EP241 Computer Programming

Topic 2

Dr. Ahmet BİNGÜL
Department of Engineering Physics
University of Gaziantep

Modifications by
Dr. Andrew BEDDALL
Department of Electric and Electronics Engineering

Sep 2014
Content of Topic 2

Extracted from cpp.gantep.edu.tr

- 2.1 Introduction
- 2.2 Data Types
- 2.3 Identifiers
- 2.4 Variables
- 2.5 Constants
- 2.6 Basic Operators
- 2.7 Basic Strings
- 2.8 Header Files
- 2.9 Basic Intrinsic Functions
2.1 Introduction

In this lecture we will learn some fundamental elements of C++:

Data types, operators, strings and intrinsic functions.

NOTE THAT

The C and C++ programming languages are quite different from each other, even though they share some common syntax.
2.2 Data Types

A data type determines the type of the data that will be stored, in the computer memory (RAM).

C++ provides 6 fundamental data types:

- char
- int
- float
- double
- bool
- wchar_t

There are also some qualifiers that can be put in front of the numerical data types to form derivatives:

- short
- long
- signed
- unsigned

For example:

- short int
- unsigned char
The table shows the fundamental data types in C++, as well as the range of values.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Size (byte)</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Character or small integer</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>unsigned char</td>
<td></td>
<td></td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>short int</td>
<td>Short integer</td>
<td>2</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>unsigned short int</td>
<td></td>
<td></td>
<td>0</td>
<td>65,535</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>4</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td></td>
<td></td>
<td>0</td>
<td>4,294,967,295</td>
</tr>
<tr>
<td>unsigned long int</td>
<td></td>
<td></td>
<td>0</td>
<td>18,446,744,073,709,551,615</td>
</tr>
<tr>
<td>float</td>
<td>Single precision floating point number (7 digits)</td>
<td>4</td>
<td>-3.4e +/- 38</td>
<td>+3.4e +/- 38</td>
</tr>
<tr>
<td>double</td>
<td>Double precision floating point number (15 digits)</td>
<td>8</td>
<td>-1.7e +/- 308</td>
<td>+1.7e +/- 308</td>
</tr>
<tr>
<td>long double</td>
<td>Quad precision floating point number (34 digits)</td>
<td>16</td>
<td>-1.0e +/- 4931</td>
<td>+1.0e +/- 4931</td>
</tr>
</tbody>
</table>

[*] only on 64 bit platforms.

Note that the unqualified `char`, `short`, `int`, `(long int)` are signed by default. And `unsigned` integers are always positive and so have a larger positive range.
2.3 Identifiers

An identifier is a string of alphanumeric characters. It is used for naming variables, constants, functions, structures and classes.

A valid identifier
- must begin with a letter or underscore (_),
- can consist only of letters (a-z, A-Z), digits (0-9), and underscores.
- should not match with any C++ reserved keywords which are:

  asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while
Remember to use only the English alphabet:

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

According to these rules, the following are valid identifiers:

- mass
- peynir
- pos12
- speed_of_light
- SpeedOfLight
- isPrime

while the following are not valid:

- 2ndBit
- speed of light
- yağmur
- c++
- float
NOTE THAT

C++ is case sensitive.

That is, it distinguishes uppercase letters from lowercase.

So, Food and food are different identifiers.
2.4 Variables

- A variable is a symbolic name indicating a location in your computer's memory where you can store a value which can later be retrieved.

- A variable has to be a valid identifier.

- When you define a variable in C++, you must tell the compiler what kind of variable it is: `int`, `double`, `char`, etc.

- All variables must be declared before use.
- Example declarations

```c
int i, j;
long k;
float w, x, y, z;
double speed, dragForce;
```

- When a variable is declared, you can *initialize* it in two alternative but equivalent ways

```c
int cake = 122;
```

or

```c
int cake(122);
```
Example: Declaration of variables

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

int main () {
    int x = 22, y = 11, z;
    z = x - y;
    cout << "z = " << z << endl;

    int p = 3;
    int q = x*y*z - 2*p;
    cout << "q = " << q << endl;
}
```

```
z = 11
q = 2656
```
In C++, a variable can have either *local* or *global* scope.
2.5 Constants

- Variables can be made constant with the `const` qualifier. Constants must be initialized at the point of declaration.

```c
const double PI = 3.1415926, TWOPI = 2.0*PI;
const int EOF = -1;
```

- Symbolic constants are defined via the `#define` preprocessor directive.

