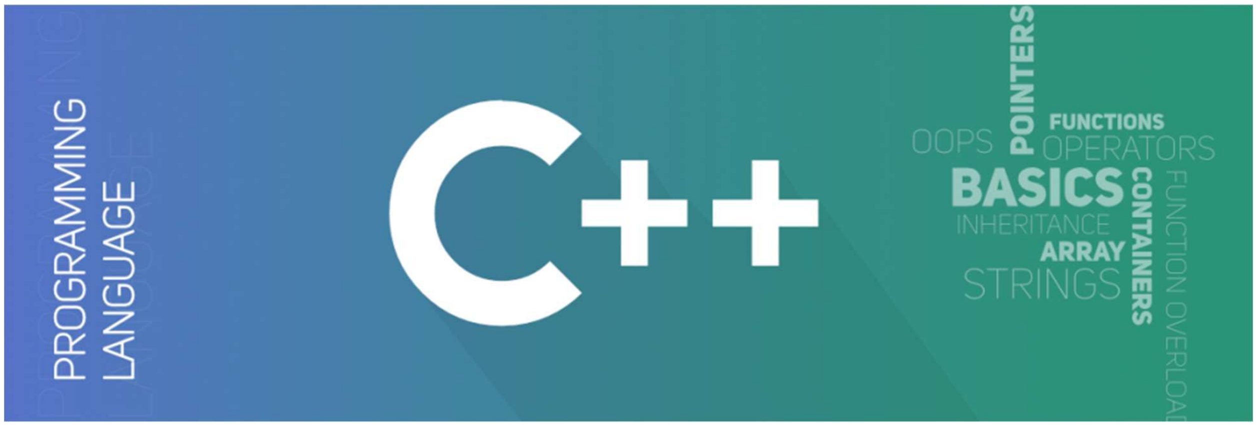
INFORMATION TCHNOLOGY II

C++ PROGRAMMING LANGUAGE

Mathematics 1st Stage



Lecturer: Azhi A. Salih

C++ PROGRAMMING LANGUAGE

In simple terms, C++ is a sophisticated, efficient and a general-purpose programming language based on C. It was developed by **Bjarne Stroustrup** in 1979.

Many of today's operating systems, system drivers, browsers and games use C++ as their core language. This makes C++ one of the most popular languages today.

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

int main()
{
    cout<<"Hello World!";     //print message;
    return 0;
}
```

Comments

A **comment** is a programmer-readable note that is inserted directly into the source code of the program. Comments are ignored by the compiler and are for the programmer's use only.

In C++ there are two different styles of comments, both of which serve the same purpose: to help programmers document the code in some way.

Single-line comments

The // symbol begins a C++ single-line comment, which tells the compiler to ignore everything from the // symbol to the end of the line. For example:

```
std::cout << "Hello world!"; // Everything from here to the end of the line is ignored
```

Multi-line comments

The /* and */ pair of symbols denotes a C-style multi-line comment. Everything in between the symbols is ignored.

```
/* This is a multi-line comment.
   This line will be ignored.
   So will this one. */
```

Variables

Variable Declaration and initialization

Variable Definition

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows

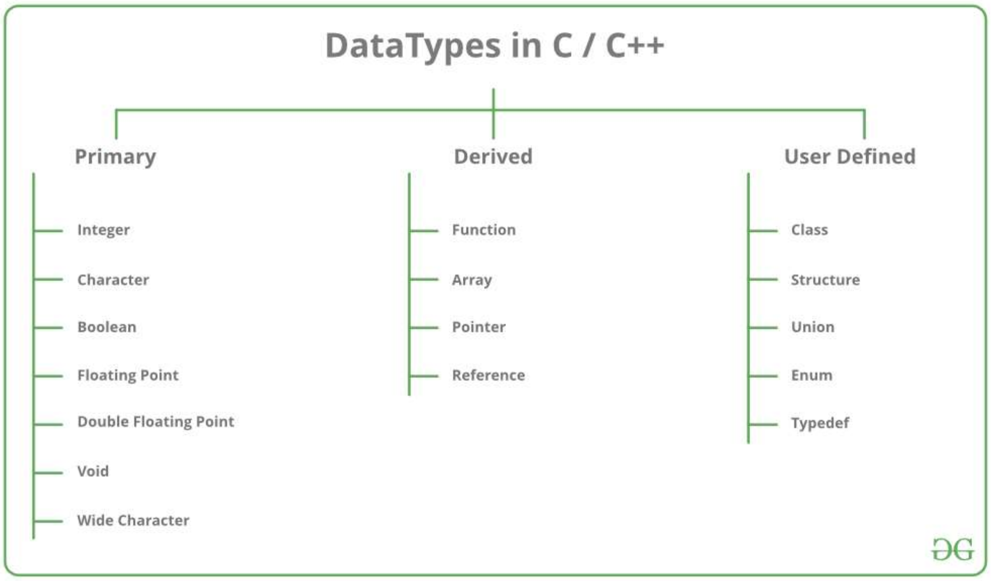
```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

