Introduction

C++ is a object-oriented programming (OOP) language used with application or system software, client-server application drivers, and embedded firmware. It is developed by Bjarne Stroustrup. It is a compiled language. It is an extension of the C language. Hence the basic syntax and code structure of C, as well as C++, are the same.

How does C++ make working so easy?

There are some features of OOP's. Which by default make C++ a very powerful and irreplaceable language.

1. Encapsulation:

Encapsulation is "Wrapping of data in a single unit". Like a class, its wraps data member and methods under a single class

2. Polymorphism:

Poly means many, morphism means forms. Polymorphism means the ability to take many forms. This feature enables to use the same name with different functionality depending upon object passed to it.

3. Inheritance:

Inheritance is the main aspect of OOP. This feature gives the idea of code reusability. Inheritance adds extra features without modifying existing class. Its extends class and gives the benefit of using members of a previous class.

What can you do with C++?

Most of the programmer come across the task which needs more security and ability to be flexible enough they select C++. It has a power that we can use it anywhere like making Anti-virus, games, server-side programming and the most important

one is Operating System. Also, list not finish here, we can build editors, databases, compilers, real-life complex application and list goes on and on…

Advantages

1. **Obliviously the first one is it's an Object Oriented programming language (OOP).**
2. **It uses Pointer concept which is really powerful.**
3. **It is easy to learn**
4. **It is quite fast as compared to other high-level languages.**
5. **This is platform independent. Support many devices.**
6. **It has a robust performance.**
7. **It has a rich functional library.**

Required Skills

Initially, when we are talking about computer languages my suggestion is to always start C++ as your first language. To become a software developer or to learn C++ you must know the OOPs concept. A computer doesn't understand the high-level language that what we speak like English. It only knows binary 0s and 1s. Some fundamental knowledge of how language communicates with the computer. It's an added advantage that how the computer works.

Why should we use C++?

The idea of getting a job with C++ is not a thinking process of the people these days. The people who are currently looking for a job or which are already in a job will not think about C++.As a requirement comes there are fewer programmers who really know it. This is the main reason to learn it. And the other point of view is As many times I say its performance which makes it irreplaceable at some point.

Features of C++

Here is the list of most important features in C++ that can be used to developed high-performing applications:

1. Simple

C++ is one of the most-simple languages when it comes to programming. It is also easy to understand and learn as it originated from the C programming language.

2. Object-Oriented Programming

One of the most important features because of which C++ got famous. Everything is treated as objects in C++ that's why it is called object-oriented programming. Objects are used for performing all kinds of functionalities.

3. Portability

C++ is not platform independent but we can say it is portable enough to run on different machines by adding some or no changes at all. On different operating systems you can run the same code. Write code for one time and use it for every time you need that particular functionality. We can't say that it's completely platform-independent. So don't confuse C++ with Java because Java is completely platform-independent. For example, you have written code in Linux but wants to run in Windows so C++ code will run on both without any hindrance! Simple and straight.

4. Mid-Level Programming Language

C++ programming language is a collection of special features of low-level languages and high-level languages. It can be used to develop applications based on the required level of programing language that is low or high.

5. Rich Library

C++ library is full of in-built functions that save a huge amount of time in the software development process.

6. Case Sensitive

As C++ is originated from C, it is also purely CASE sensitive that means lowercase and uppercase characters written in code will have completely different meaning and will be treated differently.

7. Compiler-Based

As no interpretation is done in C++ code it is considered to be a compiler-based language that makes it faster than other programming languages like Java, etc. Without compilation, you can't execute any C++ code.

8. Dynamic Memory Allocation

Due to pointer support in the C++ language. Memory allocation can be easily done dynamically rather than static. Can free memory anytime by using the free() function.

9. Recursion

Due to code reusability features, we can call any function within a function saving memory space by not writing the same code again and again. Code reusability is there for every function.

10. Fast

As compared to other programming languages. Execution time and Compilation time of a code written in C++ language are faster than any other programming language.

11. Pointers

As we all know pointers hold the address of another variable and we can access the address of any variable using concept or pointers which increases performance. C++ also provides the use of pointers in the software development process.

C++ Data Types

C++ Data types define the type of data that a variable can hold. It can be user-defined data type like integer, float, double, char, or built-in data type like union, enum, struct or can be a derived data types like functions, pointers, arrays. Data types should be defined before the execution as it informs the compiler the type of data specific variables holds. Integer data type can hold only integer values, it cannot hold the float values or string values.

A data type is to let know the variable, what type of element it is and definitely going to determine the memory allocation of that variable. We are aware that each data type has a different memory allocation. There are three different C++ data types namely; Primitive, Derived and User Defined. Let's go ahead and learn about them.

Data Types in C++

Here are three different data types in c++ which are explained below:

1. Primitive Data Types

These are pre-defined in c++, also called the built-in data types. We can directly use them to declare the variables.

a. Integer: Usually defined by "int". We can know the size of memory allocated and how the variable is declared as below. صحيح

b. Character: Usually defined by "char". We can know the size of memory allocated and how the variable is declared as below. حرف

c. Floating Point: Usually defined by "float". We can know the size of memory allocated and how the variable is declared as below.حقيقي

d. Boolean: Usually defined by "bool". We can know the size of memory allocated and how the variable is declared as below

e. String: Usually defined by "String". We can know the size of memory allocated and how the variable is declared as below.

Variables in C++ المتغيرات

Variables in C++ acts as a memory location, it is nothing but the name of the container or element that stores the data or values that are being used in the program later for execution. It can be defined using the combination of letters digits, or special symbols like underscore(_), defined by using the data types like char, int, float, double. Variables can be anything except the reserved keyword, the first letter of the variables must start with the letter only.

