Flow Control

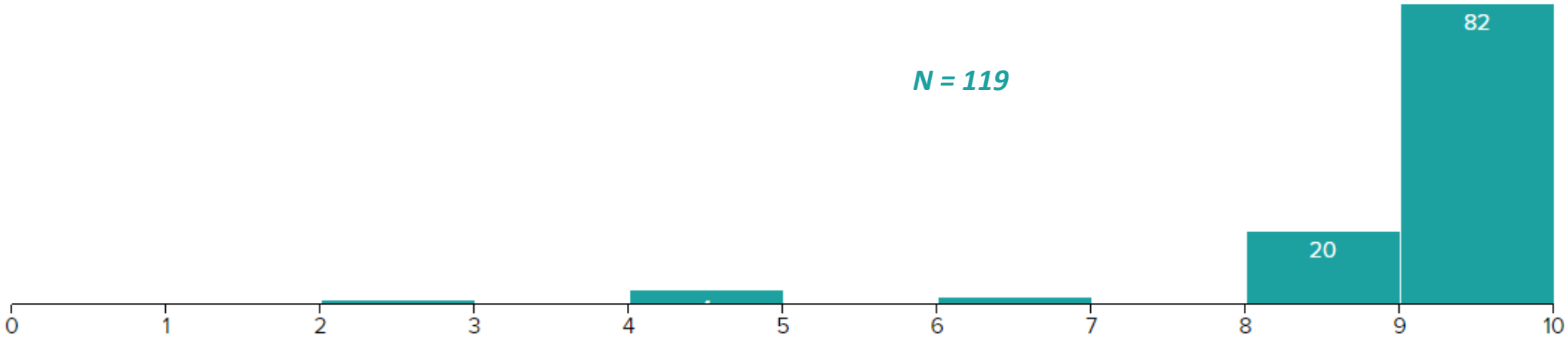
CS 16: Solving Problems with Computers I
Lecture #4

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- New lab (#2) and new homework (#2) are out!
- Homework 1 and Lab 1 were due yesterday
  - How did lab1 go?
- Quiz 1 and Quiz 2
  - How did Quiz 1 go?

Quiz 1



- Mean: **9.27/10**
- Median: **10/10** (not bad!)
- Remember to keep this in perspective: there are 7-8 more quizzes…

Class Exercise

- **What will this print out?**

A. 0
B. 1
C. true
D. false
E. Nothing

```
int a = 44, b = 9;
bool c;
c = (a == b);
cout << c;
```

Class Exercise

- **What will this print out?**

  A. 0

  B. 1

  C. 44

  D. 9

  E. Nothing

```
int a = 44, b = 9;
if (a <= b)
{
    cout << a;
}
```

Lecture Outline

- Re: GitHub
- Compiling Programs in C++
- Input and Output Streams

- Simple Flow of Control
- IF/ELSE Statements
- Loops (While ; Do-While ; For)

Re: GitHub

- Lab2 has 3 exercises & you don't need GitHub to do them
- **BUT!** GitHub will help you to review them with your TAs/ULAs/me
- You don't have to use Unix to use GitHub
  - You can use your Web Browser
  - You just need to have final copies of your lab programs on your home (i.e. *local*) computer
- Lab2 has a section that tells you how to make an account and link it to this class
  - Don't worry if you don't get it right away – you will eventually
  - Use the lab hours and office hours to ask away! ☺

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>
```

*The first **$** indicates a UNIX prompt (you don't type this…)*

- 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`

- 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

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;`
  - You can use either in CS16

- 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:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);

double price = 78.5;
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

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!

Demo!

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.

int ←→ double

Q: What is 9/4 ??!!!
A: It depends!

What happens with variable **z** here?

```
int x(9);
double y(4), z;
z = x / y;
cout << z;
```

This prints out: 2.25

And what happens with variable **p** here?

```
int n(4);
double m(9), p;
p = m / n;
cout << p;
```

This prints out: 2.25

And with variable **c** here?

