History of C++

C++ programming language was developed in the year 1980 by Mr.Bjarne Stroustrup at the well established Bell Laboratories of the American Telephone & Telegraph (AT&T) company positioned in the United State of America.

Bjarne Stroustrup is the founder of the C++ Programming Language.

He developed C++ Programming Language in order to integrate the Object-Oriented style of programming into C Language without having it to make any significant change to the C Fundamentals.

Features of C++

C++ Programming Language is loaded with many performance-oriented features which are mentioned as follows:

- Rich Library

C++ language incorporates multiple built-in arithmetic and logical functions along with many built-in libraries which make development faster and convenient.

- Object-Oriented

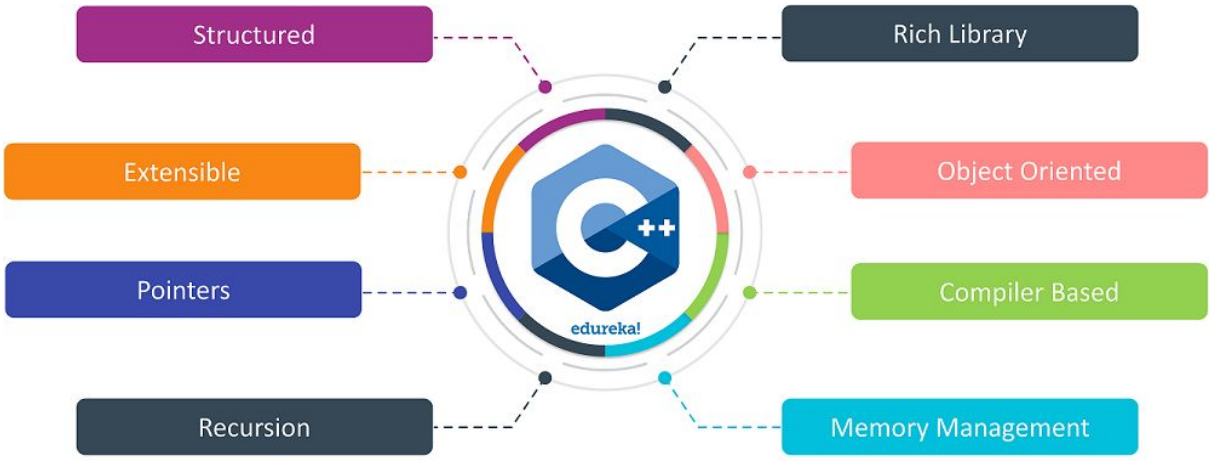
C++ Programming Language is designed to be an object-oriented programming language. OOPS feature makes development and maintenance easier whereas in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

- Compiler Based

C++ Programming Language is a compiler-based programming language, it means without compilation no C++ program can be executed. First, we need to compile our program using the compiler and then we can execute our program.

- Memory Management

C++ provides the best in class memory management. It can both allocate and deallocate memory dynamically at any instance of time.



- Recursion

C++ Language supports function back-tracking which involves recursion. In the process of recursion, a function is called within another function for multiple numbers of times.

- Pointers

C++ enables users to directly interact with memory using the pointers. We use pointers in memory, structure, functions, arrays, stack and many more.

- Extensible

C++ Programming Language is highly extensible because of its easily adaptable features.

- Structured

C++ Language supports structured programming which includes the use of functions. Functions reduce the code complexity and are totally reusable.

C++ Installation

Let us follow the steps below in order to install C++ in our local systems.

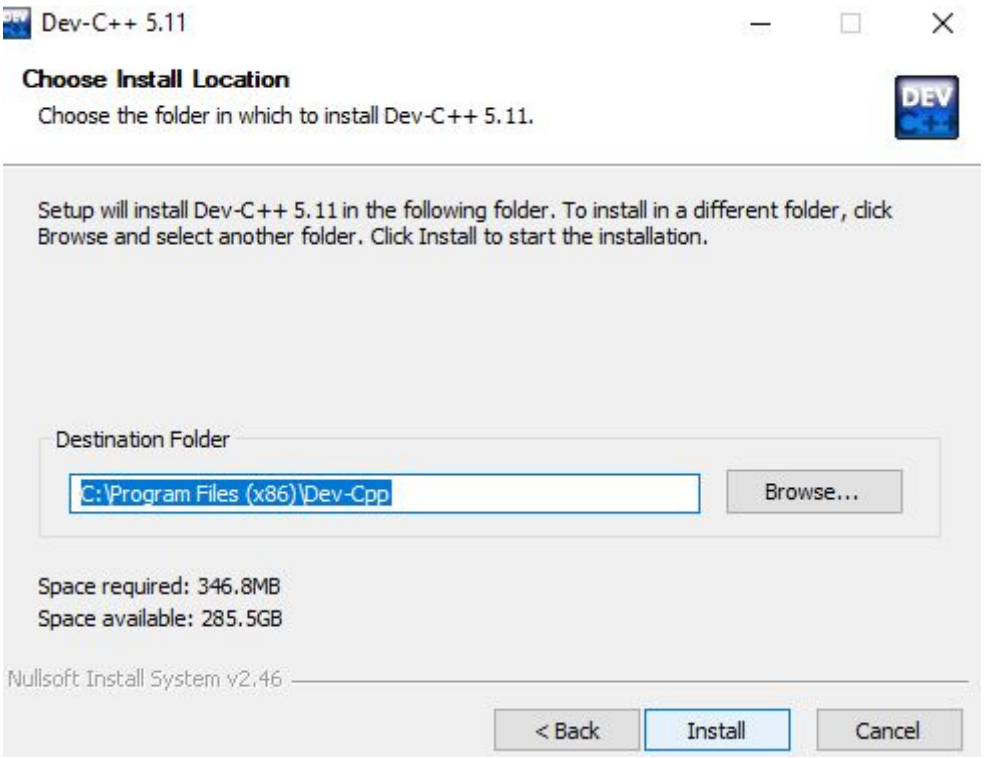
- Step 1: Select your preferred Text Editor.

C++ can be used using various Text editors like TurboC, CodeBlocks, Dev.C++, Visual Studio, Eclipse and many more.

- Step 2: Installing MinGW on to your Local System.

After selecting your text editor, you need a compiler to interpret your code to the computer system. We need MinGW for the same. Once you download MinGW,

you need to set the environment variables for the same.



Execute your first C++ Program

After installing C++ compiler and a Text Editor of your choice, you can directly go ahead and execute your first basic C++ Program, The Hello World Program.

```
1
2       #include <iostream.h>
3
4       #include<conio.h>
5
6       void main()
7
8       {
            clrscr();
            cout << "Hello World..! Welcome to C++ Programming.";
            getch();
        }
//Output:
```

Hello World..! Welcome to C++ Programming.

Structure of a C++ Program

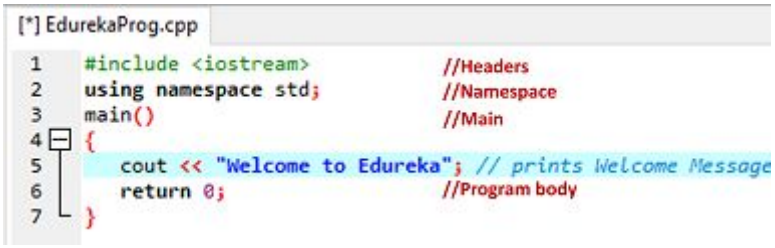
The Basic Structure of a C++ program is as follows:

- **C++ Headers:**

The lines starting with Hash Symbol(#) are considered as Header Files. All the Header Files used in a program have their own unique meaning. for example, the header file #include <iostream.h> instructs the preprocessor to perform standard input and output operations, such as writing the output of our first program (Hello World) to the screen.

- **Namespace:**

A namespace allows grouping of various members like classes, objects, functions and various C++ tokens under a single name. various users can create separate namespaces and thus can use similar names of the members. This avoids the compile-time error that may exist due to identical-name conflicts.



- **Class Definition:**

The class is a user-defined data structure declared with keyword class. It includes data and functions as its members whose access is governed by the three access specifiers namely, private, protected and public. By default access to members of a C++ class is private.

- **Member Function Definition:**

Member functions are the operators and functions that are declared as members of a particular class. Member Functions are also called friends of the main class.

- **Main Function:**

main() function is known as the entry point for every C++ program. When a C++ program is executed, the execution control goes directly to the main() function.

