CMPT 125 Assignment 3

- Due Jan. 30 at 3:20pm
- Please submit the assignment in pdf format on CourSys
- You may write or type your answers
  - If written, please take clear pictures and convert them into a single pdf

Question 1 (25 marks)
The Fibonacci sequence is given by 1, 1, 2, 3, 5, 8, 13, 21, .... The first two numbers are 1, and subsequent numbers are the sum of the previous two. If Fib(k) represents the kth number in the sequence, then we have Fib(1) = Fib(2) = 1, and Fib(k) = Fib(k-1) + Fib(k-2) for k > 2.

a) Consider the Fib and main functions below.

```c
int Fib(int k) {
    if (k <= 2) {
        return 1;
    }
    return Fib(k-1) + Fib(k-2);
}

int main() {
    int m = 4;
    int n = Fib(m);
    return 0;
}
```

Draw the progression of the function call stack for the program. Each time there is a new function call or a function returns, there should be a depiction of the call stack. For example, if the first line of the main function is “int m = 2;”, then the first few call stacks may look like this:

<table>
<thead>
<tr>
<th>Call stack</th>
<th>Last function call (if applicable)</th>
<th>Last function return (if applicable)</th>
<th>Function parameters (for function that is called or returned)</th>
<th>Function return value (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>m = 2; n = ?</td>
<td>Fib</td>
<td>N/A</td>
<td>k = 2</td>
<td>N/A</td>
</tr>
<tr>
<td>m = 2; n = ?</td>
<td>Fib</td>
<td>N/A</td>
<td>k = 2</td>
<td>1</td>
</tr>
</tbody>
</table>
Note that in the last row of the above table, the function \texttt{Fib} called with parameter \( k = 2 \) has finished execution, and returns a value of 1. \texttt{Fib} with parameter \( k = 2 \) has been popped from the call stack, which is why it is not shown.

Answer this question in the format of the 5-column table shown above. Make sure to keep track of all local variables in every function currently in the call stack.

b) Write a different version of \texttt{Fib}, called \texttt{FibIter}, that also returns the \( k \)th Fibonacci sequence, but only uses iteration. Use a loop; no recursion allowed in this part! The function prototype is as follows. Please use comments to help the reader understand your code.

\begin{verbatim}
    int FibIter(int k) {
    }
\end{verbatim}
Question 2 (25 marks)
Loops and iterations can be turned into recursions, and vice versa. In class, we studied iterative implementations of selection sort. In this question, we explore a purely recursive implementation.

To do this effectively, in our mind we divide up the array called arr into two subarrays: arr[0…mid-1] denoting the subarray from index 0 to index mid-1, and arr[mid…len-1] denoting the subarray from index mid to len-1. Next, we follow the following steps.

a) First, write a function called findMinpos, which recursively finds the position in the subarray arr[mid…len-1] with the minimum value (this was called minpos in class). The function prototype is given below. Remember: no loops allowed!

```c
int findMinpos(int arr[], int mid, int len) {
    // Finds the position in subarray arr[mid...len-1] with minimum value (minpos)
}
```

b) Next, write a function called swapMinpos, which swaps arr[mid] and arr[minpos]. The function prototype is given below:

```c
void swapMinpos(int arr[], int mid, int len) {
    // Swaps minimum of subarray arr[mid...len-1] with arr[mid]
}
```

c) Finally, write a recursive selection sort function, called selSortR. There are many possible implementations. For this problem, the function selSortR will put the smallest mid elements in the array arr in sorted order in the subarray arr[0…mid-1]. The function prototype, as well as the main function one may use to call selSortR, are given below. Remember: no loops allowed!

```c
void selSortR(int arr[], int mid, int len) {
    // Puts smallest mid elements in arr in sorted order in arr[0...mid-1];
}

void main() {
    int arr[14] = {488888, 3, 5, 0, 23, 12124, 6, 7, 2, 1121, 0, 92, 5, 8};
    selSortR(arr, 14, 14);
    /* This will put the smallest 14 elements in the first 14 positions, which will sort the entire array */
    for (int i = 0; i<14; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
}
```