Flow Control in C++ 1

CS 16: Solving Problems with Computers I
Lecture #4

Ziad Matni
Dept. of Computer Science, UCSB

Announcements

- **Homework #3 due today**
  - Please take out any staples or paper clips
- **Lab #2 is due on Friday AT NOON!**
  - Use submit.cs

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

- Re: Homeworks
  - Please mark your papers cleanly and clearly, especially if you are using pencil
  - We will post grades for HW1, HW2, Lab1 by end of the week on GauchoSpace

- Re: TA Office hours
  - TA Magzhan Zholbaryssov has changed his office hours to **Tuesday 8am - 10am**

Lecture Outline

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

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
  - Example:

  Program has to calculate taxes owed to the IRS

  Taxes owed are 20% of income, if income < $30,000

  OR they are 25% of income, if income >= $30,000

  How would a program do this calculation?

IF/ELSE Statements

- Recall the general syntax of IF/ELSE statements from earlier courses:

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

If the expression is TRUE, then only the "true statement" gets executed

Implementing IF/ELSE Statements in C++

- As simple as:

Where's the semicolon??!?

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

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

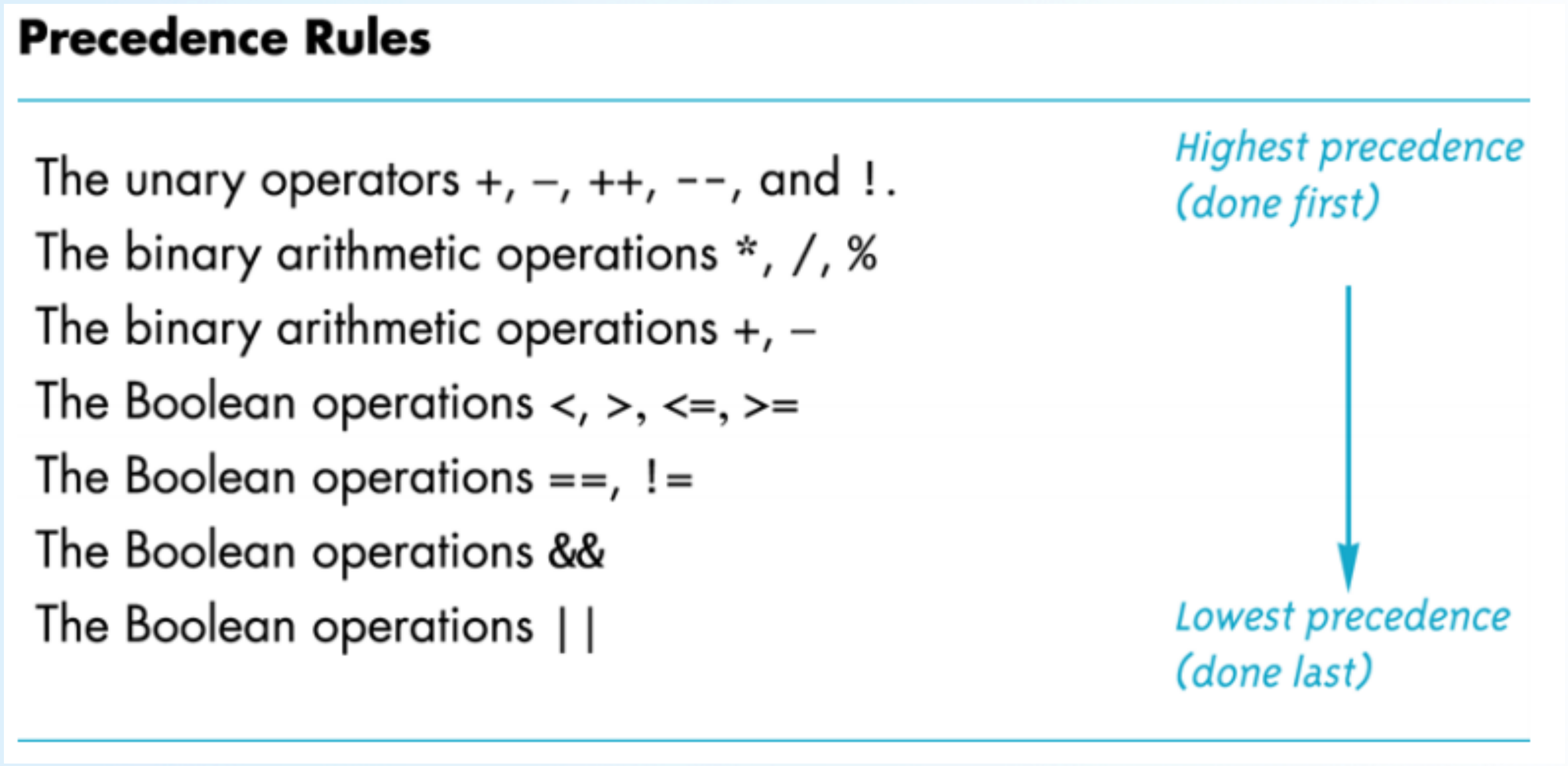
    X && (Y || Z) is NOT = (X && Y) || Z

Order of Operation for Booleans

- It's easiest to use parentheses when expressing Boolean conditions
  - Makes it less confusing for later debug, etc…