Identifiers

Identifiers in C++ Language are none other than the names assigned to variables and functions used in a C++ Program. The valid identifiers in a C++ program are declared using the alphabets from A to Z, a to z and 0 to 9. The Identifiers cannot include special symbols and white spaces, but, it can include underscores(_)

Some basic examples of valid Identifiers are as follows:

int A;

float x;

char Edureka_Happy_Learning;

Keywords

Similar to Identifiers, we do have some special and reserved words in a C++ Program which have special meaning. Such words are called Keywords. Some of the Keywords used in C++ Programming Language are as follows.

wchar_t	default	break	case	char	const	continue	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while
delete	dynamic_cast	const_cast	catch	class	namespace	mutable	inline
export	explicit	template	static_cast	reinterpret_cast	public	false	friend
protected	private	true	try	typeid	typename	using	virtual

New Keywords Introduced from C++ Language

asm	delete	virtual	template	static_cast	reinterpret_cast	true	class
explicit	typeid	wchar_t	inline	friend	false	virtual	const_cast
operator	typename	dynamic_cast	mutable	public	private	bool	try
this	using	new	namespace	protected	throw	catch	

White Space

Whitespace can be defined as the term used in C++ to define *blank spaces, tab spaces, newline characters* and *comments.* Whitespace separates one statement from another and enables the compiler to identify where one element in a statement, such as an int, ends and the beginning of the next element.

Example:

```
char Edureka_Happy_Learning;

vehicles = cars + bikes + busses; // Get the total number of vehicles
```

Comments

C++ Program comments are user-written statements that are included in the C++ code for explanation purposes. These comments help anyone reading the source code. All programming languages allow for some form of comments. C++ supports single-line and multi-line comments.

Single-Line Comments:

```
#include <iostream>

using namespace std;

main() {

  cout << "Welcome to Edureka"; // prints Welcome Message

  return 0;

}
```

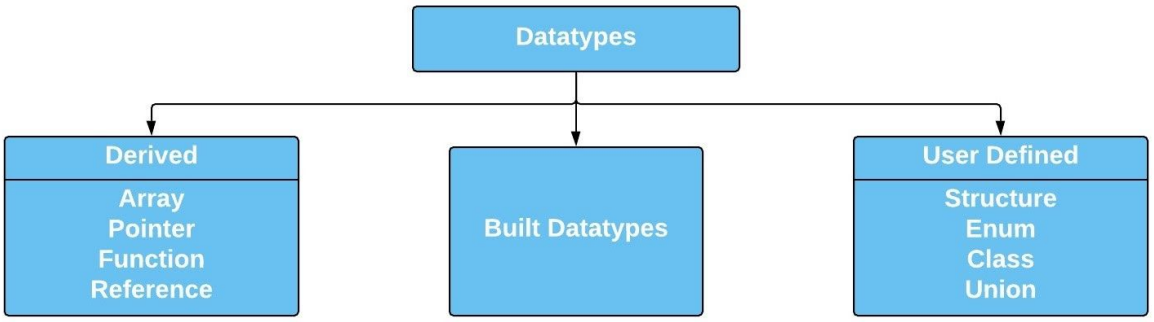
Multi-Line Comments:

```
/* Comment out printing of Happy Learning

cout << "Happy Learning"; // prints Happy Learning Message

*/
```

Next, we will discuss Datatypes

Datatypes

Types of Datatypes available in C++ Language are as follows:



Derived Datatypes

The following table shows the derived datatype

Built-in Datatypes

The following table shows the built-in datatype and the maximum and minimum value which can be stored in the designated variables.

Type	Bit Width	Typical Range
char	1 byte	-127 to 127 or 0 to 255
signed char	1 byte	-127 to 127
unsigned char	1 byte	0 to 255
short int	2 bytes	-32768 to 32767
signed short int	2 bytes	-32768 to 32767
unsigned int	2 bytes	0 to 65535
int	4 bytes	-2147483648 to 2147483647
signed int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295
long int	4 bytes	-2,147,483,648 to 2,147,483,647
signed long int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	0 to 4,294,967,295

float	4 bytes	+/- 3.4e +/- 38
double	8 bytes	+/- 1.7e +/- 308
long double	8 bytes	+/- 1.7e +/- 308
wchar_t	4 bytes	1 wide character

User-Defined Datatypes

The following table shows the User-Defined datatypes.

Structure

The Structure is known as a collection of variables of different datatype under a single name. It is completely similar to a normal class that holds a collection of data of different data types.

Example:

```
1
2       #include <iostream>
3
4       using namespace std;
5
6       struct Point {
7           int a, b;
8
9       };
10
11
12
13      int main()
        {
            struct Point p1 = { 0, 10 };
            p1.a = 25;
            cout << "a = " << p1.a << ", b = " << p1.b;
            return 0;
        }
//Output:
```

a = 25, b = 10

Enum

The enumeration in C++ is a user-defined datatype. It is designed to assign names to integral constants, the names increase the readability of a program and also the program becomes easy to maintain.

Example

```
1
2       #include<stdio.h>
3
4           enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
5
6       int main()
7
8       {
9
            enum week day;

            day = Wed;

            printf("%d",day);

            return 0;

        }
//Output:

2
```

Class

A class in C++ is actually a user-defined datatype. it's declared with the keyword class at the side of its members with personal, protected or public access specifiers. By default access to members of a C++ class is private.

Example:

```
1
2       #include <iostream>
3
4       using namespace std;
5
6       class Student {
7
8          public:
9
10            int ID;
11
12            string Name;
13
14      };
15      int main() {
            Student s1;
            s1.ID = 12012009;
            s1.Name = "Arjun";
            cout<<s1.ID<<endl;
            cout<<s1.Name<<endl;
            return 0;
        }
```

//Output:

12012009

Arjun

Union

The Union is completely similar to a structure in terms of declaration and usage but all the members of a union occupy the same space in memory that is, the memory location is shared among the union members.

```
1
2       #include<iostream>
3
4       using namespace std;
5
6       typedef union type {
7
8           int a;
9
10          float b;
11
12      }
13      my_type;
14
15
16
17      int main()
18
        {
            my_type x;
            x.a = 5;
            cout << "a : " << x.a << endl;
            x.b = 7.3;
            cout << "a : " << x.a << endl;
            cout << "b : " << x.b << endl;
            return 0;
        }
//Output:
a : 5
a : 1089051034
b : 7.3
```

Variables

The variable in C++ is a user-defined name for a particular memory in the computer reserved to store data and process it. The variables in C++ are divided into two types based on their scopes.

- Global Variables

Global variables are defined outside of all the functions. The global variables will hold their value throughout the lifetime of your program. That is, a global variable is available for use throughout your entire program after its declaration.

Example:

```
1
2       #include <iostream>
3
4       using namespace std;
5
6
7
8       int g;
9
10
11      int main () {
12
13        int a, b;

          a = 10;

          b = 20;

          g = a + b;

          cout << g;

          return 0;

        }
```

//Output:

30

- Local Variables

Local variables are declared inside a function. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Example:

```
1
2       #include <iostream>
3
4       using namespace std;
5
6
7       int main () {
8
9         int a, b;
10
11        int c;
12
          a = 10;

          b = 20;

          c = a + b;

          cout << c;

          return 0;

        }
```

//Output:

30

Modifiers in C++

Modifiers in C++ allows the char, int, and double data types to have modifiers preceding them. A modifier is used to alter the meaning of the base type so that it more precisely fits the needs of various situations.

The datatype modifiers available in C++ are:

- signed
- unsigned
- long
- short

To understand the functionality of a Modifier, Let us execute the following program.

```
1
2       #include <iostream>
3
4       using namespace std;
5
6
7       int main() {
8
9         short int i;
10
11        short unsigned int j;

          j = 50000;

          i = j;

          cout << i << " " << j;

          return 0;

        }
//Output:

-15536 50000
```

The above result is because the bit pattern that represents 50,000 as a short unsigned integer is interpreted as -15,536 by a short.

Type Qualifiers in C++

The type qualifiers provide additional information about the variables they precede. There are three different Type Qualifiers in C++ Language.

- const

Objects of type const cannot be changed by your program during execution.

- volatile

The modifier volatile tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.

- restrict

A pointer qualified by restrict is initially the only means by which the object it points to can be accessed. Only C99 adds a new type qualifier called restrict.

Operators in C++

An operator is a special symbol that is used to take up a specific operation. C++ language consists of many types of operations like arithmetic, logical, bit-wise and many more. The following are the types of operators in C++ language.