Variables are the most important part of any programming language. Any programming language is incomplete without a variable. We can also say that without variables, the program cannot run. Like any other programming language,

the C++ language also needs variables to run their program. Variables are not used to run the program, instead, they are used to store the value or string. Without storing value, the program cannot run. Hence, variables are known for the backbone of the programming language. In C++ any word except the keywords is used as a variable. To define variables we need to specify the type for the variable. Type can be anything int, double, char, float, long int, short int, etc. int is used to store integer value i.e. 5, 19, 519, 1000. Char is used to storing the character or string i.e. a, educate. Float is used to store the float values like 2.3, 3.679, 9.45. Long int is used to store long integer values. In this article, we are going to discuss how to initialize and declare the variables in the C++ language. And the types of variables.

Rules and Regulations for Defining Variables in C++ Language القواعد لكتابه المتغيرات

- **Variables can be a mixture of digits, special characters like and percent (&), underscore (_) or string.**
- **Upper and lower cases are treated as different variables as C++ is case sensitive language. Educba and eduCBA will be treated as two different variables.**
- **C++ variables must be started with the character. It will not consider the number as a first character. 6educba is not a valid variable because it starts with the number where educba6 can be a valid variable as it started with the character.**

- **variables in C++ language should not be a keyword. for, this, if, else, while, do, char, this, etc are the keywords that are used for the specific purpose. These keywords cannot be used as a variable in C++.**
- **Blank spaces are not allowed for the variables. Edu cba is not valid as there is space between edu and cba where educba is valid variable or edu_cba is also a valid variable as underscore sign is used to join the variable.**

Variables Work in C++ Language

- **Declaration of variables informs compiler the types of data variables will be used in the program.**
- **Declaration of variables names informs compiler the name of the variables that are used to store the value in the program.**
- **While declaring variables I tell the compiler the storage that variables need. The compiler does not have to worry about the storage until it is declared.**

Declare Variables in C++ Language

Variables can be declared first before starting with the programs. The syntax for declaration of a variable is as follows:

```
data_type  variable_name;
```

where

data_type: Defines types of data for storing value. Data types can be int, char, float, double, short int, etc.

variable_name: Defines the name of the variables. It can be anything except the keyword.

For example:

1. int cab;

For example 1, int is a data type and cab is a variable nameOnce the variables are declared, the storage for those variables has been allocated by the compiler as it will be used for the program.

Program to Illustrate the Declaration of Variables in C++ Language

```
#include<iostream>

using namespace std;

int  main()

{    int x, y, z;

x = 10;      y = 3;

z = x + y;

cout << "Sum of two numbers is: " << z;

rturn 0;}
```

Initialize Variables in C++ Language

In C++, variables can be initialized by assigning the values at the time of declaration. The syntax for initialization of variables in C++ language is –

```
data_type variable_name = value;
```

For example,

1. **int x = 10;**
2. **char b = ‘eduCBA’**

In example 1, we initialized variable x with value 10. In example 2, we have initialized b as a character with eduCBA value.

Types of Variables in C++ Language

There are 5 types of variables in C++ language which are as follows:

1. Local Variables

Local variables are declared inside the function. Local variables must be declared before they have used in the program. Functions that are declared inside the function can change the value of variables. Functions outside cannot change the value of local variables. Here is an example

```
int main()
{
int x = 2; //local variable
}
```

2. Global Variables

Global variables are declared outside the functions. Any functions i.e. both local function and global function can change the value of global variables.

```
int y = 10; //global variable

int main()

{

int x = 5; //local variable

}
```

3. Static Variables

These variables are declared with the word static.

Example is given as follows,

```
int main()

{

int x = 5; //local variable

static y = 2; //static variable

}
```

4. Automatic Variables

Automatic variables are declared with the auto keyword. All the variables that are declared inside the functions are default considered as an automatic variable.

```
int main()

{
```

```
int x = 20; //local variable (Automatic variable)

auto y = 12; //automatic variable

}
```

5. External Variables

By using the extern keyword, external variables are declared.

```
extern z = 4; //external variable
```

Introduction on Keywords in C++

Keywords are the reserved keywords that are defined by the compiler to perform the internal operation, written in lowercase. Keywords have some meaning which is defined by the compiler to accomplish a task in code, they cannot be used as a variable in programming. C++ provides 64 keywords – for, break, continue, switch, int float, double, char, try, catch, while, etc.

List of C++ Keywords

Below is the list of keywords used in the C++ language.

(Auto, double, int, struct, Break, else, long, switch, Case, enum, register, typedef, Char, extern, return, union, Const, float, short, unsigned, Continue, for, signed, void, Default, goto, sizeof, volatile, Do, if, static, while, Asm, dynamic_cast, namespace, reinterpret_cast, Bool, explicit, new, static_cast, Catch false, operator,

template, Class, friend, private, this, Const_cast, inline, public, throw, Delete, mutable, protected, true, Try, typeid, typename, using, virtual, wchar_t)

Function "cin" in c++:

It is an input function (meaning giving values to the variables if not given by the program) .

For example:

int x;

Here we did not give a value of x. We just reserved a place in the memory of type int ,So it is given a value by cin

Cin >> x;

Difference between : Cin >> Cout <<

Example of Keywords and cin

Program for implantation of If keyword is given as follows:

```
#include<iostream>

using namespace std;       int main()

{   int n;

cout << "Enter number:" << endl;

cin >> n;

if(n > 0){

cout << "You have entered positive number";

}     return 0;

}
```

Arithmetic Operators in C++

Arithmetic operators are used to performing some mathematical operations. Like any other operator, C++ also supports arithmetic operators to perform some mathematical actions like addition, subtraction, multiplication, etc. In this article, we are going to discuss those operators supported by the C++ language. Below is the list of different operators explained in more detail.

