المقدمة:

١- الخوارزمية
٢-التسلسل
٣-الاختيار
٤-التكرار

المقدمة:

- الخوارزميات العودية البسيطة"Simple Recursive".
- خوارزميات البرمجة الديناميكية"Dynamic Programming"
- الخوارزميات التراجعية"Backtracking"
- خوارزميات فرق تسد"Divide-and-conquer"
- خوارزميات الجشع"Greedy"
- خوارزمية هجوم القوة العمياء "Brute Force Attack"
- الخوارزمية العشوائية "Randomized algorithm".

1-Sequence

- The sequence is exemplified by sequence of statements place one after the other - the one above or before another gets executed first.

Example.1: Write an algorithm to find the area of a circle of radius r?

Area of a circle (A) = $\pi.r^2$

Input: Radius (r)of the circle.

output: Area (A) of the circle

Algorithm:

Step1: Start

Step2: Read\input the radius r of the circle

Step3: A ← PI* r * r

Step4: Print Area (A)

Step5: End

Example.2: Write an algorithm to read two numbers and find their sum?

Input: First num1, Second num2.

Output: Sum of the two numbers.

Algorithm:

Step1: Start
Step2: Read\input the first number (num1)

Step3: Read\input the second number(num2)

Step4: Sum num1+num2

Step5: Print Sum

Step6: End

2. Branching or Selection

The branch refers to a binary decision based on some condition. If the condition is true, one of the two branches is explored; if the condition is false, the other alternative is taken.

Example.4: write an algorithm to find the greater number between two input numbers?

Input: First number A, Second number B.

Output: greater number (C).

Algorithm:

Step1: Start
Step2: Read/input A and B
Step3: If A greater than or equal B then C=A
Step4: If B greater than or equal A then C=B
Step5: Print C
Step6: End

3-Loop or Repetition

The loop allows a statement or a sequence of statements to be repeatedly executed based on some loop condition. It is represented by the ‘while’ and ‘for’ constructs in most programming languages.

You must ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers. The loop is also known as the repetition structure.

Example.6: Write an algorithm to print even numbers between 0 and 99?

Input: First number A, Second number B.

Output: Even numbers (I).

Step1: Start

Step2: Read/input A, and B

Step3: I ← 0

Step4: Print I

Step5: I ← I+2

Step6: If (I <=98) then go to Step 4

Step7: End

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الاولى – البرمجة المهيكلة
الفصل الأول
المخططات الانسيابية

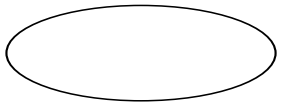
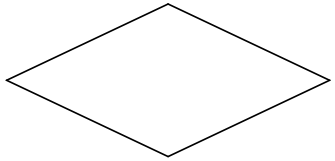
تقديم : أ.م.د حيدر ناصر خريبط

The Flowchart

- (Dictionary) A schematic representation of a sequence of operations, as in a manufacturing process or computer program.
- (Technical) A graphical representation of the sequence of operations in an information system or program.
  - Information system flowcharts show how data flows from source documents through the computer to final distribution to users.
  - Program flowcharts show the sequence of instructions in a single program or subroutine. Different symbols are used to draw each type of flowchart.

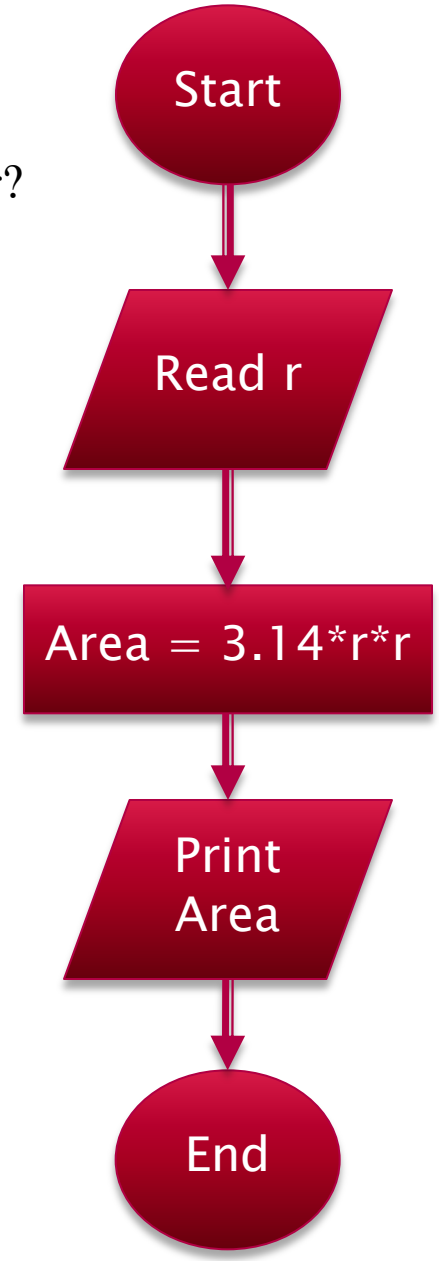
Flowchart Symbols

Basic

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

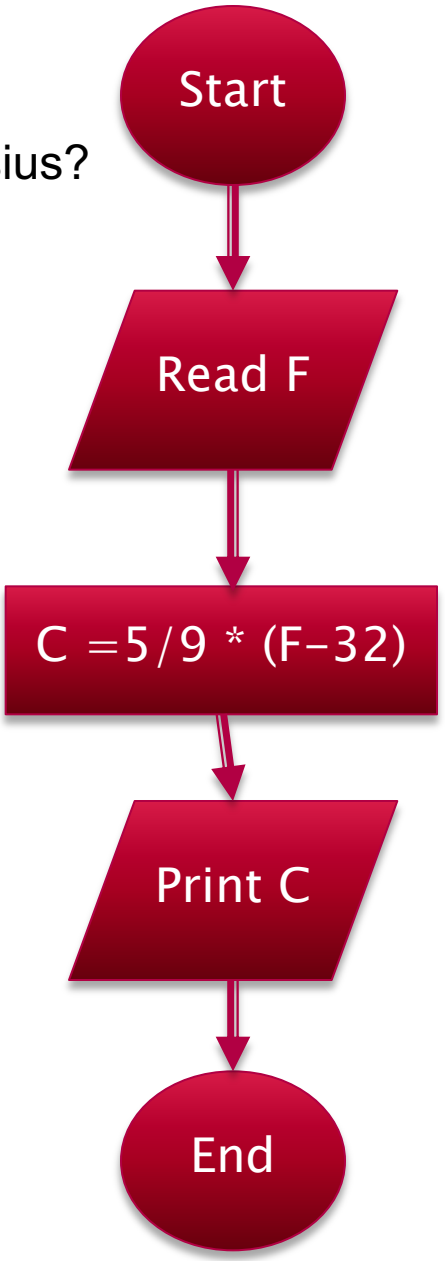
Examples

EX1: Draw a flow chart to find the area of a circle of radius r?