- Binary Operator
    - Arithmetic Operators
    - Relational Operators
    - Logical Operators
    - Bit-wise Operators
    - Assignment Operator
- Unary Operator
- Ternary Operator

Arithmetic Operators

There are following arithmetic operators supported by C++ language. Assume variable A holds 20 and variable B holds 40, then

Operator	Description	Example
+	Checks if the values of two operands are equal or not, if yes then condition becomes true.	A + B will give 60
–	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	A – B will give -20
*	Checks if the value of the left operand is greater than the value of right operand, if yes then condition becomes true.	A * B will give 800
/	Checks if the value of the left operand is less than the value of right operand, if yes then condition becomes true.	B / A will give 2
%	Checks if the value of the left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	B % A will give 0
++	Checks if the value of the left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	A++ will 21

—	Checks if the value of the left operand is less than or equal to the value of right operand, if yes then condition becomes true.	A– will give 19

Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of the left operand is greater than the value of right operand, if yes then condition becomes true.	(A>B) is not true.
<	Checks if the value of the left operand is less than the value of right operand, if yes then condition becomes true.	(A<B) is true.
>=	Checks if the value of the left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A>=B) is not true.
<=	Checks if the value of the left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A<= B) is true.

Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

A	B	A&B	A||B	A^B
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

The Bitwise operators supported by C++ language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then the results will be as shown below

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111.

Assignment Operators

There are following assignment operators supported by C++ language.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B
+=	Add AND assignment operator, It adds right operand to the left operand and saves the result to the left operand.	C += A is C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and saves the result to the left operand.	C -= A is C = C – A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and saves the result to the left operand.	C *= A is C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and saves the result to the left operand.	C /= A is C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and saves the result to the left operand.	C %= A is C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is C = C << 2
<<=	Right shift AND assignment operator.	C >>= 2 is C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is C = C | 2

C++ Operators according to their Precedence

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

Conditional Statements

Conditional statements

Conditional statements are used to execute statement or group of statements based on some conditon. If the condition is true then C++ statements are executed other wise next statement will be executed.

Different types of Conditional Statements in C++ Language are as follows:

1. If statement
2. If-Else statement
3. Nested If-else statement
4. If-Else If ladder
5. Switch statement

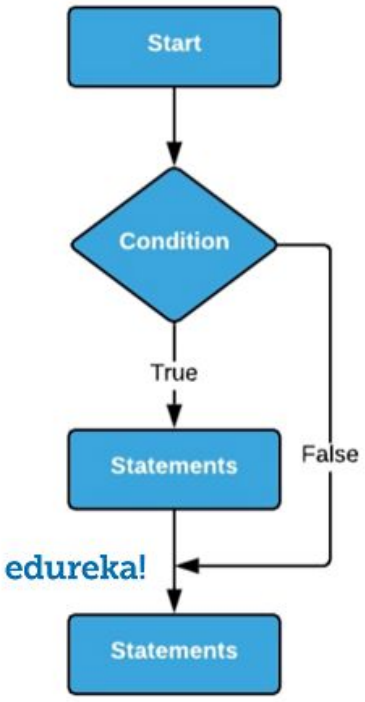
If statement

The single if statement in C++ language is used to execute the code if a condition is true. It is also called one-way selection statement.

Syntax

```
1
2       if(boolean_expression) {
3
           // statement(s) will execute if the boolean expression is true

        }
```

Flowchart



Example:

```
#include <iostream>
using namespace std;
int main () {
  int a = 100;
  if( a < 200 )
  {
    cout << "a is less than 200" << endl;
  }
  cout << "value of a is: " << a << endl;
  return 0;

}
```

//Output:

a is less than 200

value of a is: 100

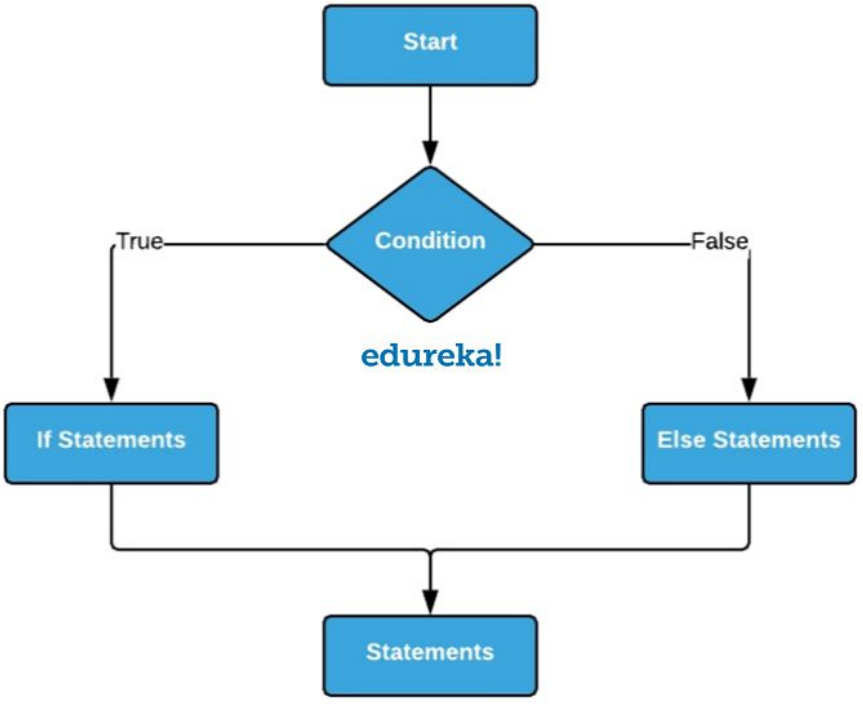
If-Else statement

The if-else statement in C language is used to execute the code if the condition is true or false. It is also called two-way selection statement.

Syntax

```
1
2       if(boolean_expression) {
3
4          // statement(s) will execute if the boolean expression is true
5
        } else {

          // statement(s) will execute if the boolean expression is false

        }
```

Flowchart



Example:

```
1       #include <iostream>
2       using namespace std;
3
4       int main()
5       {
6           int a = 100, b = 200;
7           if (b > a)
8           {
9               cout << "b is greater" << endl;
10          }
11          else
12          {
13              cout << "a is greater" << endl;
14          }
15          system("PAUSE");
16
        }
//Output:
```

b is greater

Press any key to continue . . .

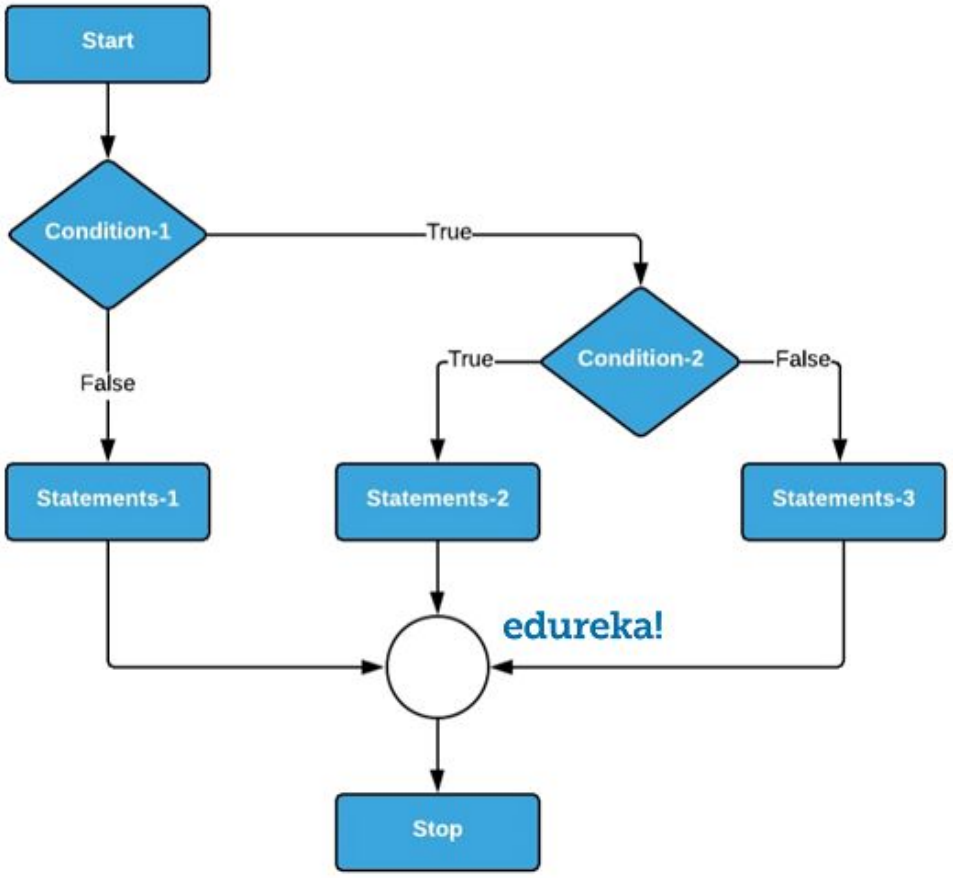
Nested If-else statement

The nested if-else statement is used when a program requires more than one test expression. It is also called a multi-way selection statement. When a series of the decision are involved in a statement, we use if-else statement in nested form.