1. **Addition Operator (+): It is used to add two operands. Suppose X and Y are two operands, this plus operators will add up these two operands X + Y.**
2. **Subtraction Operator (-): It is used to subtract two operands. Suppose X and Y are two operands, then this minis operator will subtract the value of the second operand from the first operand.**
3. **Multiplication Operator (*): It is used to multiply two operands. Suppose X and Y are two operands then this multiplication operator will multiply X with Y.**
4. **Division Operator (/): It is used to numerator by the denominator. Suppose X and Y are two operands, this division operator divides the numerator by denominator.**
5. **Modulus Operator (%): It is used to give the remainder of the division. Suppose X and Y are two operands then this modulus operator first divides the numerator by denominator and gives the remainder.**

6. **Increment Operator (++): It is used to increment the value of the variable by 1. Suppose X is the operand, then this increment operator will add the value of X by 1.**
7. **Decrement Operator (–): It is used to decrementing the value of the variable by 1. Suppose X is the operand, this decrement operator will decrement the value of X by 1.**

Examples of Arithmetic Operators in C++

Example :

```
#include <iostream>

using namespace std;

int main()  {

int X, Y, total;

cout << "Enter the value of X: ";

cin >> X;

cout << "Enter the value of Y: ";

cin >> Y;

total = X + Y;            // total = X - Y;

cout << "Addition of X and Y is: " << total;

return 0;

}
```

Example

```
#include <iostream>

using namespace std;

int main() {

int X;

cout << "Enter the value of X: ";

cin >> X;

X++;                          //X--     ++X     --X

cout << "Incremented value of X: " << X;

return 0;

}
```

Assignment Operators in C++

Here, let us begin on knowing about assignment operators in C++. As the name already suggests, these operators help in assigning values to variables. We discuss these with respect to operators and operands. These operators help us in allocating a particular value to the operands.

The main simple assignment operator is '='. We have to be sure that both the left and right sides of the operator must have the same data type. We have different levels of operators. Let us learn about each of those in C++. Assignment operators

are a part of binary operators. Examples for these are: =, +=, -=, *=, /=, %=. Let us learn about each of these with examples. There are three levels of operators.

- **Unary Operators**
- **Binary Operators**
- **Ternary Operators**

Assignment operators are a part of binary operators.

Examples for these are: =, +=, -=, *=, /=, %=. Let us learn about each of these with examples.

Examples

```
#include <iostream>

using namespace std;

int main() {

int a,b;        char c;         float d;

a=10;        b=10.5;         c='R';        d=5.85;

cout<<" Value of a is: "<<a<<endl;

cout<<" Value of b is: "<<b<<endl;

cout<<" Value of c is: "<<c<<endl;

cout<<" Value of d is: "<<d<<endl;

}
```

So, as an example, we can understand that the assignment operator '=' is just going to assign values based on the data type using the operands and variables. Above, we can see that for the value b which is given data type of int, gives only the value till the decimal point. Only if we give the data type float like the variable d, the complete value gets displayed. So, we can say that data type also plays an important role in setting and displaying the values of different operands based on their values.

Examples

```
#include <iostream>

using namespace std;

int main(){

int a,b,c;

a=10;      b=5;       c=5;

a+=6;

b+=20;

c=c+20;

cout<<" Value of a is: "<<a<<endl;

cout<<" Value of b is: "<<b<<endl;

cout<<" Value of c is: "<<c<<endl;

}
```

As observed above, the example of how we got the value for variable c, is the process of how the operator '+=' works. As per the operand, first, the left side operand is added to the left side operand and then the final value is being assigned to the left side operand. This is how '+=' is used.

Relational Operators in C++

Relational operators are also known for comparison operators. Relational operators are used to relating the condition, that is it compares the two values and prints the result. In this article, we are going to see those relational operators in C++ with the help of examples. There are total 6 relational operators ==, !=, <, >,<=, >= which are explained below:

1. Less than Operator (<)

This operator is called less-than the operator. It checks whether the value of the left operand is less than the value of the right operand or not. If it satisfies the condition then, it returns true as a value else it returns false.

2. Greater than Operator (>)

This operator is called greater than the operator. It checks whether the value of the left operand is greater than the value of the right operand. If it satisfies the condition it returns true as value else it returns false.

3. Less than or Equal to Operator (<=)

This operator is called less than or equal to the operator. It checks whether the value of the left operand is less than or equal to the value of the right operand. If it satisfies the condition it returns true as value else it returns false.

4. Greater than or Equal to Operator (>=)

This operator is called as greater than or equal to the operator. It checks whether the value of the left operand is greater than or equal to the value of the right operand. If it satisfies the condition it returns true as value else it returns false.

5. Equal to Operator (==)

This operator is called as is equal to the operator. It checks whether the value of the left operand is equal to the value of the right operand. If it satisfies the condition it returns true as value else it returns false.

6. Not Equal to Operator (!=)

This operator is called as is not equal to the operator. It checks whether the value of the left operand is not equal to the value of the right operand. If it satisfies the condition it returns true as value else it returns false.

Example :

```
#include <iostream>

using namespace std;

int main()

{

int X, Y;
```

```
cout << "Enter the value of X: ";

cin >> X;

cout << "Enter the value of Y: ";

cin >> Y;

if(X > Y)                //<    <=    >=    ==    !=

{

cout << "X is greater than Y";

}

else

{

cout << "Y is greater than X";

}

return 0;

}
```

Example :

```
#include <iostream>

using namespace std;

int main ()

{
```

```
int age;

cout << " Please enter your age here: " ;

cin >> age;

if ( age < 0 || age > 125 )       //&&

{

cout << " Ha Ha Ha You're lying - you CANNOT be that age. Impossible " <<

endl;

}

else

{

cout << " Great! Thanks for providing your age ! " << endl;

}

return 0;   }
```

Example : NOT Boolean Operator !

This Boolean operator is represented by "!" in C++ programming language and it is also known as logical NOT operator. This operator has no such conditions on both sides. In fact, it has only one aim to invert the value of the given Boolean expression as only one single expression can be prefixed to it. In a simpler word, we can say that in regular English writing we only use not when we don't want something or we

can say that not in favor like opposition. “If an expression needs to be proved false or true depending upon the expression prefixed to it always use NOT operator.”

```
#include <iostream>

using namespace std;

int main ()

{

bool  initiate;

cout << " Hey ! Do you really want to initialise the application ? [0: No] [1: Yes] "

<< endl;

cin >> initiate ; // 0 input is false', and 1 is 'true'

if ( !initiate )

{

cout << " Then why would you open the application ? Haha Funny, too bad, I'm

starting now anyway. " << endl;

}

cout << " Application initialized. " << endl;

// you can continue writing the main program code here

return 0;

}
```

if else Statement in C++

If else statement is a conditional statement. It is used to check the condition and based on the condition it executes the loop. Working of if else statement in C++ language is easy. if-else statement is used when we need to execute the same piece of code, if the given condition is true and execute another piece of the code if the condition is false. if and the if-else statement is the same, the only difference is in if statement the statement executes if the condition is true or else it stops the program whereas, an if-else statement, the statement is executed if the condition is true or else it executes the statement following the else. In this article, we are going to discuss the conditional statement in C++ language i.e. if else statement.

