جامعة
القاهرة الجديدة
التكنولوجية

NEW CAIRO
TECHNOLOGICAL
UNIVERSITY

TUE – The Technological Universities in Egypt

NCTU – New Cairo Technological University

Faculty of Industry and Energy Technology

Information Technology Department

Second-Year

Course: Programming Essentials in C++

Lecture 2

Presented by

Dr. Ghada Maher

Contents:

Escape Sequence to format the Displayed Text

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\'	Single quote. Use to print a single quote character.
\"	Double quote. Used to print a double quote character.

Creating Comments in C++ program

- We can put comment on our programs using
- // to comment a single line

```
/*
to comment multiple lines
Mulitple
Lines
*/
```

Interacting With User: Take Input From User

- In C++ , we use cin >> Variable; To accept an input from the
- To use cin >>, we must use #include <iostream >
- #include iostream notifies the preprocessor to include in the program the contents of the input/output stream header file iostream
- A variable is a location in the computer s memory where a value can be stored for use by a program.
- All variables must be declared with a name and a data type before they can be used in a program.
- Declaration

  Data Type Identifier:
  - Int width ;
  - Float salary ;

Example 1:

Write a program to find the Area of a rectangle

The area of the Rectangle are given by the following formula:

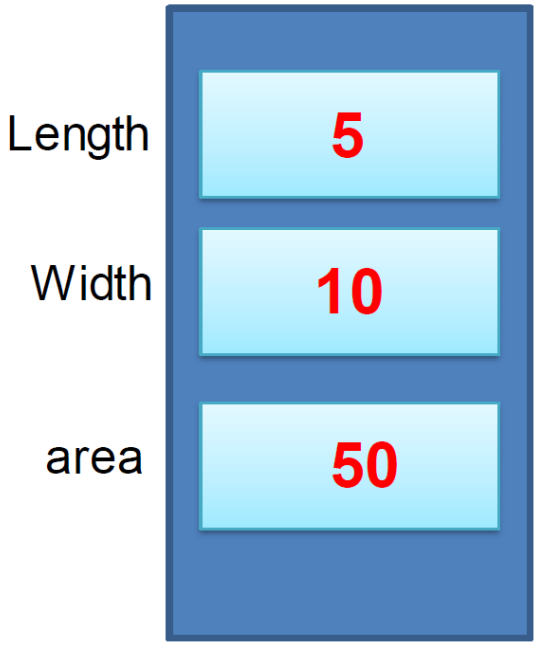
Area = Rectangle Length * Rectangle Width.

Input :
Rectangle Length , Rectangle Width.
Processing :
Area = Rectangle Length * Rectangle Width.
Output :
Print Out The area.

C++ program

```
#include<iostream>
Using namespace std;
Int main()
{
Float length,width,Area;
Cout<<"Enter the length";
Cin>>length;
Cout<<"Enter the width";
Cin>>width;
Area= length*width;
Cout<<" The area of rectangle = "<<Area;
Return 0;
}
```

Run:
Enter the length 5
Enter the width 10
The area of rectangle =50



Example 2:

Write a program to find the perimeter and area of a square

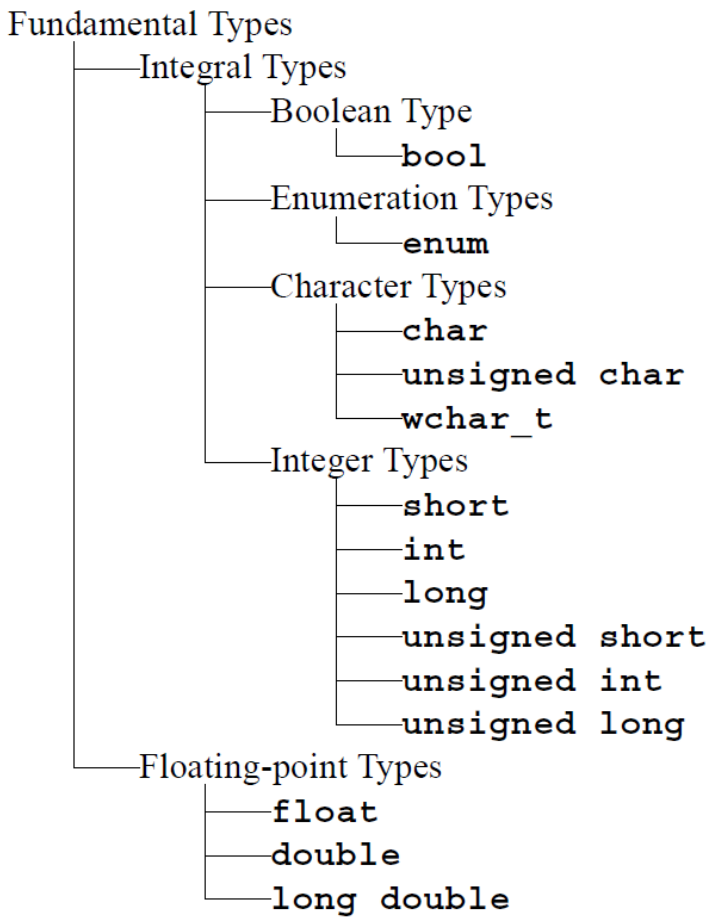
- Input:
  - Square Side Length
- Processing:
  - perimeter = Side Length * 4
  - area = Side Length * Side Length
- Output:
  - Print Out The Perimeter and Area.

C++ program

```
#include<iostream>
Using namespace std;
Int main()
{
Float Side Length ,Area;
Cout<<”Enter the Side Length ”;
Cin>> Side Length;
perimeter = Side Length * 4
area = Side Length * Side Length
Cout<<“ The perimeter =“<< perimeter<<endl;
Cout<<“ The area =“<< area<<endl;
Return 0;
}
```

C++ Fundamental Data Types

char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)



Declaring & Initializing Variables

- Initialization means to give a variable an initial value.

- Variables can be initialized when declared:

```
int first= 13 , 10
char ch ='
double x= 12.6
```

- All variables must be initialized before they are used in an arithmetic operatation

  But not necessarily during declaration

Example 3:

```
#include<iostream>
Using namespace std;
Int main()
{
int x,y;
int x=5;
Y+=x;
cout<<y;
```

Run: ????

Rules on Variable Names

- Do not use reserved words as variable names
  (e.g. if, else, int , float, case, for,

- The first character has to be a letter or underscore. It can not be a numeric digit.

- The second and the other characters of the name can be any letter, any number, or an underscore "__".

Examples:

- Some valid names:
  my_name, m 113 1 , salary, _at

- Some invalid names:
  my name, 1 stmonth

Example 4: Boolean Variables


```
int main()
{ // prints the value of a boolean variable:
  bool flag=false;
  cout << "flag = " << flag << endl;
  flag = true;
  cout << "flag = " << flag << endl;
}
```

```
flag = 0
flag = 1
```

Note that the value `false` is printed as the integer `0` and the value `true` is printed as the integer `1`.

Example 5: Character Variables


```
int main()
{ // prints the character and its internally stored integer value:
  char c='A';
  cout << "c = " << c << ", int(c) = " << int(c) << endl;
  c='t';
  cout << "c = " << c << ", int(c) = " << int(c) << endl;
  c='\t';  // the tab character
  cout << "c = " << c << ", int(c) = " << int(c) << endl;
  c='!';
  cout << "c = " << c << ", int(c) = " << int(c) << endl;
}
```

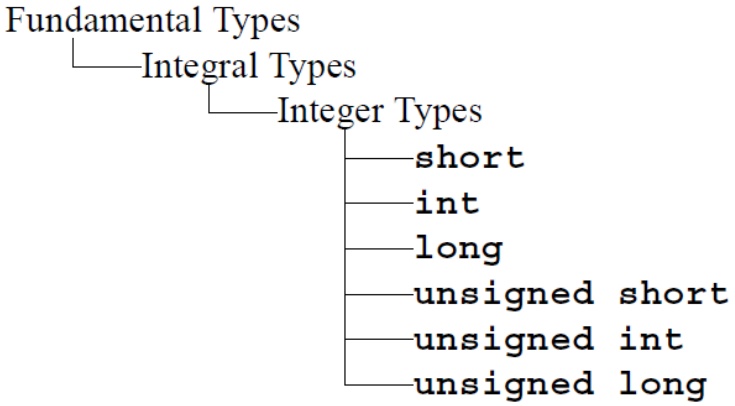
```
c = A, int(c) = 65
c = t, int(c) = 116
c =     , int(c) = 9
c = !, int(c) = 33
```