```c
#define PI 3.1415926
#define MAX 100
#define NEWLINE '\n'
```
- Enumeration, constants:

```cpp
enum { RED, GREEN, BLUE };
```

is shorthand for

```cpp
const int RED = 0, GREEN = 1, BLUE = 2;
```

- Enumeration starts by default with zero but we can override this

```cpp
enum { RED = 1, GREEN = 3, BLUE = 7 };
```

- If not assigned explicitly, each value is one greater than previous.

```cpp
enum { RED = 1, GREEN, BLUE };
```

is equivalent to

```cpp
enum { RED = 1, GREEN = 2, BLUE = 3 };
```
*Integer literal constants can be represented by three different bases: base-10 (decimal), base-8 (octal) and base-16 (hexadecimal)*

```
int i = 75;  // default base-10
int i = 0113;  // base-8
int i = 0x4B;  // base-16
int i = 0x4b;  // base-16
```

*Floating point literals express numbers with decimals and/or exponents. The symbol E or e is used in the exponent.*

```
x = 123.456;  // decimal real number
x = 1.23456e2;  // exponent (means 1.23456x10^2)
e = 1.6E-19;  // exponent (means 1.6x10^{-19})
A = 6.02e23;  // exponent (means 6.02x10^{23})
```
For string literals, we can use single quote for a character, and double quotes for a one or more than one characters.

- 'A'  // a single character
- "B"  // a single character
- "Hello World"  // a set of characters

There are additional character literal called *escape codes*.

<table>
<thead>
<tr>
<th>Escape Code</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>alert (beep)</td>
<td>cout &lt;&lt; &quot;Error !\a&quot;;</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
<td>cout &lt;&lt; &quot;Gazi\nantep&quot;;</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
<td>cout &lt;&lt; x &lt;&lt; '\t' &lt;&lt; y;</td>
</tr>
</tbody>
</table>

In C++, there are only two valid Boolean literals *true* and *false*. These are expressed as values of type *bool*.
Example: Using `enum` and escape codes

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

int main (){  
  int m;
  enum {Jan=1, Feb, Mar, Apr, May,
        Jun, Aug, Sep, Oct, Nov, Dec};

  m = Apr;

  cout << "m =\t" << m << endl;
  cout << "Physics\nEngineer\n";
  cout << "Hello!\a" << endl;
}
```

```
m  =   4
Physics
Engineer
Hello!
```
2.6 Basic Operators

Operators are special symbols that perform operations on the variables and constants.

**Arithmetic Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>13 + 5</td>
<td>18</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
<td>13 − 5</td>
<td>8</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>13 * 5</td>
<td>65</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>13 / 5</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (remainder from x/y)</td>
<td>13 % 5</td>
<td>3</td>
</tr>
</tbody>
</table>

*operator precedence: ( ) ,  * and / ,  + and −*

\[
\begin{align*}
2 - 3 & \times 4 + 2 = -8 \\
2 \times 3 & + 4 - 2 = 8 \\
2 & \times (3 + 4) - 2 = 12 \\
3 & \times 5 / 3 = 5 \\
10 & / 2 \times 3 = 15 \quad \text{evaluate left-to-right} \\
(5 & + (11-5) \times 2) \times 4 + 9 = 77
\end{align*}
\]
Assignment Operator (\(\text{=}\))

```c
int x, y;
x = 2;
y = 5*x;    // y = 10
x = x + 4;  // x = 6
y = y/2;    // y = 5
```

chained assignment

```c
m = (n = 66) + 9; // n = 66 and m = 75
x = y = 22;      // x = 22 and y = 22
```
Compound Assignment Operators (+=, -=, *=, /=, %=)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>add and assign</td>
<td>x += 3</td>
<td>x = x + 3</td>
</tr>
<tr>
<td>-=</td>
<td>subtract and assign</td>
<td>x -= 5</td>
<td>x = x - 5</td>
</tr>
<tr>
<td>*=</td>
<td>multiply and assign</td>
<td>x *= 4</td>
<td>x = x * 4</td>
</tr>
<tr>
<td>/=</td>
<td>divide and assign</td>
<td>x /= 2</td>
<td>x = x / 2</td>
</tr>
<tr>
<td>%=</td>
<td>find reminder and assign</td>
<td>x %= 9</td>
<td>x = x % 9</td>
</tr>
</tbody>
</table>

Note that \( x *= a+b \) expands to \( x = x * (a+b) \)
which is generally not the same as \( x = x * a+b \)
Similarly \( x /= a+b \) expands to \( x = x / (a+b) \)

Q: What are the values of \( s \) and \( p \) after the compound assignment
in the following code?

```
int k = 2, s = 3, p = 4, q = 4;
s += 2 + k - 1;
p *= 2 * k - 1;
q = q * 2 * k - 1;
```
Increase and Decrease by 1 (++, --)

- The following are equivalent in functionality
  \[
  x = x + 1; \\
  x += 1; \\
  x++; \\
  \]

  The ++ operator is used in the name "C++" because it increments the C programming language. That means, C++ has everything that C has and more!