```
int a(9), b(4);
double c;
c = a / b;
cout << c;
```

This prints out: 2

Flow of Control

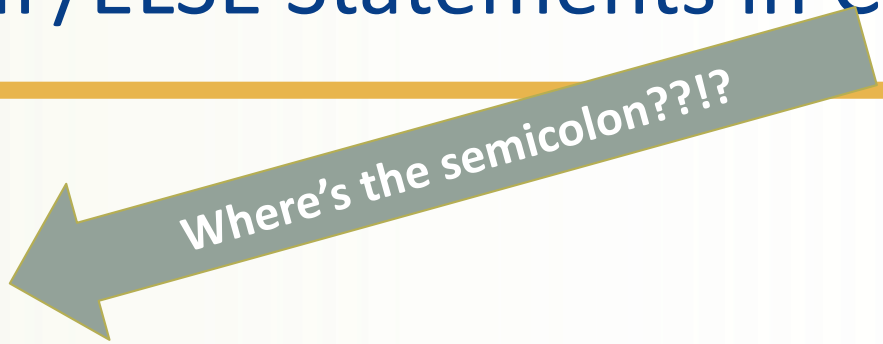
- Another way to say: ***The order in which statements get executed***

- 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;
}
```

Curly Braces {…}

- Curly braces are rules optional if they contain *only 1 statement*
- But we still use them for convention

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

```
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 **{ … }**

Boolean Statements in IF/ELSE

Demo!

```
if ( (x >= 3) && (x < 6) )  //assume ints
   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: **var = 3;**

= = is the **equality** operator
- Used to compare values
- Example: **if (var == 3) res = 10;**

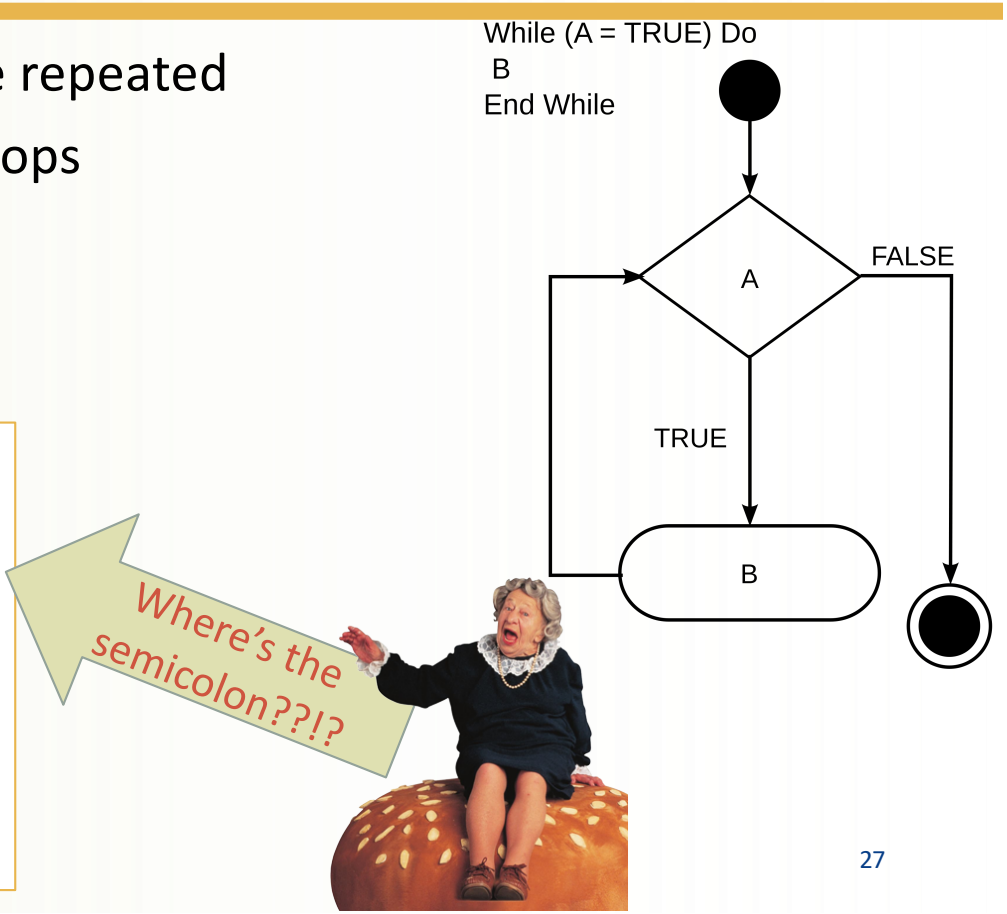
- The compiler *will actually accept this logical error*: **if (var = 3) res = 10;**
  - ***Why?***
  - It's an error of logic, not of syntax
  - But it stores 3 in **var** *instead of* ***comparing*** **var** to 3
  - Since the result is 3 (non-zero), the expression is true, so **res** gets assigned 10

