CSC172 LAB 10

POINTERS TO FUNCTIONS

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_programming .

Every student must hand in their own work, but every student must list the name of their lab partner if any on all labs.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

The textbook should present examples of the code necessary to complete this lab. However, collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

2 Pointers to functions

The goal of this lab it to gain familiarity with pointers in the 'C' programming language.

1. Begin this lab by implementing a simple C program with a “main” function that fills an array with random values. A sample program is shown on the next page. You may refer to the code in Kernighan and Ritchie if you need help.

```
#include <stdio.h>
#define MAX 10

int myrand(); // function prototype for myrand()

int a[MAX];
int main(int argc, char *argv[]) {
    int i,t,x,y;
    /* fill array */
    for (i=0; i < MAX; i++)
    {
        a[i]=myrand();
        printf("%d\n",a[i]);
    }
}

int rand_seed=10;
int myrand() // implementation of myrand()
{
    rand_seed = rand_seed * 1103515245 + 12345;
    return (rand_seed / 65536) % 32768;
}
```

2. Add a simple bubblesort function to your code as shown below. You will need to add a prototype for the function before the main as was done with the rand() function.

```
void bubble_sort(int m, int a[])
{
    int x,y,t;
     for (x=0; x < m-1; x++)
     {
        for (y=0; y < m-x-1; y++)
        {
            if (a[y] > a[y+1])
            {
                t=a[y];
```

```
                a[y]=a[y+1];
                a[y+1]=t;
            }
        }
    }
}
```

Add code to your main method to sort the random array and print it out.

3. Write a two functions `int gtcmp(int x, int y)` and `int ltcmp(int x, int y)`. The first should return 1 in x > y and zero otherwise. The second should return 1 if x < y and zero otherwise.
4. Replace the direct comparison `if (a[y] > a[y+1])` in bubblesort with a call to the function `if (gtcmp(a[y], a[y+1]))` in order to test them.
5. Referring to section 5.11 in K&R. Alter the declaration and prototype of bubblesort to take an additional parameter. This parameter should be a pointer to a function that takes two ints as parameters and returns an int. Change the call to bubblesort from main to use the comparason functions. (i.e. `bubble_sort(MAX,a,gtcmp);` or `bubble_sort(MAX,a,ltcmp);`) You will need to modify the comparison step in bubblesort to use the passed in function rather than an explicit function name.
6. Modify your main method so that it first prints out the unsorted random array, then calls bubblesort with the gtcmp method passed in, then prints the sorted array, then calls bubblesort with the ltcmp method passed in, and finally prints the reverse sorted array.

3 Hand In

Hand in the source code from this lab at the appropriate location on the blackboard system at my.rochester.edu. You should hand in a single compressed/archived (i.e. “zipped” file that contains the following.)

1. A plain text file named README that includes your contact information, your partner's name, a brief explanation of the lab (A one paragraph synopsis. Include information identifying what class and lab number your files represent.). And a one sentence explaining the contents of all the other files you hand in.
2. The source code files representing the work accomplished for this lab. All source code files should contain author and partner identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

4 Grading

Each section (1-6) accounts for 15% of the lab grade (total 90%)

(README file counts for 10%)