Variable initialization

Example:

```
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

C++ Data Types



Data types in C++ is mainly divided into three types:

1. **Primitive Data Types:** These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
   - Integer
   - Character
   - Boolean
   - Floating Point
   - Double Floating Point
   - Valueless or Void
   - Wide Character
2. **Derived Data Types:** The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
   - Function
   - Array
   - Pointer
   - Reference
3. **Abstract or User-Defined Data Types**: These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:
   - Class
   - Structure
   - Union
   - Enumeration
   - Typedef defined DataType

Primitive Data Types

- **Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
- **Boolean**: Boolean data type is used for storing boolean or logical values. A boolean variable can store either *true* or *false*. Keyword used for boolean data type is **bool**.
- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.
- **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically requires 8 byte of memory space.
- **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
- **Wide Character**: Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by **wchar_t**. It is generally 2 or 4 bytes long.

Variable assignment and initialization

```
int x; // define an integer variable named x
int y, z; // define two integer variables, named y and z
int width; // define an integer variable named width
width = 5; // copy assignment of value 5 into variable width
int a = 5, b = 6; // copy initialization

int a, b = 5; // wrong (a is not initialized!)
```

Introduction to iostream: cout, cin, and endl:

1-Output statements

cout << variable-name;

//Meaning: print the value of variable <variable-name> to the user

cout << “any message “;

//Meaning: print the message within quotes to the user

cout << endl;

//Meaning: print a new line

Example:

```
        cout << a;

        cout << b << c;

        cout << “This is my character: “ << my-character << “ he he he”

                << endl;
```

```
#include <iostream>

int main()
{
   int x=5; // define integer variable x, initialized with value 5
   cout << x; // print value of x (5) to console
   return 0;
}
```

This produces the result:

```
5
```

2-endl

endl prints a newline character to the console (causing the cursor to go to the start of the next line)

For example:

```
#include <iostream>

int main()
{
   cout << "Hi!" << endl; endl will cause the cursor to move to the next line of the console
  cout << "My name is Alex." <<endl;

   return 0;
}
```

Result:

```
Hi!
My name is Alex.
```

3-Input statements (cin) :

`cin` (which stands for “character input”) reads input from keyboard using the **extraction operator (>>)**. The input must be stored in a variable to be used.

cin >> variable-name;

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;

cin >> b >> c;

cin >> x;

cin >> my-character;
```

```
#include <iostream>
int main()
{
   cout << "Enter a number: "; // ask user for a number
   int x ; // define variable x to hold user input (and zero-initialize it)
   cin >> x; // get number from keyboard and store it in variable x
   cout << "You entered " << x << '\n';
   return 0;
}
```

