Flow Control in C++

Conditionals & Loops

CS 16: Solving Problems with Computers I
Lecture #4

Ziad Matni
Dept. of Computer Science, UCSB

Announcements

- Homework #3 due today
- Homework #4 is assigned
- Lab #2 is due on Tuesday AT NOON!

- Class is closed to new registration
- No more switching lab times

- Student **Shen, Jinxu** please identify yourself!

Note on Turning In Homework

From Now On…

PLEASE <u>STAPLE</u> YOUR HOMEWORK PAGES ☺

Lecture Outline

- Simple Flow of Control
- IF/ELSE Statements
- Review of Boolean Operators
  - Truth Tables
- Loops
  - While
  - Do-While
  - For
- Notes on Program Style

Notes on the **cmath** Library

- Standard math library in C++
- Contains several useful math functions, like

`cos( ), sin( ), exp( ), log( ), pow( ), sqrt( )`

- To use it, you must import it at the start of your program

  #include <cmath>

- You can find more information on this library at: http://www.cplusplus.com/reference/cmath/

Flow of Control

- Another way to say:
  The order in which statements get executed

- Branch:
  (verb) How a program chooses between 2 alternatives
  – Usual way is by using an *if-else* statement

```
if (Boolean expression)
   true statement
else
   false statement
```

Implementing IF/ELSE Statements in C++

- As simple as:

```
if (income > 30000)
{
    taxes_owed = 0.30 * 30000;
}
else
{
    taxes_owed = 0.20 * 30000;
}
```

Where's the semicolon??!?

Curly braces are optional if they contain only 1 statement

IF/ELSE in C++

- To do additional things in a branch, use the { } brackets to keep all the statements together

```
if (income > 30000)
{
  taxes_owed = 0.30 * 30000;
  category = “RICH”;
  alert_irs = true;
} // end if part of the statement
else
{
  taxes_owed = 0.20 * 30000;
  category = “POOR”;
  alert_irs = false;
} // end else part of the statement
```

Groups of statements (sometimes called a **block**) kept together with **{ … }**

Review of Boolean Expressions: *AND, OR, NOT*

- Since flow control statements depend on Booleans, let's review some related expressions:

AND operator (&&)

- (expression 1) && (expression 2)
- True if *both* expressions are true

OR operator (||)

Note: no space between each '|' character!

- (expression 1) || (expression 2)
- True if *either* expression is true

NOT operator (!)

- !(expression)
- False, if the expression is true (and vice versa)

Truth Tables for Boolean Operations

AND

X	Y	X && Y
F	F	F
F	T	F
T	F	F
T	T	T

OR

X	Y	X || Y
F	F	F
F	T	T
T	F	T
T	T	T

NOT

X	! X
F	T
T	F

IMPORTANT NOTES:

