CSE 232 Spring 2018

Programming Project 06

Assignment Overview
This assignment is worth 50 points (5.0% of the course grade) and must be completed and turned in before 11:59pm on Monday, March 12th , 2018. That's two weeks because of Spring Break. You have a week to do the project before you go, so we recommend that you get it done before the break. There will be no extensions to the day the project is due (it's due the Monday after you return from Spring Break).

Background, Fibonacci Sequence
You probably have all seen, or at least heard about, the Fibonacci sequence, named after Italian mathematician Leonardo of Pisa, known as Fibonacci. It is a sequence based on a starting set of numbers and a way to calculate subsequent numbers.

The starting sequence for the Fibonacci numbers requires ***two*** *seed* numbers, typically 1 and 1 (though there is some variation if you look at the web page https://en.wikipedia.org/wiki/Fibonacci_number). All subsequent numbers are based on the sum of the ***two*** (which is important for our purposes, see below) previous numbers. It is often written as a recurrence relation: $F_n = F_{n-1} + F_{n-2}$ Thus the Fibonacci sequence is:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 …

Where the seed numbers required to start the sequence are marked in bold. That is, the seed numbers are required to be present before the calculations can begin.

Fibonacci n-step sequences
But why should we restrict ourselves to 2-seeds and the sum of the 2-previous numbers. Why not 3, 6, 9 etc? Sure, why not. We can generalize to **Fibonacci n-step** numbers where **n** represents both the number of seeds required and the number of previous values that are added to create the next number in the sequence. Here are some examples

2	fibonacci	***1 1*** 2 3 5 8 13 21 34 55 89 144 233 377 610 ...
3	tribonacci	***1 1 2*** 4 7 13 24 44 81 149 274 504 927 1705 3136 ...
4	tetranacci	***1 1 2 4*** 8 15 29 56 108 208 401 773 1490 2872 5536 ...
5	pentanacci	***1 1 2 4 8*** 16 31 61 120 236 464 912 1793 3525 6930 ...
6	hexanacci	***1 1 2 4 8 16*** 32 63 125 248 492 976 1936 3840 7617 ...
7	heptanacci	***1 1 2 4 8 16 32*** 64 127 253 504 1004 2000 3984 7936 ...
8	octonacci	***1 1 2 4 8 16 32 64*** 128 255 509 1016 2028 4048 8080 ...
9	nonanacci	***1 1 2 4 8 16 32 64 128*** 256 511 1021 2040 4076 8144 ...
10	decanacci	***1 1 2 4 8 16 32 64 128 256*** 512 1023 2045 4088 8172 ...

The numbers in ***bold italics*** are the seed numbers required to start the sequences. Thus for n=5, we require 5 seeds, each of which is the sum of ***all*** the previous numbers, starting with 1, 1. For n=5, that gives 1, 1, 2, 4, 8 where, for example, 8 is the sum of all the previous numbers (1,1,2,4). After that, the sequence continues by providing each subsequent number based on the sum of the 5 previous numbers. For n=5, the 10^{th} number (index 9) is 236, the sum of 8+16+31+61+120 (the 5^{th} 6^{th}, 7^{th}, 8^{th},9^{th} numbers in the sequence).

Fibonacci number encoding

There is a encoding of numbers called a Fibonacci coding (https://en.wikipedia.org/wiki/Fibonacci_coding), which is a way to encode an integer as a binary number based on the Fibonacci sequence. The Fibonacci sequence (and, as it turns out, all Fibonacci n-step sequences) is **complete**, which means that any integer can be represented as a sum of some combination of the numbers in a Fibonacci (or Fibonacci n-step) sequence. A binary code of a number is therefore a string where the 1's in the string indicate which elements are being used.

Note, we ignore the index 0 number(1), as it is repeated at index 1(1).

For example, for Fibonacci sequence (n=2), we have the following:

index	0	1	2	3	4	5	6	7	8	9	10	11
Fib val	-	1	2	3	5	8	13	21	34	55	89	144

A nstep encoding (for n=2) of the number 100 would be: 0010100001

index	0	1	2	3	4	5	6	7	8	9	10	11
bit	-	0	0	1	0	1	0	0	0	0	1	-

Which represents adding the index:3 (3), index:5 (8) and index:10 (89) numbers to form 100, which it does. Notice that a trailing 0 (for the index 0) is not relevant and is consequently dropped.

It's a backwards encoding!

Notice that the encoding is ***backwards*** from what we normally think of. In the integer 123, leftmost digit, the '1', is the most significant digit. It represents the largest value (1 x 100). In the encoding. In our nstep encoding, the most significant digit is the rightmost bit. It represents the largest value in the string.

Being greedy

How do we figure out for a particular nstep sequence which numbers to use to form our binary string. We do so by using what is called a ***greedy algorithm***. Greedy is a term in computer science that roughly means doing the "best" thing in the present situation. For us, it means selecting the biggest nstep number less than or equal to the number we are encoding. In the n=2 100 example, we pick 89 (index 10) number first because it is the biggest number less than or equal to 100. If we then subtract 89 from 100, we are left with 11. We are again greedy and pick the biggest number less than or equal to 11. That's 8 (index 5), which we subtract leaving 3. We then pick the index 3 number, 3, leaving 0 and we are done. A greedy algorithm for nstep sequences is guaranteed to find an encoding for an integer.

nstep encoding
There are actually some nice properties of a Fibonacci (n=2) encoding, but we don't care. We are going to allow an encoding of any nstep sequence as a binary number.

Your Tasks
Complete the Project 6 by writing code for the following functions. Details of type for the functions can be found in `proj06_functions.h` (provided for you, see details below):

Functions
`string vec_2_str(const vector<long>& v)`

- returns a string that represents the values in the vector
- each element in the string is separated by a ","
    - no "," after the last element

`vector<long> gen_nstep_vector(long limit, long nstep)`

- generates a vector containing the `nstep` sequence
    - the last element of that vector should be the biggest long less than or equal to the `limit` parameter

`string num_to_nstep_coding(long num, long nstep)`

- returns a binary string which represents `num` as the `nstep` sequence.
- uses `gen_nstep_vector`
    - remember, index 0 of the generated vector is not used (it is redundant, as index 1 already represents the value of 1)

```
long nstep_coding_to_num (const string& coding,
                          const vector<long>& nstep_sequence)
```

- converts the `nstep` coded binary string to a long
    - utilizes the vector `nstep_sequence`, which contains the `nstep` sequence used to do the decoding

Deliverables
`proj06/proj06_functions.cpp` -- your completion of the functions described above. Only `proj06/proj06_functions.cpp` is turned in to Mimir.

1. Remember to include your section, the date, project number and comments.
2. Please be sure to use the specified directory and file name.

Manual Grading
In lab05 you worked with Code Style. The TAs will apply the things that you learned in your Code Review lesson on the code you submit. 4 points are reserved (out of 50) for this.

Assignment Notes

1. Mimir allows us to test the functions in `proj06_functions.cpp` individually and we will do that.

2. You can (and should!) write your own `pro06_main.cpp` to test your code, but only `proj06_functions.cpp` needs to be turned into Mimir
3. `proj06_functions.h` is provided in the project06 directory. It will be used in testing. When we do testing, we will use the file we provide. If you change and submit your own version, Mimir will ignore your version and use the file originally provided.