# AST20105 Data Structures & Algorithms

# Lab 6 – Stack and Queue

## A. Submission Details

In this lab, you are required to submit **TWO** C++ program to solve the given problem shown in the section "Exercise". To make the program implementation easier, you are suggested to use Visual Studio 2012 or later version instead of doing it using pencil and paper. After you have completed the task, submit your program files (i.e. StackMain.cpp & QueueMain.cpp for this lab) via Canvas on or before November 6, 2016. For details, please refer to the following:

You are reminded to double-check your solutions to verify everything is correct before submission. You are also required to put your name, student ID, and lab number at the beginning of your source files.

**Important: You are only able to submit your work once.**

## B. Objective

The objective of this lab is to give you a revision on Stack and Queue data structure. Please note that all the concepts reviewed during this lab are very useful for your further study. So, you are highly encouraged to pay attention to what your lab instructor tells you. Apart from this, you will be asked to work on a lab question so as to help you get familiar with Stack and Queue application.

The following gives greater details about this lab.

Part 1: A quick review of features and different implementation of Stack and Queue.
Part 2: Lab question about application of Stack and Queue.

## C. Review of concepts

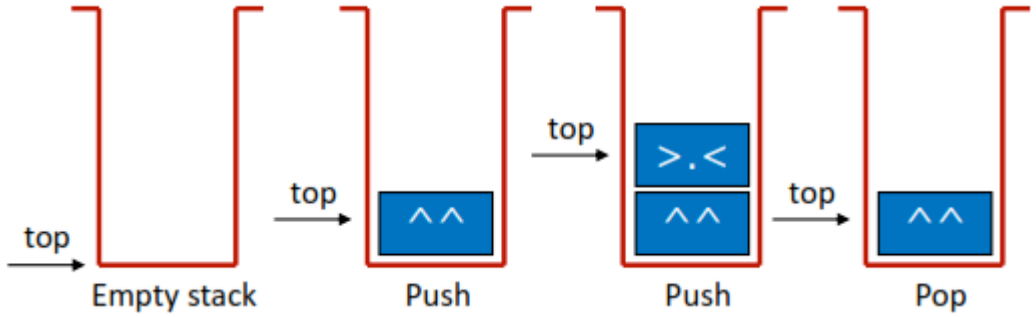
### 1. Stack:

Recall, a stack is a list that can handle a collection of elements, but with Last in, First out or First in, Last out restriction.

- Element can only be inserted and deleted at ONE END (i.e. the top of the list)
- Last inserted element will be the first to be examined or deleted.
- First inserted element will be the last to be examined or deleted.

**Fundamental operations:**

1. Push: Insert an element to the top of the stack.
2. Pop: Delete an element from the top of the stack.
3. Top: Examine the element at the top of the stack.
   Note: The element at the top is NOT DELETED.




**Stack implementations:**

Stack can be implemented with (1) array or (2) linked list. Stack implemented using array has fixed size, while stack implemented using linked list has flexible size and will never be full.

**StackArr class:**

```
class StackArr
{
  private:
     int maxTop;
     int stackTop;
     double* values;
  public:
     StackArr(int size = 10);
     ~StackArr();
     bool isEmpty() const;
     bool isFull() const;
     double top() const;
     void push(const double& x);
     double pop();
     void displayStack() const;
};
```

**StackLL class:**

```
class SinglyList
{
   private:
     Node* head; // a pointer to the first node in the list
     friend class StackLL;
   public:
     SinglyList();     // constructor
     ~SinglyList();    // destructor
     // isEmpty determines whether the list is empty or not
     bool isEmpty();
     // insertNode inserts a new node at position "index"
     Node* insertNode(int index, double x);
     // findNode finds the position of the node with a given value
     int findNode(double x);
     // deleteNode deletes a node with a given value
     int deleteNode(double x);
     // displayList prints all the nodes in the list
     void displayList() const;
};

class StackLL : public SinglyList
{
   public:
     StackLL();
     ~StackLL();
     double top() const;
     void push(const double& x);
     double pop();
     void displayStack() const;
};
```

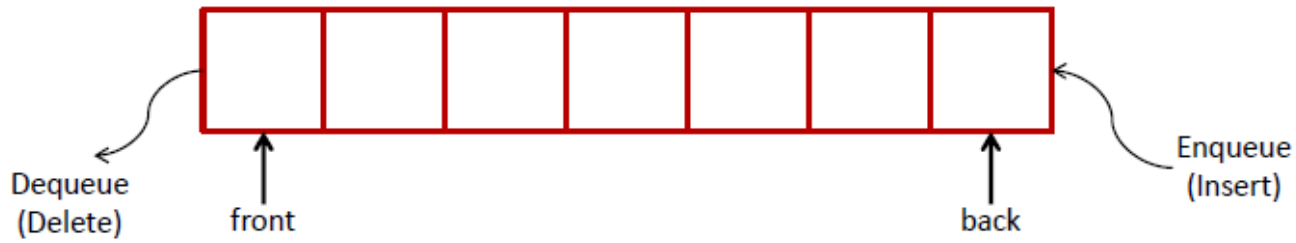
**2. Queue:**

Recall, a queue is a list that can handle a collection of elements, but with First in, First out or Last in, Last out restriction.

- Element can only be inserted at ONE END and be deleted at the OTHER END.
- First element will be the first to be examined or deleted.
- Last element will be the last to be examined or deleted.