Syntax

```
1
2       if( boolean_expression 1) {
3
4         // Executes when the boolean expression 1 is true
5
6         if(boolean_expression 2) {

            // Executes when the boolean expression 2 is true

          }

        }
```

Flowchart



Example:

```
1       #include <iostream>
2       using namespace std;
3
4       int main ()
5       {
6         int a = 1000;
7         int b = 2500;
8         if( a == 100 )
9         {
10          if( b == 200 )
11
12          {
13
14            cout << "Value of a is 1000 and b is 2500" << endl;
15
16          }
17
18      }
```

```
        cout << "The Exact value of a is: " << a << endl;

        cout << "The Exact value of b is: " << b << endl;

        return 0;

    }
```

//Output:

The exact value of a is: 1000

The exact value of b is: 2500

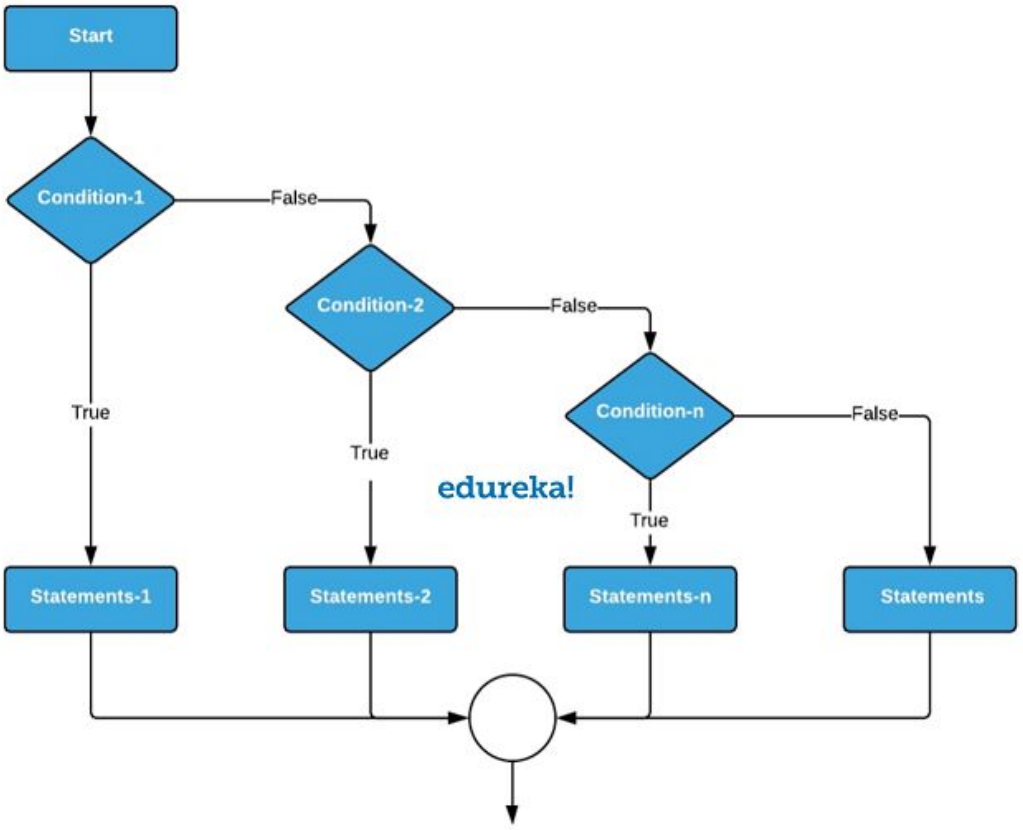
Else-if Ladder

The if-else-if statement is used to execute one code from multiple conditions. It is also called multipath decision statement. It is a chain of if..else statements in which each if statement is associated with else if statement and last would be an else statement.

Syntax

```
1   if(boolean_expression 1)
2   {
3     // When expression 1 is true
4   }
5   else if( boolean_expression 2)
6   {
7     // When expression 2 is true
8   }
9   else if( boolean_expression 3)
10  {
11    // When expression 3 is true
12  }
13  else
14  {
15    // When none of expression is true
16  }
```

Flowchart


Start
Condition-1
False
Condition-2
False
Condition-n
False
True
True
True
edureka!
Statements-1
Statements-2
Statements-n
Statements

Example:

```
1       #include <iostream>
2       using namespace std;
3
4       int main(){
5          int score;
6          cout << "Enter your score between 0-100n"; cin >> score;
7          if(score >= 90)
8          {
9             cout << "YOUR GRADE: An"; } else if (score >= 50 && score < 90)
10         {
11            cout << "YOUR GRADE: Bn"; } else if (score >= 35 && score < 50)
12         {
13            cout << "YOUR GRADE: Cn";
14         }
15         else
16         {
17            cout << "YOUR GRADE: Failedn";
18         }
19         return 0;
20      }
```

//Output:

Enter your score between 0-100

60

YOUR GRADE: B

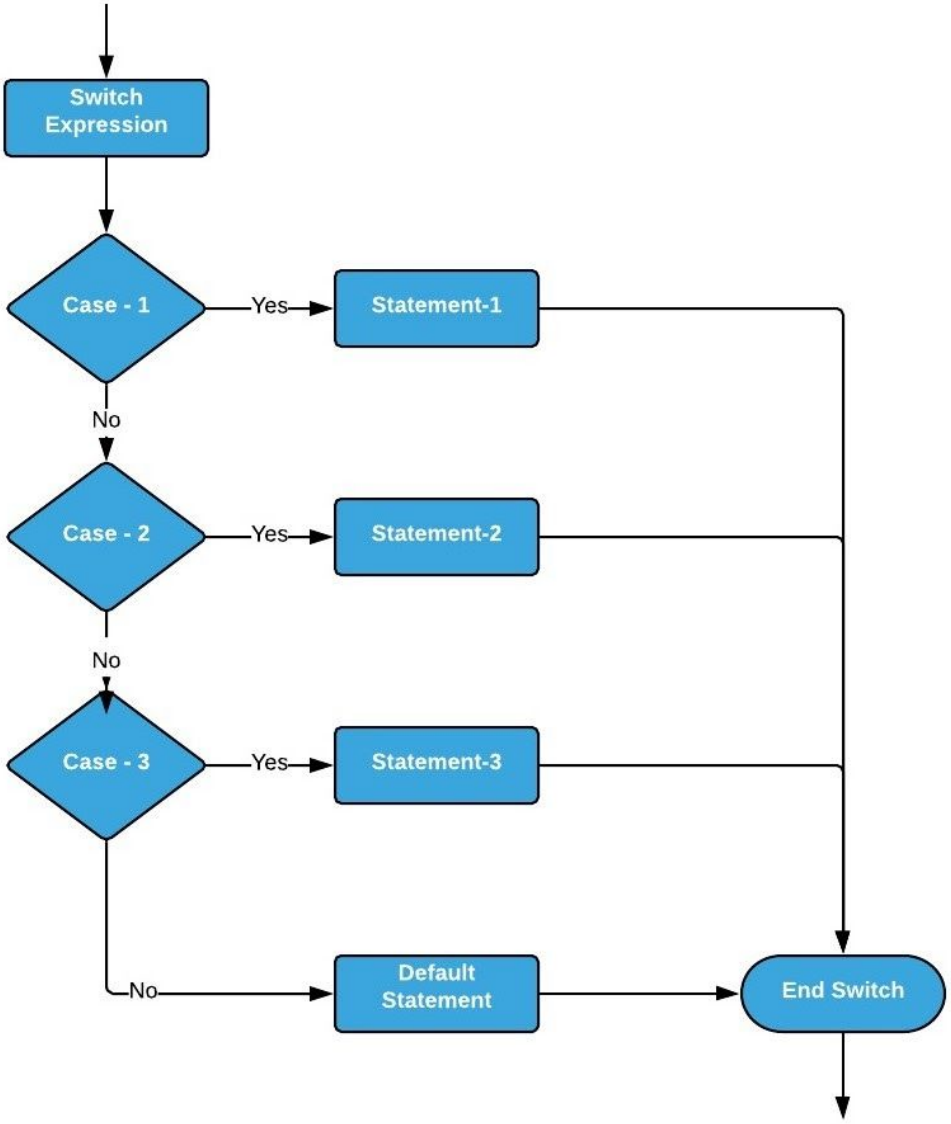
Switch statement

Switch statement acts as a substitute for a long if-else-if ladder that is used to test a list of cases. A switch statement contains one or more case labels which are tested against the switch expression. When the expression match to a case then the associated statements with that case would be executed.