- If parenthesis are omitted from Boolean expressions, the default precedence of operations is:
  - Perform ! operations first
  - Perform relational operations such as < next
  - Perform && operations next
  - Perform | | operations last

Precedence Rules on Operations in C++



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 == 3) || ( x < 0) )
   y = 10;
```

- The variable **y** will be assigned the number 10 if the variable **x** is either equal to 3 or if it is a negative number

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

Note: NOT operators can be confusing, so use them sparingly

- 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)

Translating Inequalities from Math into C++

- Be careful translating inequalities to C++
- If the Math expression is "**if $x < y < z$**", then it translates into C++ as:

```
if ( ( x < y ) && ( y < z ) )
```

NOT

```
if ( x < y < z )
```

- How would you translate the following Math expressions?
  - $b^2 \leq 4ac$
  - $x^3 – x^2 + 1 \neq 0$

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)

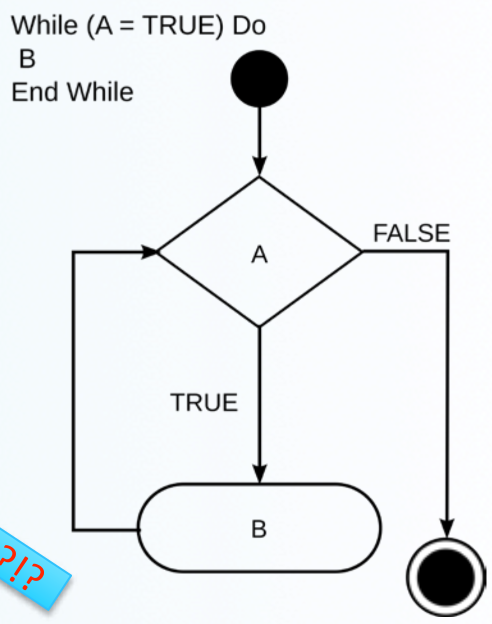
- The compiler will actually accept this logical error: **if (x = 3)**
  - It's an error of logic, not of syntax
  - But it stores 3 in **x** instead of comparing x and 3
  - Since the result is 3 (non-zero), the expression is true

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:

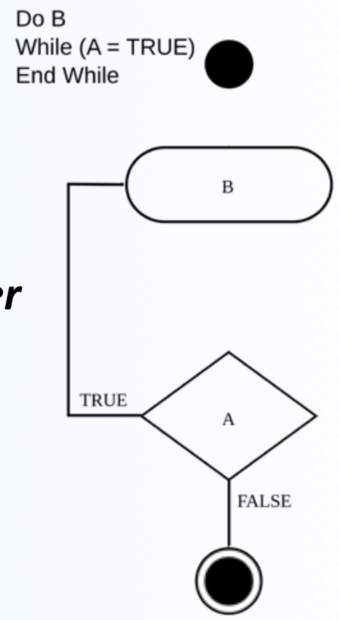
```
Hello Hello Hello
```

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

Why is there a semicolon??!?



- Output is:

```
Hello
```

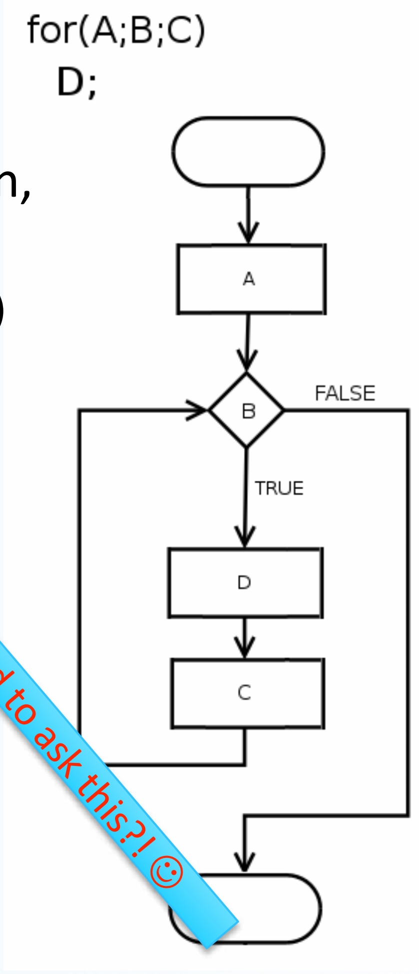
Simple Loops3 *for*

- 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 = 1; count < 5; count++)
{
  cout << "Hello ";
}
```

- Output is:

```
Hello Hello Hello Hello
```



Do I need to ask this?! ☺

Increments and Decrements by 1

- Note that:

  x += 1 is equivalent to
  x++ is equivalent to
  x = x + 1

- Note that:

  x -= 1 is equivalent to
  x-- is equivalent to
  x = x - 1

NOTE:
The ++ and -- operators only work for inc/dec by ***1***.

The other operators can create inc/dec by ***any number***

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)
```

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 $\phi = 1.61803$), 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…

Golden Ratio!

TO DOs

- Readings
  - The rest of Chapter 3, of textbook
- Homework #4
  - Due on Thursday, 10/6 and submit in class
  - Has a programming question that requires planning ahead!
- Lab #2
  - Due Friday, 10/7, at noon

</LECTURE>