Unit: I

What is c++?

- C++ is a cross platform language that can be used to create high performance applications.
- C++ was developed by Bjarne stroustrup as an extension to the c language.
- C++ gives programmers a high level of control over system resources and memory.
- C++ supports object oriented programming. The 4 major pillars of oops used in c++ are Inheritance, Polymorphism, Encapsulation, Abstraction.

Why do we use c++?

- C++ is one of the world's most popular programming language.
- C++ is fun and easy to learn.
- C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
- This language allows developers to write clean and efficient code for large application and software development, game development and operating system programming.

Syntax:

```
#include<iostream>
using namespace std;
int main()
{
cout<<"Hello world!";
return 0;
}
```

iostream: iostream header file library which stores the definition of the cin and cout methods that we have used for input and output. #include is a preprocessor directive using which we import header files.
namespace: we can use names for object and variable from the standard library.
cout: It uses output/ print text.

Data Types in C++

c++ supports a wide variety of data type and the programmer can select the data type appropriate to the needs of the applications. Data types specify the size and the types of values to be stored.

C++ Supports the following data types

- Primary or Built-in or Fundamental Data Type
- Derived Data Type
- User-Defined Data Type

Primitive Data Types

These data types are built-in or predefined data types and can be used directly by the user to declare variables.

Primitive data types available in c++ are following:

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point

Derived Data Types

Derived data types that are derived from the primitive or built-in datatypes are referred to as derived data types.

- Function
- Array
- Pointer
- Reference

Abstract or User-Defined Data Types

Abstract or User-Defined data types are defined by the user itself. Like defining a class in c++ or a structure.

C++ provides the following user defined datatypes:

- Class
- Structure
- Union
- Enumeration
- Typedef defined

Data Type	Size (In Bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 to 4	1 wide character

Manipulators in C++

Manipulators are helping functions that can modify the input/output stream. It does not mean that we change the value of a variable,it only modifies the I/O stream using insertion << and extraction>> operation.

Types of Manipulators

There are various types of manipulators:

Manipulators Without Arguments:

The most important manipulators defined by the iostream library are provided below.

endl: It is defined in ostream. It is used to enter a new line and after entering a new line and after entering a new line it flushes the output stream.

ws: It is defined in istream and is used to ignore the whitespaces in the string sequence.

ends: It is also defined in ostream and it inserts a null character into the output stream. It typically works with std::ostream, when the associated output buffer needs to be null-terminated to be processed as a c string.

flush: It is also defined in ostream and it flushes the output stream it forces all the output written on the screen or in the file. Without flush the output would be the same but may not appear in real time.

Manipulators With Arguments:

Some of the manipulators are used with the augment like setw(val),setfill(*) and many more. These all are defined in the <iomanip> header file.

The some important manipulators in <iomanip> are:

setw(val): It is used to set the field width in output operations.
setfill(c): It is used to fill the character ‘c’ on output stream.
setprecision(val): It set val as the new value for the precision of floating point value.
setbase(val): It is used to set the numeric base value for numeric values.
setiosflags(flag): It is used to set the format flags specified by parameter mask.

Type Conversion in C++

Type conversion is the process that converts the predefined datatype of one variable into appropriate data type. The main idea of type conversion is to convert 2 different data type variables into a single datatype to solve mathematical and logical expressions easily without and data loss.

Types of Type Conversion:

There are 2 types of type conversion:
Implicit Type Conversion
Explicit Type Conversion

Implicit Type Conversion

The implicit type conversion is the type of conversion done automatically by the compiler without any human effort. It means an implicit conversion automatically converts one data type into another type based on the same predefined rules of the c++ compiler. Hence it is also known as automatic type conversion.
Example:

```
#include<iostream>
using namespace std:
int main()
{
 int a=10;
float b=12.2;
double c;
c= a+b;
cout<<c;
}
```

Output: 22.2

In the above example, there are 3 variables a,b,c. Where a is int type, b is float type, and result variable is c is double type that stores a and b variables. But the c++ compiler automatically converts the lower rank data type. Value into higher type before resulting sum of 2 numbers.

Why we use implicit conversion?

It avoids the data loss, overflow or sign loss in implicit type conversion of c++.

Explicit Type Conversion
Conversion that requires user intervention to change the data type of one variable to another is called the explicit type conversion.

In other word, An explicit conversion allows the programmer manually changes or typecasts the data type from one variable to another type. Hence it is also known as type casting.

