C++ Basic

Contents at a glance:

Chapter-1

Introduction to C++ Programming Language

Introduction of C++:

C++ is a cross-platform language that can be used to create high-performance applications.It was developed by Bjarne Stroustrup, as an extension to the C language.It gives programmers a high level of control over system resources and memory.The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17

C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a **compiled** language.

Simple:

It is a simple language in the sense that programs can be broken down into logical units and parts, has a rich library support and a variety of data-types.

Object-Oriented:

One of the strongest points of the language which sets it apart from C. Object-Oriented support helps C++ to make maintainable and extensible programs. i.e. Large-scale applications can be built. Procedural code becomes difficult to maintain as code-size grows.

Applications of C++:

C++ finds varied usage in applications such as:

- Operating Systems & Systems Programming. e.g. *Linux-based OS (Ubuntu etc.)*
- Browsers *(Chrome & Firefox)*
- Graphics & Game engines *(Photoshop, Blender, Unreal-Engine)*
- Database Engines *(MySQL, MongoDB, Redis etc.)*
- Cloud/Distributed Systems

First program in C++:

Let’s starts with very simple program in C++.Try it yourself in different ways.

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

Output:

```
Hello World!
```

Chapter – 2

Basic Input/Output

Basic Input/Output :

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

Standard output stream (cout):

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

Let's see the simple example of standard output stream (cout):

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

Output:

```
Hello World!
```

Standard input stream (cin):

The **cin** is a predefined object of **iostream** class. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

```
#include <iostream>
using namespace std;
int main() {
  cout << "Enter your age:"<<a;
  cin>>a;
  return 0;
}
```

Output:

Input is 19, so output

Enter your age : 19

Chapter – 3

Variable

Variable:

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.Let's see the syntax to declare a variable:

```
type variable_list;
```

Example:

```
int x=5;
float  f;
```

Variables declaration:

A typical variable declaration is of the form:

```
// Declaring a single variable

type variable_name;

// Declaring multiple variables:

type variable1_name, variable2_name, variable3_name;
```

A variable name can consist of alphabets (both upper and lower case), numbers and

the underscore ‘_’ character. However, the name must not start with a number.

Types of variables

There are three types of variables based on the scpoe of a variable:

- Local Variables
- Instance Variables
- Static Variables

Local Variables:

A variable defined within a block or method or constructor is called local variable.

- These variable are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.
- Initialisation of Local Variable is Mandatory.

Instance Variables:

Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables.
- If we do not specify any access specifier then the default access specifier will be used.

Static Variables: Static variables are also known as Class variables.

- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- If we access the static variable like Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name to class name automatically.

Chapter - 4

Data types

Data types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.There are three types of data types are namely:

* Built-in data type
* User-defined data type
* Derived data type

i) Build-in Data types

Integer:

Keyword used for integer data types is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

Character:

Character data type is used for storing characters. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

Boolean:

Boolean data type is used for storing boolean or logical values. A boolean variable can store either true or false. Keyword used for boolean data type is bool.

Floating Point:

Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space.

Double Floating Point:

Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating

point data type is double. Double variables typically requires 8 byte of memory space.

Void:

Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.

Wide Character:

Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by wchar_t. It is generally 2 or 4 bytes long.

Datatype Modifiers

As the name implies, datatype modifiers are used with the built-in data
types to modify the length of data that a particular data type can hold. Data type modifiers available in C++ are:

* Signed
* Unsigned
* Short
* Long

// C++ program to sizes of data types

```
#include<iostream>
using namespace std;
int main()
{
cout << "Size of char : " << sizeof(char)
                    << " byte" << endl;

cout << "Size of int : " << sizeof(int)
      << " bytes" << endl;

cout << "Size of short int : " << sizeof(short
int) << " bytes" << endl;

cout << "Size of long int : " << sizeof(long int)
       << " bytes" << endl;

cout << "Size of signed long int : " <<
sizeof(signed long int) << " bytes" << endl;
```

```
cout << "Size of unsigned long int : " <<
sizeof(unsigned long int) << " bytes" << endl;

cout << "Size of float : " << sizeof(float)
 << " bytes" <<endl;

cout << "Size of double : " << sizeof(double)
 << " bytes" << endl;

cout << "Size of wchar_t : " <<
sizeof(wchar_t)<< " bytes" <<endl;

    return 0;
}
```

Output:

```
Size of char : 1 byte
Size of int : 4 bytes
Size of short int : 2 bytes
Size of long int : 8 bytes
Size of signed long int : 8 bytes
```

Size of unsigned long int : 8 bytes
Size of float : 4 bytes
Size of double : 8 bytes
Size of wchar_t : 4 bytes

ii)User-defined data types

User Defined Data type in c++ is a type by which the data can be represented. The type of data will inform the interpreter how the programmer will use the data. A data type can be pre-defined or user-defined.

