Compiling C++ Programs
Flow Control in C++

CS 16: Solving Problems with Computers I
Lecture #3

Ziad Matni
Dept. of Computer Science, UCSB

Lecture Outline

- Compiling Programs in C++
- Input and Output Streams
- Simple Flow of Control
- IF/ELSE Statements
- Loops (While ; Do-While ; For)
- Multiway Branching and the `switch` command

Compile vs. Run Time Errors

Compile Time Errors

- Errors that occur *during* ***compilation*** *of a program*.

Run Time Errors

- Errors that occur *during the* ***execution*** of a program
- Runtime errors indicate bugs in the program (bad design) or unanticipated problems (like running out of memory)
- Examples:
  - Dividing by zero
  - Bad memory calls in the program (bad memory address)
  - Segmentation errors (memory over-flow)

Compiling Programs in C++
(on a UNIX/Linux OS Machine, like those in CSIL)

- Use the built-in compiler program **g++**
- At the prompt, do the following:

```
$ g++ <source code> -o <object code>
```

- Where:
  - Source code = Your C++ program file. Always has the extension .cpp
  - Object code = What the output executable file will be called.

- The default version of C++ used by our CSIL g++ is ver. 14.
  - You can force the compiler to use another version with the `-std` option (e.g. `-std=c++11`)

Compiling Programs in C++
(on a UNIX/Linux OS Machine, like those in CSIL)

- For example:

```
$ g++ myProg.cpp -o myProg
```

- Now, to run your program, you run the executable (object file), like this:

```
$ ./myProg
```

  - The "./" tells the Linux OS that the file "myProg" is found in the current directory

- Make sure that you are in the correct directory where your program is!
  - For example, when you first log-in, you will be in your "home" directory
  - If you program lies within a directory called "myPrograms", for example, do this before you compile anything:

```
$ cd myPrograms
```

  - "cd" tells the Linux OS that you want to "change directory" to myPrograms

Can I Access These CSIL Machines *Remotely*?

- Yes! Know your username and password for CSIL before hand
- Let's say they're "**jimbo**" and "**i<3cs16**", respectively

- You will now use that information to remotely log into a CSIL machine
  - These are called **csil-<number between 01 and 42>**
  - For example: **csil-10**

Can I Access These CSIL Machines *Remotely*?

- Using a Mac – it's easy:
  - Open up **Terminal** and do the following:

```
$ ssh jimbo@csil-10.cs.ucsb.edu
```

  - Answer whatever questions come up, like your password, etc…
  - You are now logged into that CSIL machine! Do your work and then remember to exit using

```
$ exit
```

- Using a Windows machine – it's easy, but needs some initial setups:
  - Download **putty** – a free program ---OR--- set up you Windows 10 machine for **bash-shell** and run that instead
    - Google how to do that – it's akin to Mac's Terminal application
  - Do the same exact steps as with the Mac instructions

Inputs and Outputs

Data Streams - Definitions

- **Data stream:** a sequence of data
  - Typically in the form of characters or numbers
- **Input stream:** data for the program to use
  - Typically (standard) originates at the keyboard, or from a file
- **Output stream:** the program's output
  - Destination is typically (standard) the display, or other times to a file

Examples of Use (cout)

```
cout << number_of_bars << " candy bars\n";
```

- This sends two items to the monitor (display):
  - The value of **number_of_bars**
  - The quoted string of characters **" candy bars\n"** (note the starting space)
  - The '**\n**' causes a ***new line*** to be started following the 's' in bars
- A new *insertion operator* **(<<)** must be used **for each item of output**
- Note: do **not** use single quotes for the strings

Escape Sequences

- Tell the compiler to treat certain characters in a special way
  - **** (back-slash) is the escape character
- Example: To create a newline in the output, we use
  - **\n** – as in, `cout << "\n";`
  - An alternative: `cout << endl;`

- Other escape sequences:
  - **\t** horizontal tab character
  - **** backslash character
  - **\”** quote character
  - **\a** audible bell character

For a more complete list of escape sequences in C++, see:

http://en.cppreference.com/w/cpp/language/escape

Formatting Decimal Places

A common requirement when displaying numbers.

