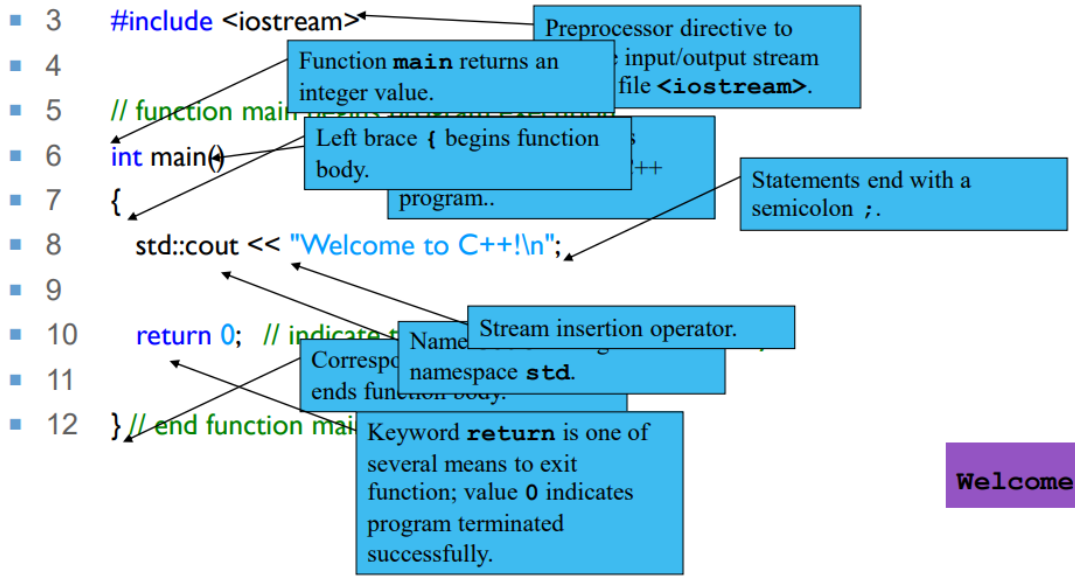
Chapter1. C++ Concept Revision

First program in c++




```
Welcome to C++!
```

Escape Sequence

هي رموز خاصة تستخدم داخل النصوص.

الرمز	المعنى
\n	سطر جديد
\t	Tab
\r	الرجوع لبداية السطر
\a	إصدار صوت تنبيه
\\	طباعة \
\"	طباعة "

Example

```
cout << "Welcome\nto\n\nC++!";
```

result

```
Welcome
to

C++!
```

QUENS

Variables

- Location in memory where value can be stored
- Common data types
  - `int` - integer numbers
  - `char` - characters
  - `double` - floating point numbers
- Declare variables with name and data type before use

```
int integer1;
int integer2;
int sum;
```

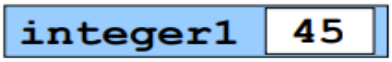
- Can declare several variables of same type in one declaration
  - Comma-separated list

```
int integer1, integer2, sum;
```

Variable names

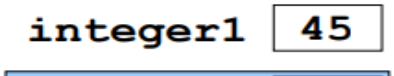
- **Correspond to actual locations in computer's memory**
- **Every variable has name, type, size and value**
- **When new value placed into variable, overwrites previous value**

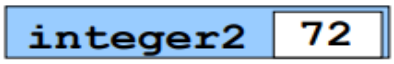
CE 100 - CS 101 - CS 112 - CS 203
CE 100 - CS 101 - CS 112 - CS 203
Tel: 41161170 - 66200958




```
std::cin >> integer1;
```

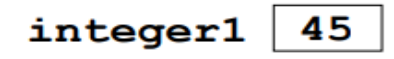
- Assume user entered 45

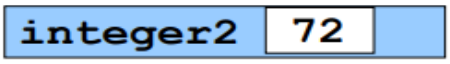





```
std::cin >> integer2;
```

- Assume user entered 72








```
sum = integer1 + integer2;
```

Arithmetic Operators

أهم العمليات الحسابية:

Operator	العملية
+	جمع
–	طرح
*	ضرب
/	قسمة
%	باقي القسمة

Operator Precedence

- Rules of operator precedence
  - Operators in parentheses evaluated first
    - Nested/embedded parentheses
      - Operators in innermost pair first
  - Multiplication, division, modulus applied next
    - Operators applied from left to right
  - Addition, subtraction applied last
    - Operators applied from left to right

Operator(s)
()
*, /, or %
+ or –

CE 100 - CS 101 - CS 112 - CS 203

ترتيب تنفيذ العمليات:

