Palindrome, Base Conversion and Mine Clusters

Lab #5 (CS1010 AY2011/2 Semester 4)

Date of release: 16 July 2012, Monday, 14:00hr.

Submission deadline: 24 July 2012, Tuesday, 23:59hr.

School of Computing, National University of Singapore

0 Introduction

Important: Please read Lab Guidelines before you continue.

This lab requires you to do 2 exercises for submission. The third exercise is only for your own practice, hence it is not compulsory for you to submit it as it will **not** be graded.

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!

The rule for **attempt mark** for this lab assignment is as follows:

- You must submit both exercises 1 and 2.
- The average mark of exercises 1 and 2 must be >= 50% of the total maximum mark.
- At most one of the exercises can have < 50% of its maximum mark, but it must be >= 20% of its maximum mark.

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. 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 objectives

- Writing recursive function.

1.2 Task statement

Watch out this 2nd of November! There might be a surge in the number of people tying their nuptial knot on that special day. Written in year-month-day format, the date 2011-11-02, or 20111102, is a **palindrome** number. A palindrome is something that reads the same in both directions.

Write a program `palindrome_numbers.c` that reads two positive integers: a start value and an end value (which is greater than or equal to the start value), and compute 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 Number of submission

For this exercise, the number of submission is **20**.

1.5 Important notes

- The skeleton program is provided here: palindrome_numbers.c.
- Hint: Extract the digits of a number and store them in an array. Then run a recursive function on the array to determine if it is a palindrome.
- You must use recursion. An iterative algorithm (loop) will not be accepted since it undermines the objective of this exercise.

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**

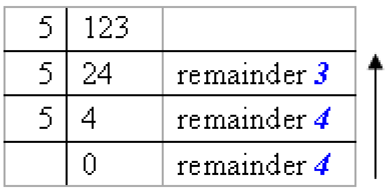
2 Exercise 2: Converting Decimal to another Base

2.1 Learning objectives

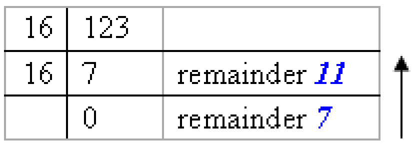
- Writing recursive function.
- Using character and string function.

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.

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
443
```

Sample run #2:

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

Sample run #3:

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

2.4 Number of submission

For this exercise, the number of submission is **20**.

2.5 Important notes

- The skeleton program is provided here: convert_base.c. This program contains a `char digit(int d)` function for your use.

- 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 algorithm (loop) 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: Mine Clusters

3.1 Learning objectives

- Two-dimensional array.
- Writing recursive function.

3.2 Task statement

This exercise will **not** be graded.

We shall revisit the minesweeper game. Recall that our minefield consists of mine-filled cells (represented by '*') and mine-free cells (represented by '-').

If two mines are adjacent, then they are part of the same **mine cluster**. Mine clusters can be quite large and may contain numerous mines. Your task is to determine how many mine clusters there are in the minefield.

You are to write a program `mine_clusters.c` to read in the following input data:

- The first line contains two positive integers *rows* (≤ 20) and *cols* (≤ 40) which denote the number of rows and columns, respectively, of the minefield.

- Followed by a (*rows* × *columns*) array representing the minefield. Each cell on the minefield is represented either by '*' (mine) or '-' (mine-free).

Your program outputs the **number of distinct mine clusters**. Two mines are parts of the same mine cluster if they are adjacent horizontally, vertically, or diagonally.

The following is an example showing a 5 × 8 minefield.

```
5 8
----*---
-**-*-*-
-*---*--
***-*---
**--***-
```

The number of mine clusters for the above example is 2. The two mine clusters are shown in different colours below. (Your program only needs to output the value 2. The figure below is for explanatory purpose only.)

```
----*---
-**-*-*-
-*---*--
***-*---
**--***-
```

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 mine_clusters.c -o mine_clusters
$ mine_clusters
4 5
-----
-----
-----
-----
0
```

Sample run #2:

```
7 18
-----*------------
----*-*-----*-----
---*---*---*-*----
--*--*--*-*---*---
---*---*---*-*----
----*-*-----*-----
-----*------------
3
```

3.4 Number of submission

For this exercise, the number of submission is **20**. Note that this exercise will **not** be graded.

3.5 Important notes

- The skeleton program is provided here: mine_clusters.c. This program contains a **`scan_minefield()`** function for your use.
- A minefield has at most 20 rows and 40 columns.
- You must use recursion. An iterative algorithm (loop) will not be accepted since it undermines the objective of this exercise.
- Entering the data interactively for this exercise is tedious, especially when the minefield is large. You may create a text file to contain your input data.
  Type the input data into a text file, let's call it `mines.in`. You may then use the UNIX input redirection symbol < to redirect input from this file:

  ```
  mine_clusters < mines.in
  ```

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): 50 minutes
- Translating pseudo-code into code: 20 minutes
- Typing in the code: 20 minutes
- Testing and debugging: 60 minutes
- **Total: 2 hours 30 minutes**

4 Deadline

The deadline for submitting all programs is **24 July 2012, Tuesday, 23:59hr**. Late submission will NOT be accepted.

Go back to CS1010 Labs page.

Aaron Tan
Saturday, October 8, 2011 10:37:16 PM SGT