CS 137: Assignment #8

Due on Friday, Nov 25, 2022, at 11:59PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2022

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 12.
- <math.h> is not allowed
- Use Valgrind when testing.

Problem 1

This new search looks for data by guessing the position of the data searched using a specific formula. It can **only** perform on an array of numbers that have been **sorted ascending**. The alleged position as the data position sought (X) can be calculated using the following formula:

Pos = int (BB + ((X – A [BB]) / (A [BA] – A [BB])) * (BA – BB))

/* the division is real division then cast the result to int */

where:

X is the item to search in array A

BB is the index of the beginning of the range to be searched (initialized to 0)

BA is the index of the end of the range to be searched (initialized to length(A)-1)

Rules to follow in this particular order:

- If X is smaller than the first value in the range or bigger than the last value in the range, stop the search
- If the data in that position (Pos) is equal to the value of data sought (X), then the search is stopped and declared successful.
- Whereas if the value of data in that position (Pos) is not equal to the value of data sought (X), then:
    - if the value of the data sought (X) is smaller than the data value in that position (Pos), then the search is continued by changing the upper boundary(BA) of the search area to be to the left of the current position(Pos)
    - Whereas if the data value sought (X) is greater than the value of data in that position(Pos), then the search is continued by changing the lower boundary(BB) of the search area to be to the right of the current position (Pos).

Create a C program `newSearch.c` that consists of a function

```
bool newSearch(int arr[], int len, int x);
```

that has a sorted array of unique (no repetition) integers (`arr`) of length `len`, and a positive integer (`x`). The function returns `true`, if `x` exists in `arr`, `false` otherwise following the steps described at the start of this question. The function also prints messages as demonstrated in the following examples. Make sure to copy-paste strings provided in `A8strings.txt` to avoid any typos.

You are to submit this file containing only your implemented function (that is, you must delete the test cases portion/main function). However, **you must keep the required included libraries.**

The sample code for testing is below.

```
1. #include <stdio.h>
2. #include <stdbool.h>
3. #include <assert.h>
4.
5. bool newSearch(int arr[], int len, int x);
6.
7. int main(void)
8. {
9.          int a[1] = {14};
10.         assert(!newSearch(a,1,10));
11.         printf("\n\n");

12.         assert(newSearch(a,1,14));
13.         printf("\n\n");

14.         int b[13] = {1,4,5,7,12,23,44,67,89,100,120,121,122};
15.         assert(!newSearch(b,13,0));
16.         printf("\n\n");

17.         assert(!newSearch(b,13,150));
18.         printf("\n\n");

19.         assert(newSearch(b,13,4));
20.         printf("\n\n");

21.         assert(newSearch(b,13,121));
22.         printf("\n\n");

23.         assert(newSearch(b,13,23));
24.         printf("\n\n");

25.         int c[20] = {-10,-
   4,5,7,12,23,44,67,89,100,120,121,122,200,210,222,300,301,303,500};
26.         assert(!newSearch(c,20,55));
            printf("\n\n");

            assert(newSearch(c,20,12));
27.         printf("\n\n");

28.         return 0;
29.  }
```

The output of the code above:

```
start with the range 14 to 14
10 not in the range between 14 and 14


start with the range 14 to 14
14 was found in position 0


start with the range 1 to 122
0 not in the range between 1 and 122


start with the range 1 to 122
150 not in the range between 1 and 122


start with the range 1 to 122
move to search in the range between 4 and 122
4 was found in position 1


start with the range 1 to 122
121 was found in position 11


start with the range 1 to 122
move to search in the range between 7 and 122
move to search in the range between 23 and 122
23 was found in position 5
```

```
start with the range -10 to 500
move to search in the range between 7 and 500
move to search in the range between 23 and 500
move to search in the range between 44 and 500
move to search in the range between 67 and 500
55 not in the range between 67 and 500


start with the range -10 to 500
move to search in the range between -4 and 500
move to search in the range between 5 and 500
move to search in the range between 7 and 500
move to search in the range between 12 and 500
12 was found in position 4
```

Problem 2

Create a C program `printsort.c` that consists of the function

```
void printsorted(char* word, int len);
```

that prints only the English letters sorted decreasingly by the number of times they appear in the string and for ties sorted by alphabetical order. The printed string should be ended with `\n` character.

For example, `"hello"` would be printed as `"lleho"`.

Assume all English letters are in lowercase.

