These activities must be completed and shown to your lab TA either by the end of this lab, or by the start of your next lab. You may work in groups of up to two people.

This is an introduction to C++ through some simple activities. You should also read the C++ Primer in your textbook and practice as much as possible.

First you will need to login to your CompSci ugrad account (it will look like r2d2 or c3p0a). If you do not have one yet, login with userid getacct (with no password) to obtain it (and if you are reading this before your lab, the webpage https://www.cs.ubc.ca/getacct/ does the same thing).

If you would like to access your lab account from your own computer, you may log in remotely using your ugrad account. If you are on Windows, install XManager. You will be able use Xshell like a UNIX terminal in the lab, and Xftp to transfer files. You can login into the ugrad servers using:

```
ssh <your-ugrad-id>@remote.ugrad.cs.ubc.ca
```

If you are on MacOS or Linux, you have a built-in Terminal where you can login using (the -Y enables graphics for gedit):

```
ssh -Y <your-ugrad-id>@remote.ugrad.cs.ubc.ca
```

Once you have logged in, at the command line enter the following to create a directory to hold your cs221 files, another subdirectory for this lab, and to navigate there (the semi-colon allows several commands on one line):

```
mkdir cs221; cd cs221
mkdir lab1; cd lab1
```

First you will compile and run two programs that demonstrate input and output in C++.

1. Copy and paste the lines from `lab1-samples.cpp.txt`. There are many ways to do this. For this lab just open a web browser and navigate to the course Lab exercises webpage, click on the file link, highlight the lines and copy them to the clipboard. Now back at the command line, enter:

   ```
   gedit q1.cpp
   ```

   Be patient, it might complain about some sub-process failing but it will invoke a fairly intuitive GUI text editor (if you are using a Mac and gedit doesn't work, you may need to download XQuartz). Paste the saved lines (Ctrl-V even on a Mac) from the clipboard into the empty file. If you have been following along with the commands, your current directory should be `~/cs221/lab1` and if it is, then just save and close. If it isn't, Save As instead (i.e. save `q1.cpp` in `~/cs221/lab1`) and after you exit gedit, enter the following to make `~/cs221/lab1` your current directory:

   ```
   cd; cd cs221/lab1
   ```

   These are the lines that should be in q1.cpp:

```
#include <iostream> // needed for cin and cout

using namespace std; // use standard functions without qualifier
```

```
float circleArea(float radius); // forward function declaration

int main() {
  float circle_radius;
  cout << "Enter radius: " << endl;
  cin >> circle_radius;
  cout << "Area is: " << circleArea(circle_radius) << endl;
  return 0;
}

float circleArea(float radius) {
  return 3.14159 * radius * radius; // = pi * r^2
}
```

You can check what is saved in the file by entering:

```
cat q1.cpp
```

To compile your program, enter:

```
g++ -Wall q1.cpp –o q1
```

After you get a clean compile, run your program by entering:

```
./q1
```

That is period-slash-executable. Your lab1 subdirectory is not in the default PATH, so you have to tell the machine where to find your executable. The period says this sub-directory, the slash is just the usual delimiter after subdirectory names, and `q1` is the name of the file you told `g++` to put the output in (that's what the `-o q1` did in the compile command above).

2. File I/O. First create a file called `infile.txt` containing at least 6 lines of text. Then create another file called `q2.cpp` containing the following code. Compile and run as in q1.

```
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

int main() {
  ifstream in("infile.txt");       // input file-stream
  ofstream out("outfile.txt");     // output file-stream
  string ss;
  // getline() puts next line in ss and discards any newline characters
  while (getline(in, ss))
    out << ss << endl; // add each line to the new file, appending endl
  cout << "End of program" << endl;
  return 0;
}
```

Notice that these files are called streams. In q1, `std::cin` used `>>` to extract a value from the input stream and store it in a variable. An `ofstream` (or `std::cout`) uses the `<<` operator to add to the output stream.

Also, a `string` object is used to hold each line of text. You might have heard of a C-string, which is a sequence of `char` that terminates in a `NULL` char (0x00), and might have seen them used in programs:

```
char* err_msg; // in some C or C++ program
```

A `string` object and a C-string are not the same thing at all. The former has constructors and methods, while the latter is actually just the address of one char by convention we interpret this as a C-string by including all characters up until a 0x00 is encountered.

