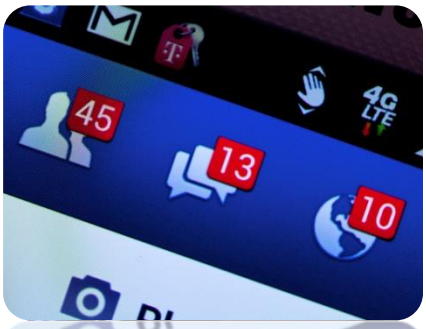
Computational Problem Solving (CS100)

Shafay Shamail, Arif Zaman and

Safee Ullah Chaudhary



Lab 3 Objectives:

- Review the usage of:
  - variables
  - **`cin`** (i.e. user input)
  - Arithmetic operations including **mod** (%)
  - String concatenation

Lab Guidelines

1. Make sure you get your work graded before the lab time ends.
2. You put all your work onto the LMS folder designated for the lab (i.e. “Lab02”) before the time the lab ends.
3. Talking to each other is NOT permitted. If you have a question, ask the lab assistants.
4. If you are a hot-shot C++ expert, you are still not allowed to use any feature of C++ that has not been covered in class or in the lab.
5. The object is not simply to get the job done, but to get it done in the way that is asked for in the lab.

Task 1: The Grid

Write a program that prints the following pattern.

```
+--+--+--+--+
|  |  |  |  |
o..o..o..o..o
|  |  |  |  |
+--+--+--+--+
|  |  |  |  |
+--+--+--+--+
|  |  |  |  |
o..o..o..o..o
|  |  |  |  |
+--+--+--+--+
|  |  |  |  |
o..o..o..o..o
```

Of course, you could simply write 13 cout statements, one for each line in the pattern.
BUT, that would not get you ANY points (see the fifth point of the "Lab Guidelines on the previous page).

We are trying to write a program so that if later you wanted a larger or smaller grid, or if you wanted to make you grid using different characters like **#==#==#==#**, you could do it without much difficulty. This is called writing "reusable" code, and while it may make the job a little bit harder, it saves a lot of time later on.

1. First define three variables: `string plus = "+", bar = "|", oh ="o";`

2. Now define three more string variables, call them **`plus3`**, **`bar3`**, and **`oh3`**, which have a length of three characters. For example **`plus3`** should contain `"+--"`.

3. Then define three more strings called **`plusline`**, **`barline`** and **`ohline`**, that contain an entire line that is to be printed.

4. Finally print the grid using **`cout`** and an appropriate combination of the above variables.

There should be NO quotation marks in your **`cout`** lines!

CHECKPOINT 1

Show your program and result to the TA.

Task 2: Debugging!

The following program is supposed to ask you for your weight and height.
It is supposed to compute a number called a Body-Mass-Index, based on the formula: $BMI = \frac{weight}{height^2}$
Where the weight is in kilograms and the height is in meters. The BMI gives some indication of the percentage of body fat (which is correlated to increased risk of Diabetes and Heart Disease)[1].

The program is given below. (You can cut and paste it from this file into CODEBLOCKS).

```
//BODY MASS INDEX CALCULATOR

#include <iostream>

using namespace std;

int main()
{     //declaring variables
      double weight = 0;
      double height = 0;
      int heightsqr;

      //prompting user for input
      cout << "Please enter your height in centimeters: ";
      cin  << weight;

      Cout << "Please enter your weight in KGs: ";
      cin  << weight;

      // squaring the height
      Heightsqr = height*height;

      weight = 90;

      //calculating the BMI
      BMI = weight/heightsqr;

      //printing the BMI using cout
      cout <<
}
```

The program has a number of errors. Even if you notice them, don't fix all of them, because we are trying to teach you some concepts here. Only fix the errors that you are asked to fix, and no more.

[1] DO NOT waste your time reading this footnote now. Later, if you are interested, you may want to read more about BMI and other indices at https://en.wikipedia.org/wiki/Body_mass_index. BMI is not a very good indicator. Other more reliable indices are use and exponent of 2.5 or 3 instead of the square of the height. Yet other indices don't use weight at all. These are hip circumference/height, or waist circumference/height. You can find references to these measures as well as what are healthy ranges of these ratios in the Wikipedia article referenced above.

Try to compile the program as it is. There will be errors detected by the compiler.
These errors are called "**compile time errors,**" because they are noticed by the compiler at the time that it is trying to compile your program.

Fix **only** these **compile time** errors appropriately.

Now when you try to run the program (after entering the numbers for height and weight) you should get an error.
This error is called a "**run time error**", because the error is noticed by the computer when the program is running. It could not have been noticed just by looking at the program, because it depends on the values of the variables in the program, when the program is running.