- ++ and -- can be used both as a prefix and as a suffix.
  \[
  a = 5; \\
  b = a++; \quad // \quad a = 6 \text{ and } b = 5 \\
  c = ++a; \quad // \quad a = 7 \text{ and } c = 7 \\
  \]

- **Q:** What are the values of `s` and `p` after the compound assignment in the following code?
  ```
  int k = 2, s = 3, p = 4;
  s += k++; \\
  p *= ++k;
  ```
Integer Division

```c
int i, j, k;
double p, q;
i = 4/2;   // i = 2 now
j = 5/2;   // j = 2 now
p = 5/2;   // p = 2.0 now
p = 5/2.0; // p = 2.5 now
q = i + p; // q = 2.0 + 2.5 = 4.5 now
k = 25.0/2; // k = 12 now
```

Type Casting

```c
int i; float f; double d;
i = int(7.25);   // i = 7
d = double(5);   // d = 5.0
f = float(7)/2;  // f = 3.5f
```
2.7 Basic Strings

- A *string* is a series of characters, such as "Hello World!"
- There are three ways to define a string variables:

```c
char *str1 = "This is string1"; // in C/C++
char str2[] = "This is string2"; // in C/C++
string str3 = "This is string3"; // in C++
```

- Strings can do some basic operations.

```c
string s1, s2, s3, s4;
s1 = "centi";
s2 = "meter";
s3 = s1;        // s3 = "centi" now
s4 = s1 + s2;   // s4 = "centimeter" now
s1 += "lmen";  // s1 = "centilmen" now
```
Example: *Using strings*