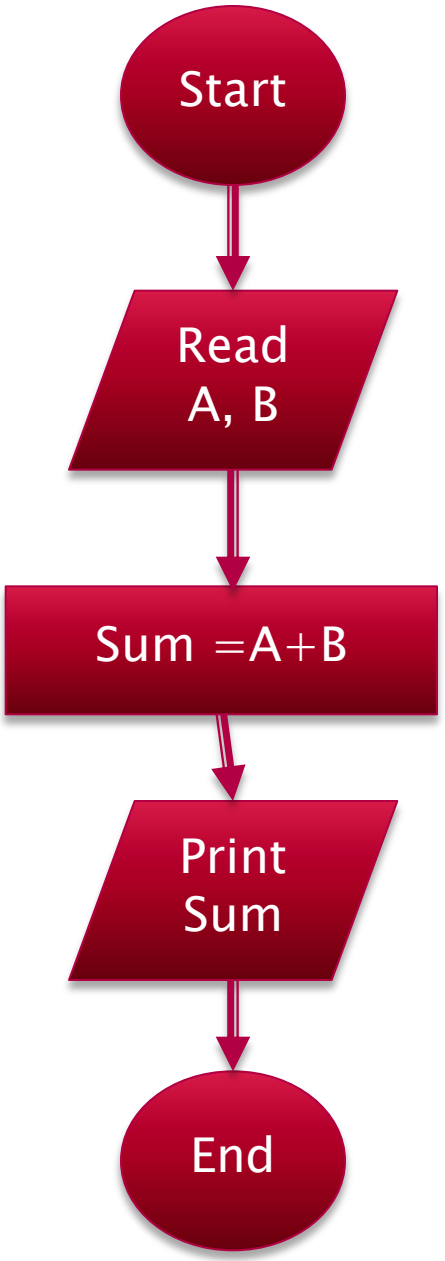
Examples

EX2: Draw a flow chart to convert temperature Fahrenheit to Celsius?



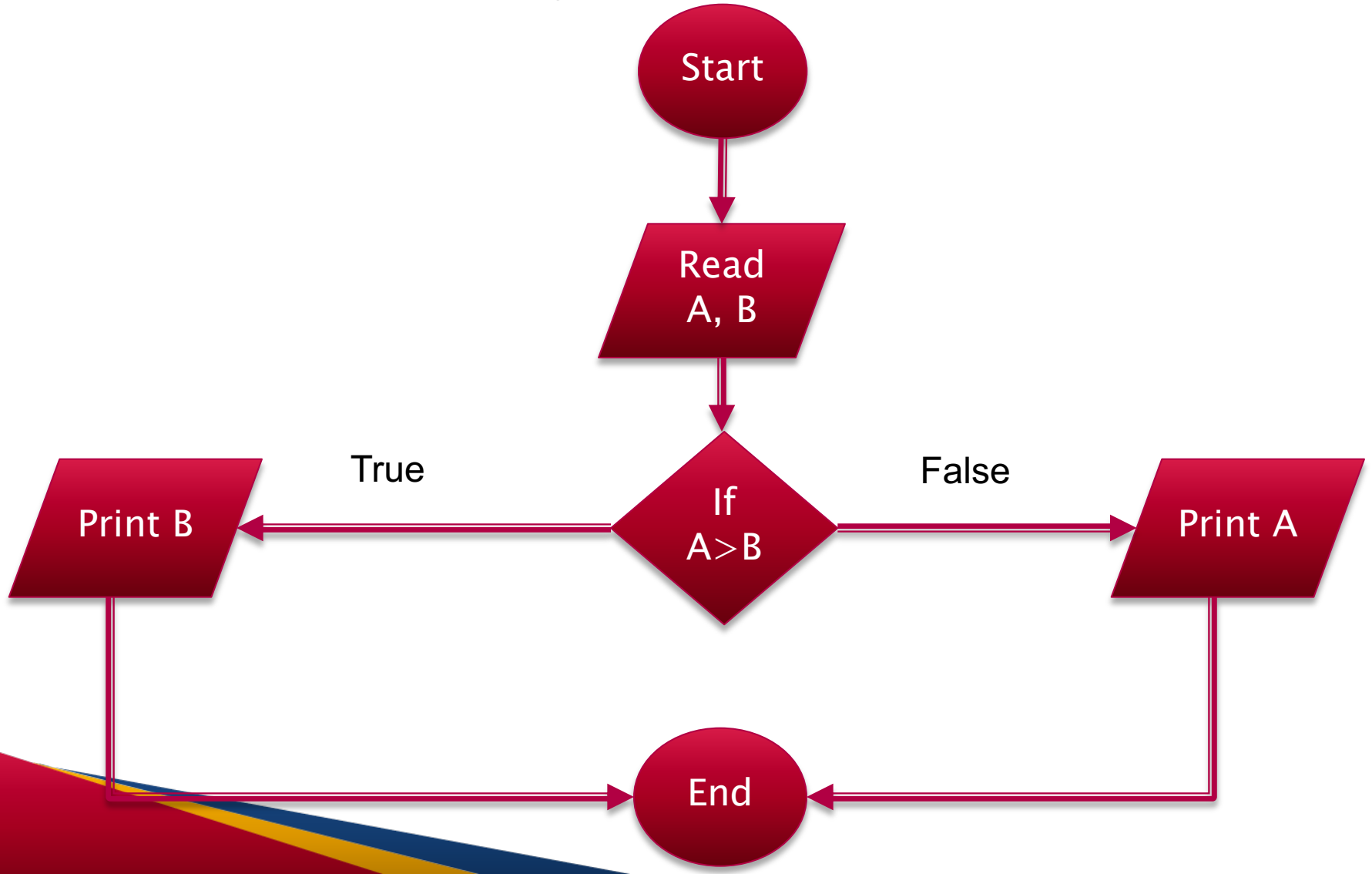
Examples

EX3: Draw a flow chart for inputs two numbers and prints sum of their value?



Examples

EX4: Draw a flow chart to find the greater number between two numbers?



Flowcharts

- Flowcharts is a graph used to show a step by step solution using **symbols** which represent a task.
- The symbols used consist of geometrical shapes that are connected by **flow lines**.
- It is an alternative to pseudo coding; whereas a pseudocode description is verbal, a flowchart is graphical in nature.

Assignment

1- Draw a flow chart for printing even numbers between 9 and 98?

2- Draw a flow chart for reading students marks for 5 subjects and then calculate their averages?

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الاولى – البرمجة المهيكلة
الفصل الثاني
Introduction to C++

تقديم : أ.م.د حيدر ناصر خريبط

Introduction to C++

When we consider a C++ program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into:-

- Class
- Object
- Methods
- Instant variables

Introduction to C++…

- **Class** − A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.

- **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.