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;
}
```



What is the Output?

Demo!

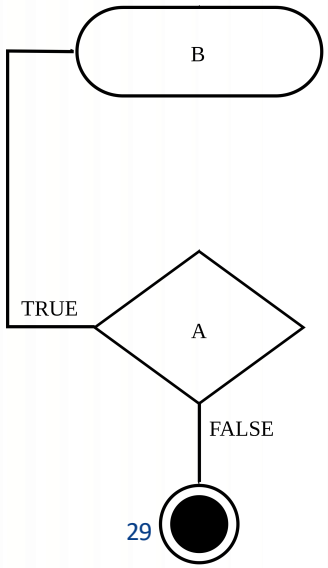
```
int count_down = 3;
while (count_down > 0)
{
   cout << "Hello ";
   count_down -= 1;
}
```

a) Hello Hello Hello
b) Hello Hello
c) Hello
d) None of the above

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??!?

What is the Output?

Demo!

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

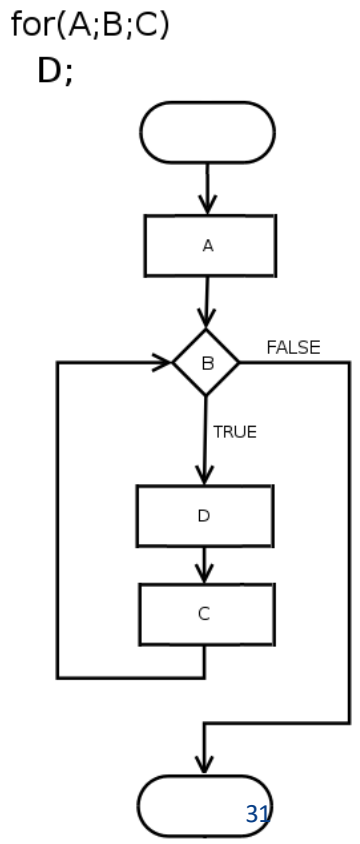
a) Hello Hello Hello
b) Hello Hello
c) Hello
d) None of the above

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) {…}

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



What is the Output?

Demo!

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

a) Hello Hello Hello
b) Hello Hello
c) Hello
d) None of the above

A Note on Styling for CS 16

I will put up Styling Guidelines on Piazza today, here's a summary:

- Always have `using namespace std;`
- Always have `return 0;` at the end of a program
- Always use **consistent** tabbing for indenting blocks
- Re: variable names - pick *either* camel case or snake case & avoid vague names like *xyz* or *var1*.
- Put curly braces { and } on separate lines
  - Allowed alteration: You can put the open brace **{** on the same line as the block starter

    e.g.
    ```
    while (number < 7)
     {
    ```
    or
    ```
    while (number < 7) {
    ```
    is ok
- Always use informative comments
  - Details in the posting…
- More about functions… see posting…

Example

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

int main() {
    int sum = 0, num = 0, count = 0; max_count = 5;
    // Ask the user for a number max_count times
    // Accumulate the sum of all input numbers, and finally print that sum
    while (j < max_count) {
        cout << "Please enter a number here: ";
        cin >> num;
        sum += num;
        j++;
    }
    cout << sum << endl;
    return 0;
}
```

YOUR TO-DOs

- ❑ Start **Lab2** today
- ❑ Do **Homework2**
- ❑ Do **Quiz2** this week (Fri.)

- ❑ Dance like you mean it (when no one's looking)

</LECTURE>