

## CMPT125, Fall 2019

### Homework Assignment 3

Due date: Friday, November 1, 2019, 23:59

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

Solve all 3 problems in the assignment. Problems 2 and 3 have two questions each.

The assignment will be graded both **automatically** and by **reading your code**.

**Do not!** add `main()` to your `assignment3.c` file. This will cause compilation errors as the `main()` function will be in the test file.

**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. Make sure that your code compiles without warnings/errors, 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

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

**Testing:** Test your code. An example of a test file is included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

### Question 1 [20 points].

*Recall the Quick Sort algorithm and its rearrange function. 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* ar, int n, int pivot_index)
```

*Test your implementation using the provided Quick Sort function.*

## Question 2 [40 points].

*In this question you may only use the following functions to access the stack. You should not make assumptions about the exact implementation details.*

```
typedef struct {  
    // not known  
} stack_t;  
  
// creates a new stack  
stack_t* stack_create();  
// pushes a given item to the stack  
void stack_push(stack_t* s, int item);  
  
// pops the top element from the stack  
// Pre condition: stack is not empty  
int stack_pop(stack_t* s);  
  
// checks if the stack is empty  
bool stack_is_empty(stack_t* s);  
  
// frees the stack  
void stack_free(stack_t* s);
```

- a) *Write a function that gets two stacks and checks if they are equal (i.e., have the same elements in the same order). When the function returns the stacks must be in their initial state.*

```
// checks if the two stacks are equal  
bool stack_equal(stack_t* s1, stack_t* s2)
```

- b) *Write a function that gets a stack and reverses the order of the elements in it.*

```
// reverse stack  
void stack_reverse(stack_t* s)
```

### Question 3 [40 points].

*In this question you may only use the following functions to access the queue. You should not make assumptions about the exact implementation details.*

```
typedef struct {  
    // not known  
} queue_t;  
  
// creates a new queue  
queue_t* queue_create();  
  
// add a given item to the queue  
void 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);  
  
// frees the queue  
void queue_free(queue_t* q);
```

- a) *Write a function that gets two queues and checks if they are equal (i.e., have the same elements in the same order). When the function returns the queues must be in their initial state.*

```
// checks if the two queues are equal  
bool queue_equal(queue_t* q1, queue_t* q2)
```

- b) *Write a function that gets a queue and makes another copy of it with the same content (i.e., in the end both will have the same elements in the same order). When the function returns the original queues must be in its initial state.*

```
// copies queue  
queue_t* queue_copy(queue_t* orig)
```