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

int main (){
    string name;
    cout << "What is your name? ";
    cin >> name;
    cout << "Hello " << name << endl;
}
```

What is your name? Mert
Hello Mert
2.8 Header Files

- The `#include` directive allows to use source code from another file.

- `#include <iostream>` refers to an external file named `iostream`, and tells the preprocessor to take the `iostream` file and insert in the current program.

- The files that are included are called header files.

- The C/C++ standard library traditionally declare their standard functions and constants in header files.

<table>
<thead>
<tr>
<th>C++ Standard Library</th>
<th>Standard Template Library</th>
<th>C Standard Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>ios</td>
<td>vector</td>
<td>casassert</td>
</tr>
<tr>
<td>iostream</td>
<td>deque</td>
<td>cctype</td>
</tr>
<tr>
<td>iomanip</td>
<td>list</td>
<td>cerrno</td>
</tr>
<tr>
<td>fstream</td>
<td>map</td>
<td>climits</td>
</tr>
<tr>
<td>sstream</td>
<td>set</td>
<td>locale</td>
</tr>
<tr>
<td></td>
<td>stack</td>
<td>cmath</td>
</tr>
<tr>
<td></td>
<td>queue</td>
<td>csetjmp</td>
</tr>
<tr>
<td></td>
<td>bitset</td>
<td>csignal</td>
</tr>
<tr>
<td></td>
<td>algorithm</td>
<td>cstdarg</td>
</tr>
<tr>
<td></td>
<td>functional</td>
<td>cstddef</td>
</tr>
<tr>
<td></td>
<td>iterator</td>
<td>cstdio</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cstdint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cstdlib</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cstring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ctime</td>
</tr>
</tbody>
</table>
# 2.9 Basic Intrinsic Functions

An *intrinsic* or a *library* function is a function provided by C++ language. For example, the `cmath` library contains mathematical functions/constants:

<table>
<thead>
<tr>
<th>Function Declaration</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>double fabs(double x);</td>
<td>absolute value of real number, $</td>
<td>fabs(-4.0)</td>
<td>4.0</td>
</tr>
<tr>
<td>int floor(double x);</td>
<td>round down to an integer</td>
<td>floor(-2.7)</td>
<td>-3</td>
</tr>
<tr>
<td>int ceil(double x);</td>
<td>round up to an integer</td>
<td>ceil(-2.7)</td>
<td>-2</td>
</tr>
<tr>
<td>double sqrt(double x);</td>
<td>square root of x</td>
<td>sqrt(4.0)</td>
<td>2.0</td>
</tr>
<tr>
<td>double pow(double x, double y);</td>
<td>the value of $x^y$</td>
<td>pow(2., 3.)</td>
<td>8.0</td>
</tr>
<tr>
<td>double exp(double x);</td>
<td>the value of $e^x$</td>
<td>exp(2.0)</td>
<td>7.38906</td>
</tr>
<tr>
<td>double log(double x);</td>
<td>natural logarithm, $\log_e x = \ln x$</td>
<td>log(4.0)</td>
<td>1.386294</td>
</tr>
<tr>
<td>double log10(double x);</td>
<td>base 10 logarithm, $\log_{10} x = \log x$</td>
<td>log10(4.0)</td>
<td>0.602060</td>
</tr>
<tr>
<td>double sin(double x);</td>
<td>sinus of x (x is in radian)</td>
<td>sin(3.14)</td>
<td>0.001593</td>
</tr>
<tr>
<td>double cos(double x);</td>
<td>cosine of x (x is in radian)</td>
<td>cos(3.14)</td>
<td>-0.999999</td>
</tr>
<tr>
<td>double tan(double x);</td>
<td>tangent of x (x is in radian)</td>
<td>tan(3.14)</td>
<td>-0.001593</td>
</tr>
<tr>
<td>double asin(double x);</td>
<td>arc-sine of x in the range [-pi/2, pi/2]</td>
<td>asin(0.5)</td>
<td>0.523599</td>
</tr>
<tr>
<td>double acos(double x);</td>
<td>arc-cosine of x in the range [-pi/2, pi/2]</td>
<td>acos(0.5)</td>
<td>1.047198</td>
</tr>
<tr>
<td>double atan(double x);</td>
<td>arc-tangent of x in the range [-pi/2, pi/2]</td>
<td>atan(0.5)</td>
<td>0.463648</td>
</tr>
<tr>
<td>M_PI</td>
<td>constant pi</td>
<td>myPI = M_PI</td>
<td>3.141592...</td>
</tr>
<tr>
<td>M_E</td>
<td>constant e</td>
<td>$x = M_E$</td>
<td>2.718281...</td>
</tr>
<tr>
<td>Function Declaration</td>
<td>Description</td>
<td>Example</td>
<td>Result</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>int abs(int x);</td>
<td>absolute value of integer number, $</td>
<td>x</td>
<td>$</td>
</tr>
<tr>
<td>int atoi(const char *s);</td>
<td>converts string to integer</td>
<td>atoi(&quot;-1234&quot;)</td>
<td>-1234</td>
</tr>
<tr>
<td>double atof(const char *s);</td>
<td>converts a string to double</td>
<td>atof(&quot;123.54&quot;)</td>
<td>123.54</td>
</tr>
<tr>
<td>void exit(int status);</td>
<td>terminates the calling process &quot;immediately&quot;</td>
<td>exit(1)</td>
<td>-</td>
</tr>
<tr>
<td>int rand(void);</td>
<td>Returns a random integer between 0 and RAND_MAX</td>
<td>rand()</td>
<td>1048513214</td>
</tr>
<tr>
<td>RAND_MAX</td>
<td>The largest number rand() will return</td>
<td>x = RAND_MAX</td>
<td>2147483647</td>
</tr>
</tbody>
</table>
Example: *Using trigonometric functions*

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

int main (){
    double beta;
    cout << "Input an angle in degrees: ";
    cin >> beta;

    // convert from degrees to radians
    beta = beta * M_PI/180.0;

    cout << "\sin(beta) = " << sin(beta) << endl;
    cout << "\cos(beta) = " << cos(beta) << endl;
    cout << "\tan(beta) = " << tan(beta) << endl;
}
```

Input an angle in degrees: 60
sin(beta) = 0.866025
cos(beta) = 0.5
tan(beta) = 1.73205
Example: Using logarithmic functions