Assignment Operators

بدل:

```
c = c + 3;
```

يمكننا كتابة:

```
c += 3;
```

وبنفس الطريقة:

```
d -= 4;   //  d = d - 4
e *= 5;   //  e = e * 5
f /= 3;   //  f = f / 3
g %= 9;   //  g = g % 9
```

```
variable operator= value
```

Increment and Decrement Operators

- Increment operator (++) - can be used instead of `c += 1`
- Decrement operator (--) - can be used instead of `c -= 1`
  - Preincrement
    - `++c / --c`
  - Posincrement
    - `c++ / c--`
- If `c = 5`, then
  - `cout << ++c;` prints out 6
  - `cout << c++;` prints out 5

CE 100 - CS 101 - CS 112 - CS 203

using statements

ex. using std::cout

Eliminate use of std:: prefix

Write cout instead of std::cout

or write using namespace std; in begin of program above main

```
#include
using std::cout;    //can write using namespace std
int main()
{
cout << "Welcome ";
std::cout << "to C++!\n";
return 0;
}
```

Control Structures

- Specifies the order in which statements are to executed
- Program control
  1. Sequential execution
     - Statements executed one after the other in the order written
  2. Transfer of control
     - When the next statement executed is not the next one in sequence

CE 100 - CS 101 - CS 112 - CS 203

QUENS

1. Types of Control Structures

1. Selection structures

   C++ has three types - **`if`**, **`if/else`**, and **`switch`**

1. Repetition structures

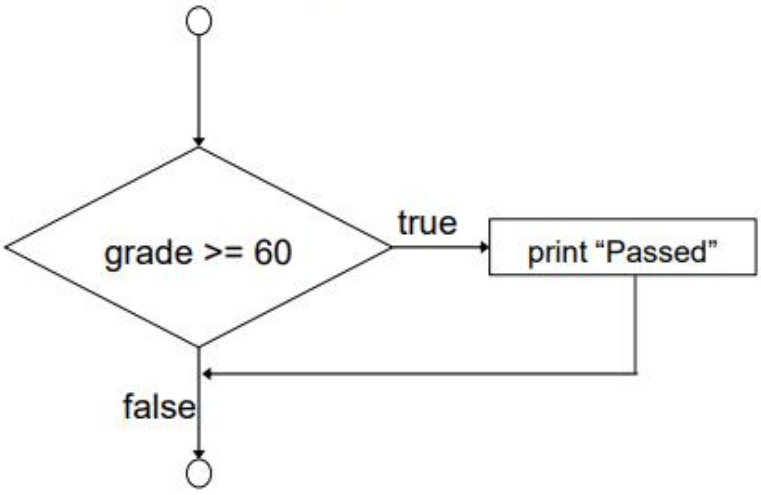
   C++ has three types - **`while`**, **`do/while`**, and **`for`**

EX

```
if ( grade >= 60 )
cout << "Passed";
```

The if Selection Structure

- Flowchart of pseudocode statement



A decision can be made on any expression.

zero - **`false`**

nonzero - **`true`**

Example:

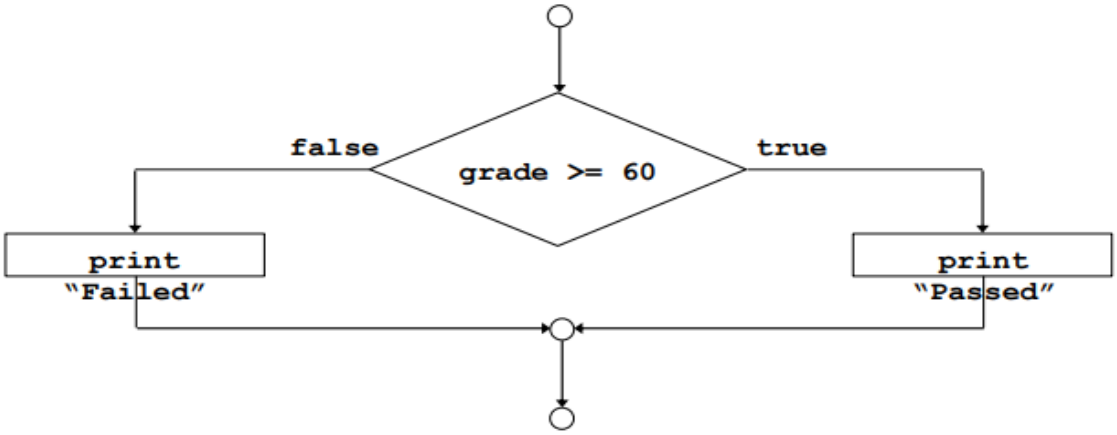
`3 - 4` is **`true`**

CE 100 - CS 101 - CS 112 - CS 203
Tel: 41161170 - 66200958

The IF/ELSE Selection Structure

- If Only performs an action if the condition is true
- if/else A different action is performed when condition is true and when condition is false

```
if ( grade >= 60 )
cout << "Passed";
else
cout << "Failed";
```