Example 6: Integer Type Ranges

This program prints the numeric ranges of the 6 integer types in C++:

```
#include <iostream>
#include <climits>     // defines the constants SHRT_MIN, etc.
using namespace std;
int main()
{ // prints some of the constants stored in the <climits> header:
  cout << "minimum short = " << SHRT_MIN << endl;
  cout << "maximum short = " << SHRT_MAX << endl;

  cout << "maximum unsigned short = 0" << endl;
  cout << "maximum unsigned short = " << USHRT_MAX << endl;
  cout << "minimum int = " << INT_MIN << endl;
  cout << "maximum int = " << INT_MAX << endl;
  cout << "minimum unsigned int = 0" << endl;
  cout << "maximum unsigned int = " << UINT_MAX << endl;
  cout << "minimum long= " << LONG_MIN << endl;
  cout << "maximum long= " << LONG_MAX << endl;
  cout << "minimum unsigned long = 0" << endl;
  cout << "maximum unsigned long = " << ULONG_MAX << endl;
}
```



Run:

```
minimum short = -32768
maximum short = 32767
maximum unsigned short = 0
maximum unsigned short = 65535
minimum int = -2147483648
maximum int = 2147483647
minimum unsigned int= 0
maximum unsigned int= 4294967295
minimum long = -2147483648
maximum long = 2147483647
minimum unsigned long = 0
maximum unsigned long = 4294967295
```

Arithmetic Operators

Example 7

This example illustrates how the arithmetic operators work.

```
int main()
{ // tests operators +, -, *, /, and %:
  int m=54;
  int n=20;
  cout << "m = " << m << " and n = " << n << endl;
  cout << "m+n = " << m+n << endl;  // 54+20 = 74
  cout << "m-n = " << m-n << endl;  // 54-20 = 34
  cout << "m*n = " << m*n << endl;  // 54*20 = 1080
  cout << "m/n = " << m/n << endl;  // 54/20 = 2
  cout << "m%n = " << m%n << endl;  // 54%20 = 14
}
```

```
m = 54 and n = 20
m+n = 74
m-n = 34
m*n = 1080
m/n = 2
m%n = 14
```

Note that integer division results in another integer: 54/20 = 2, not 2.7.

Applying the Pre-increment and Post increment Operators

Example 8:

```
int main()
{ // shows the difference between m++ and ++m:
  int m, n;
  m = 44;
  n = ++m;  // the pre-increment operator is applied to m
  cout << "m = " << m << ", n = " << n << endl;
  m = 44;
  n = m++;  // the post-increment operator is applied to m
  cout << "m = " << m << ", n = " << n << endl;
}
```

```
m = 45, n = 45
m = 45, n = 44
```

The line

```
n = ++m;  // the pre-increment operator is applied to m
```

increments m to 45 and then assigns that value to n. So both variables have the same value 45 when the next output line executes.

The line

```
n = m++;  // the post-increment operator is applied to m
```

increments m to 45 only after it has assigned the value of m to n. So n has the value 44 when the next output line executes.

Example 9: Floating-Point Arithmetic


```
int main()
{ // tests the floating-point operators +, -, *, and /:
  double x=54.0;
  double y=20.0;
  cout << "x = " << x << " and y = " << y << endl;
  cout << "x+y = " << x+y << endl;  // 54.0+20.0 = 74.0
  cout << "x-y = " << x-y << endl;  // 54.0-20.0 = 34.0
  cout << "x*y = " << x*y << endl;  // 54.0*20.0 = 1080.0
  cout << "x/y = " << x/y << endl;  // 54.0/20.0 = 2.7
}
```

```
x = 54 and y = 20
x+y = 75
x-y = 35
x*y = 1100
x/y = 2.7
```

Unlike integer division, floating-point division does not truncate the result: 54.0/20.0 = 2.7.