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

int main() {
    double x;
    cout << "a value ";
    cin >> x;

    cout << "log(x)    = " << log(x) << endl;
    cout << "log10(x)  = " << log10(x) << endl;
    cout << "exp(x)    = " << exp(x) << endl;
    cout << "pow(x,2.5)= " << pow(x,2.5) << endl;
}
```

a value 1.4
log(x)    = 0.336472
log10(x)  = 0.146128
exp(x)    = 4.0552
pow(x,2.5)= 2.3191
#include <iostream>
using namespace std;

int main (){ 
    const double NA = 6.022e23;
    const double Energy_Per_Reaction = 1.824e-18, MC = 12.0;
    double m, nC, en;

cout << "Input the mass of the carbon in kg: ";
cin >> m;

    // Number of carbon atoms in m kg 
    nC = 1000*m * NA / MC;

    // Total energy released in J 
    en = nC * Energy_Per_Reaction;

cout << "Number of C atoms = " << nC << endl;
cout << "Total energy in J = " << en << endl;

return 0;
}
Problem 1:
Gasoline engines use the heat produced in the combustion of the carbon and hydrogen in gasoline. One of the important sources of energy is the oxidation of carbon to form carbon-dioxide:

\[ \text{C} + \text{O}_2 \rightarrow \text{CO}_2 + 11.4 \text{ eV} \]

where 11.4 eV (=11.4 x 1.6 x 10^{-19} = 1.824x10^{-18} Joule) released comes from the increased binding energy of CO₂ molecule.

Write a program to find total number of carbon atoms and the total energy released when m kg of carbon is oxidized where m is the input from the keyboard.

Avagadro’s number : \( N_A = 6.022 \times 10^{23} \) atoms/mole
Atomic mass Carbon : \( M_C = 12 \) g/mole
Solution:

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

int main(){
    const double NA = 6.022e23;
    const double Energy_Per_Reaction = 1.824e-18, MC = 12.0;
    double m, nC, en;

    cin >> m;

    // Number of carbon atoms in m kg
    nC = 1000*m * NA / MC;

    // Total energy released in J
    en = nC * Energy_Per_Reaction;

    cout << "Input the mass of the carbon in kg: ", m;
    cout << "Number of C atoms = " << nC << endl;
    cout << "Total energy in J = " << en << endl;

    return 0;
}
```

Input the mass of the carbon in kg: 1
Number of C atoms = 5.01833e+25
Total energy in J = 9.15344e+07
Problem 2:
Suppose you want to make a cylinder, whose radius is \( r \) and height is \( h \), to hold 0.01 m\(^3\) of liquid soap. The sheet steel for the can costs 0.2 TL/m\(^2\). It costs 0.1 TL/m to seal circular pieces to the top and bottom of the can and along the seam.

(a) Write a program to input only radius (\( r \)) of the cylinder and output the required height (\( h \)) and the cost (\( C \)) of the cylinder.

(b) What is the cost of the cylinder that is least expensive to make?
Solution:

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

int main(){
    const double V = 0.01; // m3
    const double cs = 0.20; // TL/m2
    const double cl = 0.10; // TL/m
    double r, h, c, S, L;

    cout << "Input radius (m): ";
    cin >> r;

    h = V/(M_PI*r*r); // height
    S = 2*M_PI*r*(r+h); // surface area
    L = 4*M_PI*r + h; // total length
    c = cs*S + cl*L; // total cost

    cout << "Height = " << h << " m." << endl;
    cout << "Cost   = " << c << " TL." << endl;
}
```

Input radius (m): 0.1
Height = 0.31831 m.
Cost = 0.21 TL.
HW1:
A light ray falls on an optical glass as shown. An experiment is made to determine the refractive index of the glass. The ray hits at a the distance $x_2$ if we remove the glass and hits the at a distance $x_1$ when we insert the glass.

Write a C++ program to determine the refractive index of the glass if:

- the thickness of the glass is $h = 1$ cm
- the distance $D = 10$ cm
- the distance $x_1 = 12$ cm
- the distance $x_2 = 13$ cm.
HW2:
The period the Earth is $T = 1$ year and distance between Sun and Earth is $r = 150$ million km. Write a program to determine the mass of Sun.

Hint: Use the following relation:

$$m \frac{v^2}{r} = G \frac{mM}{r^2}$$
HW3:
Io is a moon of Jupiter, with a mass $M = 8.90 \times 10^{22}$ kg, and a radius of 1820 km. NASA wants to put a satellite in a very low orbit around Io. Write a program that calculates the orbital speed needed and period of this satellite will have.

Hint: Use the following relation:

$$m \frac{v^2}{r} = G \frac{mM}{r^2}$$

![Diagram of satellite orbiting Io]