CS4115 Week07 Lab Exercise

Lab Objective: The objective of this week's lab is to implement two versions of the Fibonacci number generator and convince ourselves that the recursive version, although easy to understand, charges a high price. Here's the lab summary:

❶ Create a directory `~/cs4115/labs/week07` and write two programs called `fib-rec` and `fib-iter` for generating the nth Fibonacci number

❷ By running both programs on various values of n convince yourself that there is the rare time when recursion is *not* the answer

❸ Submit the lab for marking

In Detail

❶ If you need to, please refer to the lectures where we discussed generating Fibonacci numbers and the associated time-complexity.

You will recall that the formula for generating Fibonacci numbers is

$$F_n = F_{n-1} + F_{n-2}, \qquad F_0 = F_1 = 1$$

This formula made for a very simple program, as was presented in the lectures. Your first task is to create a C++ file, `fib-rec.cc` that will be an implementation of the algorithm seen in class. That is, given a number, `n`, the program should generate and print out the `n`th Fibonacci number – nothing more, nothing less – using a recursive function.

Some points to keep in mind:

- Assume that there will be no tricky input and that $0 \leq n \leq 45$;
- Since you will be handing in this code with the `handin` program your entire recursive implementation must be in this single file. Yes, yes, I know, the software engineering types are screaming at me already;
- To run the program to generate the 9th Fibonacci number you should give the command

  ```
  fib-rec 9
  ```

  To do this, the program should be able to take a command-line argument of the values of `n` to run it on. For a hint of the way to read command line arguments see the file `fib-stub.cc` in the class directory `~cs4115/labs/week07`; you should see how you can convert, for example the command-line string "9" into the `int` 9;

- To compile the program, the command will be
  ```
  g++ fib-rec -o fib-rec
  ```

If you are in doubt about the correctness of your program (or the output format) then there is a program, `fib`, in the class directory, `~cs4115/labs/week07`, that will provide the definitive output.

You may also recall from the lectures that our analysis of the recursive program's running time found that it was exponential in performance. Not good. You should now write a version of the program that runs *much* faster. The trick here is to generate Fibonacci numbers iteratively so that you have them already computed when you need them. Call this program `fib-iter` with all of your code put in a file `fib-iter.cc`. Some points:

- This program should be run in the same way as the previous one: to find the nth Fibonacci number you should give the command
  ```
  fib-iter n
  ```
  So you should start with the same stub program as for the last case;
- Compilation:
  ```
  g++ fib-iter.cc -o fib-iter
  ```
- In order to store previously computed values an array of `int`s would be a good choice. I will assure you that for testing your program I will not call a larger value than $n = 45$ so you could dimension your array with something along the lines of
  ```
  int prevs[45];
  ```
  and all would be ok *here*;
- That solution should leave you a little less than fully satisfied, however, because it will only work as long as the user cooperates and doesn't force an array access to go out of bounds ($i > 45$). Can you think of a better solution?

  What is really needed is to *dynamically* create the array when you learn of n. There are two options here:
  - use the `new` operator to request an array of a given size;
  - use `vectors`, the STL implementation of magic arrays; these are arrays that grow automatically so, in effect, you never need to worry about the allocation.
- I will be inspecting your code afterwards and I will award one extra mark if you can write your program so that the storage of any previous values computed by your program is by one of the two methods above
- But this solution is still unsatisfying: we still need to be able to have a memory pile whose size grows linearly with the value of n. Can you think of a strategy that will result in a program that requires just a *constant* amount of extra space? So, no matter waht value of n is given as an argument the same amount of storage will be used.

  I will give two extra marks for a correct solution that uses this approach.

❸ As described above you should write two programs, `fib-rec` and `fib-iter`, both taking a single integer command-line argument. The programs should output simply F_n – nothing fancier. (Consult the given program for exact format.)

I will inspect your programs and, in the `fib-iter` case, I will award 1 extra mark for allocating the memory dynamically, as discussed above; if you can do it using constant memory I will award two extra marks.

The command that you should run is (exactly)

```
~cs4115/progs/handin -m cs4115 -p w07
```

If you have to, please cut-and-paste this command in order to ensure that this is the command you enter.

The deadline for submitting Week07's assignment without penalty is Wednesday Week08 at 18.00. Submissions will be accepted until Monday, Week09 at 17.00 but with a lateness penalty of 10% per day.