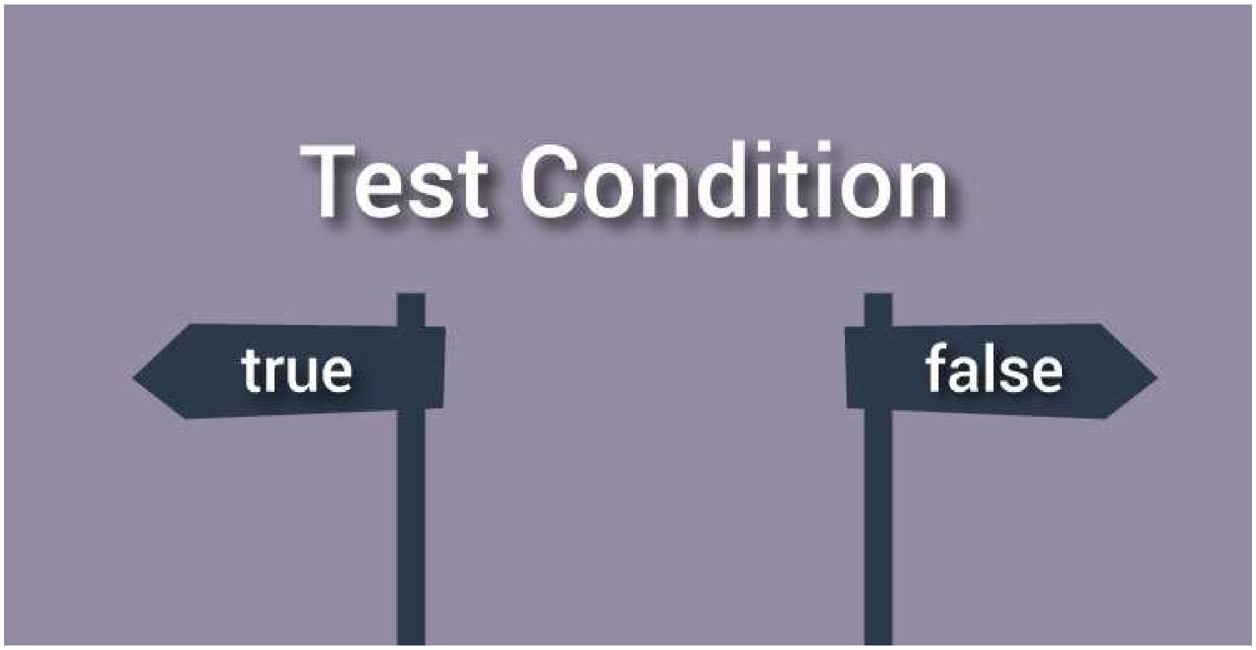
Example: Print Number Entered by User

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

int main()
{
    int number;

    cout << "Enter an integer: ";
    cin >> number;

    cout << "You entered " << number;
    return 0;
}
```

Output

```
Enter an integer: 23
You entered 23
```

C++ if, if...else and Nested if...else

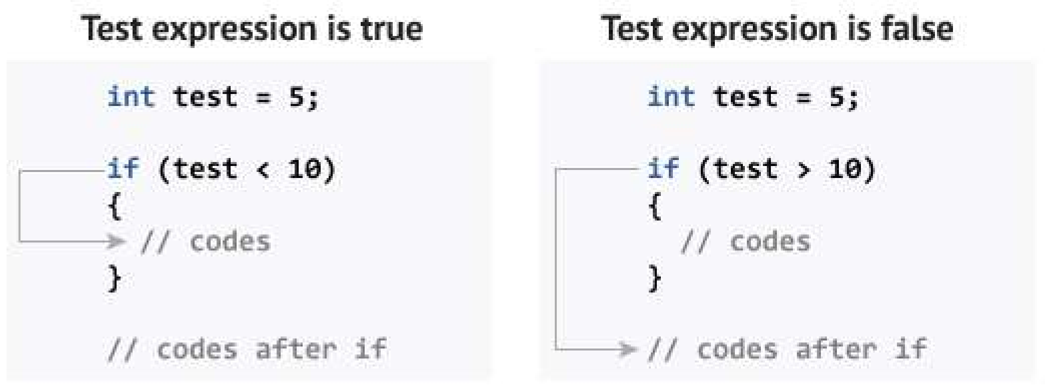


C++ if Statement

```
if (testExpression)

{

   // statements

}
```

The `if` statement evaluates the test expression inside parenthesis.
If test expression is evaluated to true, statements inside the body of `if` is executed.
If test expression is evaluated to false, statements inside the body of `if` is skipped.

How if statement works?



```
int test = 5;

if (test < 10)
{
  // codes
}

// codes after if
```

```
int test = 5;

if (test > 10)
{
  // codes
}

// codes after if
```

Flowchart of if Statement

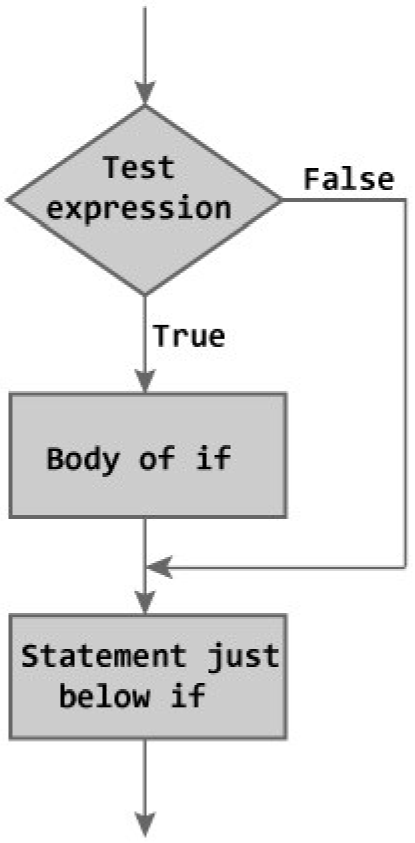


Figure: Flowchart of if Statement

Above figure describes the working of an if statement.

Example 1: C++ if Statement