EXAMPLE: Consider the following statements:

```
double price = 78.5;
cout << "The price is $" << price << endl;
```

- Do you want to print it out as:

```
The price is $78.5
The price is $78.50
The price is $7.850000e01
```

Likely, you want the 2nd option
You have to **DEFINE that format** ahead of time

Note: **endl** is the same as **"\n"**
and is part of <iostream>

Formatting Decimal Places with **cout**

- To specify fixed point notation, use:

```
cout.setf(ios::fixed)
```

- To specify that the decimal point will always be shown

```
cout.setf(ios::showpoint)
```

- To specify that ***n*** decimal places will always be shown

```
cout.precision(n)
```

--- where n can be 1, 2, 3, etc…

EXAMPLE:

```
double price = 78.5;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout << "The price is " << price << endl;
```

You usually only need to do this ONCE in a program, unless you decide to change the format later on

Inputs via **cin**

- **cin** is an input stream bringing data from the keyboard
- The *extraction operator* (**>>**) removes data to be used and can be used more than once

EXAMPLE:

```
cout << "Enter the number of bars in a package\n";
cout << " and the weight in ounces of one bar.\n";
cin >> number_of_bars;
cin >> one_weight;
```

Alternative: `cin >> number_of_bars >> one_weight;`

- This code prompts the user to enter data then reads 2 data items from **cin**
- The 1st value read is stored in *number_of_bars*, the 2nd value in *one_weight*
- Data entry can be separated by spaces OR by return key when entered

Entering Multiple Data Input Items

- Multiple data items are ***best*** separated by spaces
- Data is not read until the **Enter** key is pressed
  - This allows user to make corrections

When you see this, it means I'm demonstrating code in class AND will have it available on the class website!

EXAMPLE:

```
cin >> v1 >> v2 >> v3;
```

*Requires **3** whitespace separated values*

A whitespace = space OR tab OR return

- So, user might type:

```
34 45 12<enter key>   or   34<enter key>45<enter key>12<enter key> etc…
```

Space chars.

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 **{ … }**

Boolean Statements in IF/ELSE

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

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

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

- The variable **y** will be assigned 10 if **x** is NOT larger than 5 (i.e. if **x** is 4 or smaller)
  - DESIGN PRO-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) y = 0;**

- The compiler *will actually accept this logical error*: **if (x = 3) y = 0;**
  - ***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, so y becomes 0