Introduction to C++…

- **Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

- **Instance Variables** − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

A Simple C++ Program

```
// my first program in C++

/* my hello world program in C++ with

   More comments */

#include <iostream>
using namespace std;

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

A Simple C++ Program

```
// my first program in C++

/* my hello world program in C++ with

   More comments */
```

- This is a comment line and comment block. All lines beginning with two slash signs (//) will represent comment line. Also, all lines beginning with slash followed by star and ended up with start followed by slash will represent comment block (/* */). Both comment line and comment block considered comments and do not have any effect on the behaviour of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

A Simple C++ Program

```
#include <iostream>
```

- *#include<iostream>* tells the preprocessor to include the *iostream* standard file. This specific file (*iostream*) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

A Simple C++ Program

```
using namespace std;
```

- All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library.

A Simple C++ Program

```
int main ()
```

- This line corresponds to the beginning of the definition of the *main* function. The *main* function is the point by where all C++ programs start their execution, independently of its location within the source code.

A Simple C++ Program

```
cout << "Hello World!";
```

- ***cout*: represents the **standard output** stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen)

A Simple C++ Program

```
return 0;
```

The *return* statement causes the main function to finish. *return* may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

The general format of a C++ program

```
//Introductory comment
//file name, programmer, when written or modified
/* what program does……
……..
……..
……. */

#include <iostream>

using namespace std;

   int main()
   {
        constant declarations
        variable declarations
        executable statements
   }
```

Example 1

Write a C++ program that prints on screen the following sentences:

{****Shatt Al-arab University College***}

{* Computer Science Department*}

{**First Year**}

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "{****Shatt Al-arab University College***}" << endl;
    cout << "    {* Computer Science Department *}"
    <<endl;
    cout << "              {**First Year**}" << endl;
   return 0;
}
```

Cin: is the **standard input**

- ***Cin***: is the **standard input** device is usually the keyboard. Handling the standard input in C++ is done by applying the overloaded operator of extraction (>>) on the cin stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age ;
cin >> age;
```

Example 2

Write a C++ program that enter any integer value and find its double, the screen output should be like:

```
Please enter an integer value: 10
The value you entered is 10 and its double is 20
```

```
/* A C++ program that enter any integer value
     and find its double, the screen output should be like:
           Please enter an integer value: 10
           The value you entered is 10 and its double is 20
*/

#include <iostream>
using namespace std;
int main ( )
  {
     int i;
     cout << "Please enter an integer value: ";
     cin >> i;
     cout << "The value you entered is " << i;
     cout << " and its double is " << i*2;
    return 0;
  }
```

C++ Data Types

Primitive Built-in Types

- C++ offers the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types −

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Several of the basic types can be modified using one or more of these type modifiers −

- signed
- unsigned
- short
- long

Variable Declaration in C++

- A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable definition at the time of linking of the program.

Variable Declaration in C++

- A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program.
- We have two types of variable (Global and Local)

Example

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

// Global Variable declaration:
 int a, b;
 int c;
 float f;

int main () {
   // Local Variable definition:
   int a, b;
   int c;
   float f;

   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

   cout << c << endl ;

   f = 70.0/3.0;
   cout << f << endl ;

   return 0;
}
```

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الاولى – البرمجة المهيكلة
الفصل الثاني
Introduction to C++

تقديم : أ.م.د حيدر ناصر خريبط

Initialization of variables

- When declaring a regular local variable, its value is by default undetermined. But you may want a variable to store a concrete value at the same moment that it is declared. In order to do that, you can initialize the variable. There are two ways to do this in C++:

Initialization of variables

The first one, known as c-like, is done by appending an equal sign followed by the value to which the variable will be initialized:

type identifier = initial_value ;

For example, if we want to declare an int variable called a initialized with a value of 0 at the moment in which it is declared, we could write:

int a = 0;

Initialization of variables

The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses (()):

type identifier (initial_value) ;

For example:

int a (0);

Both ways of initializing variables are valid and equivalent in C++.

Example

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
 {
   int a=5; // initial value = 5
   int b(2); // initial value = 2
   int result; // initial value undetermined
                a = a + 3;
                result = a - b;
                cout << result;
   return 0;
 }
```

Constants

- **Literal constants:**

Literal constants are literal numbers used to express particular values within the source code of a program.

- double a=5.8; // initial value = 5.8
- int b(2); // initial value = 2
- char ch = ‘H’; // initial char value= H
- bool a = true; // initial a value= 1

Constants

- **Symbolic constants:**

Symbolic constants can be declared in two different ways: using the ***#define*** preprocessor directive, and through use of the ***const*** keyword.

Constants

- You can define your own names for constants that you use very often without having to resort to memory consuming variables, simply by using the #define preprocessor directive. Its format is:

#define identifier value

For example:

#define PI 3.14159

Example

EX6: Write a C++ program to calculate the weekly pay?

WeeklyPay = workDays * workHours * payRate;

Example

```
#include <iostream>
using namespace std;
int main (void)
 {
int workDays;
float workHours, payRate, weeklyPay;
workDays = 5;

workHours = 7.5;
payRate = 38.55;

weeklyPay = workDays * workHours *
payRate;
cout << "Weekly Pay = "<< weeklyPay <<
'\n';

return 0;
   }
```

Example

- Write a C++ program to input three different types of data and outputs it?

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

{
  int n; float f; char c;
  cout << "input integer number: ";
  cin>>n;
  cout<<n<<endl;
  cout << "input float number: ";
  cin>>f;
  cout<<f<<endl;
  cout << "input character: ";
  cin>>c ;
  cout<< c ;
  return 0;
}
```

Operators in C++

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provide the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Operators in C++

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provide the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators

There are following arithmetic operators supported by C++ language −
Assume variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Example

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

int main() {
   int a = 21;
   int b = 10;
   int c ;
   c = a + b;
   cout << "Line 1 - Value of c is :" << c << endl ;
   c = a - b;
   cout << "Line 2 - Value of c is  :" << c << endl ;
   c = a * b;
   cout << "Line 3 - Value of c is :" << c << endl ;
   c = a / b;
   cout << "Line 4 - Value of c is  :" << c << endl ;
   c = a % b;
   cout << "Line 5 - Value of c is  :" << c << endl ;
   c = a++;
   cout << "Line 6 - Value of c is :" << c << endl ;
   c = a--;
   cout << "Line 7 - Value of c is  :" << c << endl ;
   return 0;
}
```

Relational Operators

- There are following relational operators supported by C++ language
- Assume variable A holds 10 and variable B holds 20, then

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 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 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 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 left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Example

