CMPT125, Fall 2020

Homework Assignment 3

Due date: Wednesday, November 11, 2020, 23:59

For this assignment you need to:

1) Implement the functions from Q1 and Q2 in **assignment3.c**.
2) Implement the functions from Q3 in **stack.c**
3) Implement the struct from Q4 in the header file **queue.h**
4) Implement the functions from Q4 in **queue.c**

Submit all four files to CourSys.

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(): 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.

Good luck!

Question 1 [25 points]

- *Implement this part in assignment3.c*

Recall the Quick Sort algorithm and its rearrange function.

[15 points] Implement the rearrange function.

```
// used for QuickSort:
// the function rearranges the elements, and
// returns the index of the pivot after the rearrangement
int rearrange(int* array, int n, int pivot_index);
```

[10 points] Implement the QuickSort algorithm. Use your rearrange function as a subroutine. For the pivot choose 3 random elements, and let the pivot be their median.

```
// QuickSort algorithm
void quick_sort(int* array, int n);
```

Question 2 [25 points]

- *Implement this part in assignment3.c*

Recall the MergeSort algorithm and its merging function.

[15 points] Implement the merging function.

```
// used for MergeSort:
// the function gets an array of length n
// and the index of the midpoint.
// the assumption is that ar[0...mid-1] is sorted
// and ar[mid...n-1] is sorted
// the function merges the two halves into a sorted array
void merge(int* array, int n, int mid);
```

[10 points] Implement the MergeSort algorithm. Use your merge() function as a subroutine.

```
// MergeSort algorithm
void merge_sort(int* array, int n);
```

Question 3 [30 points (15 points each item)]

- *Implement this part in stack.c.*

- *In this question you may only use the following functions to access the (unbounded) stack of ints. The interface for stack is provided in stack.h.*
- *The stack functions are implemented in stack.c, but you should not make assumptions about the exact implementation details. You may use only the signatures of the functions from stack.h.*

```
typedef struct {
 // not known
} stack3_t;

// creates a new stack
stack3_t* stack_create();

// pushes a given item to the stack
void stack_push(stack3_t* s, int item);

// pops the top element from the stack
// Pre condition: stack is not empty
int stack_pop(stack3_t* s);

// checks if the stack is empty
bool stack_is_empty(stack3_t* s);

// frees the stack
void stack_free(stack3_t* s);
```

a) *Write a function that gets a stack and returns the number of elements in it. When the function returns the stack must be in its initial state.*

```
// returns the number of elements in the stack
int stack_length(stack3_t* s)
```

b) *Write a function that gets two stacks and checks if all elements in s1 are* ***strictly less*** *than all elements in s2. When the function returns the stacks must be in their initial state. (If s1 or s2 is empty, the function returns true)*

```
// checks if all elements in s1 are < than all elements in s2
bool stack_strictly_less(stack3_t* s1, stack3_t* s2)
```

Question 4 [30 points - struct 6 points, each function 4 points]

- *Implement this part in queue.h and queue.c*

In this question you need to implement the ADT queue of ints.
Use the idea with 2 pointers we saw in class.

Implement the struct in the header file queue.h.
Implement the functions in queue.c.

```
typedef struct {
      // implement me
} queue_t;

// creates a new queue
// if malloc fails, returns NULL
queue_t* queue_create();

// add a given item to the queue
// if everything works ok, returns same q as the input
// if malloc fails, returns NULL
queue_t* enqueue(queue_t* q, int item);

// removes an element from the queue
// Pre condition: queue is not empty
int dequeue(queue_t* q);

// checks if the queue is empty
bool queue_is_empty(queue_t* q);

// returns the length of the queue
// if q==NULL, returns -1
int queue_length(queue_t* q);

// frees the queue
// if q==NULL, returns -1
void queue_free(queue_t* q);
```