Syntax

```
switch(expression) {
  case constant-expression  :
    statement(s);
    break; //optional
  case constant-expression  :
    statement(s);
    break; //optional

  // you can have any number of case statements.
  default : //Optional
    statement(s);
}
```

Flowchart



Example:

```
1       #include <iostream>
2       using namespace std;
3
4       int main() {
5          int x = 3;
6          switch (x)
7          {
8             case 1:
9             cout << "Choice is 1";
10            break;
11            case 2:
12            cout << "Choice is 2";
13            break;
14            case 3:
15            cout << "Choice is 3";
16            break;
17            default:
18            cout << "Choice other than 1, 2 and 3";
19            break;
20         }
21      return 0;
22      }
```

//Output:

Choice is 3

Loops in C++

A loop statement is used for executing a block of statements repeatedly until a particular condition is satisfied. The C++ Language consists of the following Loop Statements.

- For Loop
- While Loop
- Do While Loop

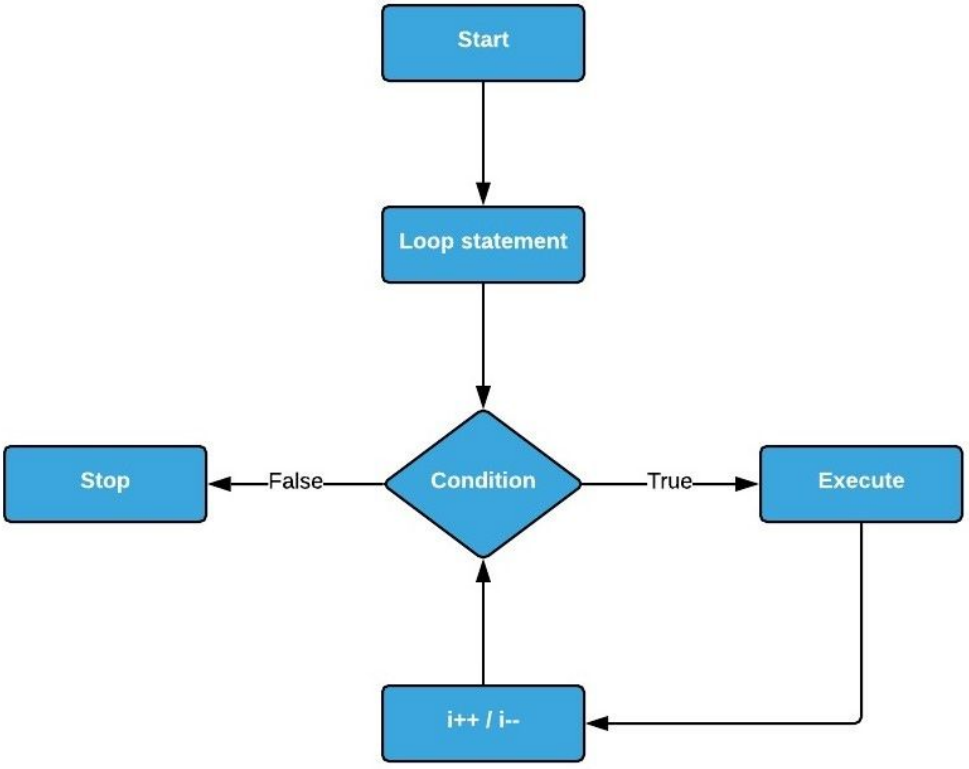
For Loop

The for loop is used to execute a particular code segment for multiple times until the given condition is satisfied.

Syntax

```
1
2       for(initialization; condition; incr/decr){
3
        //code to be executed

        }
```

Flowchart



Example:

```
1       #include <iostream>
2       using namespace std;
3       int main() {
4               for(int i=1;;i<=10;i++){
5                 cout<<i <<"n";
6               }
7           }
```

//Output:

1

2

3

4

5

6

7

8

9

10

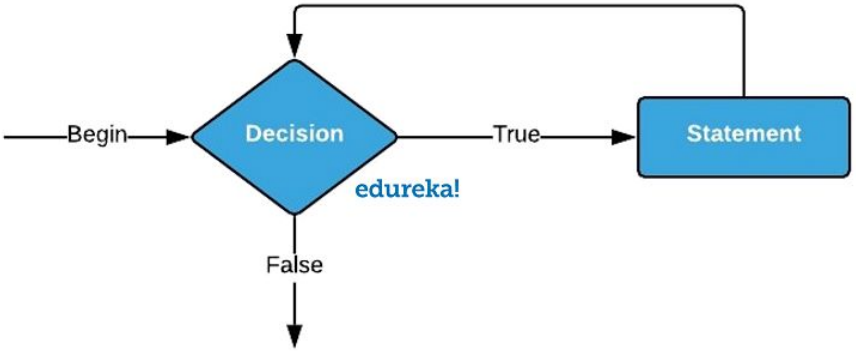
While Loop

The While loop is used to execute a code segment for multiple numbers of times until a specific condition is satisfied.

Syntax

```
1       while(condition){
2       //code to be executed
3       }
```

Flowchart



Example:

```
1       #include <iostream>
2       using namespace std;
3       int main()
4       {
5          int number, i = 1, factorial = 1;
6          cout << "Enter a positive integer: "; cin >> number;
7
8          while ( i <= number) {
9             factorial *= i;
10            ++i;
11         }
12         cout<<"Factorial of "<< number <<" = "<< factorial;
13         return 0;
14      }
```

//Output:

Enter a positive integer: 10

Factorial of 10 = 3628800

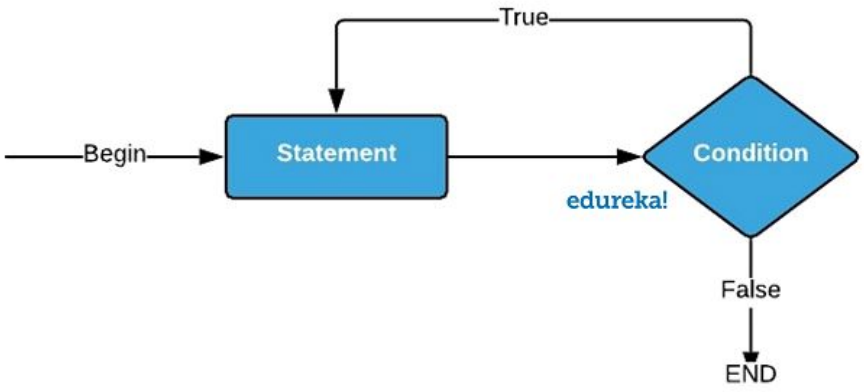
Do While Loop

Do while loop is completely similar to While Loop but the only difference is that the condition is placed at the end of the loop. Hence, the loop is executed at least for once.

