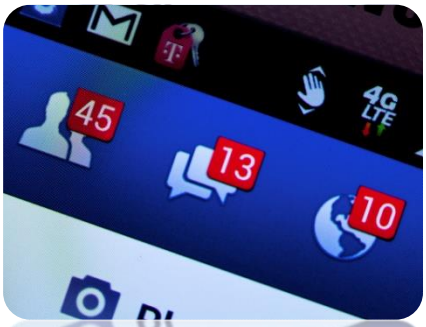
Computational Problem Solving (CS100)

Shafay Shamail, Arif Zaman and

Safee Ullah Chaudhary



Lab 5 Objective:

- Practice if statements (called conditionals)
- Pracitce While loop.

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. “Lab05”) 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 objective is not simply to get the job done, but to get it done in the way that is asked for in the lab.

Fun with Numbers

Given two integers, each in the range 10..99, if there is a digit that appears in both numbers, such as the 2 in 12 and 23, the program should print the number. (Note: division, e.g. n/10, gives the left digit while the % "mod" n%10 gives the right digit.)

```
Enter digit 1:

24

Enter digit 2:

45

The common digit is '4'
```

Cut and Paste Programming for Triangular Numbers

Triangular numbers are defined by

$1=T_1$

$2 \quad 3=T_2$

$4 \quad 5 \quad 6=T_3$

$7 \quad 8 \quad 9 \quad 10=T_4$

$11 \quad 12 \quad 13 \quad 14 \quad 15=T_5$

Here is a program that we could use to list the first few triangular numbers.

```
int n=0, t=0;
// now update n and t, so that they are ready for the next number
      n = 1;
      t = 1;
cout << "T("  << n << ") = " << t << endl;
// now update n and t, so that they are ready for the next number
      n = 2;
      t = 3;
cout << "T("  << n << ") = " << t << endl;
// now update n and t, so that they are ready for the next number
      n = 3;
      t = 6;
cout << "T("  << n << ") = " << t << endl;
// now update n and t, so that they are ready for the next number
      n = 4;
      t = 10;
cout << "T("  << n << ") = " << t << endl;
```

As you can see, there is not much point to this program. We entered in all the values and then got the computer to print them out. What would be nice is if we could replace the four blocks of four lines with

some statements that are exactly the same. That way we could cut-and-paste the same block of four lines four times, or a dozen times. Already the first and last line in every one of the blocks is the same. But the two middle lines are different for each block.

Can you think of a way to rewrite the middle two lines in the block so that every block is identical?

Hint: Try to express the next value of `n` in terms of the previous values.

Try to express the next value of `t` in terms of the previous values.

a) Once you have got the correct output, find the first twelve triangular numbers by just cutting and pasting twelve times.

CS 100 Lab 5

In our previous labs, we have been doing cut-and-paste programming for a while.

Let us introduce (without much explanation for the moment), a new programming concept.

First a review (You can skim the first three points quickly because it is a review):

We already know three kind of statements

1) Declaration statements.
   `int x=1, y=19*19;`

   They start with the name of a type (`int` or `double` or `string` or other types that we haven't yet covered).

   Then there is at least one name of a variable.

   Optionally you can assign a value to that variable.
   Optionally you can continue to declare more variables of the same time using commas to separate them.

   Declaration statements are terminated by a semi-colon;

2) Assignment statements
   `x=x*2+2;`

   They start with the name of a variable, followed by an equal sign, followed by an expression.

   An expression can be
   - a simple number (or string enclosed in quotes `" "`)
   - or it can be the name of a variable
   - or it can be two expressions joined with an operator
   - or it might be an expression enclosed within parentheses `( )`

   Assignment statements are ended with a semi-colon;

3) `if` statements
   `if (x<5) { cout << "small"; } else { cout << "big"; }`

   They start with an `if` followed by a set of parenthesis containing an expression that evaluates to true or false. This is followed by a single statement or a set of one or more statements enclosed in braces `{ }`. A single statement following the `if` should not be another if (to avoid confusion).