```
#include <iostream>
using namespace std;
int main() {
   int a = 21;
   int b = 10;
   int c ;
   if( a == b ) {
      cout << "Line 1 - a is equal to b" << endl ;
   } else {
      cout << "Line 1 - a is not equal to b" << endl ;
   }
   if( a < b ) {
      cout << "Line 2 - a is less than b" << endl ;
   } else {
      cout << "Line 2 - a is not less than b" << endl ;
   }
   if( a > b ) {
      cout << "Line 3 - a is greater than b" << endl ;
   } else {
      cout << "Line 3 - a is not greater than b" << endl ;
   }
   /* Let's change the values of a and b */
   a = 5;
   b = 20;
```

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الاولى – البرمجة المهيكلة
الفصل الثاني
Introduction to C++

تقديم : أ.م.د حيدر ناصر خريبط

Logical Operators

- There are following logical operators supported by C++ language.
- Assume variable A holds 1 and variable B holds 0, then −

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.

Example

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

int main() {
   bool a = 0;
   bool b = 1;

      cout << "Line 1 - Condition is "<< (a ||
b)<<endl ;
      cout << "Line 2 - Condition is "<< (a &&
b)<<endl ;
      cout << "Line 3 - Condition is "<< !(a &&
b)<<endl ;


   return 0;
}
```

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 will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

Example

```
#include <iostream>
using namespace std;
int  main() {
   int a = 21;
   int c =5;
   c =  a;
   cout << "Line 1: ( = ) Operator, Value of c = : " <<c<< endl ;
   c +=  a;
   cout << "Line 2: ( += )Operator, Value of c = : " <<c<< endl ;
   c -=  a;
   cout << "Line 3: ( -=) Operator, Value of c = : " <<c<< endl ;
   c *=  a;
   cout << "Line 4: (*=) Operator, Value of c = : " <<c<< endl ;
   c /=  a;
   cout << "Line 5: (/=) Operator, Value of c = : " <<c<< endl ;
   c  = 200;
   c %=  a;
   cout << "Line 6: (%=) Operator, Value of c = : " <<c<< endl ;
   return 0;
}
```

Operators Precedence in C++

- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.

- Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

Category	Operator	Associativity
Postfix	(), [ ], –, ++, – –	Left to right
Multiplicative	*, / , %	Left to right
Additive	+, –	Left to right
Relational	< ,<= ,> ,>=	Left to right
Equality	==, !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Assignment	=, +=, –=, *=, /=, %=	Right to left

Example

```
#include <iostream>
using namespace std;
int main() {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;
   e = (a + b) * c / d;
   cout << "Value of (a + b) * c / d is :" << e << endl ;
   e = ((a + b) * c) / d;
   cout << "Value of ((a + b) * c) / d is  :" << e << endl ;
   e = (a + b) * (c / d);
   cout << "Value of (a + b) * (c / d) is  :" << e << endl ;
   e = a + (b * c) / d;      //  20 + (150/5)
   cout << "Value of a + (b * c) / d is  :" << e << endl ;
   return 0;
}
```

Homework

- Find the answer of the following:

a = 100 + 200 / 10 – 3 * 10

b = 100 / 10 * 10

c = 7 + (3 * (20/4))-13

Homework

- Applying the operator precedence, and find the following expression :

3+4*4 > 5*(4+3)-1

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الاولى – البرمجة المهيكلة

الفصل الثالث

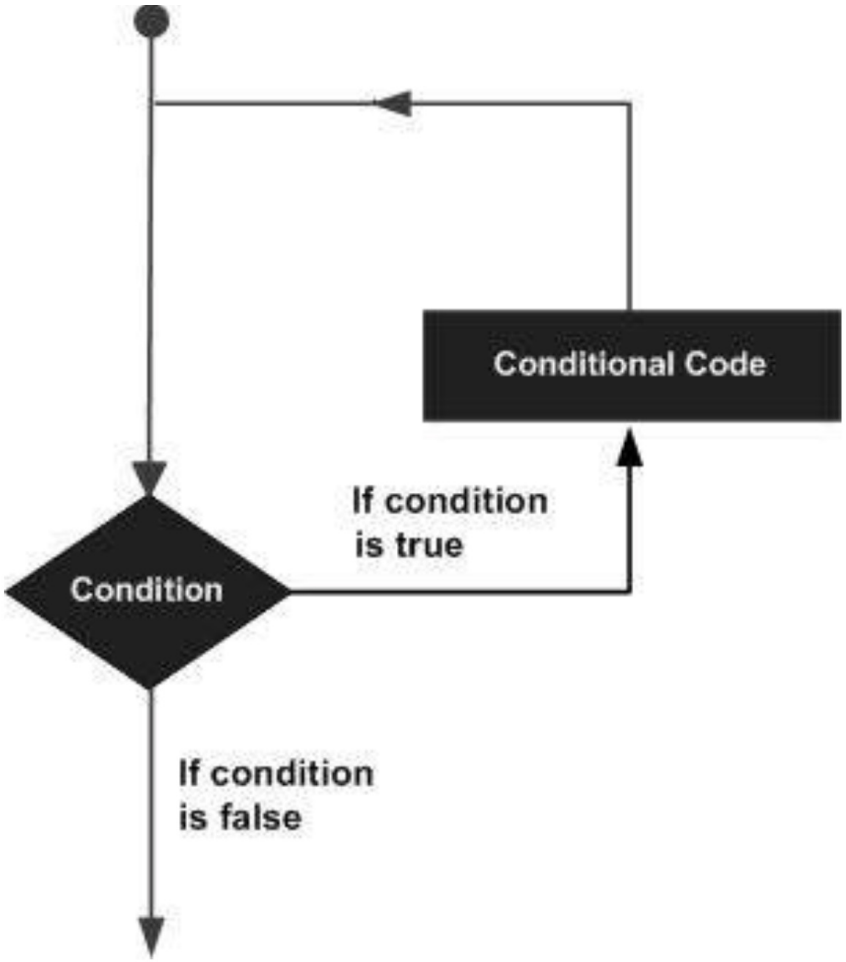
C++ Iteration (Loop)