```cpp
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;

    // checks if the number is positive
    if ( number > 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }

    cout << "This statement is always executed.";
    return 0;

}
```

Output 1

```
Enter an integer: 5
You entered a positive number: 5
This statement is always executed.
```

Output 2

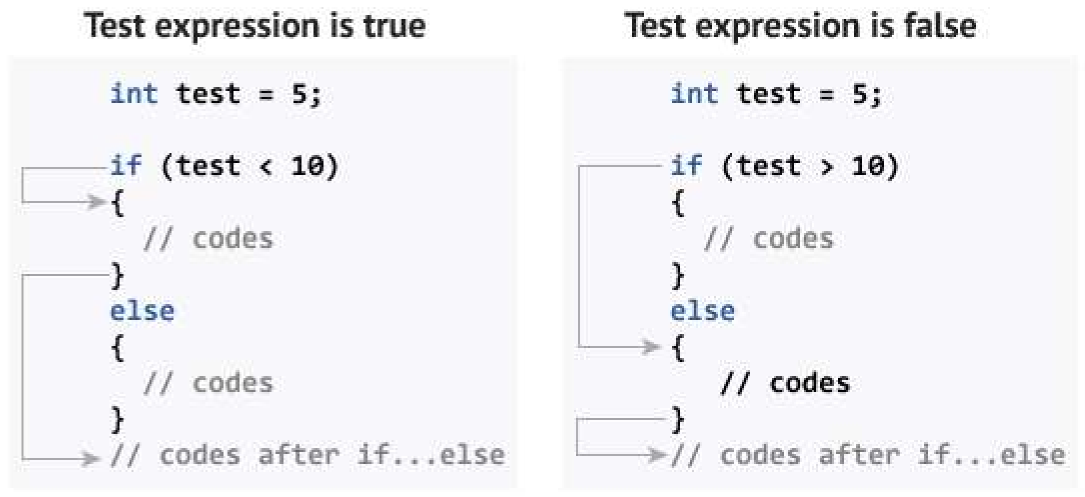
```
Enter a number: -5
This statement is always executed.
```

C++ if...else

The `if else` executes the codes inside the body of `if` statement if the test expression is true and skips the codes inside the body of `else`.

If the test expression is false, it executes the codes inside the body of `else` statement and skips the codes inside the body of `if`.

How if...else statement works?



Test expression is true

```
int test = 5;

if (test < 10)
{
  // codes
}
else
{
  // codes
}
// codes after if...else
```

Test expression is false

```
int test = 5;

if (test > 10)
{
  // codes
}
else
{
  // codes
}
// codes after if...else
```

Flowchart of if...else

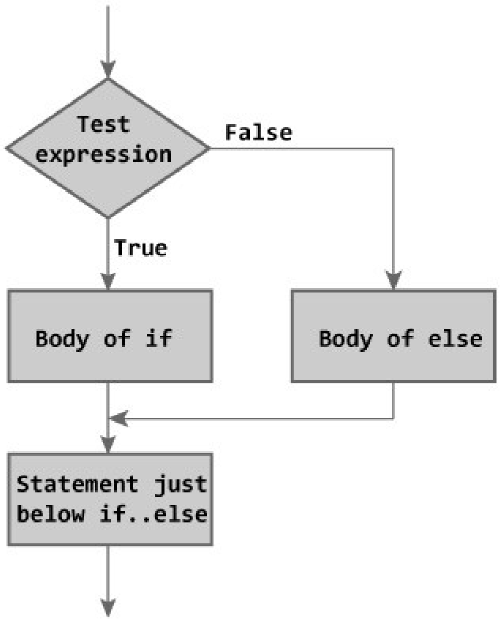


Figure: Flowchart of if...else Statement

Example 2: C++ if...else Statement

```cpp
// Program to check whether an integer is positive or negative
// This program considers 0 as positive number

#include <iostream>
using namespace std;

int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;

    if ( number >= 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }

    else
    {
        cout << "You entered a negative integer: " << number << endl;
    }

    cout << "This line is always printed.";
    return 0;
}
```

Output

```
Enter an integer: -4
You entered a negative integer: -4.
This line is always printed.
```

C++ Program to Check Whether Number is Even or Odd

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

int main()
{
    int n;

    cout << "Enter an integer: ";
    cin >> n;

    if ( n % 2 == 0)
```

```cpp
        cout << n << " is even.";
    else
        cout << n << " is odd.";

    return 0;
}
```