Types of Explicit Type Conversion

- Explicit Conversion using the cast operator
- Explicit Conversion using the assignment operator

Explicit Conversion using the cast operator
In c++ language, a cast operator is a unary operator who forcefully convert one type into another type.
Example:

```
#include<iostream>
Using namespace std;
int main()
{
float f2=6.7;
int x=static_cast<int>(f2);
cout<<”The value of x is:”<<x;
return 0;
}
```

Explicit Conversion using the assignment Operator
To convert the data type of one variable into another using the assignment operator in the c++ program.
Example:

```
#include<iostream>
Using namespace std;
int main()
{
int num_int;
double num_double=9.99;
num_int = num_double;
cout<<”num_int=”<<num_int<<endl;
cout<<”num_double=”<<num_double<<endl;
return 0;
}
```

Unit: II

Function in C++

A function is a set of statements that takes input does some specific computation and produces output. The idea is to put some commonly or repeatedly done tasks together to make a function so that instead of writing the same code again and again for different inputs we can call this function.

In other words, A function is a block of code that runs only when it is called.

Syntax:

```
Return_type function_name(parameter list)
{
body of the function
}
```

Write a program to print a number using function

```
#include<iostream>
using namespace std;
void displayNum(int n1, float n2)
{
cout<<"The int number is"<<n1<<endl;
cout<<"The float number is"<<n2<<endl;
}
int main()
{
int num1=5;
Float num2=5.5;
displayNum(num1,num2);
return 0;
}
```

Types of Function

We have 2 types of function in c++

- Built-in-functions
- User defined function

Built-in-functions

Built-in-functions are also known as library functions. We need not to declare and defined these functions as they are already written in the c++ libraries such as iostream,cmath etc. we can directly call them when we need.

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

```
int main()
{
int x,y;
pow (x,y);
cout<<pow(2,5);
return 0;
}
```
Output: 32

User defined function

C++ allows the programmer to define their own function. A user-defined function groups code to perform a specific task and that group of code is given a name.

When the function is invoked from any part of the program it all executes the codes defined in the codes defined in the body of the function.

Ex: Write a program to print the sum of 2 numbers using user defined function.

```
#include<iostream>
#include<cmath>
using namespace std;
int sum(int,int);
int main()
{
int x,y;
cout<<"Enter First Number:";
cin>>x;
cout<<"Enter Second Number:";
cin>>y;
cout<<"Sum of these two numbers:"<<sum(x,y);
return0;
}
int sum(int a, int b)
{
int c=a+b;
return c;
}
```

Output: Enter First Number: 30
Enter Second Number: 40
Sum of these two numbers:70

Ways to pass parameters

These are most popular ways to pass parameters

- Pass by Value
- Pass by Reference
- Pass by Pointer

Pass by Value

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case,changes made to the parameter inside the function have no effect on the argument.

By default c++ uses call by value to pass arguments. In general this means that code within a function can't alter the arguments used to call the function.

Example:

```
#include<iostream>
using namespace std;
void swap(int x, int y);
int main()
{
int a=100;
int b=200;
cout<<"Before swap,value of a:"<<a<<endl;
cout<<""Before swap,value of b:"<<b<<endl;
swap(a,b);
cout<<"After swap, value of a:"<<a<<endl;
cout<<"After swap,value of b:"<<b<<endl;
return0;
}
```

Output:

```
Before swap,value of a:100
Before swap,value of b:200
After swap,value of a:100
After swap,value of b: 200
```

Pass by Reference

The call by reference method of passing arguments to a function aopies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call.

Example:

```
#include<iostream>
using namespace std;
void swap(int &x, int &y);
int main()
{
```

```
int a=100;
int b=200;
cout<<”Before swap, value of a:”<<a<<endl;
cout<<”Before swap,value of b:”<<b<<endl;
swap(a,b);
cout<<”After swap,value of a:”<<a<<endl;
cout<<”After swap, value of b:”<<b<<endl;
return 0;
}
```
Output:
Before swap,value of a: 100
Before swap,value of b: 200
After swap,value of a: 200
After swap,value of b: 100

Call by Pointer

The call by pointer method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used to access the actual argument used in the call. This means that changes made to the parameter affect the passed arguments.