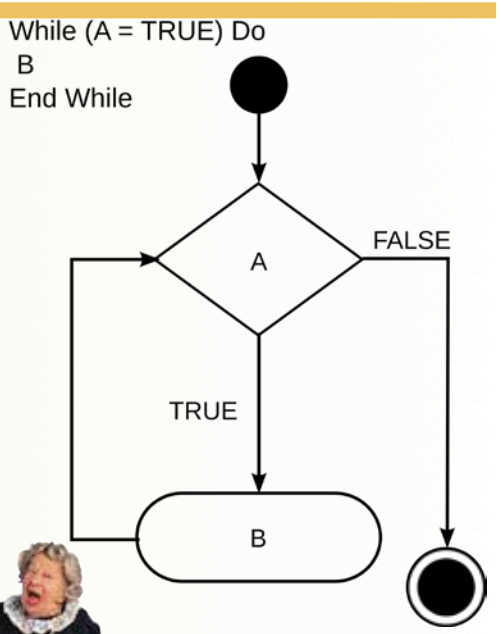
Simple Loops 1: *while*

- 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;
}
```

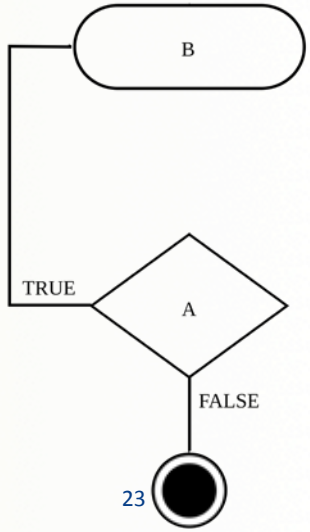
Output is:
Hello Hello Hello



Simple Loops 2: *do-while*

- Executes a block of code ***at least once***, and then repeatedly executes the block 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

Do B
While (A = TRUE)
End While




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

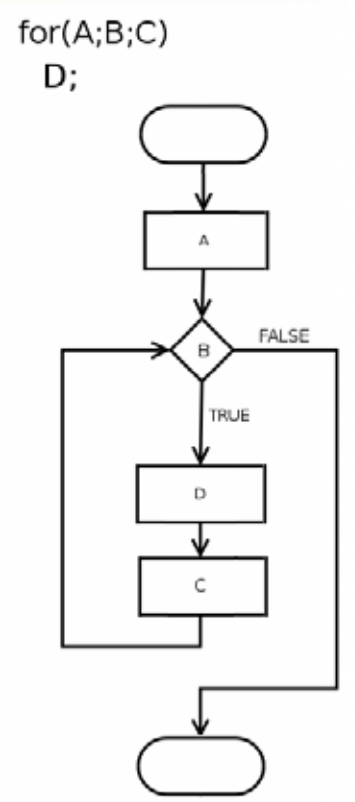
Why **is there** a semicolon here??!?

Output is:
Hello

Simple Loops 3: *for*

- 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) {…}



Output is:
Hello Hello Hello

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

Increments and Decrements by 1

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

more common → `a++`

or like this:

`++a`

Similarly, you can decrement by:

`a--` or `--a`

Some Cool Uses of **x++**

- In a while loop, you always need to increment a counter var.

Example:

```
int max = 0;
while (max < 4)
{
   cout << "hi" << endl;
   max++;
}
```

What will this print out?

Some Cool Uses of **x++**

- You can make a slight change and save a line of code!

Example:

```
int max = 0;
while (max++ < 4)
{
   cout << “hi” << endl;
}
```

When to use **x++** vs **++x**

- **x++** will assess **x** *then* increment it
- **++x** will increment **x** first, *then* assess it

- 95% of the time, you will use the first one

- In *while* statements, it **makes** a difference
- In *for* statements, it **won't make** a difference

Examples

```
for (int c = 0; c < 4; c++)
   cout << "hi" << endl;
```

Prints "hi" 4 times

```
for (int c = 0; c < 4; ++c)
   cout << "hi" << endl;
```

Prints "hi" 4 times

```
int max = 0;
while (max++ < 4)
{
   cout << "hi" << endl;
}
```

Prints "hi" 3 times

```
int max = 0;
while (++max < 4)
{
   cout << "hi" << endl;
}
```

Infinite Loops

- Loops that never stop – **must** 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 (to make the loop to end)
- **Example**: Goal: Print all positive odd numbers less than 6

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

What is the problem with this code?

```
x will never be 6! Infinite Loop!
```

What simple fix can undo this bad design?

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

Using **for-loops** For Sums

- To create an **accumulated sum**, in a for-loop:

```
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?**

Using **for-loops** For Products

- Forming an **accumulated 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
- When you DON'T have a predetermined number of iterations,
  you will want to use **while loops**

The are 3 common methods to END a while loop:

1. *List ended with a sentinel value:* Using a particular value or calculation to signal the end
2. *Ask before iterating:* Ask if the user wants to continue before each iteration
3. *Running out of input:* Using the *eof* function to indicate the end of a file
   (more on this when we discuss file I/Os)

1. List Ended With a Sentinel Value

```
cout  << "Enter a list of positive integers.\n"
      << "Place a negative integer after the list to quit.\n";
sum = 0;
cin >> number;
while (number > 0)
{
   cout << “The double of that is: ” << 2*number << endl;
   cin >> number;
}
```

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

2. Ask Before Iterating

```
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;
  }
```

YOUR TO-DOs

- ❑ Do HW3 for Thursday
- ❑ New Lab on Wednesday!

- ❑ Visit Prof's and TAs' office hours if you need help!

- ❑ Eat all your vegetables

</LECTURE>