Output

```
Enter an integer: 23
23 is odd.
```

C++ Program to Find Largest Number Among Three Numbers

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

int main()
{
    float n1, n2, n3;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    if(n1 >= n2 && n1 >= n3)
    {
        cout << "Largest number: " << n1;
    }

    if(n2 >= n1 && n2 >= n3)
    {
        cout << "Largest number: " << n2;
    }

    if(n3 >= n1 && n3 >= n2) {
        cout << "Largest number: " << n3;
    }

    return 0;
}
```

Output

```
Enter three numbers: 2.3
8.3
-4.2
Largest number: 8.3
```

Example 2: Find Largest Number Using if...else Statement

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

int main()
{
    float n1, n2, n3;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    if((n1 >= n2) && (n1 >= n3))
        cout << "Largest number: " << n1;
    else if ((n2 >= n1) && (n2 >= n3))
        cout << "Largest number: " << n2;
    else
        cout << "Largest number: " << n3;

    return 0;
}
```

Output

```
Enter three numbers: 2.3
8.3
-4.2
Largest number: 8.3
```

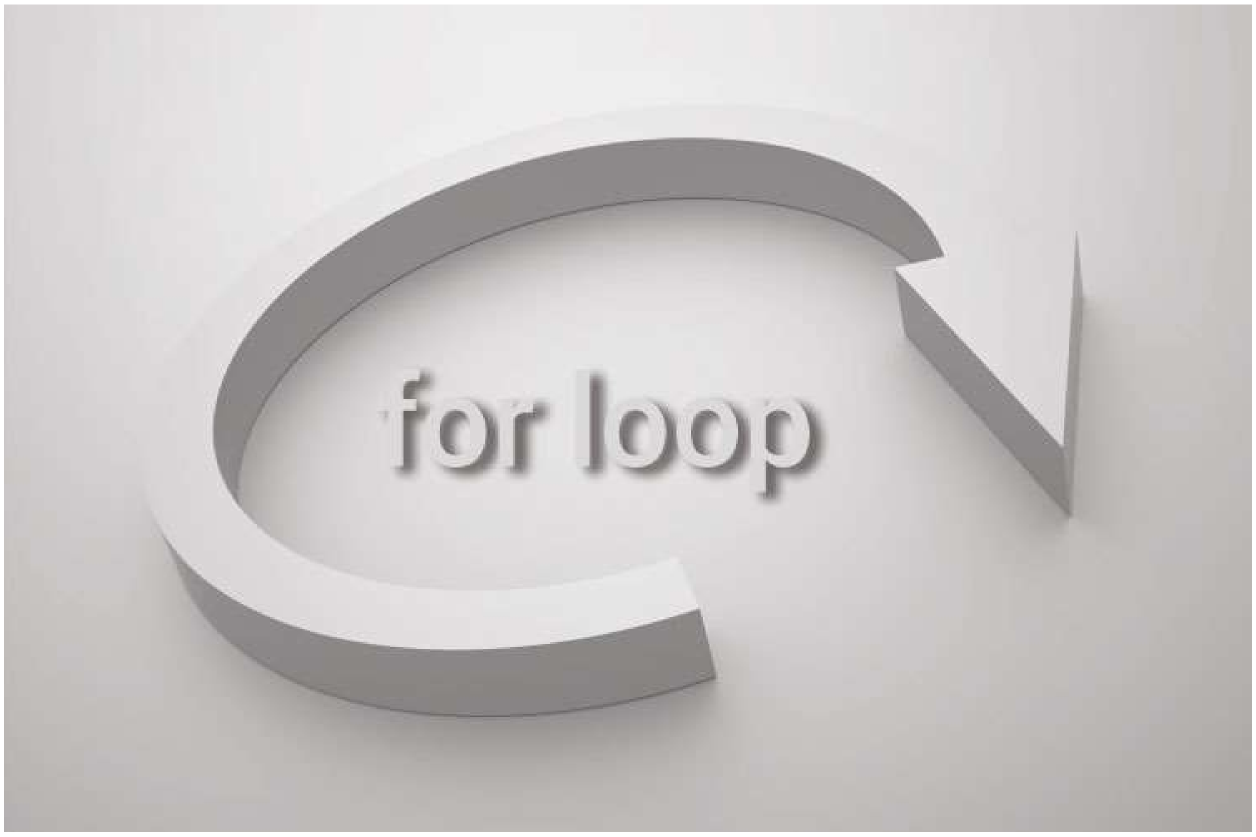
Example 3: Find Largest Number Using Nested if...else statement

