EP578 Computing for Physicists

Topic 2

C++ Basis

Department of Engineering Physics

University of Gaziantep

Course web page
`www.gantep.edu.tr/~bingul/ep578`

Sep 2011

1. Introduction

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

Data types, operators, strings and intrinsic functions.

The lecture is taken from Topic 2 of the Basic C++ tutorial

`http://www1.gantep.edu.tr/~cpp/tutorialbasic.php?topic=2`

NOTE THAT

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

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.

Tablo 2.1: *Fundamental data types and their size and ranges in the memory. The numbers are evaluated for a 32-bit system.*

Data Type	Description	Size (byte)	Lower Limit	Upper Limit
char	Character or small integer	1	-128	127
unsigned char			0	255
short int	Short integer	2	-32,768	32,767
unsigned short int			0	65,535
int	integer	4	-2,147,483,648	2,147,483,647
unsigned int			0	4,294,967,295
long int	Long integer	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long int			0	18,446,744,073,709,551,615
float	Single precision floating point number (7 digits)	4	-3.4e +/- 38	+3.4e +/- 38
double	Double precision floating point number (15 digits)	8	-1.7e +/- 308	+1.7e +/- 308
long double	Quad precision floating point number (34 digits) [*]	16	-1.0e +/- 4931	+1.0e +/- 4931

[*] 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.

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
```


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
```

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 _
```

NOTE THAT

C++ is case sensitive.

That is, it distinguishes uppercase letters from lowercase.

So, `Food` and `food` are different identifiers.

4. Variables

- Example declarations

```
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

```
int cake = 122;
```

or

```
int cake(122);
```

Program: *Declaration of variables*

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

int main () {
  short 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;

  return 0;
}
```

```
z = 11
q = 2656
```

Sayfa 9

Program: *Nested and parallel scopes*

```
#include <iostream>
using namespace std;
int k = 11; // this k is global

int main ()
{
 int k = 22; // this k is local in main()
 {
  int k = 33; // this k is local in this block
  cout << "Inside internal block: k = " << k << endl;
 }
 cout << "Inside main(): k = " << k << endl;
 cout << "Global k = " << ::k << endl;

 return 0;
} // end main() block
```

```
Inside internal block: k = 33
Inside main(): k = 22
Global k = 11
```

Sayfa 10

5. Constants

- To help promote safety, variables can be made *constant* with the **`const`** qualifier. Since **`const`** variables cannot be assigned during execution, they must be initialized at the point of declaration.

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

- *Symbolic* constants (that are not memory-consuming) are defined via the **`#define`** preprocessor directive.

```
#define PI 3.1415926
#define MAX 100
#define NEWLINE '\n'
```


- Sometimes we want to assign numerical values to words, e.g. January = 1, February = 2, and so on. C++ allows to define 'enumeration' constants with keyword **`enum`**.

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

is shorthand for

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

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

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

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

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

is equivalent to

```
enum { RED = 1, GREEN = 2, BLUE = 3 };
```

Program: *Using* ***enum*** *and escape codes*

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

int main ()
{
  short 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;

  return 0;
}
```

```
m  =      4
Physics
Engineer
Hello!
```


- 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 aditional character literal called *escape codes* or *escape sequences* which are preceded by a backslash (\).

Escape Code	Description	Example
`\a`	`alert (beep)`	`cout << "Error !\a";`
`\n`	`newline`	`cout << "Gazi\nantep";`
`\t`	`horizontal tab`	`cout << x << '\t' << y;`

- In C++, there are only two valid Boolean literals **`true`** and **`false`**. These are expressed as values of type **`bool`**.

- Integer literal constants can be represented by three different bases: base-10 (decimal), base-8 (octal) and base-16 (hexadecimal)

```
i = 75;   // default base-10
i = 0113; // base-8
i = 0x4B; // base-16
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 = 1234.56e-1; // exponent (means 1234.56x10^-1)
c = 1.6E-19;     // exponent (means 1.6x10^-19)
A = 6.02e23;     // exponent (means 6.02x10^23)
```


6. Basic Operators

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

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	13 + 5	18
-	Subtraction	13 - 5	8
*	Multiplication	13 * 5	65
/	Division	13 / 5	2
%	Modulus (reminder from x/y)	13 % 5	3

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

```
  2 - 3 * 4 + 2 = -8
  2 * 3 + 4 - 2 = 8
2 * (3 + 4) - 2 = 12
    3 * 5 / 3 = 5
   10 / 2 * 3 = 15      evaluate left-to-right
```

