CS211 – Programming Practicum Fall 2021

Programming Project 1

Due: Thursday, 9/9/21 at 11:59 pm

The overall goal of this C program is to:

1. Read in a list of integers from standard input and store those values into an array. (Stop reading the input when the value of -999 is read in.)
2. Make a copy of that array
3. Sort the copy of the array
4. Read in a second list of integers from standard input (again stop reading the input when the value of -999 is read in). For each integer:
    a. Do a linear search on that value in the unsorted array
    b. Do a binary search on that value in the sorted array
    c. Print out the following information from each search:
        i. Was the value found or not found?
        ii. How many comparisons were needed to determine if it was found or not
        iii. If found, in what position was the value found in each array

The values read in by the program will form 2 lists of integers. Each list of values will end with the terminal value of -999. You can assume that the values will only be integer values and that there will be two values of -999 in the input. For example, assume the data entered was as follows:

```
3  9    57    18  2    4   -999
57
6
18
3
-999
```

The first 6 values would be stored in the array (3, 9, 57, 18, 2, 4). The searches would be done using the values of 57, 6, 18 and 3.

Write **a C program (not a C++ program!)** that will contain the following:

- Write a function that will make a copy the values from one array to another array. You are required to use this function when making the copy of the array (see item #2 above). Suggested prototype:
  ```
  void arrayCopy (int fromArray[], int toArray[], int size);
  ```

- Write your own function that will sort an array in ascending order. You are required to use this function when sorting one of your arrays (see item #3 above). You may use whatever sorting algorithm you wish. Suggested prototype:
  ```
  void myFavorateSort (int arr[], int size);
  ```

- Write your own function that will perform a linear search on the unsorted array. You are required to use this function when performing in the linear searches (see item #4a above). **The function is to "return" two values:**
    1. The first value returned is the position in the array where the value was found or the value of -1 if the value was not found.
    2. The second value returned is the number of comparisons needed to determine if/where the value is located in the array.

  Suggested prototypes:

  ```
  int linSearch (int arr[], int size, int target, int* numComparisons);
  ```
  or
  ```
  void linSearch (int arr[], int size, int target, int* pos, int* numComparisons);
  ```

- Write your own function that will perform a binary search on the sorted array. You are required to use this function when performing in the linear searches (see item #4b above). **The function is to "return" two values.**
    1. The first value returned is the position in the array where the value was found or the value of -1 if the value was not found.
    2. The second value returned is the number of comparisons needed to determine if/where the value is located in the array.

  Suggested prototype:

  ```
  int binSearch (int arr[], int size, int target, int* numComparisons);
  ```
  or
  ```
  void binSearch (int arr[], int size, int target, int* pos, int* numComparisons);
  ```

The code inside of the main( ) function should do the following:

- read in integer input from **standard input** and store these values into **a dynamically resizing array**. The values will have a "terminal value" of -999. So you read in these values in a loop that stops when the value of -999 is read in. The use of informative prompts is required for full credit. You may not assume how many numeric values are given on each line. **The use of a scanf() with the following form is expected** to read in the values:

  ```
  scanf ("%d", &val);
  ```

- make a copy of the integer array using the arrayCopy() function described above

- sort one of the arrays (using the sort() function described above), leave the other array unsorted

Inside of main, your code should (continued):

- read in integer input from standard input (again, the use of scanf() is expected) and for each of the values read in search for the value using both search functions described above (i.e. for each value read in, perform a linear search on the unsorted array and perform a binary search on the sorted array). Using the information returned/sent back from the search functions, **print out from main():**

    1. Whether the value was found or not found,
    2. The number of comparisons needed to determine whether the value exists or not in each algorithm,
    3. And the location in the array where the number is located (if the value does exist in the array).

    The above information MAY NOT be printed out in the linear search or binary search functions. Those functions MUST RETURN/SEND BACK the information to be printed by the main function. Not doing this will SEVERELY LOWER your score on this project.

    Repeat reading in integer values and searching the arrays until the terminal value of -999 is read in. The use of informative prompts AND descriptive result output is required for full credit. Again, scanf() is expected to read the input.

You may not assume the input will be less than any set size. Thus you will need to dynamically allocated space for the array.

Dynamic Array Allocation

Dynamic Array Allocation allows the space in an array to change during the course of the execution of a program. In C, this requires the use of the malloc() function. To dynamically allocate space for 100 integers, the malloc() code would be as follows:

```
int *darr;
int size = 100;
darr = (int *) malloc (size * sizeof(int) );
```

This array can only hold 100 integers and it not really dynamic. To make it truly dynamic, we need to grow the array when we try to put more values into it than can be held by its current size. The following code will double the size of the array.

```
int *temp = darr;
darr = (int *) malloc ( size * 2 * sizeof(int) );
int i;
for ( i = 0 ; i < size ; i++)
        darr[i] = temp[i];
free (temp);
size = size * 2;
```

Running your Program

Make sure your program runs properly when compiled using gcc on the bertvm.cs.uic.edu machine! (Do not use g++ or any other C++ compiler with this program.)

Program Submission
You are to submit the program via the proper Assignments link in Gradescope. You should name your files with your UIC netID and project name. For example if your UIC netID is ptroy1, then:

The file should be named as ptroy1proj1.c

Suggestion: Using Redirection of Input and Output to help Test Your Program

To help test your program, the use of redirection of standard input from a text file is a good idea for this project. Redirection is done at the command line using the less than and greater than signs. Redirection of both input and output can be done; however, for this project, you may only want to use redirection of input.

- Assume you have a text file that is properly formatted to contain the input as someone would type it in for the input called: **proj1input.txt**
- Also assume the executable for this project is in a file in the current directory called: **a.out**
- To run the project so that it reads the input from this text file instead of standard input using redirection of input, you would type the following on command line:
  ```
  ./a.out < proj1input.txt
  ```
- To store the output sent to standard output to a file called **outfile.txt** using redirection (the input is still being read from standard input), type:
  ```
  ./a.out > outfile.txt
  ```
- To redirect both standard input and standard output , type:
  ```
  ./a.out < proj1input.txt  > outfile.txt
  ```

Note that the code inside of your program will still read from standard input. Redirection is information given to the Operating System at the command prompt. The Operating System then “redirects” a standard input read operation away from the keyboard and to the specified file while the program is running.