Syntax

```
1       do
2       {
3          statement(s);
4       } while(condition);
```

Flowchart



Example:

```
1       #include <iostream>
2       using namespace std;
3       int main(){
4          int num=1;
5          do{
6             cout<<"Value of num: "<<num<<endl;
7             num++;
8          }while(num<=6);
9          return 0;
10      }
```

//Output:

Value of num: 1

Value of num: 2

Value of num: 3

Value of num: 4

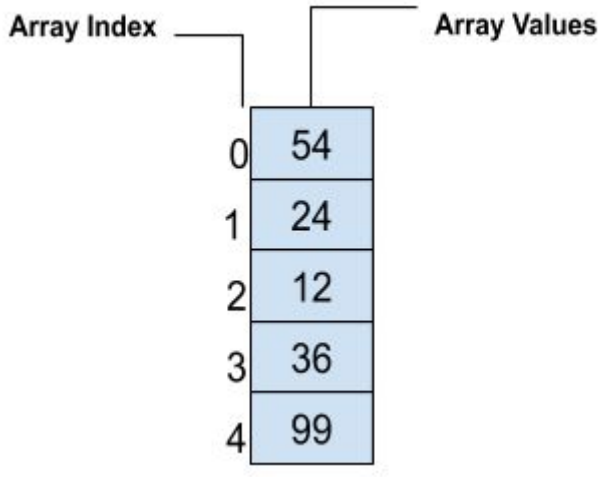
Value of num: 5

Value of num: 6

Array

An Array is a sequential collection of elements, of the same data type. They are stored sequentially in memory. An Array is a data structure that holds similar type of elements. The array elements are not treated as objects in c++ like they are in java.

Suppose there are 5 numbers we have to store in an array. So, the array will look like this:-



Arrays In C++ Imagine you are at a music record store and I tell you to arrange all the records under the label XYZ Records at one place one above the other. This sequential collection of records can be called an Array. An array is a sequential collection of elements of the same data type. In our example above, XYZ Records is the data type and all the records you collected have the same publishers. All the elements in an array are addressed by a common name.

There are two types of array:

- Single Dimensional Array
- Multi Dimensional Array

Let us start by understanding what are single dimensional arrays,

Single Dimensional Array

Syntax for declaring a *Single Dimensional Array:*

We have a data type that can be any of the basic data types like int, float or double. Array Name is the name of the array and we declare the size of the array. In our above example, the array will be,

```
1
    XYZ Record recordArray[100];
```

Let's consider another example:

```
1
        int test[20];
```
The array test will hold the elements of type int and will have a size 20.

Array Size

Array size is given at the time of declaration of the array. Once the size of the array is given it cannot be changed. The compiler then allocates that much memory space to the array.

Consider the Example

```
1       int test[20];
```
In the example above, we have an array test, of type int. We have given the array size to be 20. This means that 20 consecutive memory locations will be left free for the array in the memory.

Array Index And Initialization

A number associated with each position in an array and this number is called the *array index*. Its starts from 0 and to the last element, that is the size of the array minus one. The minus one is there because we start counting from zero and not one. Array indices always begin from zero.

Consider this example, this is the age array.

Array Value	12	41	03	13	07
Array Indices	0	1	2	3	4

Here the array contains the values 12,41,3,13,7 and the indices are 0,1,2,3,4,5. If we want to represent an element at index 4 it is represented as age[4] and the value 7 will be displayed.

By default, the array contains all zero values. Array initialization is done at the time of declaration. This can also be carried out later if the user enters the array value as and when needed.

Let us see how initialization works during declaration,

Initialization During Declaration

An array can be initialized during declaration. This is done by specifying the array elements at the time of declaration. Here the array size is also fixed and it is decided by us.

Consider the code,

```
1	#include<iostream>
2	using namespace std;
3	int main()
4	{
5	int arr[] = { 10, 20, 30, 40 } ;
6	return 0;
7	}
```

Explanation

In the above example, we create an array of type int and with the name arr. We directly specify the array elements. The size of the array is decided by counting the number of elements in our array. In this case, the size is 4.

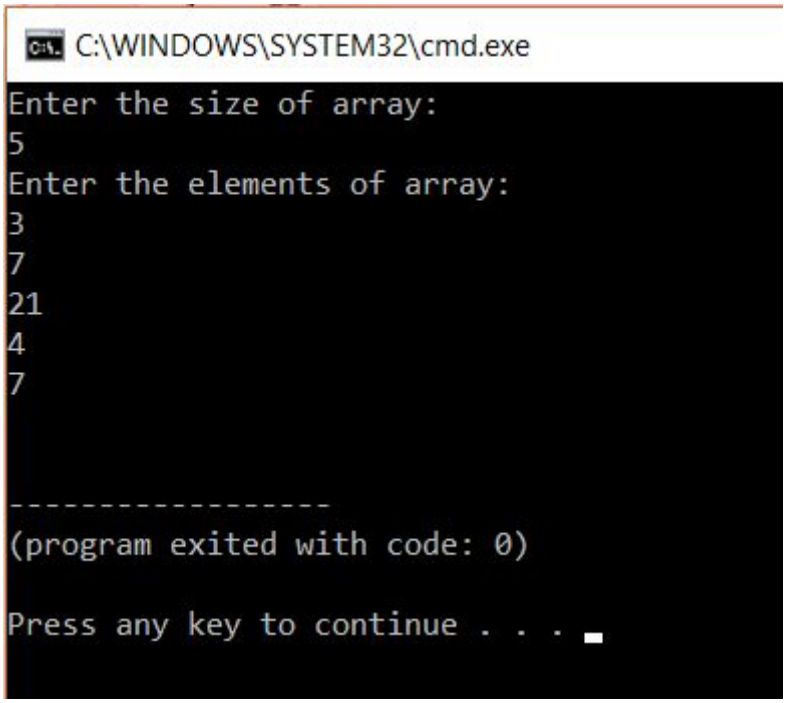
Initialization By A User

In this method, we let the user decide the size of the array. In this case, we need a variable to hold the size of the array and a for loop to accept the elements of the array. We assign a random size at the time of declaration and use only as needed. The size at the start is usually at the higher side. We have a variable i to control the for loop.

Consider the example,

```
1	#include<iostream>
2	using namespace std;
3	int main()
4	{
5	int arr[50],n,i ;
6	cout<<"Enter the size of array:"<<endl;
7	cin>>n;
8	cout<<"Enter the elements of array:"<<endl;
9	for(i=0;i<n;i++)
10	{
11	cin>>arr[i];
12	}
13	return 0;
14	}
```

Output



Explanation

In the above program, we declare an array of size 50. We then ask the user to enter the number of elements he wishes to enter in this array. We then accept the array elements entered by the user.

Displaying the Array

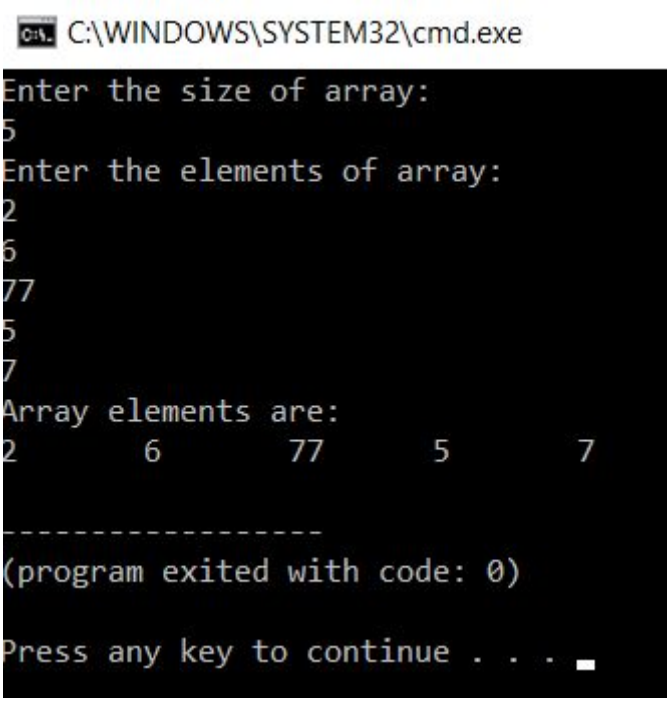
Displaying the array also requires the for-loop. We traverse to the entire array and display the elements of the array.

Here is an example,

```
1    #include<iostream>
2    using namespace std;
3    int main()
4    {
5    int arr[50],n,i ;
6    cout<<"Enter the size of array:"<<endl;
7    cin>>n;
8    cout<<"Enter the elements of array:"<<endl;
9    for(i=0;i<n;i++)
10   {
11   cin>>arr[i];
12   }
13   cout<<"Array elements are:"<<endl;
14   for(i=0;i<n;i++)
15   {
16   cout<<arr[i]<<"t";
17   }
18   return 0;
```

```
19      }
```

Output



Explanation

In the above program, we declare an array of size 50. We then ask the user to enter the number of elements he wishes to enter in this array. We then accept the array elements entered by the user. We then use a for loop again to display the array elements.

