Palindrome, Base Conversion and Sorting Points

Lab #5 (CS1010 AY2013/4 Semester 1)

Date of release: 28 October 2013, Monday, 9am.

Submission deadline: 9 November 2013, Saturday, 9am.

School of Computing, National University of Singapore

0 Introduction

Important: Please read Lab Guidelines before you continue.

The objective of the first two exercises is on recursion, and the third exercise is on array of structures.

You are required to do exercises 1 and 2 which are on recursion. As exercise 3 uses structure and the topic on structure is taught in week 12, it is optional for you to submit exercise 3. Your DL will grade only exercises 1 and 2.

The maximum number of submissions for each exercise is **15**.

If you have any questions on the task statements, you may post your queries **on the relevant IVLE discussion forum**. However, do **not** post your programs (partial or complete) on the forum before the deadline!

Important notes applicable to all exercises here:

- You should take the "Estimated Development Time" seriously and aim to complete your programming within that time. Use it to gauge whether your are within our expectation, so that you don't get surprised in your PE. We advise you to do the exercises here in a simulated test environment by timing yourself.

- Please do **not** use variable-length arrays. An example of a variable-length array is as follows:
  ```
  int i;
  int array[i];
  ```
  This is not allowed in ANSI C, as explained in Week 7 lecture. Declare an array with a known maximum size. We will tell you the maximum number of elements in an array.

- You are **NOT allowed to use global variables**. (A global variable is one that is not declared in any function.)

- You are free to introduce additional functions if you deem it necessary. This must be supported by well-thought-out reasons, not a haphazard decision. By now, you should know that you **cannot write a program haphazardly**.

- In writing functions, we would like you to include function prototypes before the main function, and the function definitions after the main function.

1 Exercise 1: Palindrome Numbers

1.1 Learning objective

- Writing recursive function.

1.2 Task statement

As you have learned, a **palindrome** reads the same forwards and backwards. Likewise for a palindrome number. Example of palindrome numbers are: 20111102, 56765, and 926629.

Write a program **`palindrome_numbers.c`** that reads two positive integers: a start value and an end value (end value is greater than or equal to the start value), and computes the number of palindrome numbers in the range from start to end, both inclusive. You may assume that the largest integer input contains **at most 9 digits**.

For example, if the start and end values are 179 and 231 respectively, then there are 5 palindrome numbers between them: 181, 191, 202, 212, and 222.

You must have a recursive function to determine if a certain number is a palindrome. **No mark will be given if you do not use recursion.**

1.3 Sample runs

Sample runs using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall palindrome_numbers.c -o palindrome_numbers
$ palindrome_numbers
Enter start and end: 1234 4321
Number of palindrome numbers = 30
```

Sample run #2:

```
Enter start and end: 987654321 999999999
Number of palindrome numbers = 1235
```

1.4 Skeleton program and Test data

- The skeleton program is provided here: palindrome_numbers.c
- Test data: Input files | Output files

1.5 Important notes

- Hint: To solve this task easily, since you know the number contains at most 9 digits, you may extract the digits of the input number and store them in an array. You can then run a recursive function on the array to determine if it is a palindrome.
- The skeleton program provided shows you that there is a driver function **`isPalindrome()`** which calls the actual recursive function **`palindromeRecur()`**. Hence, the main() function needs only call **`isPalindrome()`** Refer to Week 11 slides 36 - 38 for explanatian.
- You must use the **`isPalindrome()`** function.
- You must use recursion. An iterative (loop) algorithm will not be accepted since it undermines the objective of this exercise.

- Please note that the only recursive function required is **`palindromeRecur()`**. The part where you run from the start value to the end value can be done with a loop.

1.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 10 minutes
- Translating pseudo-code into code: 10 minutes
- Typing in the code: 15 minutes
- Testing and debugging: 25 minutes
- **Total: 1 hour**

2 Exercise 2: Converting Decimal to another Base

2.1 Learning objectives

- Writing recursive function.
- Using character and string function.

2.2 Task statement

We can convert a positive integer in decimal (base 10) to another base by doing repeated division. The two examples below show how to convert decimal value 123 to base 5 (quinary) and base 16 (hexadecimal).

5	123	
5	24	remainder *3*
5	4	remainder *4*
	0	remainder *4*

16	123	
16	7	remainder *11*
	0	remainder *7*

The answer is obtained by collecting the remainder in each step of the division, from the bottom up. Hence, $123_{10} = 443_5 = 7B_{16}$.

For bases larger than 10, the digits 10, 11, 12, ... are represented by the characters A, B, C, ... Hence the digits in hexadecimal are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

The above algorithm can be implemented using recursion.

Write a program **`convert_base.c`** to read in a positive integer and a target base (between 2 and 36 inclusive), and generate the equivalent value in the target base. Your program must use a recursive function. **No mark will be given if you do not use recursion.**

2.3 Sample runs

Sample runs using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall convert_base.c -o convert_base
$ convert_base
Enter a positive decimal integer: 123
Enter target base (2-36): 5
```

```
123 converted to base 5 = 443
```