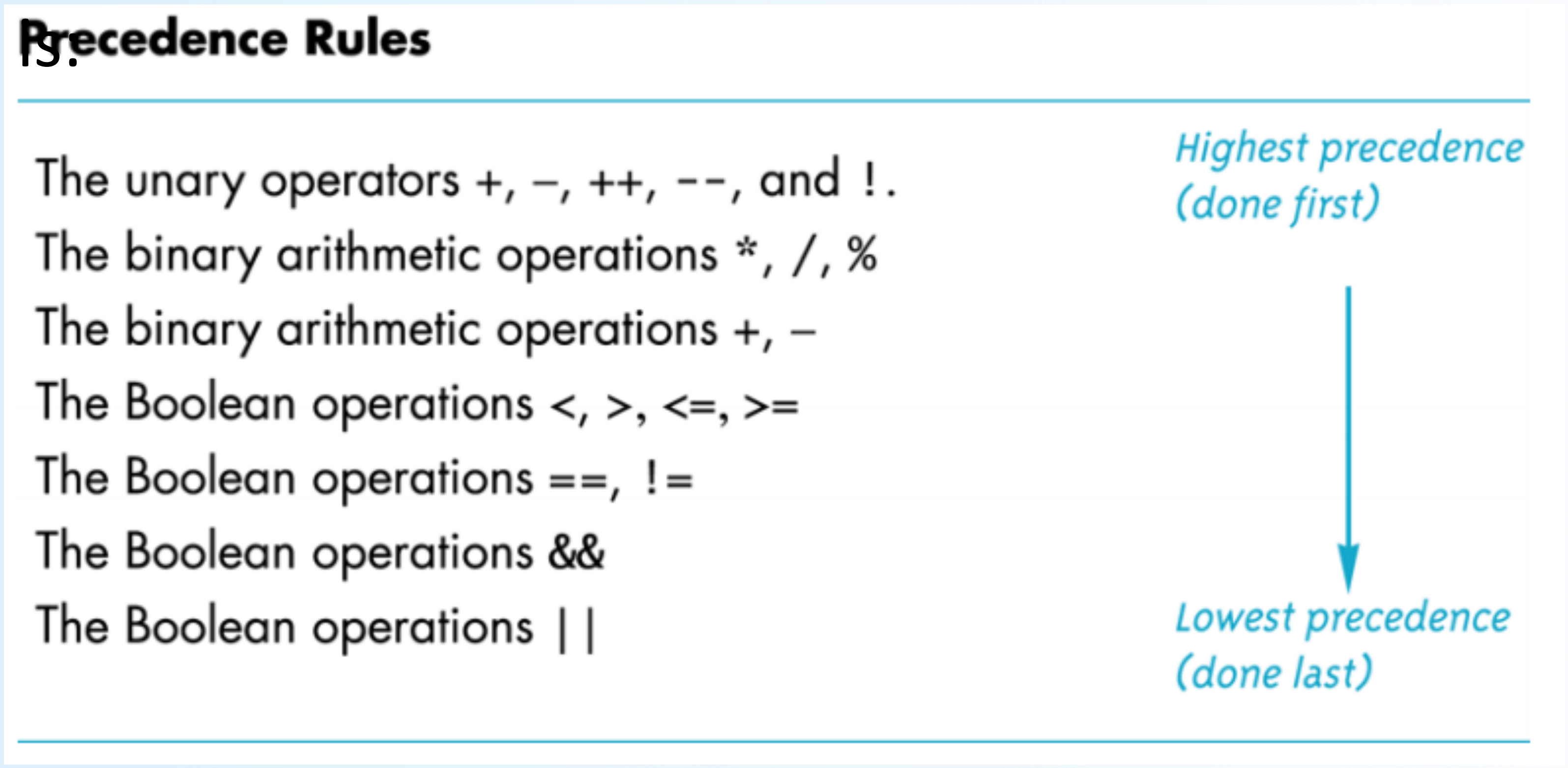
1. AND and OR are **not opposites** of each other!!
2. AND: if just one condition is false, then the outcome is false
3. OR: if at least one condition is true, then the outcome is true
4. AND and OR are **commutative, but not when mixed** (so, order matters)

    X && Y = Y && X

    X && (Y || Z) is NOT = (X && Y) || Z

Precedence Rules on Operations in C++

- If parenthesis are omitted from Boolean expressions, the default precedence of operations is:



Precedence Rules

The unary operators +, −, ++, −−, and !.	*Highest precedence (done first)*
The binary arithmetic operations *, /, %	
The binary arithmetic operations +, −	
The Boolean operations <, >, <=, >=	
The Boolean operations ==, !=	
The Boolean operations &&	
The Boolean operations ||	*Lowest precedence (done last)*

Examples of IF Statements

```
if ( (x >= 3) && ( x < 6) )
   y = 10;
```

- The variable **y** will be assigned the number 10 only if the variable **x** is equal to 3, 4, or 5

```
if !(x > 5)
   y = 10;
```

- The variable **y** will be assigned the number 10 if the variable **x** is NOT larger than 5 (i.e. if **x** is 4 or smaller)
  - DESIGN TIP: Unless you really have to, avoid the NOT logic operator when designing conditional statements