تقديم : أ.م.د حيدر ناصر خريبط

Introduction

- There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages

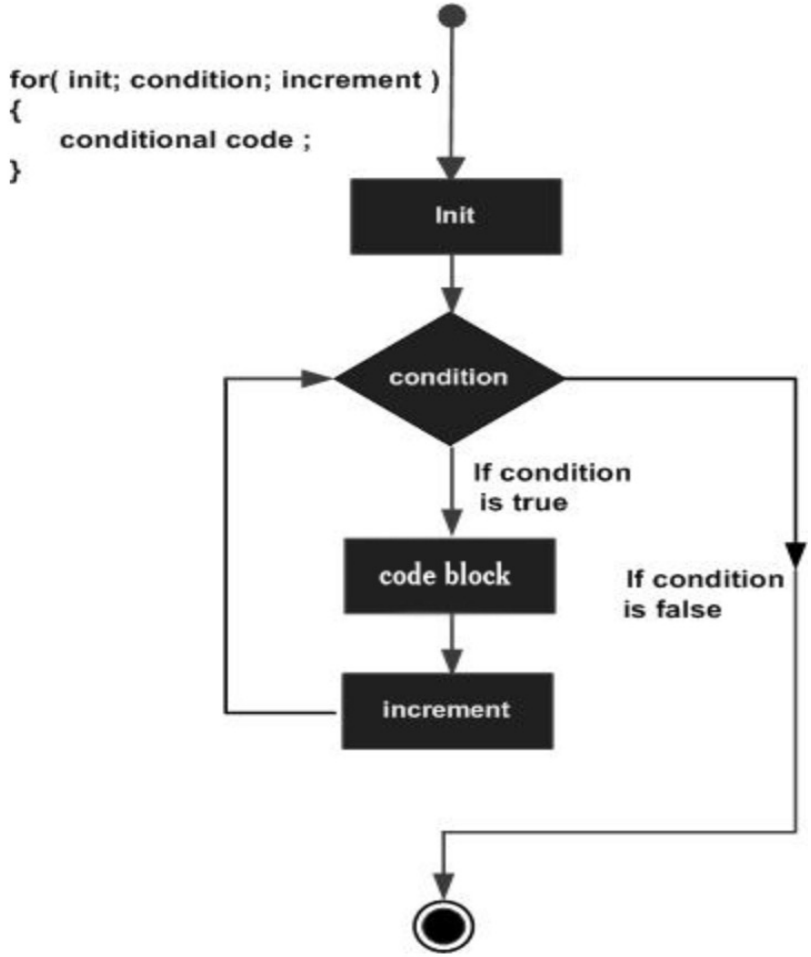
Introduction



C++ Iteration (Loop) Types

No	Statement	Description
1	for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	do...while loop	Like a 'while' statement, except that it tests the condition at the end of the loop body.
4	Nested loop	You can use one or more loop inside any another 'while', 'for' or 'do.. while' loop.

1- For loop- Flow Diagram



Flow diagram of for loop

1- For loop-Syntax

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

```
for ( init; condition; increment )
{
        statement(s);
}
```

Here is the flow of control in a for loop:-

- The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

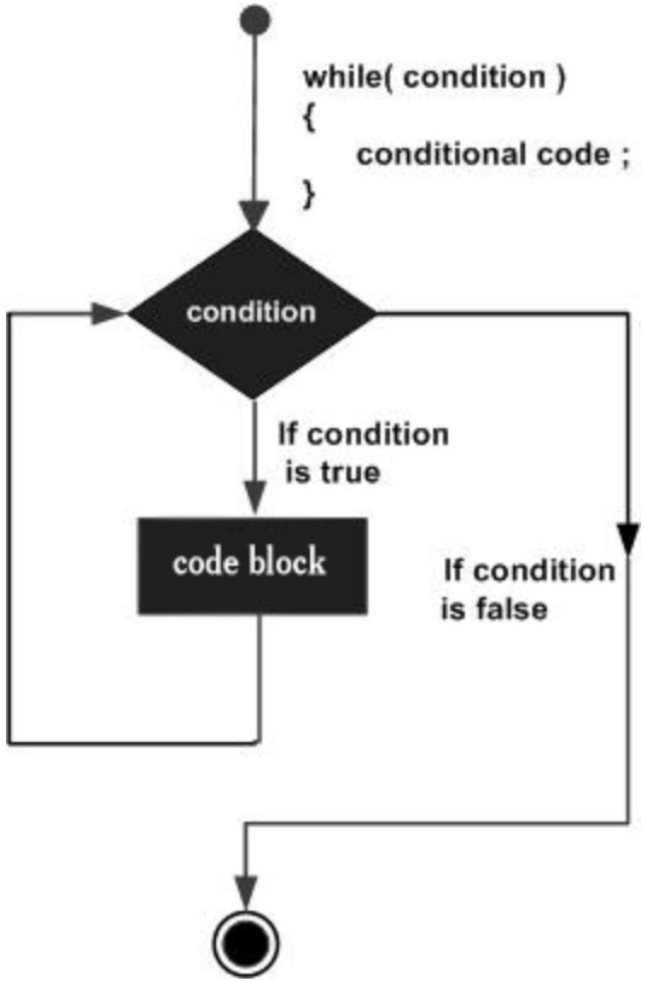
1- For loop-Syntax…

- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

1- For loop- Example:

```
#include <iostream>
using namespace std;
 int main () {
   // for loop execution
   for( int a = 10; a < 20; a = a + 1 ) {
      cout << "value of a: " << a << endl;
   }
    return 0;
}
```

2- while loop- Flow Diagram



Flow diagram of while loop

2-While loop-Syntax

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.
Syntax
The syntax of a while loop in C++ is −

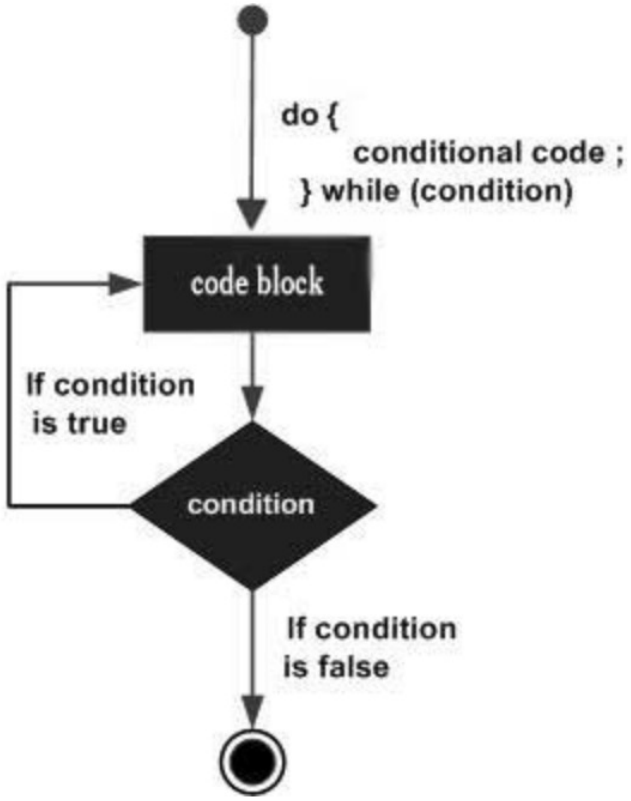
```
while(condition) {
statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
When the condition becomes false, program control passes to the line immediately following the loop.

2-While loop-Example

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

int main () {
   // Local variable declaration:
   int a = 10;

   // while loop execution
   while( a < 20 ) {
      cout << "value of a: " << a << endl;
      a++;
   }

   return 0;
}
```

3- do.. while loop- Flow diagram



Flow diagram of do ... while loop

3- do.. while loop- Syntax

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.
A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax
The syntax of a do...while loop in C++ is −

```
do {
 statement(s);
 } while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.
If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

3- do.. while loop-Example

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

int main () {
   // Local variable declaration:
   int a = 10;

   // do loop execution
   do {
      cout << "value of a: " << a << endl;
      a = a + 1;
   } while( a < 20 );

   return 0;
}
```

4- **Nested loop**

- A loop can be nested inside of another loop. C++ allows at least 256 levels of nesting.