Sample run #2:

```
Enter a positive decimal integer: 123
Enter target base (2-36): 16
123 converted to base 16 = 7B
```

Sample run #3:

```
Enter a positive decimal integer: 12345
Enter target base (2-36): 36
12345 converted to base 36 = 9IX
```

2.4 Skeleton program and Test data

- The skeleton program is provided here: convert_base.c This program contains a **`char digit(int)`** function for your use.
- Test data: Input files | Output files

2.5 Important notes

- You may assume that the longest answer consists of 31 characters.
- You may work on a simpified version first, to convert into base of 9 or less, to get the logic right, before working on the case for base > 10.
- You must not use string functions such as **`char *itoa (int value, char *str, int base)`** (or the like) to do the conversion, but you may use string functions that do not do conversion of bases, such as **`strcat()`**.
- You must use recursion. An iterative (loop) algorithm will not be accepted since it undermines the objective of this exercise.

2.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 15 minutes
- Translating pseudo-code into code: 10 minutes
- Typing in the code: 15 minutes
- Testing and debugging: 40 minutes
- **Total: 1 hour 20 minutes**

3 Exercise 3: Sorting Points

3.1 Learning objectives

- Sorting.
- Array of structures.

3.2 Task statement

Write a program **`sort_points.c`** to read in a positive integer indicating the number of 2-dimensional points, followed by the x- and y-coordinates of those points. These points should be read into an array of structures, where each structure contains the x- and y-coordinates of a point. You should name the structure type **`point_t`**.

You may assume that there are at least 2 points and at most 20 points. You may use only one of these sorting algorithms: Selection sort, Bubble sort, or Insertion sort.

Your program should then sort the array of points in ascending order of the distance of these points from the origin (0,0).

The formula to compute the distance of a point (x,y) from the origin (0,0) is given below:

$$dist(x,y) = sqrt(x*x + y*y)$$

However, you are **not** allowed to use any math function. In other word, your program should **not** include <math.h>.

3.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall sort_points.c -o sort_points
$ sort_points
Enter number of points: 5
Enter 5 points:
18 2 7 12 -5 2 8 -10 4 13
Sorted points:
(-5,2)
(8,-10)
(4,13)
(7,12)
(18,2)
```

Sample run #2:

```
Enter number of points: 6
Enter 6 points:
5 6 7 2 -3 -4 -2 10 1 12 9 7
Sorted points:
(-3,-4)
(7,2)
(5,6)
(-2,10)
(9,7)
(1,12)
```

3.4 Skeleton program and Test data

- The skeleton program is provided here: sort_points.c
- Test data: Input files | Output files

3.5 Important notes

- There are at most 20 points.

- You are to use only one of these sorting algorithms: Selection sort, Bubble sort, or Insertion sort.

3.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 20 minutes
- Translating pseudo-code into code: 15 minutes
- Typing in the code: 15 minutes
- Testing and debugging: 30 minutes
- **Total: 1 hours 20 minutes**

4 Deadline

The deadline for submitting all programs is **9 November 2013, Saturday, 9am**. Late submission will NOT be accepted.

Aaron
Wednesday, October 23, 2013 09:04:28 PM SGT