   Optionally this can be followed by the word `else` followed by a single statement or a set of statements enclosed in braces `{ }`.

Now we want to introduce a fourth kind of statement. Read this part carefully!

4) `while` statements

They look just like `if` statements, and behave a bit like them.

They start with a `while` followed by a set of parenthesis containing an expression that evaluates to true or false. This is followed by a single statement or a set of one or more statements enclosed in braces `{ }`.

(Compare this to the `if` statement and you will see that the structure of the two statements is identical.)

If the expression in the parenthesis of the `while` statement is false, then the `while` and the `if` statement both do exactly the same thing. They skip over the following statements, doing nothing at all, and go on to the next statement.

If the expression in the parenthesis of the `while` statement is true, then just like the `if` statement, the statement (or group of statements) in front of the `while` will be run. At the end of this, there is a big difference:

- The `if` statement goes on to the next statement
- The `while` returns back and does the same `while` statement once again. This means that once again it checks the condition in the parentheses to see if the expression is true or false. If it is false, it goes on to the next statement. If true, it does the same set of statements again, repeating until the condition becomes false.

Note that if the expression is something like (3<5), then this will never become false.

Even if the expression is something like (x<50) this may never become false. The expectation is that inside the statements of the while, something might happen that changes the value of x, so that x becomes 50 or larger, and so the looping stops. Otherwise the computer will loop forever.

So let us return to our triangular numbers:

```
int n=0, t=0;
while (n < 4)
{
      // now update n and t, so that they are ready for the next number
      n = ...;
      t = ...;
      cout << "T("  << n << ") = " << t << endl;
}
```

You can see that this loop will print the first few triangular numbers without any copy-pasting!

If we now want to print the 12th or the 2000th triangular number all we have to do is to change one number in the program.

a) Find the 2000th triangular number.
b) Make the program stop at the first triangular number that is larger than 30000.

Flying spaghetti code monster

An underpaid programmer rage quit his job but the last task assigned to him was due before he left. However, as a last act of revenge he wrote all the code perfectly but made it very difficult to read. You, the new employee, have been given this piece of code. Your job is to reformat this code to make it more readable. Format it nicely so that the brackets align, and fill the test table so you know how the conditions are planted in the code. Next, use these statements and test table to unravel the code into some simple `if (…) { … } else if ( … ) { … } else if (…) { … } else { … }` statements which are easy to understand.

```
int main()
{     int x,y=7;
      string s;
      cout << "Enter a number x: "; cin >> x;
      cout << "Enter a word s: "; cin >> s;
      while (s != "quit")

      {if(x<5){if(s == "clouds"){cout << "rain" << endl;}
      else{if(x<3){if(s=="rabbit"){cout << "egypt" << endl;}
      else{cout << "White Rabbit" <<endl;}}
      else{if(x>y){cout << "unicorn" << endl;}
      else{cout << "Horse" << endl;}}}}
      else{if (x>(y+3)){if(s=="bat"){cout << "cave" << endl;}
      else{if(s=="cow"){if(x>=20){cout << "cattle" << endl;}
      else{if(x<=10){cout << "Pegasus"<< endl;}
      else{cout << "legend" << endl;}}}
      else{cout << "elephant" << endl;}}}
      else{cout << "mouse" << endl;}}

      cout << "Enter a number x: "; cin >> x;
      cout << "Enter a word s: "; cin >> s;
      }
}
```

a) The table has been partially filled out for your understanding. Use the following format to make an appropriate test table of your code. This table is by no means exhaustive, there are other parts of the code that are tested by different combinations of inputs. Fill the table duly and compare it with the one your TA has.

When x is	and s is	program outputs
X<5	clouds	rain
X<3	rabbit	egypt
x>10	Anything**	elephant
x>=5 && x<11	Anything**	mouse

*** this means that the output does not depend on 's'.*

b) For what values of 'x' and 's' does the program print "unicorn"?

c) Once you are done with this table, work on rewriting this code into easier and more efficient `if (…) { … } else if ( … ) { … } else if (…) { … } else { … }` statements. Remember that the output of you code should match the output of this table.