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

int main()
{
    float n1, n2, n3;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    if (n1 >= n2)
    {
        if (n1 >= n3)
            cout << "Largest number: " << n1;
        else
            cout << "Largest number: " << n3;
    }
    else
    {
        if (n2 >= n3)
            cout << "Largest number: " << n2;
        else
            cout << "Largest number: " << n3;
    }

    return 0;
}
```

Output

```
Enter three numbers: 2.3
8.3
-4.2
Largest number: 8.3
```

C++ for Loop



Loops are used in programming to repeat a specific block until some end condition is met. There are three type of loops in C++ programming:

1. for loop
2. while loop
3. do...while loop

C++ for Loop Syntax

```
for(initializationStatement; testExpression; updateStatement) {

    // codes

}
```

Flowchart of for Loop in C++

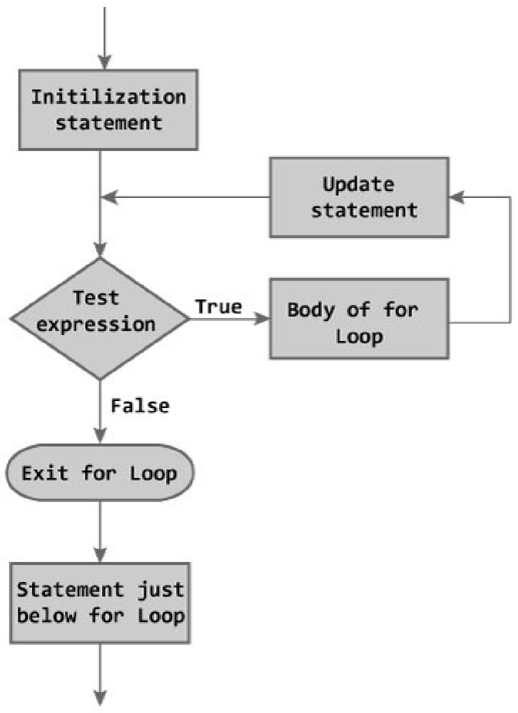


Figure: Flowchart of for Loop

Example 1: C++ for Loop

```cpp
// C++ Program to find factorial of a number
// Factorial on n = 1*2*3*...*n

#include <iostream>
using namespace std;

int main()
{
    int i, n, factorial = 1;

    cout << "Enter a positive integer: ";
    cin >> n;

    for (i = 1; i <= n; ++i) {
        factorial *= i;   // factorial = factorial * i;
    }

    cout<< "Factorial of "<<n<<" = "<<factorial;
    return 0;
}
```

Output

```
Enter a positive integer: 5
Factorial of 5 = 120
```

C++ Program to Calculate Sum of Natural Numbers

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

int main()
{
    int n, sum = 0;

    cout << "Enter a positive integer: ";
    cin >> n;

    for (int i = 1; i <= n; ++i) {
        sum += i;
    }

    cout << "Sum = " << sum;
    return 0;
}
```

Output

```
Enter a positive integer: 50
Sum = 1275
```

C++ Program to Find Factorial

For any positive number `n`, it's factorial is given by:

```
factorial = 1*2*3...*n
```

Factorial of negative number cannot be found and factorial of 0 is 1.

Example: Find Factorial of a given number

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

int main()
{
    unsigned int n;
    unsigned long long factorial = 1;

    cout << "Enter a positive integer: ";
    cin >> n;

    for(int i = 1; i <=n; ++i)
    {
        factorial *= i;
    }

    cout << "Factorial of " << n << " = " << factorial;
    return 0;
}
```

Output

```
Enter a positive integer: 12
Factorial of 12 = 479001600
```

C++ Program to Generate Multiplication Table

```cpp
nclude <iostream>
using namespace std;

int main()
{
    int n;

    cout << "Enter a positive integer: ";
    cin >> n;

    for (int i = 1; i <= 10; ++i) {
        cout << n << " * " << i << " = " << n * i << endl;
    }

    return 0;
}
```

Output

```
Enter an integer: 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

C++ while Loop

The syntax of a while loop is:

```
while (testExpression)

{

     // codes

}
```

where, `testExpression` is checked on each entry of the while loop.

How while loop works?

- The while loop evaluates the test expression.

- If the test expression is true, codes inside the body of while loop is evaluated.
- Then, the test expression is evaluated again. This process goes on until the test expression is false.
- When the test expression is false, while loop is terminated.