Assignment Operator (=)

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

chained assignment

```
m = (n = 66) + 9; // n = 66 and m = 75
x = y = 22;       // x = 22 and y = 22
```


Compound Assignment Operators (+=, -=, *=, /=, %=)

Operator	Description	Example	Equivalent to
+=	add and assign	x += 3	x = x + 3
-=	subtract and assign	x -= 5	x = x - 5
*=	multiply and assign	x *= 4	x = x * 4
/=	divide and assign	x /= 2	x = x / 2
%=	find reminder and and assign	x %= 9	x = x % 9

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++; // a = 6 and b = 5
c = ++a; // a = 7 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

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

Type Casting

```
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
```

7. Basic Strings

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

```
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.

```
string s1, s2, s2, s4, s4;
s1 = "centi";
s2 = "meter";
s3 = s1;          // s3 = "centi" now
s4 = s1 + s2;     // s4 = "centimeter" now
s1 += "lmen";     // s1 = "centilmen" now
```


Program: *Using strings*

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

int main ()
{
  string name;

  cout << "What is your name? ";
  cin >> name;
  cout << "Hello " << name << endl;

  return 0;
}
```

```
What is your name? Mert
Hello Mert
```

8. Header Files

- The **`#include`** directive allows the program 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 2.1: C++ standard library header files

C++ Standard Library	Standard Template Library	C Standard Library
ios	vector	cassert
iostream	deque	cctype
iomanip	list	cerrno
fstream	map	climits
sstream	set	clocale
	stack	cmath
	queue	csetjmp
	bitset	csignal
	algorithm	cstdarg
	functional	cstddef
	iterator	cstdio
		cstdint
		cstdlib
		cstring
		ctime

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:

Some C++ library mathematical functions and constants defined in <cmath>

Function Decleration	Description	Example	Result
`double fabs(double x);`	absolute value of real number, |x|	`fabs(-4.0)`	4.0
`int floor(double x);`	round down to an integer	`floor(-2.7)`	-3
`int ceil(double x);`	round up to an integer	`ceil(-2.7)`	-2
`double sqrt(double x);`	square root of x	`sqrt(4.0)`	2.0
`double pow(double x, double y);`	the value of x^y	`pow(2., 3.)`	8.0
`double exp(double x);`	the value of e^x	`exp(2.0)`	7.38906
`double log(double x);`	natural logarithm, $\log_e x = \ln x$	`log(4.0)`	1.386294
`double log10(double x);`	base 10 logarithm, $\log_{10} x = \log x$	`log10(4.0)`	0.602060
`double sin(double x);`	sinus of x (x is in radian)	`sin(3.14)`	0.001593
`double cos(double x);`	cosine of x (x is in radian)	`cos(3.14)`	-0.999999
`double tan(double x);`	tangent of x (x is in radian)	`tan(3.14)`	-0.001593
`double asin(double x);`	arc-sine of x in the range [-pi/2, pi/2]	`asin(0.5)`	0.523599
`double acos(double x);`	arc-cosine of x in the range [-pi/2, pi/2]	`acos(0.5)`	1.047198
`double atan(double x);`	arc-tangent of x in the range [-pi/2, pi/2]	`atan(0.5)`	0.463648
`M_PI`	constant pi	`myPI = M_PI`	3.141592...
`M_E`	constant e	`x = M_E`	2.718281...

Some standard C++ library functions and constant defined in <cstdlib>

Function Decleration	Description	Example	Result
int abs(int x);	absolute value of integer number, |x|	abs(-4)	4
int atoi(const char *s);	converts string to integer	atoi("-1234")	-1234
double atof(const char *s);	converts a string to double	atof("123.54")	123.54
void exit(int *status*);	terminates the calling process "immediately"	exit(1)	-
int rand(void);	Returns a random integer between 0 and RAND_MAX	rand()	1048513214
RAND_MAX	The largest number rand() will return	x = RAND_MAX	2147483647

Program: *Using trigonometric functions*

```
#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;

  return 0;
}
```

```
Input an angle in degrees: 60
sin(beta) = 0.866025
cos(beta) = 0.5
tan(beta) = 1.73205
```

Program: *Using logarithmic functions*

```
#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;
  return 0;
}
```

```
a value 1.4
log(x)     = 0.336472
log10(x)   = 0.146128
exp(x)     = 4.0552
pow(x,2.5)= 2.3191
```