Beware: = vs ==

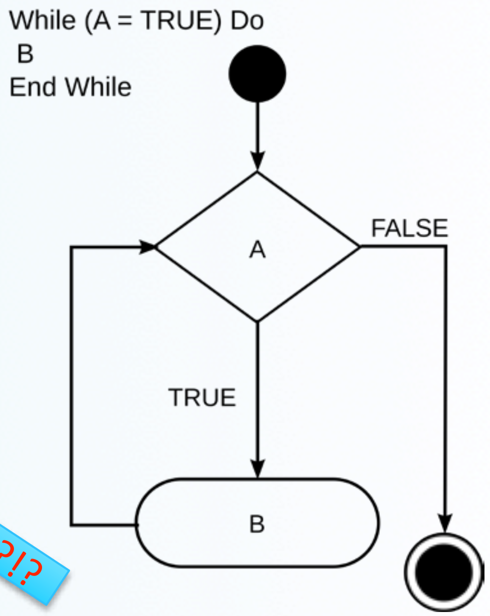
- ' = ' is the **assignment** operator
  - Used to assign values to variables
  - Example: **x = 3;**
- '= = ' is the **equality** operator
  - Used to compare values
  - Example: **if (x == 3)**
- The compiler will actually accept this logical error: **if (x = 3)**
  - ***Why?***
  - It's an error of logic, not of syntax
  - But it stores 3 in **x** instead of comparing x and 3
  - Since the result is 3 (non-zero), the expression is true

Simple Loops1 *while*

- We use loops when an action must be repeated
- C++ includes several ways to create loops
  - while, for, do…while, etc…
- The **while loop** example:

```
int count_down = 3;
while (count_down > 0)
 {
 cout << "Hello ";
 count_down -= 1;
 }
```

Where's the semicolon??!?



- Output is:

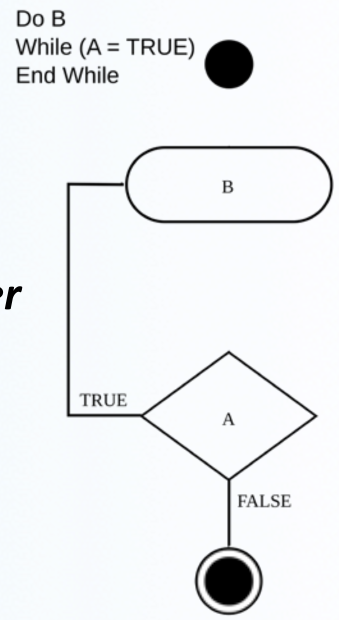
```
Hello Hello Hello
```

Simple Loops2 *do-while*

- The **do-while loop**
- Executes a block of code ***at least once***, and then repeatedly executes the block, or not, depending on a given Boolean condition at the end of the block.
  - So, unlike the while loop, the Boolean expression is checked ***after*** the statements have been executed

```
int flag = 1;
do
{
  cout << "Hello ";
  flag -= 1;
}
while (flag > 0);
```

Why is there a semicolon??!?



- Output is:

```
Hello
```

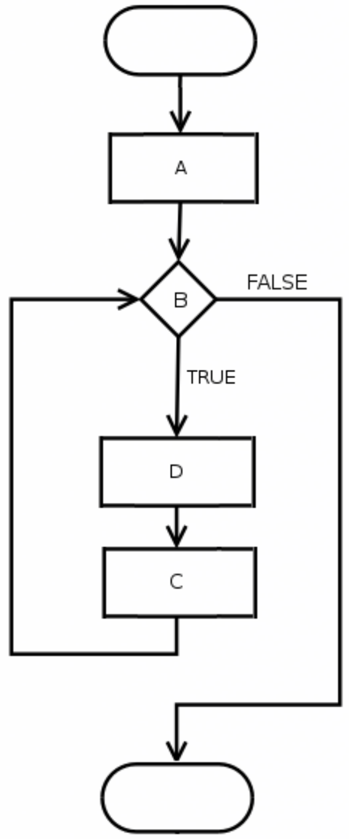
Simple Loops3 *for*

- The **for** loop
  - Similar to a while loop, but presents parameters differently.
- Allows you to initiate a counting variable, a check condition, and a way to increment your counter all in one line.
  - *for (counter declaration; check condition statement; increment rule)*

    {…}

```
for (int count = 2; count < 5; count++)
{
  cout << "Hello ";
}
```

for(A;B;C)
  D;



- Output is:

```
Hello Hello Hello
```

Increments and Decrements by 1

In C++ you can increment-by-1 like this:

```
a++
```

← *more common*

or like this:

```
++a
```

Similarly, you can decrement by:

```
a--   or   --a
```

Infinite Loops

- Loops that never stop – to be avoided!
  - Your program will either "hang" or just keep spewing outputs for ever