To check that q2 copied the file correctly, compare `input.txt` and `output.txt` side by side by entering:

```
sdiff infile.txt outfile.txt
```

3. Write another program called q3.cpp, and in it declare a global array with 10 integer elements. Compile as in q1 and q2.
   a) Write a function `fillArray()` to fill the elements of the global array with the numbers 1 through 10. Call this function from `main()` and after it returns, print the contents of the array to screen.
   b) Modify the function (and its call) so that it accepts two integers as parameters. The first integer represents the value to be assigned to the first element, the second integer represents the increment between each element. As before, the contents of the array are printed to the screen after `fillArray()` returns to `main()`. Only your work for part (b) needs to be checked by your TA for Q3.

      For example: `fillArray(4,2)` fills the array with the numbers 4, 6, 8, 10, 12, 14, 16, 18, 20, 22. `fillArray(0,5)` will fill it with 0, 5, 10, 15, 20, 25, 30, 35, 40, 45.

      Your `fillArray()` function prototypes should look like:

```
void fillArray(); // for part (a)
void fillArray(int first_value, int increment); // for part (b)
```

And your calls to `fillArray()` in `main()` should look like this:

```
fillArray(); // for part (a)
fillArray(4,2); // for part (b), you can use other numbers if you want
```

4. (from Wikipedia) The Tower of Hanoi or Towers of Hanoi is a mathematical game or puzzle. It consists of three pegs (called A, B and C), and a number, n, of disks of different sizes which can slide onto any peg. The puzzle starts with the disks neatly stacked in order of size on peg A, the smallest at the top, thus making a conical shape.

   The objective of the puzzle is to move the entire stack to peg C, obeying the following rules:

- Only one disk may be moved at a time
- Each move consists of taking the upper disk from one of the pegs and sliding it onto another peg, on top of the other disks that may already be present on that peg
- No disk may be placed on top of a smaller disk

Your task is to write code to print instructions to solve the Towers of Hanoi problem for n disks. **Hint: Use recursion!**

To move n disks from A to C:

a) Recursively move $n-1$ disks from peg A to peg B. This leaves disk n alone on peg A.
b) Move disk n from A to C.
c) Recursively move $n-1$ disks from peg B to peg C so they sit on disk n.

Don't forget to check the base case when using recursion (what is the base case in this problem?). Your program should take a small integer n as input from the command line (called a *command line argument*) and invoke the `moveDisks()` function using code like this:

```
int main(int argc, char* argv[]) {
  if (argc != 2) {
    cout << "Usage: " << argv[0] << " number_of_disks" << endl;
    return -1;
  }
  int n = atoi(argv[1]);
  moveDisks(n, "peg A", "peg B", "peg C");
  return 0;
}

// put your moveDisks() function here
```

It should produce output to solve the problem with n disks. For example, the output of `hanoi 3` may be:

```
Move disk from peg A to peg C
Move disk from peg A to peg B
Move disk from peg C to peg B
Move disk from peg A to peg C
Move disk from peg B to peg A
Move disk from peg B to peg C
Move disk from peg A to peg C
```

5. Write a program that simulates a guessing game. It should randomly generate a number from 1 to 20 and ask the user to input a guess. The game should keep running until the user gets the number correct, or otherwise indicates that they wish to end the game.

   You will need the following functions:

   - `http://www.cplusplus.com/reference/cstdlib/srand/` use `srand(0)` while debugging
   - `http://www.cplusplus.com/reference/cstdlib/rand/`

- http://www.cplusplus.com/reference/cstdlib/atoi/ alpha-to-integer

Be sure to show your work to your TA before you leave, or at the start of the next lab, or you will not receive credit for the lab!

APPENDIX: SOME OTHER USEFUL COMMANDS AND TIPS

cd .. - change directory to the parent of the current directory

cd ../.. - change directory to the parent of the parent of the current directory

cd - change directory to your home directory (the same as where you are when you first log in)

ls - list the files in the current directory

ls -alF - list all info about the files in the current directory, one per line

ls -1 > DIR.txt - that's the number 1 (and -alF has the letter l), this will list the names of the files in the directory, one per line and put the output in DIR.txt (this can be useful when creating the initial version of a README.txt file, for example)

cat DIR.txt - print the contents of DIR.txt to the screen

rm fn.ext - remove (delete) the file named fn.ext from current directory

rm -f *.* - remove all files in current directory (CAREFUL) without confirmation

rmdir dir - remove directory named dir (which MUST be empty)

bye - log off