MTE 140 - Spring 2009. Project 2.

Stacks and Queues. Due Date: Friday June 12, 11:59am.

Hanan Ayad

1 Project Description

In this project, you will implement the following ADTs: a stack using using a dynamically resizable array and a queue using an array as a circular track. You are given the following source files:

1. The interface (header) file `stackADT.h`, which has the type definitions and the function declarations for the stack ADT.
2. An incomplete implementation file `dynamicStack.c`, for which you are required to implement the body of each of the functions, based on the dynamically resizable array representation.
3. The interface (header) file `queueADT.h`, which has the type definitions and the function declarations for the queue ADT.
4. An incomplete implementation file `circularTrackQueue.c`, for which you are required to implement the body of each of the functions, based on the circular array representation.

As part of your project, you are required to analyze the **actual size in bytes** of the memory used to store the stack and queue data structures (including all struct fields) as implemented in this project. Give your answer as a number, in the case of the queue, and an exact function of the array size limit denoted here as m, in the case of the stack. Write your analysis near the struct definitions. Answers may differ based on the underlying system (32-bit versus 64-bit systems).

2 Specifications of the Stack ADT Operations

The functions you will be implementing for the stack ADT are described as follows:

1. `Stack stack_create( unsigned int size )`
   This function takes as an argument the initial size limit of the array and allocates the required memory space of the stack. If `size` is 0, an initial size of 16 is used as a default. The function appropriately initializes the fields of the created empty stack. It returns a `Stack` type which is a pointer to `struct stackTag`.
2. `void stack_destroy( Stack S )`
   This function takes as an argument a Stack S and deallocates the memory space allocated for it.
3. `int stack_isEmpty( Stack S )`
   This function takes as an argument a Stack and returns 1 if the stack is empty, 0 if it has one or more items, and -1 if S is `NULL`.

4. `int stack_length( Stack S )`
   This function takes as an argument a Stack S, and returns the number of items in S. It returns -1 if S is `NULL`.

5. `int stack_push( stackItem value, Stack S )`
   This function takes as arguments a Stack S and a stackItem `value`. If S is not full, the value is pushed onto S. If S is full, the size limit of S is doubled and the item is pushed onto the resized stack S. It returns 1 if the item is pushed, 0 if resizing fails, and -1 if S is `NULL`.

6. `int stack_pop( stackItem *itemPtr, Stack S )`
   This function takes as arguments a Stack S and a pointer to a stackItem itemPtr. If S is not empty, the top item is removed from S. The popped item is copied into the item pointed to by itemPtr. If the number of items remaining in S after popping is 1/4 the size limit of the array and if the size limit is greater than the initial size, the size of the array is halved. It returns 1 if the item is popped, 0 if the stack was empty before pop, and -1 if any of its arguments is `NULL`.

7. `int stack_peek( stackItem *itemPtr, Stack S )`
   This function takes as arguments a Stack S and a pointer to a stackItem itemPtr. If S is not empty, the top item is copied into the space pointed to by itemPtr. It returns 1 of S is not empty, 0 if it is empty, and -1 of any of its arguments is `NULL`.

8. `void stack_print( Stack S )`
   This function takes as an argument a Stack S and prints the stack items sequentially, in the order from the top to the bottom of the stack.

3 Specifications of the Queue ADT Operations

The functions you will be implementing for the queue ADT are described as follows:

1. `void queue_initialize( Queue *Q )`
   This function takes as an argument a pointer Q to a Queue (which is a struct), and initializes its fields appropriately.

2. `int queue_isEmpty( Queue *Q )`
   This function takes as an argument a pointer Q to a Queue and returns 1 if the queue is empty, 0 if it has one or more items, and -1 if Q is `NULL`.

3. `int queue_isFull( Queue *Q )`
   This function takes as an argument a pointer Q to a Queue and returns 1 if the queue is full, 0 if it has one or more empty positions, and -1 if Q is `NULL`.

4. `int queue_length( Queue *Q )`
   This function takes as an argument a pointer Q to a Queue and returns the number of items in the queue. It returns and -1 if Q is `NULL`.

5. `int queue_enqueue( queueItem value, Queue *Q )`
   This function takes as arguments a queueItem `value` and pointer Q to a Queue. It inserts the value at the rear of the queue (after the last item). It returns 1 if the item is enqueued, 0 if the queue is full, and -1 if Q is `NULL`.

6. `int queue_dequeue( queueItem *itemPtr, Queue *Q )`
   This function takes as arguments a pointer `itemPtr` to an queueItem and a pointer Q to a Queue.

It removes the value at the front of the queue and copies it into the value pointed to by `itemPtr`. It returns 1 if the item is dequeued, 0 if the queue was empty before dequeueing, and -1 if any of its arguments is `NULL`.

7. `void queue_print( Queue *Q )`
   This function takes as an argument a pointer Q to a Queue and prints the queue items sequentially, in the order from the front to the rear of the queue.

4 Documentation and Programming Style

It is required that your name and ID appear at the top of all your submitted text files. Your code should be clear and understandable; you are required to use proper spacing and indentation to enhance the clarity of your code, use meaningful variable names, and useful comments. You should include comments for each function indicating its specifications. This includes a description of what it does, its parameters, return values, and any assumptions that it makes.

5 Submission Requirements

For the submission of this project, you are required to follow the steps outlined below:

1. Create a directory with the name `uwuserid_NNNNNNNNp1`, where `uwuserid` is your UW user ID, `NNNNNNNN` is your UW student ID number, and p1 denotes project 1.

2. Save all the sources files under this directory (do not create subdirectories). The files are: `stackADT.h`, `dynamicStack.c`, `queueADT.h`, `circularTrackQueue.c`.

3. Implement the project and generate all the object (.o) files. You are not required to sumit a `main` function for this project (but you will need to write one for yourself to test your functions. You can use the debugger to reduce the number of `printf` used in testing your functions).

4. Zip the directory into one .zip file with the same name. That is, `uwuserid_NNNNNNNNp1.zip`. To check that this step was done successfully, create a new `test` directory and unzip the `.zip` file under `test`. The unzipping should result in creating a subdirectory under test with the name `uwuserid_NNNNNNNNp1`, with all the project files being extracted under this subdirectory.

5. Submit the file `uwuserid_NNNNNNNNp1.zip` into the dropbox designated for project 2 on UW-ACE (under MTE 140).