CMPT125, Fall 2021

Homework Assignment 3
Due date: Wednesday, November 5, 2021, 23:59

You need to implement the functions in ***assignment3.c***.
Submit only the ***assignment3.c*** file to Canvas.

Solve all 4 problems in the assignment.

The assignment will be graded both **automatically** and by **reading your code**.
.
Correctness: Make sure that your code compiles without warnings/errors,
and returns the required output.

Readability: Your code should be readable. Add comments wherever is necessary.
If needed, write helper functions to break the code into small, readable chunks.

Compilation: Your code MUST compile in CSIL with the Makefile provided.
If the code does not compile in CSIL the grade on the assignment is 0 (zero).
Even if you can't solve a problem, make sure it compiles.

Helper functions: If necessary, you may add helper functions to the assignment1.c file.

main() function: do not add main(). Adding main() will cause compilation errors, as the main() function is already in the test file.

Using printf()/scanf(): Your function should have no unnecessary printf() statements. They may interfere with the automatic graders.

Warnings: Warnings during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Testing: An example of a test file is included.
Your code will be tested using the provided tests as well as additional tests.
You are *strongly encouraged to write more tests* to check your solution is correct, but you don't need to submit them.

You need to implement the functions in ***assignment3.c***.
If necessary, you may add helper functions to the assignmen32.c file,
but you should not add main() to the file.
Submit only the ***assignment3.c*** file to Canvas.

Question 1 [30 points]
In this question you will implement QuickSort of ints that sorts the numbers in the non-decreasing order.

[15 points] Implement the rearrange function used for QuickSort using the O(n) time algorithm we saw in class with two pointers.
The function gets as input an array, and index of the pivot.
The function rearranges the array, and returns the index of the pivot after the rearrangement.

```
int rearrange(int* A, int n, int pivot_index);
```

[10 points] Implement the QuickSort algorithm.

- For n<=2 the algorithm just sorts the (small) array *(smaller number first).*
- For n>=3 the algorithm uses the rearrange function with the pivot chosen to be the median of A[0], A[n/2], A[n-1].

```
void quick_sort(int* A, int n);
```

Question 2 [40 points]
Write a function that gets an array of points (see definition in assignment3.) and sorts the array using qsort().
Given two points $a=(a_x,a_y)$ and $b=(b_x,b_y)$ we compare them as follows:

1) *if $(a_x)^2+(a_y)^2 < (b_x)^2+(b_y)^2$, then a should come before b in the sorted array.*
2) *if $(a_x)^2+(a_y)^2 = (b_x)^2+(b_y)^2$, then we compare the points by the x coordinate.*

Remark: For a point $a=(a_x,a_y)$ the quantity $((a_x)^2+(a_y)^2)^{½}$ is the distance of a from the (0,0). That is, we sort the points according to their distance to (0,0), and if for points at the same distance, then we sort them according to the first coordinate.

You will need to implement the comparison function, and apply qsort() on the array with this comparison function.

<u>For example</u>:

- *Input: [(3,2), (7,1), (1,1), (3,4), (5,0), (7,1)]*
- *Expected output: [(1,1), (3,2), (3,4), (5,0), (7,1), (7,1)]*

<u>Explanation</u>:
(1,1) is first because $1^2+1^2=2$ is the smallest
(3,2) is next because $3^2+2^2 = 13$ is the second smallest
Both (3,4) and (5,0) have the same sum of squares, 25, but 3<5 so (3,4) comes before (5,0)
(7,1) is lest because $7^2+1^2=50$ is the largest

***Note that we do allow some points in the array to be equal.*

```
void sort_points(point* A, int length);
```

Question 3 [30 points]

Write the following three functions:

*a) The function g*ets an array A of length n of ints, and a boolean predicate pred.
It returns the smallest index i such that pred(A[i])==true.
If no such element is not found, the function returns -1.

```
int find(int* A, int n, bool (*pred)(int));
```

*b) The function g*ets an array A of length n of ints, and a function f.
It applies f to each element of A.

```
void map(int* A, int n, int (*f)(int));
```

c) The function gets an array A of length n of ints, and a function f. The function f gets 2 ints and works as follows:

1. Start with accumulator = A[0]
2. For i=1...length-1 compute accumulator=f(accumulator, A[i])
3. Return accumulator

For example, if f computes the sum of the two inputs, then reduce() will compute the sum of the entire array.

```
int reduce(int* A, int n, int (*f)(int,int));
```