Lab 2 Objectives:

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

Declaring Variables:

A variable is a storage location paired with an associated identifier(name). It is called variable because you can change the value stored.

A variable has a:

- Name
    - Used to refer to the variable. The program runs exactly the same instructions if we change all occurrences of one name to another.
    - A valid name must be a sequence of
        - Letters
        - Digits
        - _ (The underline character)
        - And it must begin with a letter
- Type
    - Specifies what kind of "values" can be stored in the variable.
    - For example:
        - **`int`** (can store integers less than 1 billion = 1,000,000,000)
        - **`double`** (can store real numbers with about 19 digits of accuracy)
        - **`char`** (can store a single character)
        - **`string`** (can store a sequence of characters of any length)
- Value
    - This is what is stored inside the variable.
    - Whenever you write a variable in any formula, it will be replaced by its value and then the computation will occur.
    - When you store a new value, the old value is lost forever.
    - A variable can only contain one value at a time.
- Storage location and size
    - An **`int`** stores some 9 digits, but a **`double`** keeps 19, so doubles need more space and so have a bigger size.
    - A **`char`** stores one character, but a **`string`** stores as many as needed, so a string does not have a fixed size, and a character has a size that is quite small (even smaller than an **`int`**).
    - The exact location where a variable is kept is quite unimportant to us right now, but matters to the computer, and we may get around to learning a bit about it later.

Declaration Statements

- Declaration statements are of the form
  - *type variable_name*;
  - *type variable_name1*, *variable_name2*, *variable_name3*;

Where type is one of **`int`**, **`double`**, **`char`** or **`string`** (or other types we will learn about later).
And variable names are composed of letters, digits and underlines as described above.
Note that if you need to declare several variables of the same type, you can do that in one line, by using the second form of the declaration statement.

Assignment Statements

- Assignment statements are of the form
  - *Variable_name = expression*;

Where the expression is composed of

- literals
- variable names
- operations like +, -, * and /

A literal is a plain number without a decimal (**`int`**), or with (**`double`**), or one character enclosed in single quotes (**`'a'`**) or a string enclosed in double quotes (**`"Hello"`**).
Every valid expression can be "evaluated", and converted to a single value.

Exact Syntax Matters

Computers expect things to be written exactly as instructed. Any change will usually result in errors. We have learnt about two types of statements up till now:

Declaration with Assignment

You can combine the above two type of statements into one by using to syntax:

- type variable_name = expression;

You can even combine several such statements in a single line, as long as all variables are of the same type, for example:

```
double pi=3.14159265358979846264338327795, radius = 1.414214;
```

Keep in mind two extremely Important etiquettes of programming.

- Descriptive variable names
- Use comments to describe what you are doing

Try to avoid variable names such as a, b, x, I, aa, t etc. because they make no sense. Use proper descriptive variable names e.g. length, tank_length, cup_height etc.

Lab Guidelines:

All labs from today will be graded.

- Make sure you get your work graded before the lab time ends.
- You put all your work onto the LMS folder designated for the lab (i.e. "Lab02") before the time the lab ends.
- Talking to each other is NOT permitted. If you have a question, ask the lab assistants.
- 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.
- 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: Volume of a Sphere

a) Declare three variables:

Variable Name	Type
`PI`	`const double`
`radius`	`Int`
`volume`	`Int`

Write a program that prints the names of your variables and their values in the following format:

```
PI = 3.14;
RADIUS = 30;
```

Now, find the volume of a sphere using the formula

$$V = \frac{4}{3}\pi r^3.$$

With radius 30 and print the result in the following format:

```
VOLUME = 113039
```

Note that the value in the box is not the correct answer.
It is meant just to show what the format of the output should be.

NOTE THAT THE WORD CONST IN FRONT OF A DECLARATION TELLS C++ THAT THIS VARIABLE IS NOT GOING TO EVERY VARY, THROUGHOUT THE PROGRAM. THIS MEANS THAT IF YOU EVER TRY TO PUT SOME VALUE INTO THIS VARIABLE, C++ WILL GIVE YOU AN ERROR. THIS HELPS AGAINST SOMEONE ACCIDENTALLY CHANGING THE VALUE OF PI TO BE SOMETHING OTHER THAN WHAT YOU WOULD EXPECT.

THE USUAL CONVENTION IS TO USE NAMES IN ALL CAPITAL LETTERS FOR CONSTANTS.

b) Now change the value of radius to 1000 and print the result.

CHECKPOINT 1
STOP

Show your program and result to the TA.

Task 2: Making a Parking Lot out of a Garden

A gardener takes care of a garden of length 107 ft and width 81 ft. The administration has decided to convert the garden into a parking lot. They will pave it using square tiles that measure 2.5 ft by 2.5 ft. Compute how many complete tiles can be placed and what will be the area left.

Hint: Besides **+ - *** and **/**, there are many other operators in C++.
One of them is the **%** sign, which is called the mod operator.
If you write **a % b**, when both a and b are ints, the answer is the remainder after dividing a by b.
For example, in C++
13 / 5 has a value of 2
13 % 5 has a value of 3
Which means that 13 divides into 5 exactly 2 times with a remainder of 3.

CHECKPOINT 2
STOP

Show your program and result to the TA.

Task 3 Reusable Paving Program

Now, take both sides of the garden as input from the user using cin and repeat task 2.

Hint: Just like cout writes output from the program to the screen,
cin takes the input from the keyboard to the program.
Try the following simple program

```
int apples = 0;
cin >> apples;
```

This will open the screen, and then wait until you type some number and then press Enter.
If you type a non-number, there could be problems.
Sometimes it is confusing the have the computer just silently waiting. You wonder what went wrong.
So a better way to do this program is to write:

```
int apples = 0;
cout << "How many apples? ";
cin >> apples;
```

It informs the user that the computer is waiting for some input, as well as what input the computer is waiting for.

CHECKPOINT 3

STOP

Show your program and result to the TA.

Task 4: Old is Gold

Now we return to our first lab where we made shapes using repeated cout statements.
For this task, you are supposed to

1. initialize a string variable and set its value to '*'
2. print the string
3. add one more asterisk in the string
4. repeat steps 2 and 3, 99 more times.

Draw a right angle triangle of hundred lines. (Take a look at the **Bonus** at the end of this assignment before you start working on this task)
Your output should look something like this (but much larger):

```
*
**
***
****
*****
```

Task 4.1:

Now repeat task 4 but instead of using '*', take character input from the user.

CHECKPOINT 4
STOP

Show your program and result to the TA.

Bonus:

Since we have not learnt any clever C++ ways to repeat, the only way you can do it now is by copying and pasting the steps 2 and 3, 99 more times. This requires one copy and 99 pastes, so a total of 100 copy/paste operations.

On the other hand, if you are clever, you might notice that after you have copied once and pasted four times, you now have five copies. Instead of continuing blindly for the next 95 pastes, you might stop and copy the entire set of 5. Now every paste will paste 5 copies, so you only have to do the pasting 19 more times. This comes to a total of 1 copy + 4 pastes + 1 copy + 19 pastes = 25 copy/pastes. You saved yourself some work!

A good computer programmer is lazy! Let's find out who is the laziest. The person to outline a scheme to make the smallest number of copy/paste operations to get exactly 100 copies of the lines will get a sweet. If you think you have a good scheme,

- Write it up carefully, with a count of the total number of copy/pastes
- On LMS, click the email tab.
- Next to the "To:" click on Roles.
- Select "Teaching Assistant"
- Send your written explanation in the body of the email
- Finally inform a TA that you have emailed a solution.