You are to submit this file containing only your implemented function (that is, you must delete the test cases portion). However, **you must keep the required included libraries.**

IMPORTANT: make sure you free any temporarily allocated memory by your implementation. We will test for memory leak!

Sample code for testing:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
  printsorted("hello world!!!", strlen("hello world!!!"));
  return 0;
}
```

Output:

`llloodehrw` // ended by `\n` character

Problem 3

Create a C program `sort.c` that consists of the function

```
void sort(void *arr, int n, size_t elemsize,
          bool (*compare)(const void *a, const void *b),
          void (*swap)(void *a, void *b));
```

that sorts the array `arr` of length `n`, where each element in arr is of size `elemsize`, in a certain order by applying the function pointed at by `compare` function pointer, and swap the values by using the function that `swap` function pointer is pointing at. You want to write a solution that only when compare(p1,p2) is true and p1 comes after p2 in the original array, then it swaps.

Check the tests below to understand the role of the compare function in sorting the data.

You must NOT use any pre-defined C function for sorting, such as `qsort`.

You may use any algorithm defined in the course notes.

You are to submit this file containing only your implemented function (that is, you must delete the test cases portion). However, **you must keep the required included libraries.**

Do not submit any function you defined for testing, such as pointCompare1 & 2 etc. and pointSwap below.

For this question, since we are using void pointers, you might get lots of warnings (not errors) when using `gcc`, thus you may use `gcc -w sort.c` to compile your solution and ignore the warnings about void pointers. We will do the same when testing your solution on marmoset.

Tip: to calculate the address of the value in index `j` in array `void* arr` where the size of each value is `elemsize`:

```
((char *) arr) + j * elemsize
```

The sample code for testing is below.

```
1. #include <stdio.h>
2. #include <stdbool.h>
3.
4. struct point
5. {
6.   int x;
7.   int y;
8. };
9.
10.  void sort(void *arr, int n, size_t elemsize, bool (*compare)(const
11.         void *a, const void *b), void (*swap)(void *a, void *b));
12.
13.  bool pointCompare1(struct point *p1, struct point *p2)
14.  {
```

```
15.         if (p1->x < p2->x) return true;
16.         else return false;
17. }
18.
19. bool pointCompare2(struct point *p1, struct point *p2)
20. {
21.         if (p1->x > p2->x) return true;
22.         else return false;
23. }
24.
25. bool pointCompare3(struct point *p1, struct point *p2)
26. {
27.         if (p1->x <= p2->x) return true;
28.         else return false;
29. }
30.
31. bool pointCompare4(struct point *p1, struct point *p2)
32. {
33.         if (p1->x >= p2->x) return true;
34.         else return false;
35. }
36.
37. void pointSwap(struct point *p1, struct point *p2)
38. {
39.          struct point temp = *p1;
40.         *p1=*p2;
41.         *p2=temp;
42. }
43.
44. int main (void)
45. {
46.   struct point points[4] = {{10,5},{0,0},{-4,-5},{0,10}};
47.   sort(points,4, sizeof(struct point), pointCompare1, pointSwap );
48.   for (int i=0; i<4; i++)
49.   printf("%d %d\n", points[i].x, points[i].y);
50.   printf("\n\n");
51.
52.   struct point points2[4] = {{10,5},{0,0},{-4,-5},{0,10}};
53.   sort(points2,4, sizeof(struct point), pointCompare2, pointSwap );
54.   for (int i=0; i<4; i++)
55.         printf("%d %d\n", points2[i].x, points2[i].y);
56.   printf("\n\n");
57.
58.   struct point points3[4] = {{10,5},{0,0},{-4,-5},{0,10}};
59.   sort(points3,4, sizeof(struct point), pointCompare3, pointSwap );
60.   for (int i=0; i<4; i++)
61.         printf("%d %d\n", points3[i].x, points3[i].y);
62.   printf("\n\n");
63.
64.   struct point points4[4] = {{10,5},{0,0},{-4,-5},{0,10}};
65.   sort(points4,4, sizeof(struct point), pointCompare4, pointSwap );
66.   for (int i=0; i<4; i++)
67.         printf("%d %d\n", points4[i].x, points4[i].y);
68.   printf("\n\n");
69.   return 0;
70. }
```

The output of the code above:

```
-4 -5
0 0
0 10
10 5


10 5
0 0
0 10
-4 -5


-4 -5
0 10
0 0
10 5


10 5
0 10
0 0
-4 -5
```