- Syntax

The syntax for a **nested for loop** statement in C++ is as follows −

```
for ( init; condition; increment ) {
   for ( init; condition; increment ) {
      statement(s);
   }
   statement(s); // you can put more
statements.
}
```

4- Nested loop…

- The syntax for a **nested while loop** statement in C++ is as follows −

```
while(condition) {
   while(condition) {
      statement(s);
   }
   statement(s); // you can put more
statements.
}
```

- The syntax for a **nested do...while loop** statement in C++ is as follows −

```
do {
   statement(s); // you can put more
statements.
   do {
      statement(s);
   } while( condition );

} while( condition );
```

4- Nested loop- Example

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

int main () {
   int i, j;

   for(i = 2; i<100; i++) {
      for(j = 2; j <= (i/j); j++)
         if(!(i%j)) break; // if factor found, not prime
         if(j > (i/j)) cout << i << " is prime\n";
   }

   return 0;
}
```

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الاولى – البرمجة المهيكلة
الفصل الثالث
C++ Decision making statements

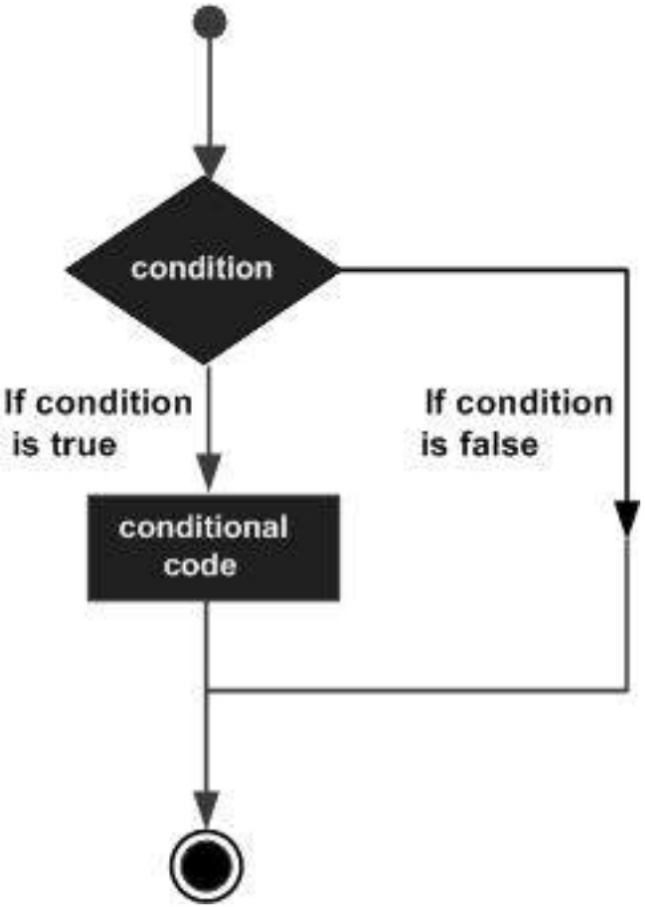
تقديم : أ.م. د حيدر ناصر خريبط

C++ decision making statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages −

C++ Decision making statements..



C++ Decision making statements..

Sr.No	Statement & Description
1	if statement An ‘if’ statement consists of a boolean expression followed by one or more statements.
2	if...else statement An ‘if’ statement can be followed by an optional ‘else’ statement, which executes when the boolean expression is false.
3	switch statement A ‘switch’ statement allows a variable to be tested for equality against a list of values.
4	nested if statements You can use one ‘if’ or ‘else if’ statement inside another ‘if’ or ‘else if’ statement(s).

1- **If** statement

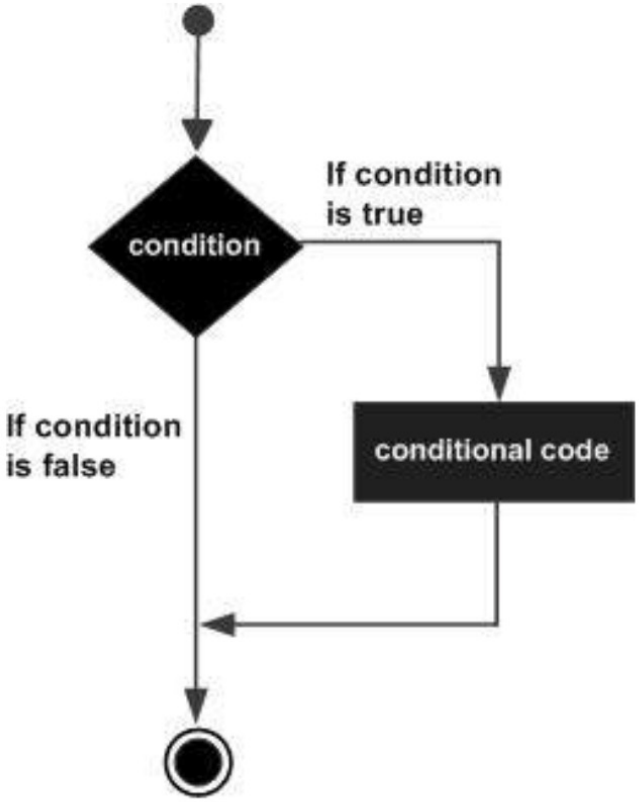
An **if** statement consists of a Boolean expression followed by one or more statements.

Syntax

The syntax of an if statement in C++ is :-

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

1-**If** statement Flow Diagram



1-**If** statement example:-

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

int main () {
   // local variable declaration:
   int a = 10;

   // check the boolean condition
   if( a < 20 ) {
      // if condition is true then print the following
      cout << "a is less than 20;" << endl;
   }
   cout << "value of a is : " << a << endl;

   return 0;
}
```

2- If else statement

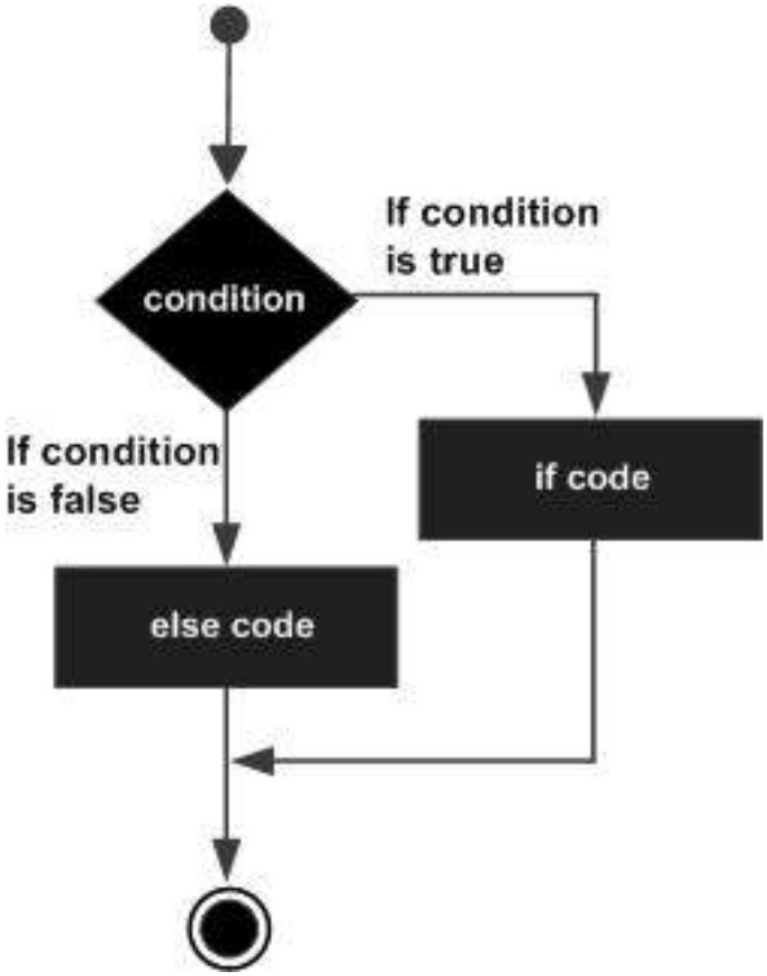
An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.

