CISC 1110 (CIS 1.5)
Introduction to Programming Using C++

Spring 2012

Instructor : K. Auyeung

Email Address: kenny@sci.brooklyn.cuny.edu

Course Page: http://www.sci.brooklyn.cuny.edu/~kenny/cisc1110

Class Hours: MW 2:40 – 4:44PM 4428N

Agenda

- Programming Tips
- cin statement
- Control Structures
- Relational Operators
- Logical Operators
- if statements
- Using brace { … }

Programming Tips

- Save & Compile often
- Flash drives
- Basic Program Structure
- Experiment with language, read errors
- Comments

```
/*
        Name
        Assignment
        Class
*/
```

Reading Data from the Keyboard

- **cin** is an object of class istream that represents the standard input stream.
- By default, most systems have their standard input set to the keyboard
- We write characters to it using the insertion operator (ostream::operator **>>**)

cin >> … >> … >> … ;

- Examples

```
cin >> age;

cin >> price >> grade;
```

Sample Code

```
#include <iostream>

using namespace std;


int main()
{

   int age, month;
   float payRate;
   char grade;


   // Entering data using multiple cin statements
   cout << "Enter pay rate ";
   cin >> payRate;

   cout << "Enter age and month of bday ";
   cin >> age >> month;

   cout << endl << endl << "You are " << age << " y.o. and born on " << month << endl << endl;


   // cin statement with multiple parameters
   cout << "Enter all 3 in one line (Age,Month,Pay,grade): ";
   cin >> age >> month >> payRate >> grade;

   cout << endl << endl << "Age " << age << endl;
   cout << "Month " << month << endl;
   cout << "Pay Rate " << payRate << endl;
   cout << "Grade " << grade << endl << endl;

   return 0;
}
```

Control Structures

- Allows programs repeat code or make decisions
- Compound-statement, sometimes called a block, is a group of statements separated by '`;`' and bounded by braces '`{}`'
- `if-else…, switch, while, do-while` loops are some of the control structures in C++
- `if` statements allows a program to make decisions
- A block of code is executed if a condition(s) is met
- There are 3 forms of if statements
  - `if`
  - `if-else,`
  - `if-else-if`

Relational Operators

- Relational operators are used to compare two values
- They can be used to compare numbers or characters
- Comparing characters uses the ASCII Codes
- The relational operators look like operators in math, except for equality:

==	equality	!=	inequality
>	greater than	<	less than
>=	greater than or equal to	<=	less than or equal to

Logical Operators (!, &&, ||)

- ! NOT
- && AND
- || OR

p	q	p && q
T	F	F
T	T	T
F	F	F
F	T	F

p	q	p || q
T	F	T
T	T	T
F	F	F
F	T	T

p	!p
T	F
F	T

BOOLEAN VARIABLES

- there is one more simple, or primitive, data type, and that is called bool
- bool comes from boolean with comes from George Boole, who was an English mathemetician who lived in the first part of the 1800's
- he formalized a type of logic that is now called ``Boolean Logic''
- this logic formalism operates on values that are true or false
- so a bool variable has one of two values: true, which is represented by 1, or false, which is represented by 0

LOGICAL OPERATORS

- boolean expressions combine bool variables and represent things that are either true or false
- in C++, there are three logical operators that are used with bool variables and boolean expressions:

&&	And
||	Or
!	not

- and can be used to put together multiple conditions, for example:

```
bool raining = false;
bool cloudy = true;
bool umbrella = ( raining && cloudy );
cout << "should I take an umbrella? " << umbrella << endl;
umbrella = ( raining || cloudy );
cout << "should I take an umbrella? " << umbrella << endl;
```

- the first question is answered 0 or ``false''
- the second question is answered 1 or ``true''
- explanation follows on the next page when we introduce truth table

TRUTH TABLE

- And

a	b	a && b
True True False False	True False True False	True False False False

- Or

a	b	a || b
True True False False	True False True False	True True True False

- Not

a	**!a**
False **True**	**True** **False**

`if` Statement

```
if (condition) {

   // execute this block

}
```

- example

```
if ( x > y ) {
    cout << "x is bigger than y\n";
}
```

- see *ifsample1.cpp*

`if-else` Statement

```
if (condition)  {
    // execute this block when the condition is true
}
else  {
    // execute this block when the condition is false
}
```

- example

```
if ( x > y ) {
   cout << "x is larger‟ << endl;
}
else {
   cout << "y is larger (or the same as x)‟ << endl;
}
```

- see *ifelsesample1.cpp*

`if-else-if` Statement

```
if (condition1)  {
    // executed when condition1 is true
}
else if (condition2)  {
    // executed when condition2 is true
}
else if (condition3)  {
    // executed when condition3 is true
}
.
.
.
else  {
    // executed when all the above are false
    // default statement
}
```

To Brace or Not To Brace…

- Example #1

```
int speed = 30;

if (speed < 55)
     speed = speed + 30;

     speed = speed / 2;

cout << endl << "Your speed is " << speed << endl;
```

- Example #2

```
int speed = 80;

if (speed < 55)
     speed = speed + 30;

     speed = speed / 2;

cout << endl << "Your speed is " << speed << endl;
```