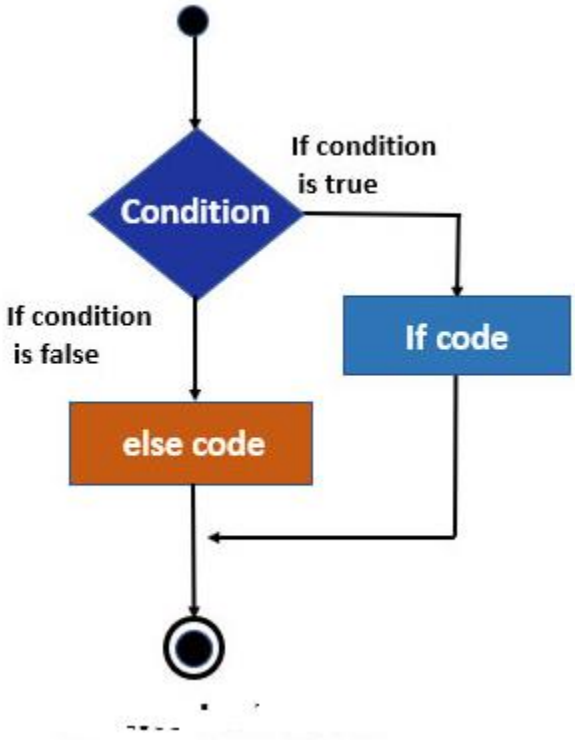
Syntax of if else Statement :

```
if(condition){
statement; }
else {
statement; }
```

if and else are the two keywords used to declare the if else statement. condition is a parameter that is used to evaluate the decision. if statements are declared inside the parenthesis of if and else statement is declared inside the parenthesis of else.

Flowchart of if else statement in C++

Below is the flow chart that defines the working of the if-else statement in a stepwise manner:



Here the condition is defined by using the diamond sign. The flow chart states that first it checks the condition and if the condition is true it transfers the flow control to the if statement and if the condition is false, it transfers the flow control to else statement.

How if else statement works in C++?

As we have discussed earlier the concept is easy to understand. In if else statement, first, it checks the condition and if the condition is true, the code inside the if the body is executed and else statement is skipped. and if the condition is false then it skips the if statement and executes the else body.

Examples:

Code:

```
#include <iostream>

using namespace std;

int main()

{

int n = 50;

if  (n <= 50)

{

cout << "Given number is less than or equal to 50";   }

else    {

cout << "Given number is greater than 50";  }

return 0;

}
```

Explanation

Here we have written a simple program to check whether the number is less than or equal to 50. First, we have an initialized variable n to 50. if keyword checks the condition i.e. n <= 50. here we have already Initialized n to 50. So here the condition is true, so it will print the Given number is less than or equal to 50.

if we change the value of n to 75, the condition becomes false and it will execute the else statement And then it will print the given number is greater than 50.

Example :

```
#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter a number ";

cin >> n;

if(n%2 == 0)

{cout << "Entered number is even";}

else    {

cout << "Entered number is odd"; }

return 0;

}
```

Explanation

Here we have written a program to check the even and odd number using is else statement. variable n is declared and allows the user to enter the value. variable n stores the value entered by the user. If statement checks the condition n%2 == 0

that declared to check the even number. if the number entered by the user satisfies the condition, it will print the Entered number is even. Else it executes the else statement and prints Entered number is odd.

Example

```
#include <iostream>

using namespace std;

int main()  {

int n;

cout << "Enter a age ";

cin >> n;

if(n >= 18)

{

cout << "Eligible for voting";   }

else   {

cout << "Not eligible for voting";    }

return 0;    }
```

Explanation

Here we have written a program to check the eligibility for voting. Variable sn is declared and allow a user to enter their use. Variable stores the age of user into

variable n. if statement checks the condition i.e. n >= 18. If the age is greater than or equal to 18, it will print Eligible for voting. if the age is less than 18, it will print Not eligible for voting.

Else if in C++

As we are already familiar with 'if and else' statements, let us now move a bit further into the concept of else if conditions. As the name already suggests that these statements deal with the conditions one after another. Now let us the concept of utilizing this concept in C++ programming language.