Composite Assignment Operators

Example 10:

```
int main()
{ // tests arithmetic assignment operators:
  int n=22;
  cout << "n = " << n << endl;
  n += 9;  // adds 9 to n
  cout << "After n += 9, n = " << n << endl;
  n -= 5;  // subtracts 5 from n
  cout << "After n -= 5, n = " << n << endl;
  n *= 2;  // multiplies n by 3
  cout << "After n *= 2, n = " << n << endl;
  n /= 3;  // divides n by 9
  cout << "After n /= 3, n = " << n << endl;
  n %= 7;  // reduces n to the remainder from dividing by 4
  cout << "After n %= 7, n = " << n << endl;
}
```

```
n = 22
After n += 9, n = 31
After n -= 5, n = 26
After n *= 2, n = 52
After n /= 3, n = 17
After n %= 7, n = 3
```

Example 11: This program tells you how much space each of the 12 fundamental types uses:


```
int main()
{ // prints the storage sizes of the fundamental types:
  cout << "Number of bytes used:\n";
  cout << "\t          char: " <<           sizeof(char) << endl;
  cout << "\t         short: " <<          sizeof(short) << endl;
  cout << "\t           int: " <<            sizeof(int) << endl;
  cout << "\t          long: " <<           sizeof(long) << endl;
  cout << "\t unsigned char: " <<  sizeof(unsigned char) << endl;
  cout << "\tunsigned short: " << sizeof(unsigned short) << endl;
  cout << "\t  unsigned int: " <<   sizeof(unsigned int) << endl;
  cout << "\t unsigned long: " <<  sizeof(unsigned long) << endl;
  cout << "\t   signed char: " <<    sizeof(signed char) << endl;
  cout << "\t         float: " <<          sizeof(float) << endl;
  cout << "\t        double: " <<         sizeof(double) << endl;
  cout << "\t   long double: " <<    sizeof(long double) << endl;
}
```

```
Number of bytes used:
                  char: 1
                 short: 2
                   int: 4
                  long: 4
         unsigned char: 1
        unsigned short: 2
          unsigned int: 4
         unsigned long: 4
           signed char: 1
                 float: 4
                double: 8
           long double: 8
```

Example 12: Simple Type Casting

This program casts a `double` value into `int` value:

```
int main()
{ // casts a double value as an int:
  double v = 1234.56789;
  int n = int(v);
  cout << "v = " << v << ", n = " << n << endl;
}
```

```
v = 1234.57, n = 1234
```

The `double` value 1234.56789 is converted to the **`int`** value 1234.

When one type is to be converted to a "higher" type, the type case operator is not needed. This is called *type promotion*. Here's a simple example of *promotion* from **`char`** all the way up to **`double`**:

Example 13: Promotion of Types

This program promotes a `char` to a `short` to an `int` to a `float` to a `double`:

```
int main()
{ // prints promoted vales of 65 from char to double:
  char c='A';   cout << "  char c = " << c << endl;
  short k=c;    cout << " short k = " << k << endl;
  int m=k;      cout << "   int m = " << m << endl;
  long n=m;     cout << "  long n = " << n << endl;
  float x=m;    cout << " float x = " << x << endl;
  double y=x;   cout << "double y = " << y << endl;
}
```

Run:

```
  char c = A
 short k = 65
   int m = 65
  long n = 65
 float x = 65
double y = 65
```

The integer value of the character `'A'` is its ASCII code 65. This value is converted as a **`char`** in `c`, a **`short`** in `k`, an **`int`** in `m`, and a **`long`** in `n`. The value is then converted to the floating point value 65.0 and stored as a **`float`** in `x` and as a **`double`** in `y`. Notice that **`cout`** prints the integer `c` as a character, and that it prints the real numbers `x` and `y` as integers because their fractional parts are 0.

Example 14: Integer Overflow

This program repeatedly multiplies n by 1000 until it overflows.

```
int main()
{ // prints n until it overflows:
  int n=1000;
  cout << "n = " << n << endl;
  n *= 1000;  // multiplies n by 1000
  cout << "n = " << n << endl;
  n *= 1000;  // multiplies n by 1000
  cout << "n = " << n << endl;
  n *= 1000;  // multiplies n by 1000
  cout << "n = " << n << endl;
}
```

```
n = 1000
n = 1000000
n = 1000000000
n = -727379968
```

This shows that the computer that ran this program cannot multiply 1,000,000,000 by 1000 correctly.