Syntax

The syntax of an if...else statement in C++ is −

```
if(boolean_expression) {
 // statement(s) will execute if the boolean expression is true
} else { // statement(s) will execute if the boolean expression is false
}
```

2-**If** else statement Flow Diagram



2-**If** else statement example:-

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

int main () {
   // local variable declaration:
   int a = 100;

   // check the boolean condition
   if( a < 20 ) {
      // if condition is true then print the following
      cout << "a is less than 20;" << endl;
   } else {
      // if condition is false then print the following
      cout << "a is not less than 20;" << endl;
   }
   cout << "value of a is : " << a << endl;

   return 0;
}
```

2-if...else if...else Statement..

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

An if can have zero or one else's and it must come after any else if's.

An if can have zero to many else if's and they must come before the else.

Once an else if succeeds, none of he remaining else if's or else's will be tested.

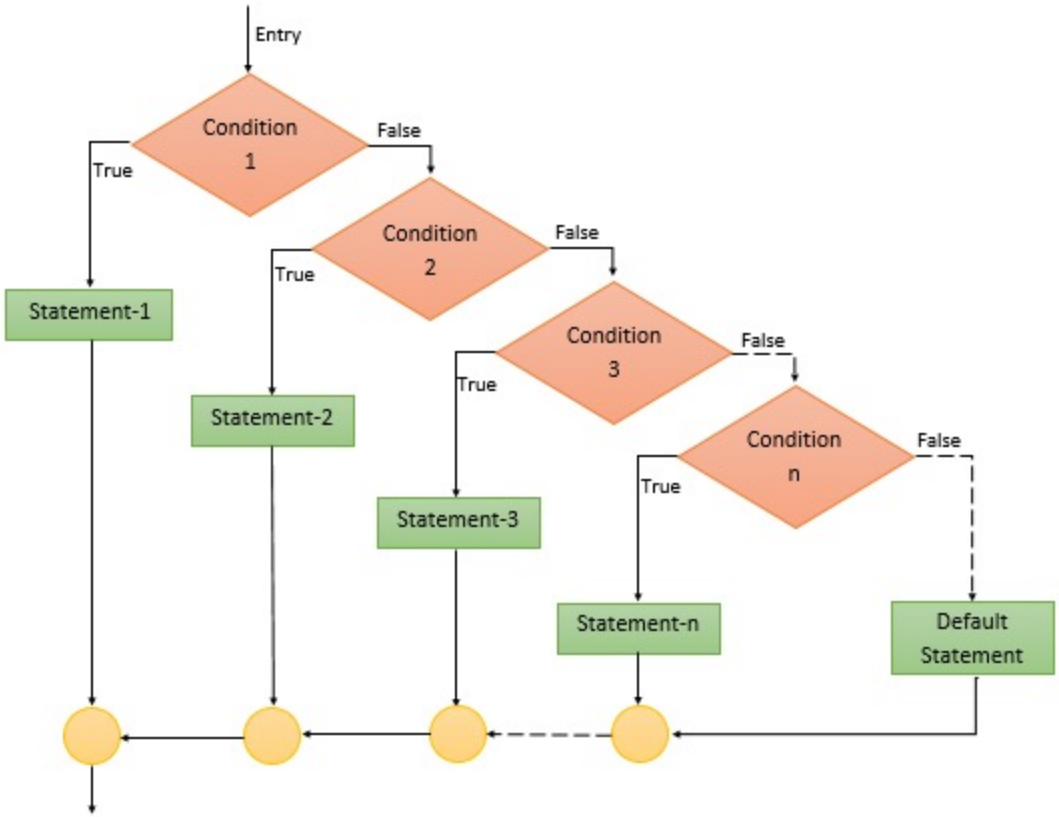
2-if...else if...else Statement…

Syntax

The syntax of an if...else if...else statement in C++ is −

```
if(boolean_expression 1) {
   // Executes when the boolean expression 1 is true
} else if( boolean_expression 2) {
   // Executes when the boolean expression 2 is true
} else if( boolean_expression 3) {
   // Executes when the boolean expression 3 is true
} else {
   // executes when the none of the above condition is true.
}
```

2-if...else if...else flow Diagram..



2-if...else if...else example:-

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

int main () {
   // local variable declaration:
   int a = 100;

   // check the boolean condition
   if( a == 10 ) {
      // if condition is true then print the following
      cout << "Value of a is 10" << endl;
   } else if( a == 20 ) {
      // if else if condition is true
      cout << "Value of a is 20" << endl;
   } else if( a == 30 ) {
      // if else if condition is true
      cout << "Value of a is 30" << endl;
   } else {
      // if none of the conditions is true
      cout << "Value of a is not matching" << endl;
   }
   cout << "Exact value of a is : " << a << endl;

   return 0;
}
```

3- A **switch** statement

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax

The syntax for a **switch** statement in C++ is as follows −

3- A **switch** statement ..

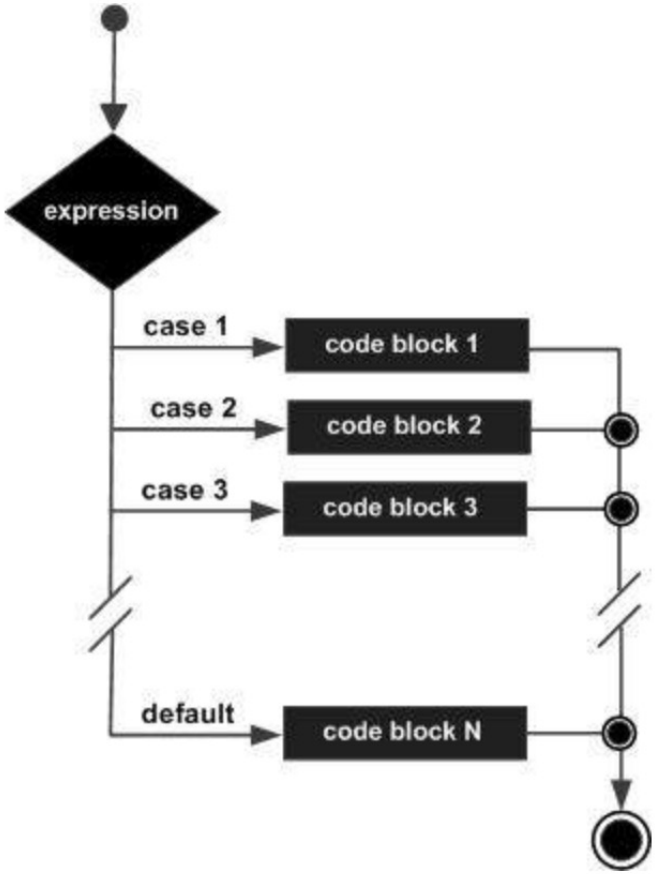
```
switch(expression) {
case constant-expression :
      statement(s);
      break; //optional
case constant-expression :
      statement(s);
      break; //optional
// you can have any number of case statements.

      default : //Optional
      statement(s);
 }
```