Flowchart of while Loop

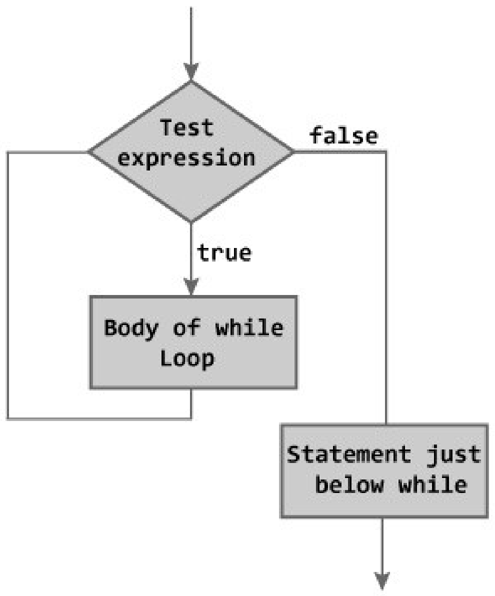


Figure: Flowchart of while Loop

Example 1: C++ while Loop

```
// C++ Program to compute factorial of a number
// Factorial of n = 1*2*3...*n

#include <iostream>
using namespace std;

int main()
{
    int number, i = 1, factorial = 1;

    cout << "Enter a positive integer: ";
    cin >> number;

    while ( i <= number) {
        factorial *= i;      //factorial = factorial * i;
        ++i;
    }

    cout<<"Factorial of "<< number <<" = "<< factorial;
    return 0;
}
```

Output

```
Enter a positive integer: 4
Factorial of 4 = 24
```

C++ do...while Loop

The do...while loop is a variant of the while loop with one important difference. The body of do...while loop is executed once before the test expression is checked.

The syntax of do..while loop is:

```
do {

   // codes;

}

while (testExpression);
```

Flowchart of do...while Loop

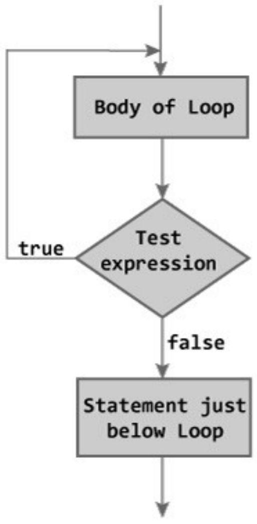


Figure: Flowchart of do...while Loop

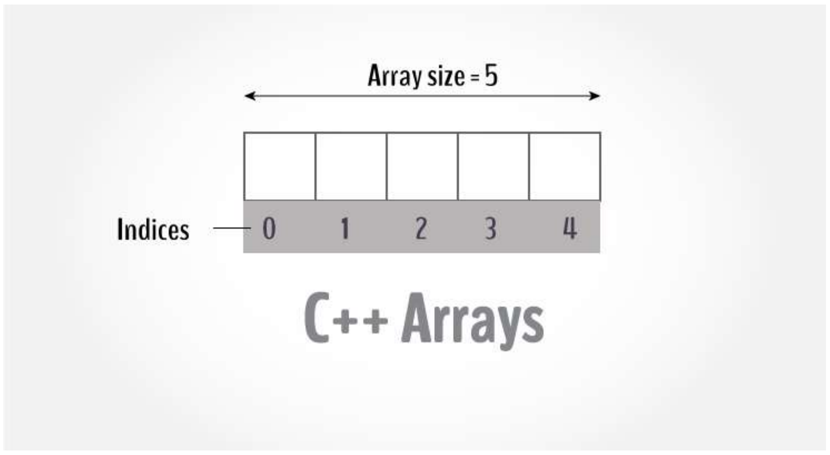
Example 2: C++ do...while Loop

```cpp
// C++ program to add numbers until user enters 0

#include <iostream>
using namespace std;

int main()
{
    float number, sum = 0.0;

    do {
        cout<<"Enter a number: ";
        cin>>number;
        sum += number;
    }
    while(number != 0.0);

    cout<<"Total sum = "<<sum;

    return 0;
}
```

Output

```
Enter a number: 2
Enter a number: 3
Enter a number: 4
Enter a number: -4
Enter a number: 2
Enter a number: 4.4
Enter a number: 2
Enter a number: 0
```

Arrays in C++

- C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- 

- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

  Type arrayName [arraySize];

  Int x [10] ;

  Double y[5] ;

Array declaration and Initialization

- **int arr[10];**   // Array declaration by specifying size
- **int arr[ ] = {10, 20, 30, 40}** // Array declaration by initializing elements , Compiler creates an array of size 4.

above is same as "int arr[4] = {10, 20, 30, 40}"

- **int arr[6] = {10, 20, 30, 40}** // Array declaration by specifying size and initializing

elements , Compiler creates an array of size 6, initializes first 4   elements as specified by user and rest two elements as 0.

