# Programming Project 04

This assignment is worth 40 points (4% of the course grade) and must be **completed and turned in before 11:59 on Monday, February 15th , 2016.**

**Assignment Overview**
This assignment will give you more with functions and introduce the use of the string data type.

**Background**
Remember the Fibonacci sequence? It is a simple, self-generating, infinite sequence of integers. You provide two seed integers called $F_0$ and $F_1$ , traditionally with both seeds initialized to the value 1. Subsequently, each succeeding number is the sum of the previous two:

```
F0  F1   F2 F3  F4 …
1,  1,  2, 3, 5, 8, 13, 21, 34, 55, 89, 144 …
```

Thus $F_2 = F_1 + F_0$, $F_3 = F_2 + F_1$, and in general, $F_n = F_{n-1} + F_{n-2}$

**Fibonacci Strings**
Turns out you can do the same things with strings. You provide two string values as $F_0$ and $F_1$, then you generate the next string in the sequence by doing $F_2 = F_1 + F_0$ and, as before, in general $F_n = F_{n-1} + F_{n-2}$

- Here the + sign indicates concatenation (conveniently)
- Again, you must provide the first two string seeds (which, by the way, could be strings of more than one letter)
- **Order matters here**: $F_1$ concatenated with $F_0$, $F_2$ concatenated with $F_1$, etc.

```
F0  F1  F2  F3    F4      F5          F6
a,  b, ba, bab, babba, babbabab, babbababbabba, …
```

They can get pretty long, so that is something we have to pay attention to. It turns out that Fibo strings have all kinds of interesting theoretical characteristics, but we probably should focus a little more on some string manipulation.

**Project Description / Specification**
We are going to write some functions to generate and manipulate fibo strings. These functions will exercise our ability to work with strings

*Warning*
Again, In this project as in the last we provide *exactly* our function specifications: the function name, its return type, its arguments and each argument's type. The functions will be tested independent of the main program you provide. In fact, we will often provide you with a main program to include in your file (or a separate main program file when we start to develop mult-file programs). If you do not follow the function specifications, these independent tests of your functions will fail. Do not change the function declarations!

**function**: `main`**:**
***We provide this*** as part of the project. Place it in your Geany project and then add your functions above in the same file. Feel free to modify it (change which strategies are used in pairs for example), change

it so you can get better debugging information, change values in the input file etc. We will not grade the main function! However, the main function does follow our function specifications. Don't mess with the function specifications!

**function** `fibo_string`:
Generate a fibo string as described.

- Three arguments
  - The two seed strings f0 and f1
  - The length required of the resulting fibo string
- return a string, a fibo string in the sequence of the required length

- If the string in the function is longer than the required length, return a substring starting at 0 of the required length. You basically need to generate strings in the Fibo sequence until you find some string Fibo string $f_i$ that is as long as (or longer than) provided length. If the right length return it, if not shorten the string to the right length and return it

**function**: `find_substring`:

- two arguments
  - a fibo string
  - a target string we are trying to find in the fibo string
- return is a long, the count of the number of times the target occurred in the fibo string

Could be a count of 0

**function**: `lcs`:

- Two arguments:
  - The two string arguments
- return is string

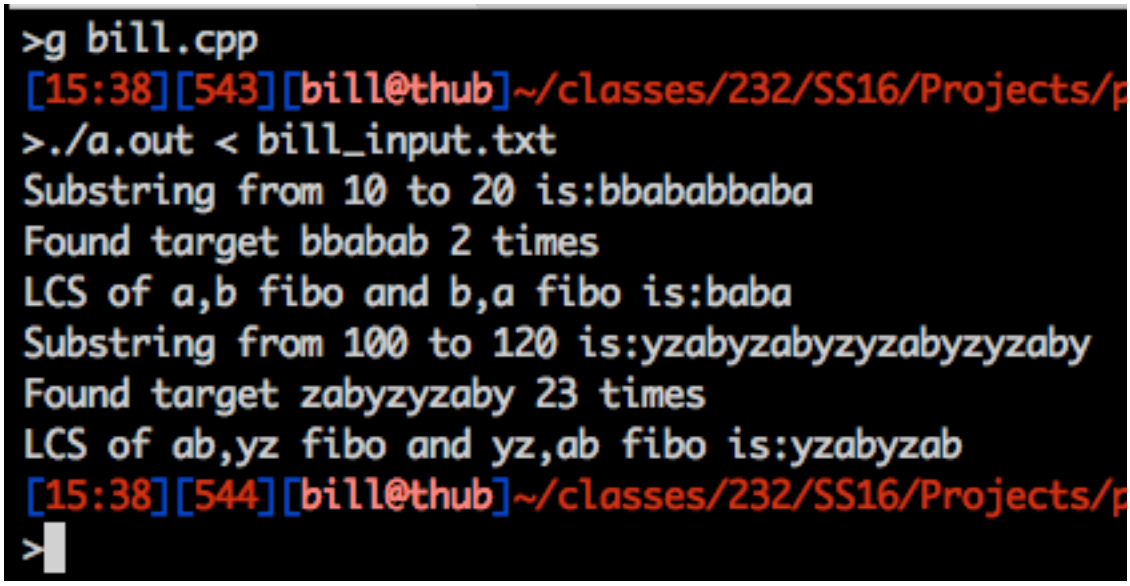
This looks for the longest common substring (lcs) in the two strings:
This is a brute force (that is, very expensive algorithm). In outline:

- lets designate the two strings `a_str` and `b_str`
- select the shortest of the two, let's say it's `b_str`
- generate every substring of `b_str`
  - for each substring of `b_str` look for that substring in `a_str`
  - record the longest such substring.
- look at the notes for more details.

**Deliverables**
`proj04.cpp` -- your source code solution (*remember to include your section, the date, project number and comments in this file*).

1) Be sure to use "proj04.cpp" for the file name (or handin might not accept it!)
2) Save a copy of your file in your CS account disk space (H drive on CSE computers). This is the only way we can check that you completed the project on time in case you have a problem with handin.
3) Electronically submit a copy of the file.

**Sample output:**




# Notes

**Finding all substrings**

Let's look at an example. What are all the substrings of the string `"abcd"` ? Here is a procedure:

1. Start with the first letter, `'a'`.
    a. `"a"` is a substring of length 1
    b. Now generate every substring beginning with `"a"`, of the various lengths: length 2 `("ab")` , then length 3 `("abc"),` then length 4 `("abcd")` up to the end of the string.
2. Those are all the substrings starting with the first letter, so we move on to the second letter `'b'` and repeat the process
    a. `"b"` is a substring of length 1
    b. Now generate every substring beginning with `"b",` of the various lengths: length 2 `("bc")` , then length 3 `("bcd"),` up to the end of the string.
3. Continue this process:
    a. Move to the next letter
    b. Generate all the substrings beginning with that letter up to the length of the string.

In order, you would get the following substrings. Convince yourself that this is all of them.
`"a", "ab", "abc", "abcd", "b", "bc", "bcd", "c", "cd", "d"`

By the way, this is a **terribly expensive algorithm**. If the string lengths get long, then the time to solve it using this algorithm makes it impossible to use. In your algorithms class you will learn better approaches, but this works for now (and stands as a bad example in the future).

**Find the Longest common substring**

For every substring, ***find*** (hint hint) that substring in the other string. For every substring tested remember the longest:

- Repeats are possible, in which case any would do.