Fix **only** these **run time** errors appropriately.

Now, by hand select a few values (at least 3) of height and weight, where you can easily compute the BMI. These do not have to be "reasonable" values; just use any that are easy to compute. Try to consider all possibilities.

For example if height is 300 cm (which is equal to 3m) and weight is 90 kg, then the BMI should be $90/3^2 = 10$.

Use a few values where you can, by hand, compute what the answer should be. Check to see that the program works as described.

You should notice that there is still a problem. This kind of an error is one that is not noticed by the computer at all. There is a problem with your programming. Sometimes it is simply a typing mistake, while at other times it may be a mistake in the logic of your solution.

Software Testing

*Each set of input values, along with the expected result is called a **test case**. For any large software, usually thousands of test cases are written. After any major change in software, each of test cases is checked (automatically) to see that it still outputs the correct results. This is useful to see if any new addition does not break some other part of the program.*

Fix the **remaining errors** appropriately.

What would happen if you tried to find the BMI of a mouse, with height 2.5 cm and weight 0.01 kg?

Your final output should look contain units, so its format should look like:

```
BMI = 23.5 kg/(m^2)
```

Note: this is just an example. The number in your result would be dependent on the values you input .

CHECKPOINT 2

STOP

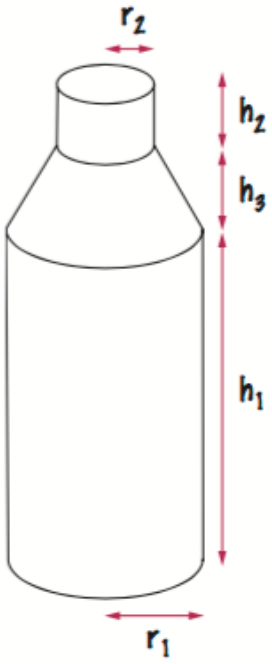
Show your program and results to the TA.

You now know how to debug faulty code. Good job!

Task 3: Bottle-Neck

a) The shape of a bottle is approximated by two cylinders of radius r_1 and r_2 and heights h_1 and h_2, joined by a cone section of height h_3.
The formula for the total volume of the bottle is,

$$V = \pi\left(h_1 {r_1}^2 + \frac{h_3}{3}({r_1}^2 + r_1 r_2 + {r_2}^2) + h_2 {r_2}^2\right)$$



Write a C++ program which computes the volume of the bottle with

$r_1 = 7$, $r_2 = 2$,
$h_1 = 15$, $h_2 = 3$ and $h_3 = 4$,
using a value of $\pi = 3.1416$

b) Now repeat the above task but instead of using fixed values of radius, ask the user to input values for the five variables to compute the volume of the bottle.

Hint: use cin, as in the previous program.

CHECKPOINT 3

STOP

Show your program and result to the TA.

Task 4: Counting!

I am sure that you didn't expect that after having made it to LUMS, "the premier University …", you would be sitting in class learning how to count! Actually, counting is a rather advanced topic that you may learn more about in your Discrete Mathematics class next year (if you take it). Later there are further courses and even graduate course in advanced counting!

But here, let us start with simple counting. First I want to write a program that simply counts to 100. The first two lines should look something like:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
```

1. To do this, let us use a variable called **`count`** that starts off as 0.
2. First we print the variable using a cout statement.
3. Then we "increment" the **`count`** variable, so that it now is one more than whatever value it previously had.
4. Then we simply copy these two statements (2 and 3) and paste them a 100 more times.

To begin with just copy paste three or four times, until things start to look right. Once it seems like it is working, print all the numbers from 0 to 100.

> ### *Debugging*
>
> *When we are first writing a program we are in the "debugging" phase. At this time, it is useful to keep things simple. Start with small and easy inputs and sizes.*
>
> *Once you have the logic of the program working, then it is time to go into the "production" phase, where you can solve the real problem that you wanted to do.*

Now we want to do something fancier. We want to count by sevens. The output should look like:

```
0 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2
```

How can we do this easily? The important hint is to count just as before, but to output not simple **`count`**, but instead try to output **`count % 7`**. You might wonder why I told you how to do this (I am a cruel teacher, and cruel teachers enjoy making students work hard). The reason is because this won't work, but it is a step in the right direction. With a bit of work you can fix it so that it works.

Remember a % b returns the remainder after dividing a by b. So 4 % 7 is 4, and 14 % 7 is 0.

CHECKPOINT 4

STOP

Show your program and result to the TA.