CS 271 Lab 3: Pointers

Goals:

- **Become proficient at writing and using makefiles.**
- **Become proficient in working with pointers in C.**
- **Master the process of swapping.**
- **Develop an understanding of the selection sort and insertion sort algorithms.**

Grading (50 points total):

1. 5 points - All programs follow documentation and style requirements.
2. makefile - 5 points
3. sortingfunctions.h - 5 points
4. sortingfunctions.c - 10 points (5 points per function)
5. helperfunctions.h - 5 points
6. helperfunctions.c - 10 points
7. lab3.c - 5 points

Step by Step Lab Instructions

Warning: This lab assignment requires Linux. If you have Linux installed on your own computer and you have the GNU gcc compiler, you may try using your own computer but you should still test your files on the computer science machines (SH 118 or 118B) before submitting.

Organize your files into folders by lab assignment.

1. **Create a file named lab3.c. Copy the following code into your program. This is the initial version of your "test program". You will modify this file as you go through the process of writing and testing the functions in this assignment.**

   The comments are there to show examples of what you might do to test your other functions.

```
#include <stdio.h>
#include <stdlib.h>
#include "helperfunctions.h"
#include "sortingfunctions.h"
#define ARRAYSIZE 10

int main (void) {

   // dynamically allocate memory space for an array
   int * arrayPtr = (int *) malloc (ARRAYSIZE * sizeof(int));
```

```
    // fill the array with random integers

    // print the array, 10 elements per line

    // sort the array using selection sort

    // print the array, 10 elements per line

    // fill the array again with random integers

    // print the array

    // sort with insertion sort

    // print the array

}
```

2. Create a file called sortingfunctions.h Write a preprocessor wrapper (following convention for constant name) and then insert the following prototypes.

```
int selectionSort ( int * const data, int size );
int insertionSort ( int * const data, int size );
```

3. Create a file called helperfunctions.h Write a preprocessor wrapper (following convention for constant name) and then insert the following prototypes.

```
void swap ( int * num1, int * num2 );
void fillArray( int * const data, int size, int min, int max );
void neatPrint ( int * const data, int size, int numPerLine, int fieldSize );
```

4. Create a file called sortingfunctions.c Write the implementation of the two functions:

```
int selectionSort ( int * const data, int size );
```

- sorts the array using the selection sort algorithm

  Caution: there are a lot of programs out there on the web that claim to be selection sort written in C. The class syllabus prohibits copying code from web sites. A grade of zero will be assigned for this assignment if the TA or instructor sees evidence that you have copied code.

- You cannot use the bracket [ ] notation to access array elements.

- The return value is a count of the number of comparisons that are made. Specifically, you should count the number of times that you have an if statement that compares the value of an array element.

```
int insertionSort ( int * const data, int size );
```

- sorts the array using the insertion sort algorithm

  Caution: there are a lot of programs out there on the web that claim to be insertion sort written in C. The class syllabus prohibits copying code from web sites. A grade of zero will be assigned for this assignment if the TA or instructor sees evidence that you have copied code.

- You cannot use the bracket [ ] notation to access array elements.

- The return value is a count of the number of comparisons that are made. Specifically, you should count the number of times that you have an if statement that compares the value of an array element.

5. Create a file called helperfunctions.c Write the implementation of the three functions:

```
void swap ( int * num1, int * num2 );
```

- Exchanges the contents of the two memory locations.

```
void fillArray( int * const data, int size, int min, int max );
```

- Fills the array elements with integers from min to max (inclusive).

- You cannot use the bracket [ ] notation to access array elements.

```
void neatPrint ( int * const data, int size, int numPerLine, int fieldSize );
```

- Prints the array elements in nice, neat columns using the parameters numPerLine and fieldSize to control the layout.

- You cannot use the bracket [ ] notation to access array elements.

6. Create a makefile to compile your program and produce an executable called lab3. Be sure to include the "all" and "clean" targets.

7. Compile, test, and debug your program as needed.

8. Zip all the files together into a zip file. This must be a ZIP file, not a tar or other file type.