typical application has many tasks
  - The real computation and data processing- Sometimes several objects performing on an equivalent shared resource in parallel.
  - Connecting with the user via a Graphical User Interface (GUI).
  - Input / output (I/O) operations.
  - These all tasks are better to be handled concurrently.
- Starting from the main thread you can create “child” threads
  - Child thread : The spawned thread
  - Parent thread : the thread that spawned

C++ Web Programming :-

CGI stands for Common Gateway Interface may be a set of standards that outline how information is interchanged from the web server and passing the web users request to an application program and to pick up data back to the user. When any user call for a web page then the server sends back the requested page. The Web server typically passes the shape information to a small application program that processes the data and may send back a approval message. This method or convention for passing data back and forth between the server and thus the appliance is called as the common gateway interface (CGI) and it’s a component of the Web's Hypertext Transfer Protocol (HTTP).

Browsing the web :-

For knowing the concept of Common Gateway Interface let us take a view at the scenario that takes place when users browse something on the web using a specific URL.

1. The browser you are using contacts the HTTP web server and demands for the URL.
2. The web server will parse the URL and will search for the file name; if the requested file is found, immediately sends back that file to the browser or else sends an error message.
3. The web browser takes the response from a web server and displays either file received or an error message.

If you're developing an internet site and you are required a Common Gateway Interface application to regulate then you can specify the name of the application within the URL which means uniform resource locator that your code in an HTML file.

Server Configuration :-

Before using the CGI programming it is mandatory that programmers should make sure that the Web server supports CGI and is well configured for handling CGI programs. By convention CGI files will have an extension as .cgi though they are C++ executable. Apache Web Server is configured to run CGI programs in /var/www/cgi-bin by default. Developers need to have a web server up and running in order to run any CGI program like Perl or shell and so forth.

Example of CGI program :

```
#include <iostream>
void main ( )
{
cout << "Content-type:text/html\r\n\r\n";
cout << "<html>\n";
cout << "<head>\n";
cout << "<title>Hello Tutorials </title>\n";
cout << "</head>\n";
cout << "<body>\n";
cout << "<h3> <b> First CGI program </b> </h3>\n";
cout << "</body>\n";
cout << "</html>\n";
}
```

We compile the above program and give this executable a suitable name along with the extension .cgi. This file requires to be kept in /var/www/cgi-bin directory and it has following content. Prior to running your CGI program ensure that you have change mode of the file

using chmod 755 cplusplus.cgi UNIX command to make the file executable. The above C++ program composes its output on STDOUT file that is on the screen. There are some other HTTP headers which are generally used in CGI programs and they are:

1. **Content-type**: It is a MIME string which defines the pattern of the file being returned.
2. **Expires: Date**: It defines the date the information of the current web page becomes false.
3. **Location: URL**: The URL that has to be returned rather than the URL that is requested.
4. **Last-modified**: Date: This is the date of last modification of the resource.
5. **Content-length**: N: The length in bytes of the data being returned. This value 'N' is used by the browser to report the approximated download time.
6. **Set-Cookie**: String: It is used to set the cookie passed through the string.