Class Work

We will solve two problems

Problem 1:

Gasoline engines use the heat produced in the combustion of the carbon and hyrdogen in gasoline. One of the important sources of energy is the oxidation of carbon to form carbon-dioxide:

$$C + O_2 \rightarrow CO_2 + 11.4\ eV$$

where 11.4 eV (=11.4 x 1.6 x 10^{-19} = 1.824x10^{-18} Joule) released comes from the increased binding energy of CO_2 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 = 6.022x10^{23} atoms/mole
Atomic mass Carbon : M_C = 12 g/mole

Solution:

```
Input the mass of the carbon in kg: 1
Number of C atoms  = 5.01833e+25
Total energy in J  = 9.15344e+07
```

```
#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 2:

Write a computer program to compute the range and time of flight of a projectile given the initial speed v_0, and angle of elevation θ.

The simplistic solution are

Range: $R = v_0^2 \sin(2\theta) / g$

Time of flight: $T = v_0^2 \sin^2(\theta) / 2g$

Solution:

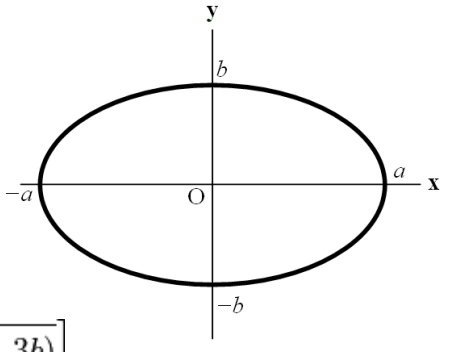
```
#include <iostream>
#include <cmath>
using namespace std;

int main(){
  const double g = 9.81;
  double v0, theta, R, T;

  // get the values
  cout << "Input the speed (in m/s): ";
  cin  >> v0;
  cout << "Input the angle of elevation (in degrees): ";
  cin  >> theta;

  // convert angle into radian
  theta = theta * M_PI/180.0;

  // calculate R and T
  R = v0*v0 * sin(2.0*theta)/g;
  T = pow(v0*sin(theta),2.0) / (2*g);
  cout << "Projectile range = " << R << " m." << endl;
  cout << "Time of flight   = " << T << " s." << endl;
}
```

Homeworks

Solve the following problems. You have to prepare a pdf document and sent it to me until next lecture.
E-mail: bingul[at]gantep.edu.tr (*replace* [at] *with* @)

1. How many data types are in C++?
2. What is the difference between `short int` and `int`?
3. What is the difference between `double` and `float`?
4. How many ways to define constants in C++?

5. The figure shows an ellipse whose semi-major axis is length a, semi-minor axis is length b. Write a C++ program that inputs the values of a and b, and outputs area (A) and circumference (C) of the ellipse.

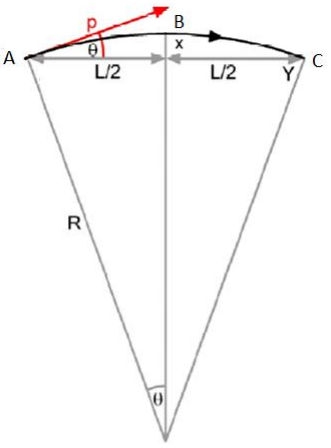


Here

$$A = \pi ab$$

$$C \approx \pi \left[3(a+b) - \sqrt{(3a+b)(a+3b)} \right]$$

6. Consider that a radioactive source emits alpha-particles with momentum of *p*.
They enter (at point A) to a region containing uniform magnetic field *B* = 1.5 T (out of page) as shown in figure. The particles follows the arc ABC. By measuring sagita (x and distance *L*) one can calculate the radius of curvature of the arc and, therefore, the momentum of the particles.



Write a C++ program to input *x* and *L* and output the radius of curavature (*R*) in cm and momentum (*p*) in MeV/c of the alpha-particles. Use relativistic kinematics.
Typical order of *x* (and of *L*) is cm.

Sayfa 35

7. In a Compton Scattering experiment, X-rays of wavelength λ = 10 pm are scattered from a target. Write a program to find the wavelength in pm of the x-rays scattered through the angle θ and maximum the maximum kinetic energy in eV of the recoil electrons where θ is input from the keyboard.

Hint: for θ = 45^o, λ' = 10.7 pm and KE_{max} = 40.8 eV.

Sayfa 36