- The loop body should contain a line that will eventually cause the Boolean expression to become false

- **Example**: Goal: Print all positive odd numbers less than 6

```
x = 1;
while (x != 6)
{
  cout << x << endl;
  x = x + 2;
}
```

- What simple fix can undo this bad design?

```
while ( x < 6)
```

Sums and Products

- A common task is reading a list of numbers and computing the sum
  - Pseudocode for this task might be:

```
sum = 0;
repeat the following this_many times
        cin >> next;
        sum = sum + next;
end of loop
```

- Let's look at it as a for-loop in C++ …

for-loop for a sum

- The pseudocode from the previous slide is implemented as

```
int sum = 0;
for(int count = 0; count < 10; count++)
   {
        cin >> next;
        sum = sum + next;
   }
```

- Note that "sum" must be initialized prior to the loop body!
   - Why?

for-loop For a Product

- Forming a **product** is very similar to the sum example seen earlier

```
int product = 1;
for(int count = 0; count < 10; count++)    {
        cin >> next;
        product = product * next;
}
```

- Note that "product" must be initialized prior to the loop body
  - Product is initialized to 1, not 0!

Ending a While Loop

- A for-loop is generally the choice when there is **a predetermined number of iterations**
- But what about ending while loops?
- The are 3 common methods:
  - *Ask before iterating*
    - Ask if the user wants to continue before each iteration
  - *List ended with a sentinel value*
    - Using a particular value to signal the end of the list
  - *Running out of input*
    - Using the *eof* function to indicate the end of a file

Ask Before Iterating

- A **while loop** is used here to implement the ask before iterating method to end a loop.

```
sum = 0;
char ans;

cout << "Are there numbers in the list (Y/N)?";
cin >> ans;

while (( ans == 'Y')  || (ans == 'y'))
{
       //statements to read and process the number

   cout << "Are there more numbers(Y/N)? ";
   cin >> ans;
 }
```

List Ended With a Sentinel Value

- A **while loop** is typically used to end a loop using the list ended with a *sentinel* value method

```
cout << "Enter a list of nonnegative integers.\n"
     << "Place a negative integer after the list.\n";
sum = 0;
cin >> number;
while (number > 0)
   {
      //statements to read/process number
      cin >> number;
   }
```

  – Notice that the sentinel value is read, but not processed at the end

Running Out of Input

- The while loop is typically used to implement the running out of input method of ending a loop

```
ifstream infile;          ← We'll cover ifstream objects later in the course
infile.open("data.dat");
while (! infile.eof( ) )
    {
    // read and process items from the file
    // File I/O covered in Chapter 6
    }
        infile.close( );
```

Nested Loops

- The body of a loop may contain any kind of statement, ***including another loop***
  - When loops are nested, all iterations of the inner loop are executed for each iteration of the outer loop
  - *ProTip:* Give serious consideration to making the inner loop a function call to make it easier to read your program

Example of a Nested Loop

```
int students(100)
double grade(0), subtotal(0), grand_total(0);
for (int count = 0; count < students; count++) {
    cout << "Starting with student number: " << count << endl;
    cout <<
          "Enter his/her grades. To move to the next student, enter a negative number.\n"
    cin >> grade;
    while (grade >= 0)      {
          subtotal = subtotal + grade;
          cin >> grade;
    } // end while loop
    cout << "Total grade count for student " << count << "is " << subtotal << endl;
    grand_total = grand_total + subtotal;
    subtotal = 0;
} // end for loop

cout << "Average grades for all students= " << grand_total / students << endl;
```

Notes on Program Style

- The goal is to write a program that is:
  - easier to read
  - easier to correct
  - easier to change
- Items considered a group should look like a group
  - Use the { … } well
  - Indent groups together as they make sense
- Make use of comments
  - //   for a single line comment
  - /* …. */   for multiple line comments
- If a number comes up often in your program (like ϕ = 1.61803), consider declaring it as a constant at the start of the program:
  - `const double PHI = 1.61803;`
  - Constants, unlike variables, cannot be changed by the program
  - Constants can be int, double, char, string, etc…

TO DOs

- Readings
  - **The rest of Chapter 3 in textbook**

- Homework #4
- Lab #2
  - Both due Tuesday

</LECTURE>