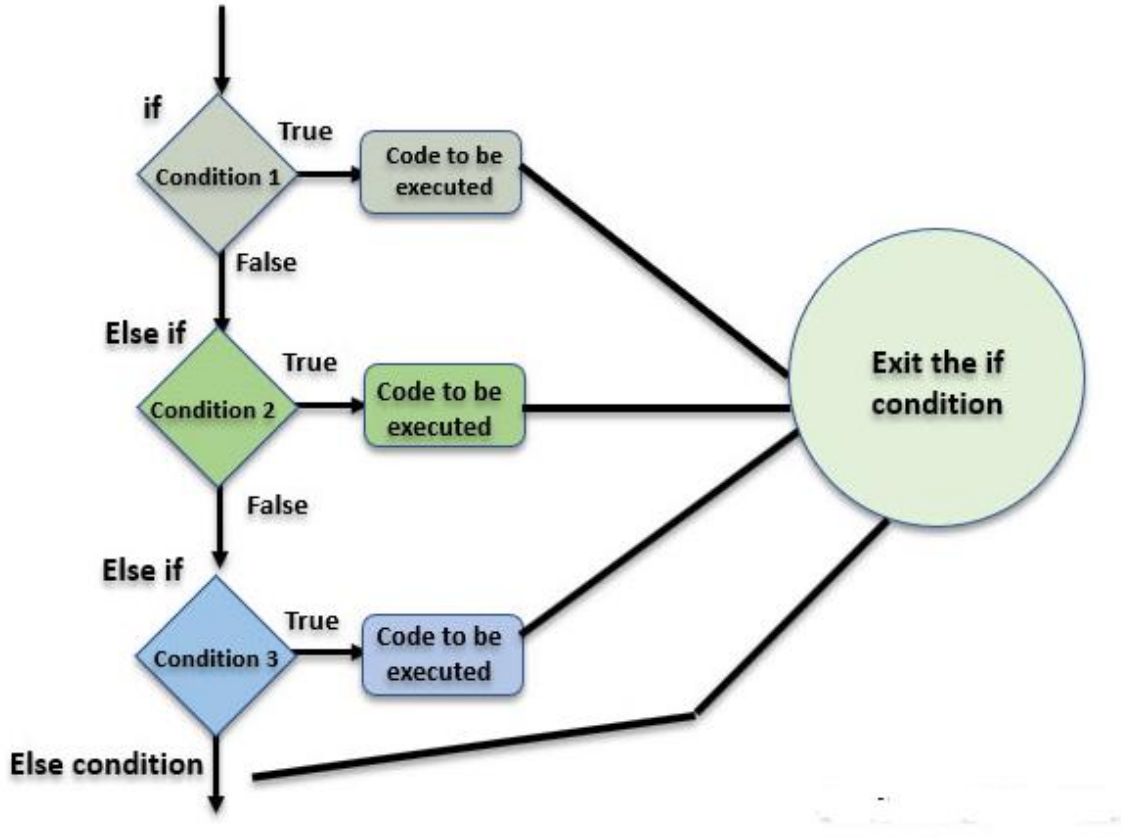
Syntax:

```
if (first condition)  {
// code to be executed.  }
else if (second condition)  {
//code to be executed  }
else if (third condition)  {
//code to be executed  } …..
else  {
//code to be executed when none of the above conditions is true  }
```

By the above syntax, we can understand there would be a first if condition and a code to be executed under it. Then we can have multiple conditions, which are to be

checked upon. And if none of the conditions work then we are going to execute the code which is written in the else block.

Flow Chart



As per the flow chart, we can note there would be a first if condition. According to the Boolean expression we can have, 'else if' condition is to be executed or the code inside the true condition is executed and the program compilation comes out of this if-else condition loop.

Else-If in C++

The condition in C++ is written in below format:

```
If(condition1)  {

Cout<<"code 1";   }

Else if(condition 2)  {

Cout<<"Code 2";  }

Else if(condition 3)    {

Cout<<"Code 3";  }  …….

Else    {

Cout<<"code to be executed if none of the above conditions is true";  }
```

'Else if' condition is the same for all the programming languages. Here in C++ we only have the syntax level changes to the code but as per the logic, there would be no much change.

Let us have a few examples below to understand working with 'else if' statement conditions in C++.

Examples of Else If in C++

Example :

```
#include <iostream>

using namespace std;

int main()

{
```

```
int a;

cout<<"Enter any number between 1 to 50: ";

cin>>a;

if(a >=0 && a<=10)  {

cout <<" Number chosen is between 0 and 10 ";    }

else if(a >10 && a<=20)    {

cout <<" Number chosen is between 10 and 20 ";    }

else if(a >20 && a<=30)    {

cout <<" Number chosen is between 20 and 30 ";    }

else if(a >30 && a<=40)    {

cout <<" Number chosen is between 30 and 40 ";    }

else if(a >40 && a<=50)    {

cout <<" Number chosen is between 40 and 50 ";    }

else    {

cout<<"The number you chose is out of given range";    }

}
```

Example :

```
#include <iostream>

using namespace std;

int main()  {

int a;

cout<<"Enter any number: ";

cin>>a;

if(a%2 == 0)    {

cout <<" Number chosen is divisible by 2 ";  }

else if(a%3 == 0)   {

cout <<" Number chosen is divisible by 3 ";    }

else if(a%7 == 0)    {

cout <<" Number chosen is divisible by 7";      }

else if(a%11 == 0)    {

cout <<" Number chosen is divisible by 11 ";    }

else if(a%13 == 0 )    {

cout <<" Number chosen divisible by 13 ";    }

else if(a%17 ==0)    {

cout<<" Number chosen is divisible by 17 ";    }

else    {
```

```
cout<<"The number chosen is not divisible by 2, 3, 7, 11, 13 and 17";     }

}
```

As explained in the flow chart also, if the starting condition gets matched then the code inside that condition is executed and the compiler comes out of that if-else if loop. So, that is why once the divisible condition of 2 is done, the compiler comes out of the 'if' conditions and displays is the required output.

Example :

```
#include <iostream>

using namespace std;

int main()

{

int a,x;

x=1;

cout<<"Original x value is: "<<x<<endl;

cout<<"Enter any number between 1 and 5: ";

cin>>a;

if(a==1)     {

cout <<" This is the main if condition " <<endl;
```

```
x=x+1;
cout<< " Value of x is increased by 1 : "<<x;    }
else if(a==2)    {
cout <<" This is second else if condition "<<endl;
x=x+2;
cout<< " Value of x is increased by 2 : "<<x;     }
else if(a==3)    {
cout <<" This is third else if condition " <<endl;
x=x+3;
cout<< " Value of x is increased by 3 : "<<x;    }
else if(a==4)     {
cout <<" This is fourth else if condition "<<endl;
x=x+4;
cout<< " Value of x is increased by 4 : "<<x;     }
else if(a==5 )     {
cout <<" This is fifth else if condition "<<endl;
x=x+5;
cout<< " Value of x is increased by 5 : "<<x;     }
else      {
```

```
cout<<"The number choosen is not in between 1 to 5"<<endl;

x=0;

cout<< " Value of x is made to zero : "<<x;     }

}
```

Switch Statement in C++?