#Accessing Array Elements in C++

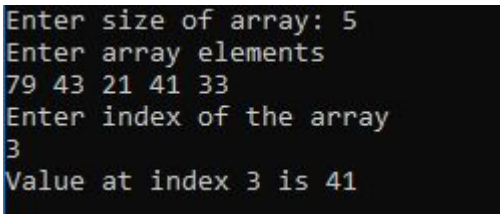
To access any specific value: For this case, we have to mention the array index, which means which locations value we want to show.

For example, see the below code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n, i;
   cout<<"Enter size of array: ";
   cin>>n;
   int array[n]; //declaring array with size
   cout<<"Enter array elements\n";
   for(i=0;i<n;i++)
   {
      cin>>array[i];
```

```
    }

    int id;
    cout<<"Enter index of the array\n";
    cin>>id;

    cout<<"Value at index "<<id<<" is "<<array[id];
    return 0;
}
```

See the following output.



To access all the elements: The best way to show all the elements in an array is by using the loop. The below program shows how to access all the elements of an array using a loop:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n, i;
    cout<<"Enter size of array: ";
    cin>>n;
    int array[n]; //declaring array with size
    cout<<"Enter array elements\n";
    for(i=0;i<n;i++)
    {
        cin>>array[i];
    }
    cout<<"All entered array values are: \n";
    for(i=0;i<n;i++)
    {
        cout<<array[i]<<" ";
    }
    return 0;
}
```

See the following Output.

```
Enter size of array: 5
Enter array elements
24 35 33 17 2
All entered array values are:
24 35 33 17 2
```

#No index bound checking in Array

There is no index of bound checking in Array in C++. C++ throws a garbage value or unexpected value when the index is out of bounds or out of range.

The following program shows the index bound checking.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n, i;
   cout<<"Enter size of array: ";
   cin>>n;
   int array[n]; //declaring array with size
   cout<<"Enter array elements\n";
   for(i=0;i<n;i++)
   {
      cin>>array[i];
   }

   int id;
   cout<<"Enter index of the array\n";
   cin>>id; //enter out of bound value

   cout<<"Value at index "<<id<<" is "<<array[id];
   return 0;
}
```

See the following output.

```
Enter size of array: 4
Enter array elements
32 42 11 3
Enter index of the array
42
Value at index 42 is 0
```

#Multi-Dimensional Array

We will learn how to create a 2D array and how to access those elements.

Syntax

```
DataType Array_name[Size 1][Size 2]
```

See the following code.

```
int arr[2][3]
```

Here we have declared a 2D array with 2 rows and 3 columns.

Storing and accessing elements of 2D array

The 2D array looks something like this diagram.

Column-> Row	0	1	2
0	15	4	32
1	74	23	2

In the above diagram, the index of 15 is arr[0][0] and the index of 2 is arr[1][2] means 2nd row and 3rd column.

Note: Row and column index starts from 0.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int x,y;
   cout<<"Enter row number and column number :";
   cin>>x>>y;
   int arr[x][y],row,col;
   cout<<"Enter value of array\n";

   for(row=0; row<x; row++)
   {
      for(col=0; col<y; col++)
      {
         cin>>arr[row][col];
      }
   }
```

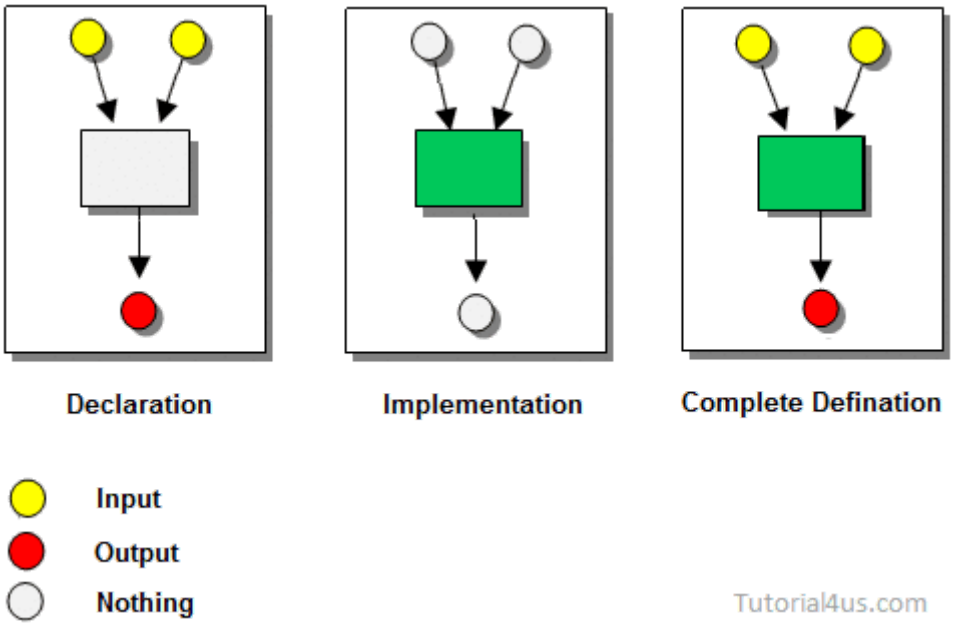
```
   cout<<"Value of array are:\n";
      for(row=0; row<x; row++)
   {
      for(col=0; col<y; col++)
      {
         cout<<arr[row][col]<<" ";
      }
      cout<<"\n";
   }
   return 0;
}
```

See the following output.

```
Enter row number and column number :3 4
Enter value of array
12 13 16 47
22 23 24 25
30 31 32 33
Value of array are:
12 13 16 47
22 23 24 25
30 31 32 33
```

Function in C++

A function is a group of statements that together perform a specific task. Every C++ program has at least one function, which is main().



Why use function ?

Function are used for divide a large code into module, due to this we can easily debug and maintain the code. For example if we write a calculator programs at that time we can write every logic in a separate function (For addition sum(), for subtraction sub()). Any function can be called many times.

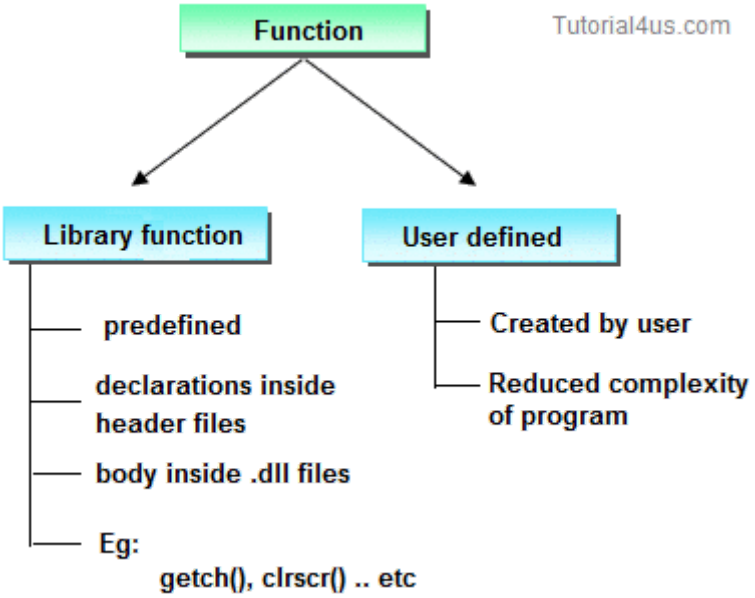
Advantage of Function

- Code Re-usability
- Develop an application in module format.
- Easily to debug the program.
- **Code optimization:** No need to write lot of code.

Type of Function

There are two type of function in C++ Language. They are;

- Library function or pre-define function.
- User defined function.



Library function

Library functions are those which are predefined in C++ compiler. The implementation part of pre-defined functions is available in library files that are .lib/.obj files. **.lib or .obj** files are contained pre-compiled code. printf(), scanf(), clrscr(), pow() etc. are pre-defined functions.

Limitations of Library function

- All predefined function are contained limited task only that is for what purpose function is designed for same purpose it should be used.
- As a programmer we do not having any controls on predefined function implementation part is there in machine readable format.
- In implementation whenever a predefined function is not supporting user requirement then go for user defined function.

User defined function

These functions are created by programmer according to their requirement for example suppose you want to create a function for add two number then you create a function with name sum() this type of function is called user defined function.

Defining a function.

Defining of function is nothing but give body of function that means write logic inside function body.