Example:

```
#include<iostream>
using namespace std;
void swap(int *x,int*y);
int main()
{
int a=100;
int b=200;
cout<<”Before swap, value of a:”<<a<<endl;
cout<<”Before swap,value of b:”<<b<<endl;
swap(&a,&b);
cout<<”After swap, value of a:”<<a<<endl;
cout<<”After swap, value of b:”<<b<<endl;
return 0;
}
```
Output:
Before swap, value of a:100
Before swap, value of b: 200
After swap,value of a: 200
After swap, value of b: 100

Function Overloading in C++

Function overloading is a feature of object oriented programming where 2 or more function can have the same name but different parameters. When a function name is overloaded with different jobs it is called function overloading.

In Function overloading function name should be the same and the arguments should be different.

Example:

```
#include<iostream>
using namespace std;
void add(int a,int b)
{
cout<<”sum=”<<(a+b);
}
void add(double a, double b)
{
cout<<endl<<”sum=”<<(a+b);
}
int main()
{
add(10,2);
add(5.2,6.2);
return 0;
}
```

Output:

```
Sum=12
Sum = 11.5
```

Inline Function in C++

C++ provides inline functions to reduce the function call overhead. An inline function is a function that is expanded inline when it is called. When the inline function is called whole code of the inline function gets inserted at the point of inline function call. This substitution is performed by the c++ compiler at compile time. An inline function may increase efficiency if it is small.

Syntax:

```
inline return_type function_name(parameters)
{
//function code
}
```

Example:

```
#include<iostream>
using namespace std;
inline void displayNum(int num)
{
cout<<num<<endl;
}
int main()
{
displayNum(5);
displayNum(8);
displayNum(666);
return 0;
}
```

Output: 5
8
666

Friend Function in C++

A friend function can be granted special access to private & protected members of a class in c++. They are the non member function that can access and manipulate the private & protected members of the class for they are declared as friends.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

Syntax:

```
Class class_name
{
Friend datatype function_name(arguments);
};
```

Example:

```
#include<iostream>
using namespace std;
class Distance
{
private:
int meter;
friend int addFive(Distance);
public:
```

```
Distance():meter(0){}
};
int addFive(Distance d)
{
d.meter=5;
return d.meter;
}
int main()
{
Distance D;
cout<<"Distance:"<<addFive(D);
return 0;
}
```

Static Variables

- A static variable associated with the class has only one copy per class but not for each object. An instance of a class does not have static variables.

- Static variables can be accessed by static or instance methods

- Memory is allocated when the class is loaded in the context area at run time.

```
// C++ program to demonstrate static
// variables inside a class

#include <iostream>
using namespace std;

class GfG {
public:
    static int i;

    GfG(){
        // Do nothing
    };
};

int GfG::i = 1;

int main()
{
    GfG obj;
```

```
    // prints value of i
    cout << obj.i;
}
```

Non-Static Variables

- Non-static variables will have one copy each per object. Each instance of a class will have one copy of non-static variables.

- Instance variables can be accessed only by the instance methods.

- Instance variables are allocated at compile time.

Static Member Function

A static function in C++ is a member function of a class that is associated with the class itself rather than with an instance or object of the class. This means that a static function can be called without creating an instance of the class.

The static keyword is used to define a static function in C++. A static function can access only static data members and other static functions of the class. It cannot access non-static data members or non-static member functions of the class.

Syntax: static data_type data_member;

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

class Box
{
   private:
   static int length;
   static int breadth;
   static int height;

   public:

   static void print()
   {
      cout << "The value of the length is: " << length << endl;
      cout << "The value of the breadth is: " << breadth << endl;
      cout << "The value of the height is: " << height << endl;
```

```
    }
};

// initialize the static data members

int Box :: length = 10;
int Box :: breadth = 20;
int Box :: height = 30;

// Driver Code

int main()
{

    Box b;

    cout << "Static member function is called through Object name: \n" << endl;
    b.print();

    cout << "\nStatic member function is called through Class name: \n" << endl;
    Box::print();

    return 0;
}
```

Default Arguments in C++

In a function arguments are defined as the values passed when a function is called. Values passed are the source & the receiving function is the destination.

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function does not provide a value for argument.

Example:

```
#include<iostream>
using namespace std:
int sum(int x,int y, int z=0,int w=0)
{
return (x+y+z+w);
}
int main()
{
cout<<sum(10,15)<<endl;
```

```
cout<<sum(10,15,25)<<endl;
cout<<sum(10,15,25,30)<<endl;
return 0;
}

output : 25
         50
         80
```