Homework - Homework 4

Topics: Recursion, string processing.
Points: 100

Submit to Blackboard before the deadline. You will be able to revise your answers until the deadline with no penalty.

Requirements:

1. No global, static or external variables allowed. NO CREDIT if such variables are used.
2. All the theoretical questions will be answered in **2320_H4.pdf**.
3. All the functions required for this homework will be written in file **palindromic_decomp.c**.
4. In main, write code that repeatedly calls each of the functions that you have to implement:
    1. first repeatedly call the function from task 1,
    2. then repeatedly call the function in task 2 and
    3. after that repeatedly call the function from task 3.
    4. (You may want to write place holders for the unimplemented functions so that you can still run the program with the input files.)
5. **Your program should not have any memory leaks or errors. Check it with Valgrind.** See useful information and links for using Valgrind on the Code webpage (it includes instructions on how to compile the code to see line numbers for errors reported by Valgrind).
6. You can write any other helper functions or wrapper functions if you want to.
7. The program must accept user input. It will be run with input redirection.
8. You should always assume that **every function you are asked to implement has to work for all sizes and variations of the data**. The provided examples may be small. Do not write your function for those examples, write it to work with any data.

Links and references:

- Valgrind tutorial

Provided files:

1. 2320_H4.pdf (2320_H4.docx)
2. all.txt - Input file with multiple inputs for each function
3. pal_1.txt - Input file with input for the palindromic decomposition. Note how -1 was used so that the other functions will not execute.

   Files with the output produced from running the program with input redirected (one at a time) from files: all.txt and pal_1.txt files.
4. run_all.txt
5. run_pal_1.txt

Task 1 (14 points)

a) Write a function called `int increasing(int * A, int N)` that takes as argument an array and its length and returns 1 if the elements in the array are in increasing and 0 otherwise. You can modify (add remove) parameters for this function if you need to. Examples:

- for [3, 6, 8, 50, 10000] it returns 1 (increasing).
- for [3, 8, 8, 50, 50] it returns 1 (increasing, with duplicates).
- for [3,6,1,8] it returns 0 (neither increasing nor decreasing).
- for [20,16, 9, 7, 3] it returns 0 (decreasing).
- for [20,6,-1,-5] it returns 0 (decreasing).
- for an empty array it return 1
- for an array with 1 element it returns 1.
- if all the array elements are equal it should return 1.

Requirements:

1. In main (or another function that will be called in main), repeatedly read arrays (first the array size and then that many elements) and call this function, until the user gives -1 for the array size.
2. The work must be done with recursion: `increasing` may be recursive or it may call a helper recursive function. .
3. You can assume that the max array size is 100.
4. Write the recursion formula (T(n) = ... and the base case T(...) = ...) for your function. Write your answer in **2320_H4.pdf**.
5. Solve the recurrence formula and give the Θ time complexity of your code. Write your answer in **2320_H4.pdf**. - This topic (solving recurrence formulas) will be covered in our upcoming lecture.

Extra practice question for exam (not graded): What are the arguments for the 3rd recursive call, when originally called on the array [5,3,1,8,9,0].

Task 2 (19 points)

Write three recursive implementation for finding the minimum in an array. Put the code for this task in palindromic_decomp.c .
In class we wrote these methods:

```
int min_rec_1(int* A, int N){
        // crashes for N < = 0
        if (N==1) return A[0];
        int temp = min_rec_1(A,N-1);
        if (temp < A[N-1])
                return temp;
        else
                return A[N-1];
}
```

```
int min_rec_2(int* A, int N){
        // crashes for N < = 0
        if (N==1) return A[0];
        if (min_rec_1(A,N-1) < A[N-1])
                return min_rec_1(A,N-1);
        else
                return A[N-1];
}
```

1. Modify these functions (add two pointer arguments) to count the number of recursive calls, `rec_ct`, the number of base cases, `base_ct`, and the MAX achieved depth of the recursion tree that each one of these functions does.
2. Write a tail-recursive implementation, `...min_rec_tail(....)`, for finding the minimum in an array. You can assume that it will be called for arrays with at least one element in them.
   Hints: use a wrapper function in addition to the tail-recursive one. Think about all the different ways to rob a train that we talked about.
   We looked at recursion trees for function calls. In the 2320_H4.pdf you are asked to give the exact number of recursive calls and base case calls generated from one call to each of the above methods.

Task 3 (67 points)

Generate all **palindromic decompositions of a string s** (cut it into substrings that are palindromes).
For example the string `abbccd` has 4 palindromic decompositions:

```
a, b, b, c, c, d,
a, b, b, cc, d,
a, bb, c, c, d,
a, bb, cc, d,
```

Requirements:

1. In main (or another function that will be called in main), repeatedly read a string from the user, until the user enters -1. For each of these strings, print the palindromic decomposition (one per line) and after that the total number of decompositions.
2. The function that generates the decompositions and counts them **must be recursive**.
3. The maximum string length is 100.
4. In the 2320_H4.pdf write the recurrence formula (T(n) = ... and the base case T(...) = ...) for the recursive function that generates the palindromes as a function of the string length, N.

Hint: Compare your problem with the queens problem, especially in the way they build the solution. For the 8-Queens problem the solution is the 1D array of rows for each column. What will your solution be? How do you represent the solution and how do you 'grow it'?
Hint: Your solution must show the substrings. To achieve that, you can store the actual substrings, or just the indexes of where each substring starts. That is for a specific decomposition like: `a, bb, cc, d,` you can store the substrings`a, bb, cc, d,` or the indexes `0,1,3,5` that refer back into the original string

```
abbccd
012345
```

and can be used as follows: first substring is from indexes [0,1) (i.e. "a"), second one is in [1,3) (i.e. "bb"), third one is in [3,5) ("cc") and fourth one is in [5,end] (i.e. "d")
Hint: Remember the usage of things like &text[4] in hw2. They may come in handy here as well.
Algorithm guideline

```
    ... pal_decomp(......)
        if current string has length 0 print recorded decomposition and return
        for each prefix,pre, of the current string
                if pre is a palindrome, record pre and call pal_decomp(...) on the remaining part of the string (and whatever other arguments you need)
```

Extra practice question for exam (not graded): What are the arguments for the 3rd recursive call, when originally called on the string `abbccd`

How to submit

The assignment should be submitted via Blackboard.
Place all your files in a folder called `hw4`. Zip that folder (it will produce a file called `hw4.zip`). Submit `hw4.zip`.
No other forms of compression are accepted, contact the instructor or TA if you do not know how to produce .zip files. The zipped directory should contain the following documents:

- **palindromic_decomp.c**
- **2320_H4.pdf** with instructions for how to compile and run your code, on omega and the recurrence formulas and time complexity of your functions.

As stated on the course syllabus, programs must be in C, and must run on omega.uta.edu.

IMPORTANT: Pay close attention to all specifications on this page, including file names and submission format. Even in cases where your answers are correct, points will be taken off liberally for non-compliance with the instructions given on this page (such as wrong file names, wrong compression format for the submitted code, and so on). The reason is that non-compliance with the instructions makes the grading process significantly (and unnecessarily) more time consuming. Contact the instructor or TA if you have any questions.

Back to Homework page.