Accessing Array Elements:

Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1

```
 int main()
{
  int arr[5];
  arr[0] = 5;
  arr[2] = -10;
  arr[1] = 2
  arr[3] = arr[0];

  cout << arr[0], arr[1], arr[2], arr[3]);

  return 0;
}
```

```
                                        Output:   5  2  -10  5
```

An example of calculating a sum of n array's elements.

```
#include <iostream>
using namespace std;
int main ()  {
 int n[ 10 ]={4,6,8,9,8};
int sum=0;
 for ( int i = 0; i < 5; i++ ) {
Sum=sum+n[ i ]
}
cout << “The Sum = " << sum<< endl;
return 0;
}
```

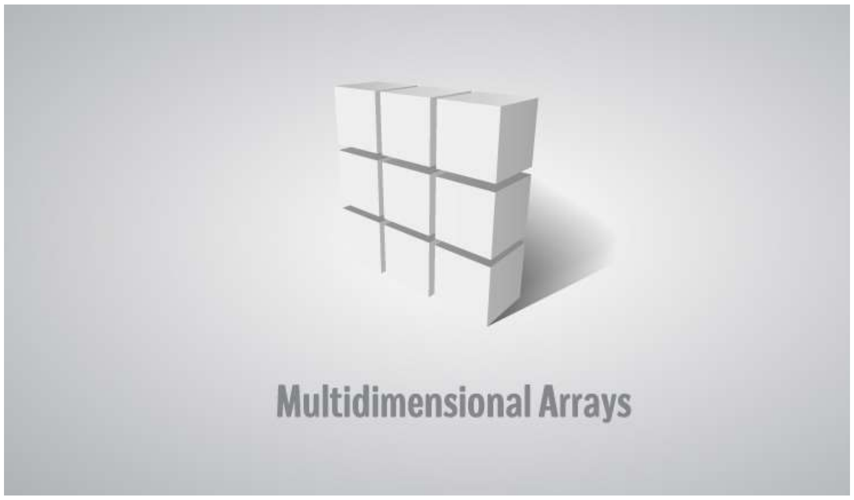
Write a c++ code to read 20 numbers from the user (use while loop)

```
#include <iostream.h>

void main() {

int i, a[100], n;

i=0; n=20;

while (i<n) {

        cout << “Input element “ << i << “: ”;

        cin >> a[i];

        i = i+1;

}

For (int i=0;i<20;i++){

Cout<<a[i] <<endl;

}

Return 0;

}
```

Questions:

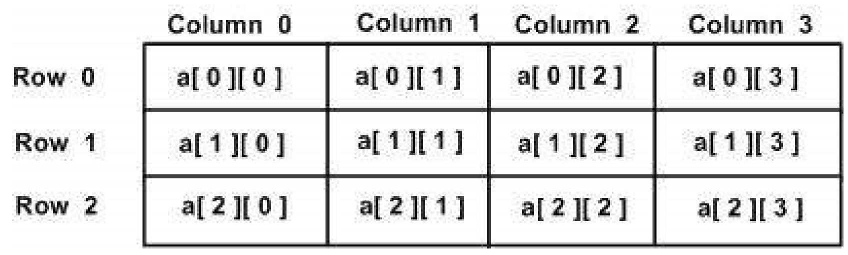
1. Write a c++ code to get the following output: 1,4 ,9 ,16 ,25. (use array).
2. Write a c++ code to read 6 numbers from the keyboard then find the maximum number among them.

Two-Dimensional Arrays



- A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows
  type arrayName [x][y];

int arr[4][3];



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initializing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
 int a[3][4] = {{0, 1, 2, 3},

              {4, 5, 6, 7},

              {8, 9, 10, 11}};
```

Or

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

```
#include <iostream>

using namespace std;
```

```
 int main () {
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
for ( int i = 0; i < 5; i++ )
   for ( int j = 0; j < 2; j++ ) {
cout << a[i][j]<< endl;
 }
 return 0;
}
```

Example 1: Two Dimensional Array

C++ Program to display all elements of an initialised two dimensional array.

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

int main()
{
    int test[3][2] =
    {
        {2, -5},
        {4, 0},
        {9, 1}
    };

    // Accessing two dimensional array using
    // nested for loops
    for(int i = 0; i < 3; ++i)
    {
        for(int j = 0; j < 2; ++j)
        {
            cout<< "test[" << i << "][" << j << "] = " << test[i][j] << endl;
        }
    }

    return 0;
}
```

Output

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
test[2][1] = 1
```