**Fundamental operations:**

1. Enqueue: Insert an element to the back of the queue.
2. Dequeue: Delete an element at the front of the queue.




**Queue implementations:**

Queue can be implemented with (1) array or (2) linked list. Queue implemented using array has fixed size, while queue implemented using linked list has flexible size and will never be full.

**QueueArr class:**

```
class QueueArr
{
  private:
    int front;
    int back;
    int counter;
    int maxSize;
    double* values;
  public:
     QueueArr(int size = 10);
     ~QueueArr();
     bool isEmpty() const;
     bool isFull() const;
     bool enqueue(double x);
     bool dequeue(double& x);
     void displayQueue() const;
};
```

**QueueLL class:**

```
class QueueLL
{
   private:
      Node* front;
      Node* back;
      int counter;
   public:
     QueueLL();
     ~QueueLL();
     bool isEmpty() const;
     void enqueue(double x);
     bool dequeue(double& x);
     void displayQueue() const;
};
```

## D. Exercise

### Part 1: Stack

1. Create a New Project and give your project the name Lab6a.

2. Download the given source files StackArr.h and StackArr.cpp from Canvas and save them to your Lab6a folder. Also import them to your project.

3. Add a source file to your project, called StackMain.cpp and implement your program according to the following:

   i. Prompt the user to input a program filename.
   ii. Open the file and check if every right brace (i.e. }), bracket (i.e. ]), and parenthesis (i.e. )) in the file correspond to its left counterpart or not.
   iii. If the program file passed the checking, output "The code is correct". Otherwise, output "The code is incorrect".

   The algorithm for checking is as follows:

   - Make an empty stack
   - Read characters until the end of program file
     1. If the character is an opening symbol, push it onto the stack.
     2. If it is a closing symbol and if the stack is empty, output "Error: Empty stack ".
     3. Otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, output "Error: Not equal".
   - At the end of file, if the stack is not empty, output "The code is incorrect". Otherwise, output "The code is correct".

4. Use the provided files, testfile1.txt and testfile2.txt to test your program. The code file testfile1.txt is correct, while the code file testfile2.txt is incorrect.

**Useful code:**

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
     ifstream inputFile("testfile1.txt");
     while(!inputFile.eof())
     {
          string str;
          getline(inputFile, str);
          cout << str << endl;
     }
     inputFile.close();
     system("pause");
     return 0;
}
```

## Part 2: Queue

1. Create a New Project and give your project the name Lab6b.

2. Download the given source files StackArr.h, StackArr.cpp, QueueArr.h and QueueArr.cpp from Canvas and save them to your Lab6b folder. Also import them to your project.

3. Add a source file to your project, called QueueMain.cpp and implement your program according to the following:

    i. Prompt the user to input a string.
    ii. Change each uppercase letter to lowercase.
    iii. Place each letter both in a queue and onto a stack.
    iv. Verify whether the input string is a palindrome (i.e. a set of letters or numbers that is the same whether read from left to right or right to left).

4. The following shows a number of program's sample input / output sessions.

```
Input a string: aibohphobia
aibohphobia is a palindrome

Input a string: level
level is a palindrome

Input a string: desmond
desmond is not a palindrome
```

**Program Submission Checklist**

Before submitting your work, please check the following items to see you have done a decent job.

| **Items to be checked** | ☑ / ☒ |
|---|---|
| 1. Did I put my name and student ID at the beginning of all the source files? | ☐ |
| 2. Did I put reasonable amount of comments to describe my program? | ☐ |
| 3. Are they all in .cpp extension and named according to the specification? | ☐ |
| 4. Have I checked that all the submitted code are compliable and run without any errors? | ☐ |
| 5. Did I submit my lab assignment to Canvas? | ☐ |

**Marking scheme:**

| Graded items | Weighting |
|---|---|
| **1. Correctness of program** | 30% |
| **2. Readability** | 15% |
| **3. Re-usability** | 10% |
| **4. Documentation** | 15% |
| **5. Delivery** | 20% |
| **6. Efficiency** | 10% |
| **Total:** | 100% |

**Grading rubric:**

| Area of Assessment | Exceptional | Acceptable | Amateur | Unsatisfactory |
|---|---|---|---|---|
| **Correctness** | The program works and meets all the specifications. | The program works and produces the correct results and displays them correctly. It also meets most of the other specifications. | The program produces correct results but does not display them correctly. | The program is producing incorrect results. |
| **Readability** | The code is exceptionally well organized and very easy to follow. | The code is fairly easy to read. | The code is readable only by someone who knows what it is supposed to be doing. | The code is poorly organized and very difficult to read. |
| **Re-usability** | The code could be re-used as a whole or each routine could be re-used. | Most of the code could be re-used in other programs. | Some parts of the code could be re-used in other programs. | The code is not organized for re-usability. |
| **Documentation** | The documentation is well written and clearly explains what the code is accomplishing and how. | The document consists of embedded comment and some simple header documentation that is somewhat useful in understanding the code. | The documentation is simply comments embedded in the code with some simple header comments separating routines. | The documentation is simply comments embedded in the code and does not help the reader understand the code. |
| **Delivery** | The program was submitted on time. | The program was submitted within 1 day of the due date. | The code was submitted within 2 days of the due date. | The code was submitted more than 2 days overdue. |
| **Efficiency** | The code is extremely efficient without sacrificing readability and understanding. | The code is fairly efficient without sacrificing readability and understanding. | The code is brute force and un-necessarily long. | The code is huge and appears to be patched together. |

Adopted from California State University (Long Beach)

http://www.csulb.edu/colleges/coe/cecs/views/programs/undergrad/grade_prog.shtml

-End-