Hence, the data types that are defined by the user are known as user-defined data types. For example; class, structure, union, Enumeration, etc.

Class:

A class in C++ is the building block that leads to Object-Oriented programming.It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

A Class is a user defined data-type which has data members and member functions.

Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

Example :

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

class fruit
{
public:
string name;
int price;
```

```
public:

void getdata(void)
{
name = "mango";
price = 100;
}

void display(void);

};

void fruit :: display(void)
{
cout<<"fruit name is
"<<name<<endl;
cout<<"1kg price is "<<price;
}


int main()
```

```
{
fruit item;
item.getdata();
item.display();
return 0;
}
```

Output:

```
fruit name is mango
1kg price is 100
```

Enumeration:

Enumeration are user-defined types that consist of named integral constants.

1) It helps to assign constants to a set of names to make the program easier to read, maintain and understand.

2) An Enumeration is declared by using the keyword “enum“.

Example:

```
#include<bits/stdc++.h>
using namespace std;
enum shape{
 circle=4,square=4
};
```

```
int main(){
   shape s1 = circle,s2 =
square ,s3 = rectangle;
   cout<<"size of circle is
"<<s1<<endl;
   cout<<"size of square is
"<<s2<<endl;
```

Output:

size of circle is 4

size of square is 4

Structure:

The structure is a user-defined data type that is available in C++.Structures are used to combine different types of data types,

just like an array is used to combine the same type of data types.

A structure is declared by using the keyword “struct“.When we declare a variable of the structure we need to write the keyword “struct in C language but for C++ the keyword is not mandatory

Syntax:

```
struct
{
   // Declaration of the struct
}
```

Example:

```
#include<bits/stdc++.h>
```

```
using namespace std;
struct colour{
    string c1 = "red";
    char c2[7] = "yellow";
    string c3 = "blue";
    string c4 = "green";
};

int main()
{
    colour col;
    cout<<"colour 1 : "<<col.c1<<endl;
    cout<<"colour 2 : "<<col.c2<<endl;
    cout<<"colour 3 : "<<col.c3<<endl;
    cout<<"colour 4 : "<<col.c4<<endl;
}
```

output:

Colour 1 : red
colour 2 : yellow
colour 3 : blue
colour 4 : green

Unions:

1)A union is a type of structure that can be used where the amount of memory used is a key factor.

2)Similarly to the structure, the union can contain different types of data types.

3)Each time a new variable is initialized from the union it overwrites the previous in C language but in C++ we also don’t need this keyword and uses that memory location.

4)This is most useful when the type of data being passed through functions is unknown, using a union which contains all

possible data types can remedy problem.It is declared by using the keyword “union“.

Example:

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

union student
{
   int roll;
   int reg;
   float marks ;
};
int main()
{
   union student r,re,m;
   r.roll = 2314;
   re.reg = 3214;
   m.marks = 40.12;
   cout<<"roll no : "<<r.roll<<endl;
```

```
    cout<<"reg no : "<<re.reg<<endl;
    cout<<"Mark : "<<m.marks<<endl;

}
```

output:

```
roll no : 2314
reg no : 3214
Mark : 40.12
```

iii) Derived data-type

The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

* Array
* Function
* Pointers
* Reference

Array:

An array is a collection of items stored at continuous memory locations. The idea of array is to represent many instances in one variable.

Example:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int x[10] = {0,1,2,3,4};
    for (int i = 0; i < 10 ;i++)

      cout<<"Array of x ["<< i <<"] is :
"<<x[i]<<endl;

}
```

output:

Array of x[0] is 0

Array of x[1] is 1

Array of x[2] is 2

Array of x[3] is 3

Array of x[4] is 4

Function:

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

Example:

```
#include <iostream>
using namespace std;
int square(int x)
{
   return x*x;
}

int main(){
    int x = 10;
    cout<<"Square of x = 10 is
"<<square(x);
    return 0;
}
```

output:

```
Square of x = 10 is 100
```

Pointer

A pointer is a variable whose value is the address of another variable.
Like any variable or constant, you must declare a pointer before you can work with it.
The general form of a pointer variable declaration is −

