# CS251 - Project 2: Stacks/Queues

**Out:** January 25, 2016 @ 9:00 am
**Due:** February 8, 2016 @ 9:00 am

## 1. Overview

Project 2 is split into three parts. In the first part, you will implement a stack class. Next you will use your stack to solve a word-search problem. Finally you will write a specialized type of queue called a double-ended queue, or deque.

## 2. Detailed Description

Part 1) Implement a stack that stores ordered pairs (i.e (2,3) or (5,8)).

- Your task in part 1a will be to implement a growable stack using a linked list format. See the lecture slides for details on this data structure.
- **You may not use C++ STL classes other than Pair for this section. Implement this on your own!**

Part 2) Use your stack to solve a word search problem.

In this part, we are going to solve a word-search problem, but with a bit of a twist. Normally in these puzzles, you are looking for a word that is written out in a straight line, such as below:

**S** A B C D
E **T** F G H
I J **A** K L
M N O **C** P
Q R S T **K**

However we want to solve something a bit trickier. In this project, we want to allow the words to be hidden in more interesting patterns, which may not be a line. Such as below:

A B C D E
**S** G **A** I J
K **T** M **C** O
P Q R **K** T
U V W X Y

Unlike above, we allow ourselves to move any direction we want from any letter we're at. Our objective in this case is to print out the location of the letters that make up the word we have found in ordered pair format. The solution for the above example is:

```
1 0
2 1
1 2
2 3
3 3
```

Thankfully, stacks provide a solution to problems like this! The rough algorithm to solve this problem works as follows:

- Begin by searching for the first letter in the word.
- If you find it, push the location onto your stack, and begin searching the surrounding cells for the next letter
- If you find the next letter, push that location onto the stack and repeat until finished.
- If, at any point, you do NOT find the letter you want, pop the last location off the stack and resume the search from the previous stage.
- If the word is found, print out the ordered pairs in the correct order
- If not, print "not found"

Additional notes for this section:

- Letters can be re-used in the search. In the given example DAD would be acceptable with the answer being:
  ```
  0 3
  1 2
  0 3
  ```

**3. Double-ended Queue**

In a standard queue, objects are inserted at the back and pulled from the front in a First In First Out fashion. There is a variation of a standard queue called a double-ended queue, often shortened to deque. As you may expect from the name, rather than inserting/deleting only on one side of the queue, deques allow insertion or deletion from both sides. The basic methods you will need to implement for this part are:

- enqueue_front: Add an item to the front of the queue
- enqueue_back: Add an item to the back of the queue
- dequeue_front: Take an item off the front of the queue
- dequeue_back: Take an item off the back of the queue
- print_array_size: Print the size of the allocated array

Your task is to implement a double-ended queue using the array-doubling strategy that was also used in the stack. As in part 1, you cannot use C++ STL classes. You must write your own.
<u>**Start with an initial array size of 10**</u>

## 4. Input/Output Format

Input and output will be handled using standard input/output (cin and cout). The provided skeleton code will handle a most of this, but you will still need to fill in the TODO sections.

Line 1: either 1, 2, or 3 declaring which part of the project is being tested.

For project part 1 the input is as follows:

- An integer n that represents the number of lines to follow
- n lines of the following format
    - The character ‘i’ (for insert) followed by 2 integers. Push these onto the stack as an ordered pair
    - The character ‘p’ (for pop). Pop an ordered pair and print the integers separated by a space. If the stack is empty print the string “empty”. (Lowercase and no punctuation please!)

For project part 2 the input is as follows:

- A line containing 2 integers (n,m) that describe the size of the word search field

- n lines each containing m characters

- one line containing the word you are searching for

- Output is the list of ordered pairs that represents the location of the word. Print “not found” if the word is not found.

For project part 3 the input is as follows:

- An integer n that represents the number of lines to follow
- n lines of the following format
    - Either the character ‘e’ or ‘d’ representing enqueue or dequeue followed by another character ‘f’ or ‘b’ representing to do the operation on the front or back of the queue follow by the integer to use in the operation
    - The character ‘s’, in which case you print the allocated size of the deque
    - All dequeued objects should be printed.

<u>**For this project, you can assume that all input we give you is well-formatted and valid.**</u>

Sample test cases:

| **Input** | **Expected output** |
|---|---|
| 1<br>5<br>i 5 2<br>i 4 3<br>p<br>p<br>p | 4 3<br>5 2<br>empty |
| 2<br>5 5<br>A B C D E<br>S G A I J<br>K T M C O<br>P Q R K T<br>U V W X Y<br>STACK | 1 0<br>2 1<br>1 2<br>2 3<br>3 3 |
| 3<br>8<br>e b 1<br>e b 2<br>e f 3<br>d b<br>d f<br>e b 5<br>d f<br>s | 2<br>3<br>1<br>10 |

**5. Grading and Programming Environment**
Assignments will be tested in a linux environment. You will be able to work on the assignments using the linux workstations in HAAS and LAWSON (use your username and password). Compilation will be done using g++ and makefiles. You must submit all the source code as well as the Makefile that compiles your provided source code into an executable named “program”.

Your project must compile using the standard g++ compiler (v 4.9.2) on data.cs.purdue.edu. For convenience, you are provided with a template Makefile and C++ source file. Note some latest features from C++14 are not available in g++ 4.9

The grading process consists of:

1. Compiling and building your program using your supplied makefile.
2. The name of produced executable program must be “program” (must be lowercase)
3. Running your program automatically with several test input files we have pre-made according to the strict input file format of the project and verifying correct output files thus follow the above instruction for “stdout” output precisely – **do not “embellish” the output with additional characters or formatting** – if your program produces different output such as extra prompt and space, points will be deducted. (It is fine however if you print your extra output to stderr if it helps in debugging.)
4. Inspecting your source code.

Input to the programming projects will be via “stdin” and output must be to “stdout”. For example, in command line form, your program will be run as:

```
program < input-test1.txt > output-test1.txt
```

The file output-test1.txt will be tested for proper output.

**Important:**
1. If your program does not compile, your maximum grade will be 50% of the grade (e.g., 5 out of 10) -- no exceptions.
2. Plagiarism and any other form of academic dishonesty will be graded with 0 points as definitive score for the project and will be reported at the correspondent office.

**Submit Instructions:**
The homework must be turned in by the due date and time using the turnin command. Follow the next steps:

1. Login to data.cs.purdue.edu (you can use the labs or a ssh remote connection).
2. Make a directory named with your username and copy your solution (**makefile, all your source code files including headers and any other required additional file**) there.
3. Go to the upper level directory and execute the following command:
   **turnin -c cs251 -p project2 your_username**
   (**Important:** previous submissions are overwritten with the new ones. Your last submission will be the official and therefore graded).
4. Verify what you have turned in by typing **turnin -v -c cs251 -p project2**
   (**Important:** Do not forget the **-v** flag, otherwise your submission would be replaced with an empty one)

**Overview / additional notes:**

- Part 1: Implement a stack using linked list format
- Part 2: Use your stack to solve a word search problem
- Part 3: Implement a deque using the array-doubling strategy

- You may **NOT** use the C++ STL classes in your stack or queue implementations!

- We provide you with some skeleton code in this project. Its use is completely optional. You can also feel free to edit it as you wish. Your project will be graded correctly as long as it compiles with the make command and I/O functions correctly.