Switch case statements are controlled statement that is regarded as a substitute for if-else statements. It is a multiway branch statement that provides a way to organize the flow of execution to parts of code based on the value of the expression. In a very basic term, a switch statement evaluates an expression, tests it and compares it against the several cases written in the code. As soon as a match with any case is found, the control will enter that case and start executing the statements written in that case until a break statement has been found. As soon as a break statement appears, the switch statement terminates and the program control exits switch.

It might sometimes happen that no case matches up with the value of the expression. For such cases, we mention a default case that will always execute in case if no match is found. The cases in a block of a switch statement are represented by different numbers or string, which is known as an identifier. The value of the expression or the value provided by the user is compared with these cases until the match is found.

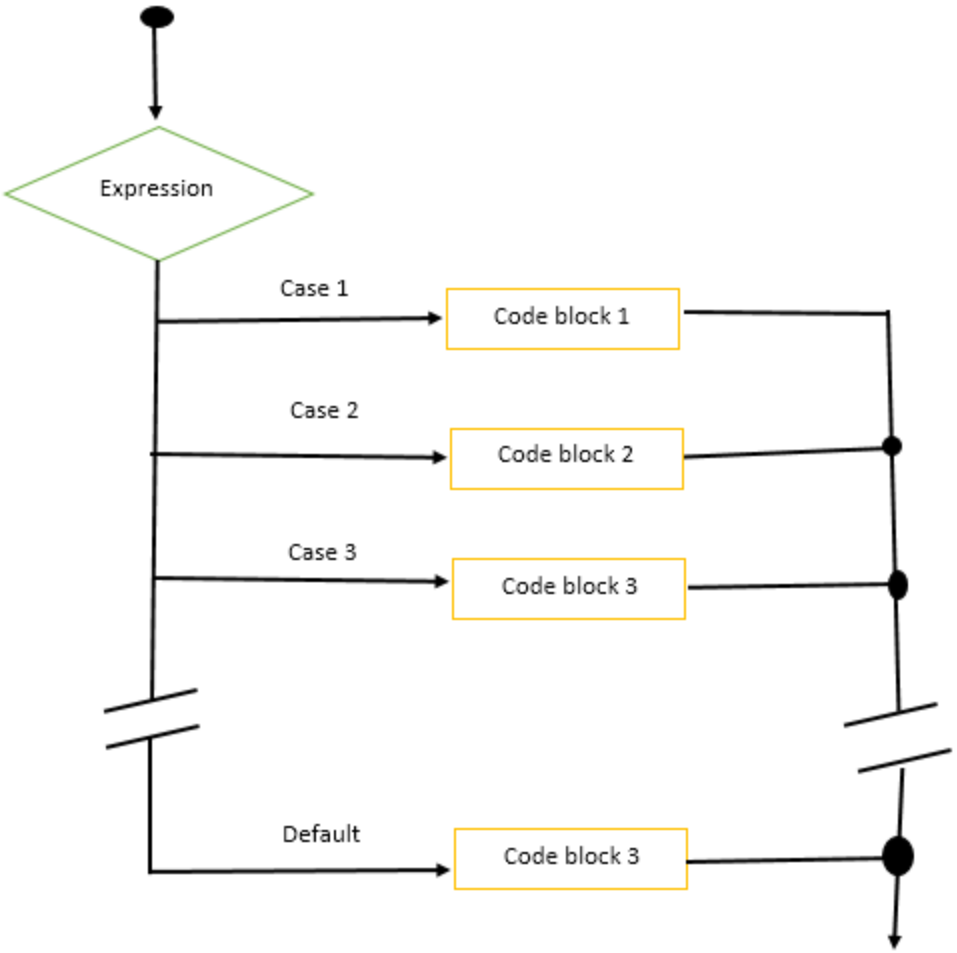
The syntax for switch statement in C++ programming language is given below-

```
switch( expression )
{
case value1:    //Block of code;    break;
case value2:    //Block of code;    break;
case valueN:    //Block of code;   break;
default:        //Block of code;     break;
```

You need to keep following things in mind when using a Switch Statement :

1. **Case labels can be an integer or a character and they should be unique**
2. **Case labels always end with a semicolon.**
3. **Even though a default case label is not mandatory, it can at most be one if defined.**
4. **You need a break statement to take the control out of the loop otherwise all the cases before a break would be executed.**
5. **The default case label does not have any specific position.**
6. **The switch statement can also be nested.**

Flowchart of Switch Statement



How does Switch Statement work in C++?

Let us understand the flow of control depicted in the flowchart above in order to gain a better understanding of the flow of execution.

An expression is passed with the switch statement which is equal to one of the values of the cases. In case the value is not equal, the default case is executed. The value of this expression is then compared with the case identifier or the first case. If the first case matches then the block of code associated with the first case is executed. Once the break is encountered, the execution stops and you will exit the switch statement.

However, if the case does not match, the execution flows to the next case. If this case matches, then the second code block executes otherwise, the flow checks the next case in a similar way. Finally, if no case matches then the default code block is executed.

Example :
This example will give more clarity about the use of switch statements.

```
#include <iostream>

using namespace std;

int main () {

char grade_report = 'D';

cout << "Your performance is: " << endl;

switch(grade_report) {

case 'A' :   cout << "Outstanding Result!\n"  << endl;   break;

case 'B' :     cout << "Excellent Result!\n"  << endl;   break;

case 'C' :    cout << "Good Result\n"  << endl;    break;

case 'D' :   cout << "Satisfying Result\n"  << endl;   break;

case 'F' :    cout << "Poor Result\n"  << endl;    break;

default :    cout << "You did not appear for exam\n"  << endl;   }

return 0;   }
```

Example :

This example depicts the use of the break statement in a switch. If the break statement is not specified after the case, the execution flow will continue until it encounters the break statement.

```
#include <iostream>

using namespace std;

int main() {

int  range_of_number=50;

switch (range_of_number) {

case 10:

case 20:

case 30:

cout << "The number is 10 or 20 or 30 " << endl;     break;

case 50:

case 55:cout << "This case also executes because there is no break " << endl;

cout << "\n" << endl;

case 60:

cout << "The number is either 40 or 50 or 60" << endl;     break;

default:

cout << "The number is greater than 60" << endl; }}
```

Example :

```
#include <iostream>

using namespace std;

int main() {

int x = 10, y = 5;

switch(x==y && x+y<10)  {

case 1:

cout << "hi" << endl;   break;

case 0:

cout << "bye" << endl;      break;

default:   cout << " Hello bye " << endl;

}}
```

Loops in C++

Loop statements in C++ execute the certain block of the code or statement multiple times, mainly used to reduce the length of the code by executing the same function multiple times, reduce the redundancy of the code. C++ supports various types of loops like for loop, while loop, do-while loop, each has its own syntax, advantages, and usage.