```
type *var-name;
```

Here, type is the pointer's base type; it must be a valid C++ type and var-name is

the name of the pointer variable. The asterisk you used to declare a pointer is the
same asterisk that you use for multiplication.

However, in this statement the asterisk is being used to designate a variable as a pointer.

Following are the valid pointer declaration –

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address.

Example:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int num[10] =
{0,1,2,3,4,5,6,7,8,9};
    int *ptr ;

 cout<<"The array values : \n";
    for (int i = 9;i > 0; i--)
```

```
        cout<< num[i] << endl;
    ptr = num;
    cout<<"The value of ptr :
"<<*ptr<<endl;

    ptr = ptr + 5;
    cout<<"The value of ptr :
"<<*ptr<<endl;

    ptr = ptr - 4;
    cout<<"The value of ptr :
"<<*ptr<<endl;
}
```

Output:

The array value :

9

8

7

6

5

4

3

2

1

The value of ptr : 0

The value of ptr : 5

The value of ptr : 1

Reference:

When a variable is declared as reference, it becomes an alternative name for an existing variable. A variable can be declared as reference by putting ‘&’ in the declaration.

Example:

```
#include <iostream>

using namespace std;

int main () {

   int    i;
   double d;
```

```
    int& i = r;
    double& d=s;

   i = 5;
   cout << "Value of i : " << i
<< endl;
   cout << "Value of i
reference : " << r  << endl;
 d = 11.7;
  cout << "Value of d : " << d <<
endl;
   cout << "Value of d
reference : " << s  << endl;

   return 0;
}
```

output:

```
Value of i : 5
Value of i reference : 5

Value of d : 11.7
Value of d reference : 11.7
```

Chapter-5

Control Structure

Control Structures are **the blocks that analyze variables and choose directions in which to go based on given parameters**. The basic Control Structures in programming languages are:
Conditionals (or Selection): which are used to execute one or more statements if a condition is met.

* Selection
* Loop
* Jump

i)Selection:

Selection statements **allow a program to test several conditions, and execute instructions based on which condition is true**. That is why selection statements are also referred to as conditional statements.

* Simple if
* If else
* If else if ladder
* Switch case

Simple if:

Simple if statement is the most simple decision-making statement.It is used to

decide whether a certain statement or block of statements will be executed or not

i.e if a certain condition is true then a block of statement is executed otherwise not.

Example:

```
#include<iostream>
#include<string.h>
using namespace std;
int main(){
    cout<<"Are you vaccinated or not ?";
    string is = "yes";
    string Answer;
    cin>>Answer;
    if (is == Answer)
      cout<<"You are allowed";
}
```

output:

input if yes,You are allowed

if-else:

1)The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t.

2)But what if we want to do something else if the condition is false.

3)Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Example:

```
#include<iostream>
using namespace std;
int main()
{
    int i;
    cout<<"Enter the candidate
age:";
    cin>>i;
    cout<<"Check that candidate
age is equal to or greater than
Eighteen"<<endl;
    if (i >= 18)
    cout<<"Yes , He is eligible";
    else
    cout<<"No , He is not
eligible";
}
```

Output:

If input is above 18

Yes,He is eligible

else,No,He is not eligible

if-else-if-ladder:

1)Here, a user can decide among multiple options.

2)The C if statements are executed from the top down.

3)As soon as 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.

4)If none of the conditions are true, then the final else statement will be executed.

Example:

```
#include<iostream>
using namespace std;
int main(){
    cout<<"What are the level of
game"<<endl;
    string is = "yes";
    string
Answer1,Answer2,Answer3;
    cin>>Answer1;
    cin>>Answer2;
    cin>>Answer3;
    if (is == Answer1)
```

```
        cout<<"EASY";
    else if (is == Answer2)
        cout<<"MEDIUM";
    else if(is == Answer3)
        cout<<"DIFFCULT";
}
```

Output:

Try it yourself,it helps to understand concept . [Hint : If-else]

Switch case:

Switch case statement evaluates a given expression and based on the evaluated value(matching a certain condition), it executes the statements associated with it.

Basically, it is used to perform different actions based on different conditions(cases).

1)Switch case statements follow a selection-control mechanism and allow a value to change control of execution.

2)They are a substitute for long if statements that compare a variable to several integral values.

3)The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Example:

```
#include<iostream>
using namespace std;
int main(){
    cout<<"SUDOKU GAME"<<endl;
    cout<<"Select your
option"<<endl;
    int option ;
    cin>>option;
    switch(option)
    {
    case 1:
        cout<<"START";
        break;

    case 2:
        cout<<"NEW GAME";
        break;

    case 3:
        cout<<"RESUME";
        break;
```

```
    default:
        cout<<"QUIT";
        break;
    }

}
```

Output:

If input 1

START

If input 2

NEW GAME

If input 3

RESUME

if input not 1,2,3

QUIT

ii)Loops

Loops in programming come into use when we need to repeatedly execute a block of statements.

There are mainly two types of loops:

1)Entry Controlled loops:
In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.

2)Exit Controlled Loops: In this type of loops the test condition is tested or evaluated at the end of loop body.

1. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false.
2. **do – while loop** is exit controlled loop.

For loop:

In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value.

If statement is true, then loop body is executed and loop variable gets updated .

Steps are repeated till exit condition comes.

Initialization Expression:

In this expression we have to initialize the loop counter to some value.

For example: int i=1;

Test Expression:

In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop.

For example: i <= 10;

Update Expression:

After executing loop body this expression increments/decrements the loop variable by some value.

For example: i++;

Example:

```
#include<iostream>
using namespace std;
int main()
{
    int i = 0;
    for(int i;i<=5;i++)
    cout<<"i is "<<i<<endl;
}
```

Output:

```
i is 0
i is 1
i is 2
i is 3
i is 4
```

While Loop

1)While studying for loop we have seen that the number of iterations is known beforehand,

i.e. the number of times the loop body is needed to be executed is known to us.

2)while loops are used in situations where we do not know the exact number of iterations

of loop beforehand.

3)The loop execution is terminated on the basis of test condition.

Example:

```
#include<iostream>
using namespace std;
int main()
{
    int i = 2;
    while ( i <= 5)
    {
```

```
        cout<<"i is "<<i<<endl;
        i++;
    }
    return 0;
}
```

output:

i is 2

i is 3

i is 4

i is 5

do-while loop:

1)In do while loops also the loop execution is terminated on the basis of test condition.

2)The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops are entry controlled loops.

Note: In do while loop the loop body will execute at least once irrespective of test condition.

Example:

```
#include<iostream>
using namespace std;
int main()
```

```
{
    int i = 6;
    do
    {
        cout<<"i is "<<i<<endl;
        i++;
    }while ( i <= 10);
    return 0;
}
```

Output:

```
i is 6
i is 7
i is 8
i is 9
i is 10
```

iii) Jump Statement

Jump statement are used to manipulate the flow of the program if some conditions are met. It is used to terminating or continues the loop inside a program or to stop the execution of a function. In C++ there is four jump statement:

* break
* continue
* goto
* return

break:

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

Syntax:

```
break;
```

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

Example

```
#include<iostream>
using namespace std;
int main()
{
 int x[10]={0,1,2,3,4,5,6,7,8,9};
    for (int n = 0;n < 10;n++)
      if (n == 5)
        break;
      else
        cout<<"value of x["<< n
<<"] is : "<<x[n]<<endl;
}
```

Output:

```
value of x[0] is 0
value of x[1] is 1
value of x[2] is 2
value of x[3] is 3
value of x[4] is 4
```

continues:

This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

As the name suggests 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.

Syntax:

continue;

Example:

```
#include<iostream>
using namespace std;
int main()
{
 int x[10]={0,1,2,3,4,5,6,7,8,9};
    for (int n = 0;n < 10;n++)
      if (n != 5)
        continue;
      else
```

```
        cout<<"value of x["<< n
<<"] is : "<<x[n]<<endl;

}
```

Output:

value of x[5] is 5

goto:

The goto statement in C/C++ also referred to as unconditional jump statement can be used to jump from one point to another within a function.

Syntax:

```
Syntax1      |   Syntax2
-------------------------
goto label;  |    label:
.            |    .
.            |    .
.            |    .
label:       |    goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label.

Here label is a user-defined identifier that indicates the target statement. The statement immediately followed after ‘label:’ is the destination statement. The

‘label:’ can also appear before the ‘goto label;’ statement in the above syntax.

Example:

```
#include<iostream>
using namespace std;
int main(){

    int x[10] =
{0,1,2,3,4,5,6,7,8,9};
    for (int n = 0;n < 10;n++)
        if (n == 3)
            goto three;
        else
            cout<<"value of x["<< n
<<"] is : "<<x[n]<<endl;

        three:
            cout<<"value of
x["<<"3"<<"] is : "<<"THREE"<<endl;
}
```

Output:

```
value of x[0] is 0
value of x[1] is 1
value of x[2] is 2
value of x[3] is THREE
```

return:

1) The return in C or C++ returns the flow of the execution to the function from where it is called.

2) This statement does not mandatorily need any conditional statements.

3)As soon as the statement is executed, the flow of the program stops immediately and return the control from where it was called.

4)The return statement may or may not return anything for a void function, but for a non-void function, a return value is must be returned.

Syntax:

return[expression];

Example:

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

int mul(int m1,float m2){
    double M = m1 * m2;
    return M;
}

void display(int R){
    cout<<"The result of
Multiplication is : "<< R;
    return ;
}

int main(){
    int v1 = 10;
    int v2 = 20;
    float R = mul(v1,v2);
```

```
    display(R);
}
```

Output:

The result of Multiplication is 200