Homework 9: Recursive functions

CS16 - Winter 2021

Due:	Thursday, March 11, 2021 (11:59 PM PST)
Points:	105
Name:	Tyler_Pruitt_____
Homework buddy:	Kristin_Cheung_____

- You may collaborate on this homework with **at most** one person, an optional "homework buddy."
- **Submission instructions:** All questions are to be written (either by hand or typed) *in the provided spaces* and turned in as a single PDF on Gradescope. If you submit handwritten solutions write legibly. We reserve the right to give 0 points to answers we cannot read. When you submit your answer on Gradescope, **be sure to select which portions of your answer correspond to which problem** and clearly mark on the page itself which problem you are answering. We reserve the right to give 0 points to submissions that fail to do this.

1. (3 points) How does a recursive function know when to stop recursing?

A recursive function knows when to stop recursing when it reaches a base case. When a recursive function reaches a base case, a value is typically returned and this results in the returning of values for all of the previous recursive function calls. This in turn ceases the recursing and the original call to the recursive function ends (i.e. returns a value, or not if it is a void function).

2. (4 points) What is a stack overflow? When can it occur? What are the consequences?

Stack overflow is when a stack is forced to expand or grow beyond it's physical limits. Since stacks are finite objects this means that the allocated memory is being forced to expand. This can happen during infinite recursions. The consequence of stack overflow is that the program will end abnormally.

3. (3 points) What is a LIFO scheme and how does it relate to stacks?

A LIFO scheme is a last-in first-out scheme in which when data is inserted into the data structure it will go on the top and when accessing information one has to go through everything placed on top of it in the data structure to access it. This last element or data being pushed on to the stack will be the first to be taken off. Stacks are data structures that work on a last-in first-out (LIFO) basis in which new information is similarly added to the top of the stack, to access/retrieve data one has to go through each layer of the stack, and while traversing the stack you can only go through one layer of information or data at a time.

4. (15 points) Write a definition of a recursive function that finds the sum of the odd integers in the first n numbers. Example, if $n = 8$, then the sum is: $(1 + 3 + 5 + 7) = 16$.

```
int sumOdds(int n)
{
    //the base case
    if (n <= 0)
    {
        return 0;
    }
    //the recursive case
    if (n % 2 != 0)
    {
        return sumOdds(n-1) + n;
    }
    else
    {
        return sumOdds(n-1);
    }
}
```

5. (25 points) Write a definition of a *recursive* function that finds the nth element in the following arithmetic numerical sequence: 3, 11, 27, 59, 123, ... I've started the program for you below, but it is missing the definition. *Hint*: First, figure out the recursive pattern as a linear equation, i.e. $a_n = x * a_{n-1} + y$. You also have to identify the base case. Example outputs would look like this (there is no repeating loop—these are 2 separate runs):

```
Which element of the sequence would you like to know? 4
Element number 4 in the sequence is 59.

Which element of the sequence would you like to know? 7
Element number 7 in the sequence is 507.
```

```
#include <iostream>
using namespace std;
int RecursiveFunc(int num);
int main() {
   int elementN;
   cout << "Which element of the sequence would you like to know? ";
   cin >> elementN;
   cout << "Element number 4 in the sequence is "
        << RecursiveFunc(elementN) << endl;
   return 0;
}

//DEFINITION HERE:

int RecursiveFunc(int num)
{
    //handle illegal input
    if (num <= 0)
    {
        cerr << "Element number must be greater than zero!\n";
        return -1;
    }
    //the base case
    else if (num == 1)
    {
        return 3;
    }
    //the recursive case
    return 2*RecursiveFunc(num-1) + 5;
}
```

6. (25 points) The Fibonacci sequence is defined as a numerical sequence of integers that are the sum of the previous 2 integers. Starting with 0 and 1, the sequence becomes: $0, 1, 1, 2, 3, 5, 8, 13, ...$

   Write the definition of 2 functions: one recursive called `Fibo` (it finds the Nth element in a Fibonacci series) and one non-recursive called `SFS` that calls `Fibo`.

   `SFS` has an integer argument `n`. The pre-condition is that `n` is assumed to be smaller than 256. The post-condition is that SFS prints out all the squares of the Fibonacci sequence of the first `n` elements. For example, calling this line in `main` (just like this): `SFS(7);` will print to standard out: `0 1 1 4 9 25 64`

   (You may assume `<cmath>` is already included.)

```
int Fibo(int n)
{
    //handle illegal input
    if (n <= 0)
    {
        cerr << "Element number must be greater than zero!\n";
        return -1;
    }
    //the base case
    else if (n == 1)
    {
        return 0;
    }
    else if (n == 2)
    {
        return 1;
    }
    //the recursive case
    return Fibo(n-1) + Fibo(n-2);
}

void SFS(int n)
{
    //handle illegal input
    if (n >= 256 || n <= 0)
    {
        cerr << "Input must be less than 256 and greater than 0!\n";
    }
    else
    {
        for (int i=1;i<=n;i++)
        {
```

```
            if (i == n)
            {
                cout << pow(Fibo(i), 2);
            }
            else
            {
                cout << pow(Fibo(i), 2) << " ";
            }
        }
    }
}
```

7. (25 points) Write a definition of a recursive function that counts the number of the letter 'a' or 'A' (i.e. either upper-case or lower-case) in a string. Specifically, given a string variable, `sentence`, when we pass that into a function `CountA(sentence)`, the function returns an integer that's a count of the number of the letter 'a' or 'A' in the variable `sentence`. This function **must** be a recursive one and cannot contain a loop.

```
int CountA(string str)
{
    int len = str.length();

    //the base case
    if (len == 0)
    {
        return 0;
    }
    //the recursive case
    if (str[len-1] == 'a' || str[len-1] == 'A')
    {
        str.erase(len-1, 1);
        return CountA(str) + 1;
    }
    else
    {
        str.erase(len-1, 1);
        return CountA(str);
    }
}
```

8. (5 points) The CS department is currently restructuring its lower-division curriculum (CS 16, 24, and 32). We want to gather feedback on the experience students are having in the current CS 16 curriculum. Please take the survey found in the following link. Your response will be recorded anonymously. To receive credit for this, please attach some proof that you took this survey (e.g., a screenshot at the end of survey).

`https://ucsb.co1.qualtrics.com/jfe/form/SV_0qVnEcqYnx9AJkV`