Syntax

```
return_type  function_name(parameter)
{
function body;
}
```

- **Return type:** A function may return a value. The return_type is the data type of the value the function returns.Return type parameters and returns statement are optional.
- **Function name:** Function name is the name of function it is decided by programmer or you.
- **Parameters:** This is a value which is pass in function at the time of calling of function A parameter is like a placeholder. It is optional.
- **Function body:** Function body is the collection of statements.

Function Declarations

A function declaration is the process of telling the compiler about a function name. The actual body of the function can be defined separately.

Syntax

```
return_type  function_name(parameter);
```

Note: At the time of function declaration function must be terminated with ';'.

calling a function.

When we call any function control goes to function body and execute entire code. For call any function just write name of function and if any parameter is required then pass parameter.

Syntax

```
function_name();
   or
variable=function_name(argument);
```

Example of function which accept two integers as an argument and return its sum. Call this function from main( ) and print the results in main( ).

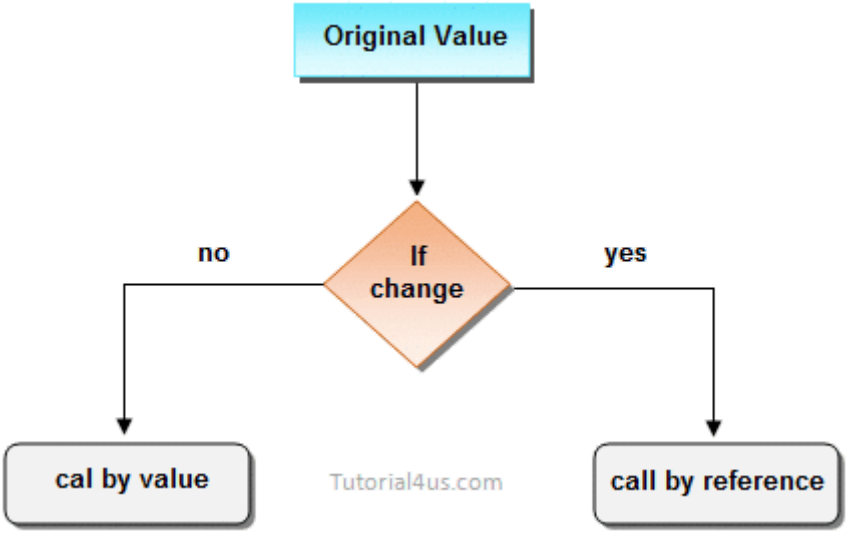
```
#include<iostream>
using namespace std;

int sum(int, int);

int main()
{
      int x,y,s;
      cout<<"Enter first number : ";
      cin>>x;
      cout<<"Enter second number : ";
      cin>>y;
      s=sum(x,y);
      cout<<"The sum is : "<<s;


      return 0;
}

int sum(int a, int b)
{

      int total;
      total=a+b;
      return total;
}
```

Note: At the time of function calling function must be terminated with ';'.

Function Arguments in C++

If a function take any arguments, it must declare variables that accept the values as a arguments. These variables are called the formal parameters of the function. There are two ways to pass value or data to function in C++ language which is given below;

- call by value
- call by reference



Call by value

In call by value, **original value can not be changed** or modified. In call by value, when you passed value to the function it is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only but it not change the value of variable inside the caller function such as main().

Program Call by value in C++

```
#include<iostream.h>
#include<conio.h>

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

void main()
{
 int a=100, b=200;
 clrscr();
 swap(a, b);  // passing value to function
 cout<<"Value of a"<<a;
 cout<<"Value of b"<<b;
 getch();

  }
```

Output

```
Value of a: 200
Value of b: 100
```

Call by reference

In call by reference, **original value is changed** or modified because we pass reference (address). Here, address of the value is passed in the function, so actual and formal arguments shares the same address space. Hence, any value changed inside the function, is reflected inside as well as outside the function.

Example Call by Reference in C++

```
#include<iostream.h>
#include<conio.h>

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

void main()
{
 int a=100, b=200;
 clrscr();
```

```
swap(&a, &b);  // passing value to function
cout<<"Value of a"<<a;
cout<<"Value of b"<<b;
getch();

  }
```

Output

Value of a: 200
Value of b: 100

Difference Between Call by Value and Call by Reference.

call by Value	call by Reference
This method copy original value into function as a arguments.	This method copy address of arguments into function as a arguments.
Changes made to the parameter inside the function have no effect on the argument.	Changes made to the parameter affect the argument. Because address is used to access the actual argument.
Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Note: By default, C++ uses call by value to pass arguments.

A function to calculate the factorial value of any integer as an argument. Call this function from main( ) and print the results in main( ).

Source Code

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

int factorial(int);

int main()
{
      int x,f;
      cout<<"Enter number : ";
      cin>>x;
      f=factorial(x);
      cout<<"The factorial is :"<<f;

      return 0;
}

int factorial(int a)
{
```

```
	int fact=1;

	while(a>=1)
	{
		fact=fact*a;
		a--;
	}

	return fact;
}
```

A function that receives two numbers as an argument and display all prime numbers between these two numbers. Call this function from main( ).

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

void showprime(int, int);

int main()
{

	int x,y;
	cout<<"Enter first  number : ";
	cin>>x;

	cout<<"Enter second  number : ";
	cin>>y;
	showprime(x,y);


	return 0;
}

void showprime(int a, int b)
{
	bool flag;

	for(int i=a+1;i<=b;i++)
	{

		flag=false;
		for(int j=2;j<i;j++)
		{

			if(i%j==0)
			{
				flag=true;
				break;
			}
		}

		if(flag==false && i>1)
```

```
                cout<<i<<endl;
        }
}
```

Raising a number to a power p is the same as multiplying n by itself p times. We can use a function to call a power that takes two arguments, a double value for n and an int value for p, and return the result as double value. Use default argument of 2 for p, so that if this argument is omitted the number will be squared. Write the main function that gets value from the user to test power function.

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

double power(double,int=2);

int main()
{
      int p;
      double n,r;
      cout << "Enter number : ";
      cin >> n;
      cout << "Enter exponent : ";
      cin >> p;
      r = power(n,p);
      cout << "Result is " << r;
      r = power(n);
      cout << "\nResult without passing exponent is " << r;

      return 0;
}

double power(double a, int b)
{
      double x = 1;
      for(int i = 1; i <= b; i++)
            x = x * a;
      return x;
}
```

A program which accept a letter and display it in uppercase letter.

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

int main()
{
      char ch;
      cout<<"Enter any character :";
      ch=getchar();
      ch=toupper(ch);
      cout<<ch;

        return 0;
```

```
}
```

Reference

The reference is an alias for a variable in a program. A variable and its reference can be used interchangeably in a program as both refer to the same memory location. Hence, changes made to any of them are reflected in the other.

Example:

```
#include<iostream>

using namespace std;

int main()

{

    int x = 10;

    int& ref = x;

    ref = 20;

    cout << "x = " << x << endl ;

    x = 30;

    cout << "ref = " << ref << endl ;

    return 0;

}
//Output:

x = 20

ref = 30
```

Further, you can find detailed documentation on Reference in the following Article.

Pointer

The pointer is a variable that can store the memory address of another variable. Pointers allow using the memory dynamically. That is, with the help of pointers, memory can be allocated or de-allocated to the variables at run-time, thus, making a program more efficient.

Example:

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

using namespace std;

void Edureka()

{

    int var = 20;

    int *ptr;

    ptr = &var;

    cout << "Value at ptr = " << ptr << "n";

    cout << "Value at var = " << var << "n";

    cout << "Value at *ptr = " << *ptr << "n";

}

int main()

{

    Edureka();

}
```

//Output:

Value at ptr = 0x6ffdd4

Value at var = 20

Value at *ptr = 20

Further, you can find detailed documentation on Pointers in the following Article.

Strings

The term string means an ordered sequence of characters. A sequence of characters can be represented using an object of a class in C++.

Example:

```
1
2       #include <iostream>
3
4
5
6       using namespace std;
7
8       int main () {
9
10
11        char ch[12] = {'H', 'e', 'l', 'l', 'o', ' ', 'w','o','r','l','d'};
12
13        string st = "Welcome to Edureka";

          std::string std_st = "Happy Learning";

          cout << ch << endl;

          cout << st << endl;

          cout << std_st << endl;

          return 0;

        }
//Output:

Hello world

Welcome to Edureka

Happy Learning
```