The FOR Repetition Structure

- The general format when using **for** loops is

```
for ( initialization; LoopContinuationTest;increment )
    statement
```

- Example:

```
for( int counter = 1; counter <= 10; counter++ )
   cout << counter << endl;
```

  - Prints the integers from one to ten

CE 100 - CS 101 - CS 112 - CS 203

QUENS

Program to sum the numbers from 1 to 100

```
1  // Fig. 2.20: fig02_20.cpp
2  // Summation with for
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int sum = 0;
11
12    for ( int number = 1; number <= 100; number += 1 )
13       sum += number;
14
15    cout << "Sum is " << sum << endl;
16
17    return 0;
18 }
```

```
Sum is 5050
```

CE 100 - CS 101 - CS 112 - CS 203

The WHILE Repetition Structure

while loop repeated until condition becomes false.

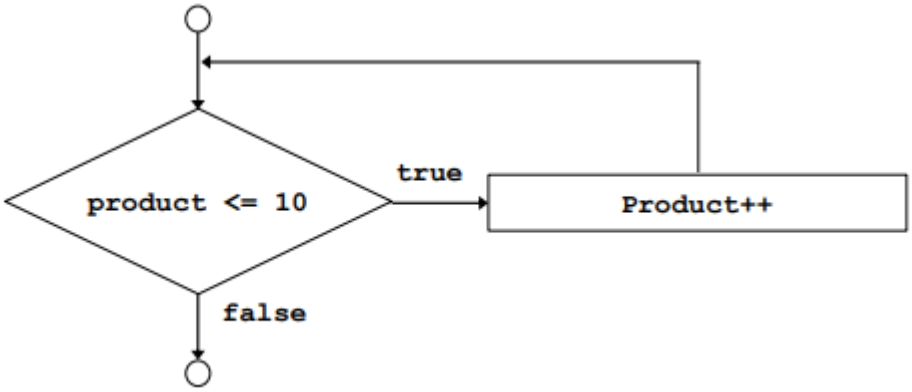
Syntax

```
initialization;
while(repetition condition)
{
 statement increment;
}
```

- Example:

```
int counter =1;          //initialization
while (counter <= 10){//repetition condition
   cout << counter << endl;
   ++counter;              //increment
   }
```

- Flowchart of **while** loop



CE 100 - CS 101 - CS 112 - CS 203
CE 100 - CS 101 - CS 112 - CS 203

- Program to sum the numbers from 1 to 5

```
2  // Summation with while
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int sum = 0;int number = 1;
12    while (number <= 5)
       { number ++;
13       sum += number;
14        }
15    cout << "Sum is " << sum << endl;
16
17    return 0;
18 }
```

```
Sum is 15
```


CE 100 - CS 101 - CS 112 - CS 203

CE 100 - CS 101 - CS 112 - CS 203

Functions In C++

- C++ allows the use of both internal (user-defined) and external functions.
- External functions (e.g., **abs, ceil, exp, pow, rand, sqrt**, etc.) are usually grouped into specialized libraries (e.g., **iostream, stdlib, math**, etc.)

Basic Approach to Programming

```
int main()
{
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
}
```

This program has one long, complex function containing all of the statements necessary to solve a problem.

CE 100 - CS 101 - CS 112 - CS 203

QUEENS

Modular Programming – Using Functions

```
int main()
{
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
}
```



```
int main()
{
    function1();
    function2();
}
```

Main function

```
int function1()
{
    Statements;
    Statements;
    Statements;
    Statements;
    Statements;
}
```

Function I

```
void function2()
{
    Statements;
    Statements;
    Statements;
}
```

Function2

User-defined functions

`main()` then uses these functions to solve the original problem.

Three main components of Function

- Function prototypes
- Function definition
- Function Call

CE 100 - CS 101 - CS 112 - CS 203

Function Prototypes

- Tells compiler **argument type** and **return type** of function
- Function prototype contains
  - Function name
  - Parameters (number and data type)
  - Return type (`void` if returns nothing)
  - Only needed if function definition after function call

