CS 211 Programming Project 1
Summer 2020

Due: Wednesday, May 20th at 11:59 pm

For your first project, write **a C program (not a C++ program!)** that will read in a given list of integers and a target integer and check if the target integer is in the correct location to match a sorted version of the input list.

Example:

*List***:** 31, **5**, 28, 8, **15**, **21**, 11, 2
Sorted: 2, **5**, 8, 11, **15**, **21**, 28, 31

Target: 15 = Yes, in order!, 11 = No, present but out of order!, 404 = Not even in the list!
Number already in sorted order = 3 / 8 = 37.5%

Your C program will contain the following:

- A function that will make a copy of the values from one array to another array. Suggested prototype:
  void copyArray (int fromArray[ ], int toArray[ ], int size);

- A function that will sort an array in ascending order. You may use whatever sorting algorithm you wish, but you must write your own code, without using any libraries. Suggested prototype:
  void myFavoriteSort (int arr[ ], int size);

- A function that will count the total number of values that match in the sorted and unsorted arrays. Suggested prototype:
  int countMatches (int arr[ ], int sorted[ ], int size);

- A function that will determine whether a target integer is either: (a) in the list, in a correctly sorted position, or (b) In the list but not in the correct position, or (c) Not in the list at all. **The function is to "return" three values.** First, return "1" if the target appears in the correct position, "0" if the target appears but not in the correct position, or "-1" if the target does not appear in the list at all. If the target is found in the list, then nfound should be set to the number of times it appears. If it is found in a correct position, then index should be set to the location of the first matching position.
  Suggested prototype:
  int findInList (int arr[ ], int sorted[ ] int size, int target, int* nfound, int* index);

Inside countMatches:

- Set pointers to the beginning of each array. Increment the pointers in a loop, counting the number of positions in which the data in the two arrays matches.
- **Note:** Both countMatches and findInList should use pointer variables to access the array elements, instead of array indices. The point is to play with pointers.

Inside findInList:

- It is easiest to search for the target through the sorted list. Set pointers at the beginning of each list. Then check to see if the value in the sorted list matches the target. If it does, increment nfound, and check to see if the same position in the unsorted list also matches the target, and if so, set the value for index. Then increment both pointers and repeat, until the data value in the sorted array is larger than the target, or until the end is reached.

Inside of main:

- Read in integer input from **standard input** and store these values into a dynamic array. This array is to grow in size if the array is full. 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, nor the total number of values contained in the input. **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 copyArray( ) function described above.

- Sort the copy array using the myFavoriteSort( ) function described above.

- Report the number of matches divided by the total number as a fraction and as a percentage, using the result of countMatches( ) and the example shown above.

- Read in integer input from standard input (again, the use of scanf() is expected) and for each of the values read in perform the findInList evaluation. Using the information returned/sent back from the search functions, **print out from main():**

  1. The target value,
  2. How many copies of the target appear in the list, and
  3. The first position in the sorted list in which it was found in the correct position.

  Repeat reading in integer values and searching the array until the terminal value of -999 is read in. The use of informative prompts AND descriptive result output is required for full credit. (Ask the user for input and report results using full sentences and proper grammar. Always specify units whenever relevant, though that may not apply to this assignment.)

  NOTE: Main should be the only function that does any input or output, including printing or scanning, except for diagnostic messages printed while debugging. None of the other functions listed here should read or write anything from/to the keyboard/screen. Failure to abide by this practice will cost points.

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 Memory Allocation

Dynamic Memory Allocation allows the space allocated to 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 *dynamicArray;
int allocated = 100;
dynamicArray = ( int * ) malloc ( allocated * sizeof( int ) );
```

This array can only hold 100 integers and is 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 = dynamicArray;
dynamicArray = ( int * ) malloc ( allocated * 2 * sizeof( int ) );
int i;
for ( i = 0 ; i < allocated ; i++ )
        dynamicArray [ i ] = temp[ i ];
free ( temp );
allocated = allocated * 2;
```

Note:

- Because the code above allocates new space, the existing data needs to be copied in to it.
- The old memory needs to be freed up (deallocated.) Failure to do so is a common error known as a "memory leak".
- For this assignment, start dynamic arrays at size 100, and double them as needed.
- **For this assignment you must use malloc( ). In particular, realloc( ) is forbidden.**

Running your C Program (not a C++ 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.)

Programming Style

Make sure your program is written in good programming style. This includes but is not limited to:

- In-line commenting
- Function header commenting
- File header commenting
- Meaningful and description variable names
- Proper use of indentation
- Proper use of blank lines
- Use of Functions

Program Submission
You are to submit the program via the proper Assignments link on Gradescope. If you create your program in multiple files (**which you should not need to do for this project**), you should create a zip file containing all your files and then submit the zip file on Gradescope. You should name your files with your net-id and project name. For example:

The file should be named as netid-proj1.c
The zip file (**if needed**) should be named as netid-proj1.zip

Creating a zip file (if needed)

1. Go the directory in which you have saved the files.
2. Select the multiple files and right click.
3. There should be a menu option to create a zip file. If you are using Mac/Linux you will see a menu option **Compress....** If you are using Windows you might need to install winzip to create a zip file

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

When testing programs, the use of redirection of standard input from a text file is often a good idea. 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, input redirection is probably sufficient.

- 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 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.