In the programming world, the loop is a control structure that is used when we want to execute a block of code, multiple times. It usually continues to run until and unless some end condition is fulfilled.

If we did not have loops, we would have to use the iterative method to print a repetitive block of statements, which would look something like this:

```
#include <iostream>

using namespace std;

int main()    {

cout << " Good morning \n";

cout << " Good morning \n";

cout << " Good morning \n";

cout << " Good morning \n";

cout << " Good morning \n";    }
```

In this example, we have printed “Good morning” five times by repeating the same set of lines.

A loop has a certain set of instructions. In a loop, we use a counter to check the condition for loop execution. In cases when the counter has not yet attained the required number, the control returns to the first instruction in the sequence of instructions and continues to repeat the execution of the statements in the block. If

the counter has reached the required number, that means the condition has been fulfilled, and control breaks out of the Loop of statements and comes outside of the loop, to the remaining block of code.

Types of Loops in C++

Now that we have seen how a Loop works, let us make it clearer by going through the types of Loops out there. In C++ programming, we have three types of Loops in C++ :

- **For Loop**
- **While Loop**
- **Do While Loop**

For Loop

Loop is an entry controlled loop, meaning that the condition specified by us is verified before entering the loop block. It is a repetition control structure. The loop written by us is run a specified number of times.

To control the loop, we use a loop variable in For loop. This variable is first initialized to some value, then we perform the check on this variable comparing it to the counter variable, and finally, we update the loop variable.

Syntax:

```
for(initialization expression; test expression; update expression)
{
```

```
// statements to execute in the loop body
}
```

Initialization Expression:

Here we initialize the loop variable to a particular value. For example, int i=1;

Test Expression:

Here, we write the test condition. If the condition is met and returns true, we execute the body of the loop and update the loop variable. Otherwise, we exit the For loop. An example for test expression is i <= 5;

Update Expression:

Once the body of the loop has been executed, we increment or decrement the value of the loop variable in the update expression. For example, i++;

Let us look at an example of For loop:

```
#include <iostream>
using namespace std;
int main(){
for (int i = 1; i <= 5; i++) {
```

```
cout << " Good morning \n";}

return 0;}
```

While Loop

While loop is also an entry controlled loop, where we verify the condition specified by us, before running the loop. The difference is being that we use For loops when we know the number of times the body of the loop needs to run, whereas we use while loops in circumstances when beforehand we do not know the precise number of times the body of the loop needs to run. The execution of the loop is terminated based on the test condition.

Syntax:

```
initialization expression;
while (test_expression)
{
// statements to execute in the loop body
update_expression;
}
```

The syntax of the loops differs only in the placement of the three expression statements.

Let us look at an example of while loop:

```
#include <iostream>

using namespace std;

int main()

{

int i = 0;    // initialization expression

while (i < 5)   // test expression

{

cout << "Good morning\n";

i++;    // update expression

}

return 0;

}
```

Do While Loop

Do while loop is an exit controlled loop, meaning the test condition is verified after the execution of the loop, at the end of the body of the loop. Hence, the body executes at least once, regardless of the result of the test condition, whether it is true or false. This happens to be the foremost difference in between while loop and do while. In while loop, the condition is tested beforehand, whereas in do while loop the condition is verified at the finish of body of the loop.

```
initialization expression;
do
{
// statements to execute in the loop body
update_expression;
} while (test_expression);
```

In do while loop, we end the body of the loop with a semicolon, whereas the other two loops do not have any semicolon to end the body of their loops.

```
#include <iostream>
using namespace std;
int main()
{
int i = 2;    // initialization expression
do{
cout << " Good morning\n";
i++;      // update expression
} while (i < 1);  // test expression
return 0;}
```

In the above-given code, the test condition says I should be less than 1 (i<1), but still the loop executes at least once, before checking for the condition, hence giving us the output "Good morning" one time.

Infinite Loop

An infinite loop or an endless loop is a loop that does not have a proper exit condition for the loop, making it run infinitely. This happens when the test condition is not written properly and it permanently evaluates to true. This is usually an error in the program.

```
#include <iostream>

using namespace std;

int main ()  {

int i;

for ( ; ; )  {

cout << "This loop runs indefinitely.\n";}}
```

This loop runs indefinitely.

Example: – Program to print the multiplication table in C++.

```
#include <iostream>

using namespace std;

int main()  {
```

```
int n, count = 0, limit;

cout << "Enter the value to find the multiplication table: ";

cin >> n;

cout << "Enter the maximum limit for multiplication table: ";

cin >> limit;

do  {

cout << n << "*" << count << " = " << n*count <<endl;

count++;  }

while(count <= limit);

return 0;

}
```

Code Explanation: Here we have written a code to print the multiplication table which the user wants to print. Here we have initialized three variables n to take the number from the user and count to count the number and limit to restrict the limit for the multiplication table. Do keyword will execute the statement. First, it calculates the multiple of the values entered by the user and it prints. The count is incremented by 1 per iteration. While checking the condition whether the count is greater than or equal to the limit or not. Based on the result it prints the multiplication table.

Example : Program to print elements of array using do while loop.

```
#include <iostream>

using namespace std;

int main()  {

int i = 0;

int array[] = {2,7,19,5,8};

do

{

cout << array[i] << endl;

i++;

}while(i <= 4);

return 0;  }
```

:

Code Explanation: Here we have written a program to print the array elements using a do while loop in C++ programming. First, we have initialized variable i to 0 and declare the array elements. do loop will print the array elements from the list. i is used as a counter to increment the value by 1. While keyword contains the condition which counts i.e. i must be less than or equal to 4.