EX.

```
double maximum( double, double, double );
```


CE 100 - CS 101 - CS 112 - CS 203
Tel: 41161170 - 66200958

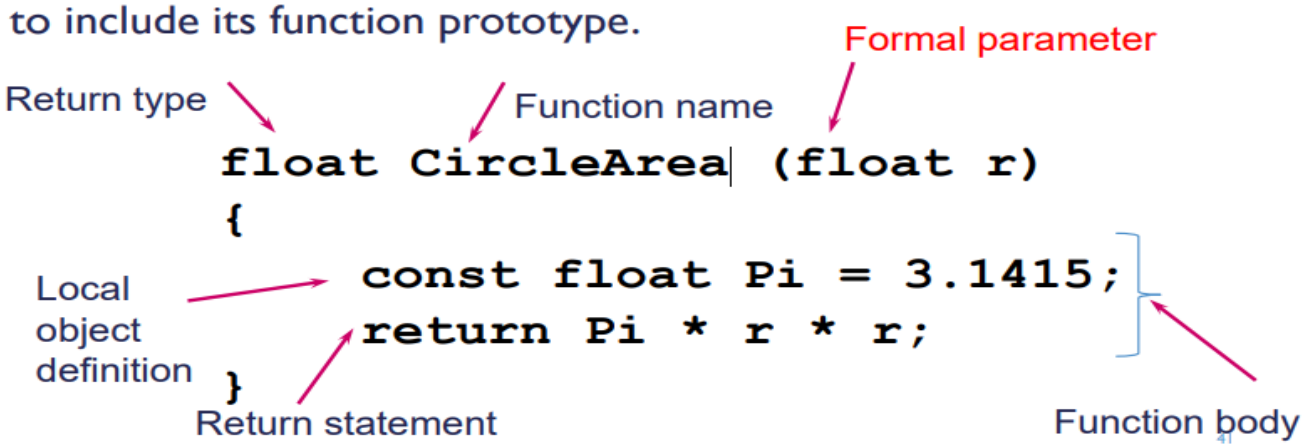
Function Definitions

- Format for function definition

  return-value-type function-name (parameter-list)
  {
  declarations and statements
  }

  - Parameter list
    - Comma separated list of arguments
      - Data type needed for each argument
  - Return-value-type
    - Data type of result returned (use **void** if nothing returned)

- The function definition can be placed anywhere in the program after the function prototypes.
- If a function definition is placed in front of main(), there is no need to include its function prototype.



CE 100 - CS 101 - CS 112 - CS 203

Calling Function

A **function call** has the following syntax:

```
<function name>(<argument list>)
```

Example -

```
A = CircleArea(MyRadius);
```

Actual parameter

To process the invocation, the function that contains the insertion statement is suspended and `CircleArea()` does its job. The insertion statement is then completed using the value supplied by `CircleArea()`.

Simple Program – Calculate Area Of A Circle

```
#include <iostream>
using namespace std;
// function declaration/prototype
float CircleArea(float r);

int main() {                    // main() function
   cout << "Enter radius: ";
   float MyRadius;
   cin >> MyRadius;
   float Area = CircleArea(MyRadius); //function call
   cout << "Circle has area " << Area;
   return 0;
}

float CircleArea(float r) {    // function definition
   const float Pi = 3.1415;
   return Pi * r * r;
}
```

```
Enter radius: 10
Circle has area 314.15
```


CE 100 - CS 101 - CS 112 - CS 203
Tel: 41161170 - 66200958

Call By Value

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

void sumdiff(int a, int b, int s, int d) {
    s = a + b;
    d = a - b;
}
int main(){
    int x=7, y=3, sum=0, diff=0;
    sumdiff(x, y, sum, diff);
    cout << "sum = " << sum << endl;    //0
    cout << "diff = " << diff << endl; //0
}
```

Call By Reference

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

void sumdiff(int a, int b, int &s, int &d) {
    s = a + b;
    d = a - b;
}
int main(){
    int x=7, y=3, sum=0, diff=0;
    sumdiff(x, y, sum, diff);
    cout << "sum = " << sum << endl;    //10
    cout << "diff = " << diff << endl; //4
}
```


Tel: 41161170 - 66200958

CE 100 - CS 101 - CS 112 - CS 203

Default Arguments

- Function call with omitted parameters
  - If not enough parameters, rightmost go to their defaults
  - Default values
    - Can be constants, global variables, or function calls
- Set defaults in function prototype

```
int myFunction( int x = 1, int y = 2, int z = 3 );
```

  - `myFunction(3)`
    - `x = 3`, `y` and `z` get defaults (rightmost)
  - `myFunction(3, 5)`
    - `x = 3`, `y = 5` and `z` gets default

QUEENS ACADEMY

Example

```
// Using default arguments.
#include <iostream>
using std::cout;
using std::endl;
// function prototype that specifies default arguments
int boxVolume( int length = 1, int width = 1, int height = 1 )
{
   return  length * width *height;
};
int main()
{
// no arguments--use default values for all dimensions
cout << "The default box volume is: " << boxVolume();

// specify length; default width and height
cout << "\n\nThe volume of a box with length 10,\n"
     << "width 1 and height 1 is: " << boxVolume( 10 );
}
```

Set defaults in function prototype.

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10
```

Function calls with some parameters missing – the rightmost parameters get their defaults.

CE 100 - CS 101 - CS 112 - CS 203
Tel: 41161170 - 66200958