Brown University ENGN2912B Fall 2017
Scientific Computing in C++
Assignment 4

1) Write a function that returns the translation vector between two 2-dimensional positions. Represent the positions as **`double`** variables and the result as a pair of **`double`** values. Pass both inputs and outputs as arguments to the function. Use references as argument types. Use the **`const`** qualifier where appropriate. Return a **`bool`** result that is **`false`** if the input points are the same. Call this function from your main program for the pair-wise distances. Read the coordinates of the two points from the command line as follows

```
assignment_04_1 x1 y1 x2 y2
```

Write a usage message if the number of arguments is not correct, and if the conversion of any of the arguments to double fails. Print the result using the following format

```
p1=(x1,y1) p2=(x2,y2) tvec=(tx,ty)
```

Use the following points to test your program: p1 = (5.31, 7.41), p2 = (413.13, 84.154), p3 = (8.815, 20.7114), p4 = (5.31, 7.41).

2) Add a first function above your main program to compute Fibonacci numbers using recursive functions. Use unsigned long as the input and output data types. The first two Fibonacci numbers are $f_0 = 0$ and $f_1 = 1$. The remaining Fibonacci numbers are the sum of the previous two numbers. Thus the sequence $(f_n)_{n\geq 0}$ is:

0, 1, 1, 2, 3, 5, 8, 13 ...

It is well known that the limit of the ratio of successive Fibonacci numbers, as n approaches infinity, is the golden ratio

$$\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \frac{1+\sqrt{5}}{2}$$

The golden ratio has many applications in the arts. For example, it is the ideal ratio of the height to the width of a doorway in order to be aesthetically pleasing.

Write a second function named `goldenRatio`, with two parameters, `precision` (input), and `firstIndex` (output), that calls your first function to approximate the golden ratio, as a function of the successive values of the Fibonacci sequence. The first parameter specifies the desired number of correct digits in the output value. The second parameter should be used to return the first index in the Fibonacci sequence for which the approximation has the desired number of correct digits. You can use the fact that you know the limit value in your implementation of this function.

Use double as the return data type. Use unsigned as the data type for the two parameters. Keep in mind that the second parameter will be used to return the value determined in the function.

Call your program with two arguments

```
assignment_04_02 cmd N
```

If the number of arguments is incorrect, print a usage message and quit.

If the `cmd` argument is equal to the string "-f": 1) Read an unsigned value N from the next argument. If the conversion of the argument to unsigned fails, print a usage message and quit; and 2) Print all the Fibonacci numbers $(f_n)_{0\leq n\leq N}$, as a table with five numbers per row. The five columns should have a width of exactly 15 spaces. Leave one space of separation between columns, and align all the numbers on the right hand side.

If the `cmd` argument is equal to the string "-g": 1) Read an unsigned value N from the next argument. If the conversion of the argument to unsigned fails, print a usage message and quit; 2) Determine the minimum number of iterations that produces the golden ratio to a floating point precision of N digits. Print the result using the following format

```
goldenRatio(N) = <value returned by your function> { firstIndex }
```

If the `cmd` argument does not match any of the previous two strings, print the usage message and quit.

3) Write a set of functions for processing strings. The strings are represented as an array of **`char`** data types with a terminating **`0`** or null character. The functions to be implemented are:

- concatenate two strings – inputs are two strings, the output is a single string with the input strings joined in a single null-terminated array. Note, there should only be one null termination on the output. The function name should be **`concat`**.
- find a substring – inputs are two strings, **`str1`** and **`str2`**. The function should determine if **`str2`** is a proper substring of **`str1`**. That is, it is possible that **`str1`** and **`str2`** are the same string, but **`str2`** can never be longer than **`str1`**. The outputs should be a **`bool`** return indicating true if **`str2`** is a substring of **`str1`**. Also the array index location of the start of **`str2`** in **`str1`** should be output as a reference to an unsigned value. The function name should be **`substr`**.
- reverse a string in place – this function has a single input and no return value. The input is a string, the result of function execution is that the input string is reversed in order. The function name should be **`reverse`**.

Declare these functions in a separate `myStringFunctions.hpp` file, implement them in a separate `myStringFunctions.cpp` file. Save the two files in the `src` directory for this assignment. Edit the `CMakeLists.txt` file so that your functions are compiled and linked with your `assignment_04_3` program. Remember to include your new header file in the file that contains your `main()` program.

Call your program as follows

```
assignment_04_3 cmd str1 [str2]
```

The first `cmd` argument should be equal to one of the following strings "-c", "-s", or "-r"; otherwise print a usage message and quit. The number of required additional arguments depends on the value of the `cmd` argument.

If the `cmd` argument is equal to "-c", two additional arguments are required. If the number of arguments is incorrect, print the usage message and quit. Run your `concat` function using the two additional command line arguments as input. Print the result using the following format

```
concat(str1,str2) = result
```

where the strings `str1`, `str2`, and `result` should be printed between double quotes, as in this example

```
concat("Hello ","World") = "Hello World"
```

Note that there is a space at the end of the first string. To specify strings with white spaces in the command line you need to use single or double quotes as well. In this example, you would call your program as follows

```
assignment_04_3 -c "Hello " World
```

If the `cmd` argument is equal to "-s", two additional arguments are required. If the number of arguments is incorrect, print the usage message and quit. Run your `substr` function using the two additional command line arguments as input. If the value returned by your function is `false`, print the result using the following format

```
substr(str1,str2) = false
```

where the strings `str1`, and `str2` should be printed between double quotes. If the value returned by your function is `truea`, print the result using the following format

```
substr(str1,str2) = true (index)
```

where the strings `str1`, and `str2` should be printed between double quotes, and `index` is the index into `str1` (regarded as an array) where the substring `str2` starts, return by your function.

If the `cmd` argument is equal to "-r", only one additional argument are required. If the number of arguments is incorrect, print the usage message and quit. Run your `reverse` function using the additional command line arguments a input. Print the result using the following format

```
reverse(str1) = result
```

where the strings `str1`, and `result` should be printed between double quotes.

Note also that C libraries exist to achieve these functions. Do not use them in your implementation. Also use **`const`** qualifiers where appropriate.