Example : Program to add numbers until the user enters 0.

```
#include <iostream>

using namespace std;

int main()

{

float f_num, Total = 0.0;

do

{

cout << "Enter a number: ";

cin >> f_num;

Total += f_num;

}

while(f_num != 0.0);

cout << "Toatal Sum = " << Total;

return 0;

}
```

Code Explanation: Here we have written a program to calculate the total of the entered numbers. For a change, here we have applied a condition that states that it will ask a user to enter a number unit it enters 0 and in the end, it calculated the total of the numbers. Note that, here we have used float data type. It enables the user to enter the decimal values.

```
# include<iostream>
using namespace std;
int main()
{
for (int i=1;1<=10;i++)
Cout<<i<<endl;
return 0;
}
```

```
# include<iostream.h>
using namespace std;
int main()
{
For (int i=1;1<=10;i+=2)
Cout<<i<<endl;
Return 0;
}
```

```
# include<iostream.h>
using namespace std;
int main()
{
Int sum=0;
For (int i=1;1<=10;i+=2)
  sum=sum+i;
Cout<<i<<endl;
Return 0;}
```

Math Functions in C++

C++ provides <math.h> library for math functions to perform the complex mathematical functions like trigonometric function, algebraic equations easily. For example, sin() function is used to calculate the value of sin, pow() the function is used to calculate the power of the value, sqrt is used to calculate the square root of the value.

Different Types of Math Functions

C++ provides a huge number of different types of math functions mentioned below with examples:

1. Maximum & Minimum function

- **max (p,q):** It will return a maximum number between p and q.
- **min (p,q):** It will return a minimum number between p and q.

C++ Code to Implement Above Functionality

```
#include <iostream>

#include <math.h>

using namespace std;

int main() {

cout << max(16,18) << "\n";

cout << min(16,18) << "\n";

return 0;

}
```

2. Power functions

- **pow (m,n):** It will calculate m raised to the power n.
- **sqrt(m):** It will calculate the square root of m.
- **cbrt(n):** It will calculate the cube root of n.
- **hypot(m,n):** It will calculate the hypotenuse of the right-angled triangle.

C++ code to implement above functionality

```
#include <iostream>

#include <math.h>

using namespace std;

int main() {

cout << pow(2,3) << "\n";

cout << sqrt(16) << "\n";

cout << cbrt(27) << "\n";

cout << hypot(3,4) << "\n";

return 0;

}
```

3. Exponential functions

- **exp(p):** It will calculate the exponential e raised to power p.
- **log(p):** It will calculate the logarithm of p.
- **log10(p):** It will calculate the common logarithm of p.

- **exp2(p):** It will calculate the base 2 exponential of p.
- **log2(p):** It will calculate the base 2 logarithm of p.
- **logb(p):** It will calculate the logarithm of p.

C++ code to implement above functionality

```
#include <iostream>

#include <math.h>

using namespace std;

int main() {

cout << exp(5) << "\n";

cout << log(8) << "\n";

cout << log10(8) << "\n";

cout << exp2(5) << "\n";

cout << log2(8) << "\n";

cout << logb(8) << "\n";

return 0;

}
```

4. Integer functions

It helps in finding the nearest integer value.

- **ceil(z):** it rounds up the value of z.
- **floor(z):** it rounds down the value of z.

- **round(z):** It rounds off the value of z.
- **fmod(z,y):** It calculates the remainder of division z/y.
- **trunc(z):** It will round off the z value towards zero.
- **rint(z):** It will round off the z value using rounding mode.
- **nearbyint(z):** It will round off the z value to a nearby integral value.
- **remainder(z,y):** It will calculate the remainder of z/y.

C++ code to implement above functionality

```
#include <iostream>

#include <math.h>

using namespace std;

int main() {

cout << ceil(4580.01) << "\n";

cout << floor(151.999) << "\n";

cout << round(518.5) << "\n";

cout << fmod(5,21) << "\n";

cout << trunc(20.25) << "\n";

cout << rint(21.25) << "\n";

cout << nearbyint(182.55) << "\n";

cout << remainder(12,36) << "\n";     return 0; }
```

5. Comparison functions

Help in comparing numbers in a quick span doesn't matter how long the number is.

Below are a few examples of Comparison functions:

- **isgreater(p,q):** It checks whether p is greater than q or not.
- **islessequal(p,q):** It checks whether p is less than or equal to q or not.
- **isgreaterequal(p,q):** It checks whether p is greater than or equal to q or not.
- **islessgreater(p,q):** It checks whether p is less or greater than y or not.
- **isunordered(p,q):** It checks whether p compared or not.

C++ code to implement above functionality

```
#include <iostream>

#include <math.h>

using namespace std;

int main() {

// cout << less(22,29) << "\n";

cout << isgreater(48,47)<< "\n";

cout << islessequal(11,5)<< "\n";

cout << isgreaterequal(19,72)<< "\n";

cout << islessgreater(59,84)<< "\n";

cout << isunordered(62,84)<< "\n";       return 0; }
```

6. Using Trigonometric Function

Functions specially used in geometric calculations. The right-angled triangle gives a relation between angle to the ratio of the length of the two sides.

- **sin(y):** It will calculate the value of sine y.
- **cos(y):** It will calculate the value of cosine y.
- **tan(y):** It will calculate the value of tangent y.
- **asin(y):** It will calculate the value of inverse sine y.
- **acos(y):** It will calculate the value of inverse cosine y.
- **atan(y):** It will calculate the value of inverse tangent y.
- **atan2(y,x):** It will calculate the value of the inverse tangent of y and x coordinates.

C++ code to implement above functionality

```
#include <iostream>

#include <math.h>

using namespace std;

int main() {

cout << sin(0) << "\n";

cout << cos(0) << "\n";

cout << tan(1) << "\n";

cout << asin(1)<< "\n";

cout << acos(0)<< "\n";
```

```
cout   <<   atan(1)<< "\n";

cout   <<   atan2(0,1)<< "\n";

return 0;

}
```