3- A **switch** statement ..

The following rules apply to a switch statement −

1) The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
2) You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
3) The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
4) When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
5) When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
6) Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
7) A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

3- A **switch** statement Flow Diagram



3- A **switch** statement example:-

```
#include <iostream>
using namespace std;
 int main () {
   // local variable declaration:
   char grade = 'D';
   switch(grade) {
      case 'A' :
         cout << "Excellent!" << endl;
         break;
      case 'B' :
      case 'C' :
         cout << "Well done" << endl;
         break;
      case 'D' :
         cout << "You passed" << endl;
         break;
      case 'F' :
         cout << "Better try again" << endl;
         break;
      default :
         cout << "Invalid grade" << endl;
   }
   cout << "Your grade is " << grade << endl;
    return 0;
}
```

4-C++ nested if statements

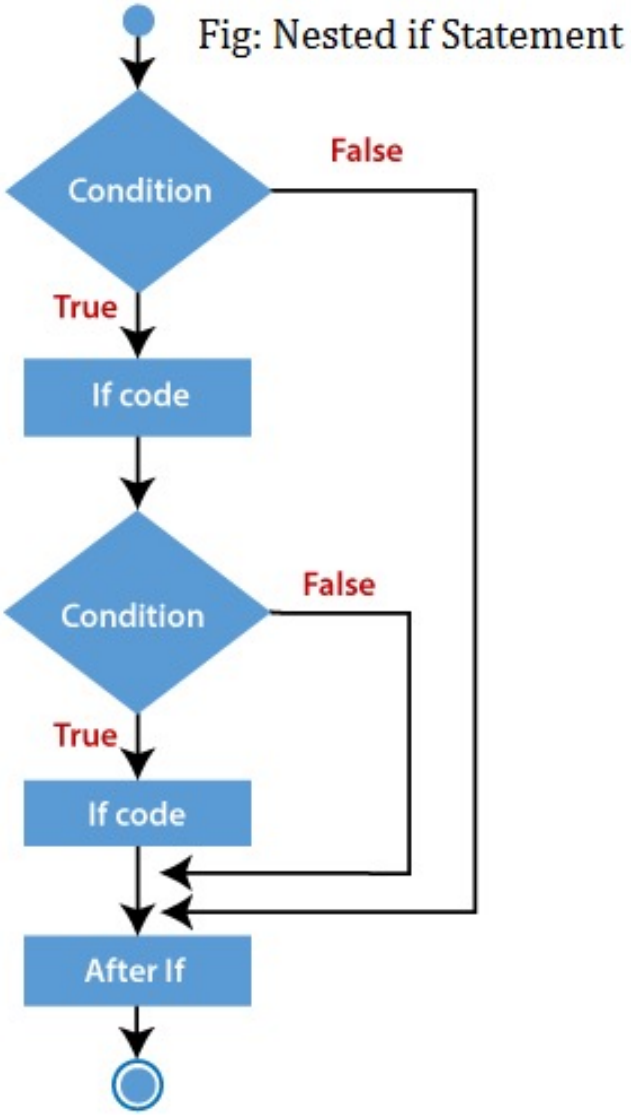
It is always legal to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Syntax

The syntax for a **nested if** statement is as follows −

```
if( boolean_expression 1) {
   // Executes when the boolean expression 1 is true
   if(boolean_expression 2) {
      // Executes when the boolean expression 2 is
true
   }
}
```

4-C++ nested if flow diagram



4-C++ nested if example:-

```
#include <iostream>
using namespace std;
 int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;
    // check the boolean condition
   if( a == 100 ) {
      // if condition is true then check the following
      if( b == 200 ) {
         // if condition is true then print the following
         cout << "Value of a is 100 and b is 200" << endl;
      }
   }
   cout << "Exact value of a is : " << a << endl;
   cout << "Exact value of b is : " << b << endl;
    return 0;
}
```

Assignment for c++ programming:-

Choose any five questions and write a c++ programming:-

1-The assignment due day is :- 11/4/2022
2-The mark for this Assignment will be 5 degree.

Thank You…

كلية شط العرب الجامعة - قسم علوم الحاسبات

المرحلة الثانية – البرمجة الكيانية

Chapter 4- C++ Function

تقديم : أ.م.د حيدر ناصر خريبط

1-Introduction

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

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

2-Introduction …

- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

- The C++ standard library provides numerous built-in functions that your program can call. For example, function **strcat()** to concatenate two strings, function **memcpy()** to copy one memory location to another location and many more functions.
- A function is known with various names like a method or a sub-routine or a procedure etc.

The general form of a C++ function definition is as follows −

Return_type Function_name (parameter list (**Parameters**))
{
Function Body
}

2-Defining a Function

- A C++ function definition consists of a function header and a function body. Here are all the parts of a function −
- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

2-Defining a Function…

- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

2-Defining a Function…

- **Function Body** − The function body contains a collection of statements that define what the function does.

3-Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

Function_name (Parameter 1, Parameter 2, Parameter n);

3-Calling a Function

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. For example :-

Function_name (Parameter 1, Parameter 2, Parameter n);

Example

```
#include <iostream>
using namespace std;
 // function declaration
int max(int num1, int num2);
 int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;
   int ret;
    // calling a function to get max value.
   ret = max(a, b);
   cout << "Max value is : " << ret << endl;
    return 0;
}
 // function returning the max between two numbers
int max(int num1, int num2) {
   // local variable declaration
   int result;
    if (num1 > num2)
      result = num1;
   else
      result = num2;
    return result;
}
```

Assignment

- **باستخدام الدوال صمم برنامج يستقبل nمن الاعداد ويطبع فقط الاعداد الزوجية منها؟**

- **اكتب برنامج بلغة C++ يحتوي على دالة تحسب قيمة ال Power؟**
  ملاحظة Pow(5,2)=25 ؟

- **اكتب برنامج بلغة C++ يحتوي على دالة تحسب Factorial؟**

Thank You…