|  |
| --- |
|  |
| Assignment 1 |
| CPTN278 |
|  |
| **Paul Fahey** |
| **1/28/2012** |

|  |
| --- |
|  |

Contents

[Introduction 3](#_Toc315861792)

[Data Structures 3](#_Toc315861793)

[Stack 3](#_Toc315861794)

[Example Application 3](#_Toc315861795)

[Permitted Operations 3](#_Toc315861796)

[Order of Efficiency 3](#_Toc315861797)

[Implementation Options 3](#_Toc315861798)

[Queue 4](#_Toc315861799)

[Example 4](#_Toc315861800)

[Permitted Operations 4](#_Toc315861801)

[Order of Efficiency 4](#_Toc315861802)

[Implementation Options 4](#_Toc315861803)

[List 4](#_Toc315861804)

[Example 4](#_Toc315861805)

[Permitted Operations 5](#_Toc315861806)

[Order of Efficiency 5](#_Toc315861807)

[Implementation Options 5](#_Toc315861808)

[Conclusion 5](#_Toc315861809)

[References 6](#_Toc315861810)

# Introduction

The purpose of this document is to compare and describe the different types of data structures used in applications such as stacks, queues, and lists. The descriptions will include examples of real life problems that use these structures, permitted operations or behaviors of each structure, how efficient each one is, and the different implementation options available to each structure. This document will also showcase the ability to gain more knowledge and practice on creating a professional looking document as it would be in industry.

# Data Structures

This section goes into brief detail about the different data structures used in programming which are stacks, queues, and lists.

## Stack

In a stack, only one element can be accessed and from only one end denoted as the top. New elements get placed on top of existing elements and whatever the last or newest element that was pushed on is the only one accessible. Stacks follow a FILO or LIFO convention which means the first element in will be the last element out and the last element in will be the first one out since only one end exists.

### Example Application

* Converting a decimal number into a binary number (Stack)

### Permitted Operations

* Top – Can only see top item of stack at all times
* Pop - Items can be removed from only one end of the structure
* Push - Items can be added onto only one end of the structure
* isEmpty – Stack contains no items
* isNotEmpty –Stack contains items

### Order of Efficiency

Stacks are considered to have an order one efficiency. In other words, when calling any of the permitted operations for a stack it will take the same amount of effort or time to do that operation no matter the size of the stack.

### Implementation Options

* Arrays
* Pointers

## Queue

In a queue, the front element in the queue can only be accessed or modified and new elements can be added after the first element. Queues follow a FIFO or LILO convention which means the first element in will be the first element out or the last element in will be the last element out.

### Example

* A line of people going onto a roller coaster

### Permitted Operations

* First - Access to first element in the queue
* Enqueue – Element is inserted at the back of the queue
* Dequeue – Element is removed, accessed, or modified from front of the queue
* isEmpty – Queue contains no items
* isNotEmpty –Queue contains items

### Order of Efficiency

Queues much like stacks are also considered to have an order one efficiency, which again when calling any of the permitted operations for a queue it will take the same amount of effort or time to do that operation no matter the size of the queue.

### Implementation Options

* Arrays
* Pointers

## List

Unlike stacks and queues, a list doesn't have any restrictions when it comes to how elements are accessed. Its considered a "sequential order of elements, any of which can be accessed without restriction." A list follows an AIAO convention, which means anything in, anything out. Elements in a list can be inserted anywhere and can be removed from anywhere in the list.

### Example

* A database of any sort

### Permitted Operations

* Locate - Access any element on the list
* Add – Adds an item to the list
* Remove – Removes an item from the list
* isEmpty – List contains no items
* isNotEmpty – List contains items

### Order of Efficiency

Unlike stacks and queues, list don't follow a set unit of efficiency. In other words, when calling any of the permitted operations the amount of effort or time will vary from the size of the list at the time of calling an operation.

### Implementation Options

* Arrays
* Pointers

*All sub sections of each structure came from in class notes except those noted.*

# Conclusion

|  |  |
| --- | --- |
| Introduction | Gave an overview of the document content |
| Data Structures | Gave descriptions on the different data structures which are stacks, queues, and lists |
| Stack | Stack can only be accessed from one end |
| Queue | Queue's front element can only be accessed and elements can be added after the front element |
| List | Elements can be accessed anywhere at any time along with adding and removing anywhere |
| Example | Gave a real life example of stack, queues, and lists |
| Permitted Operations | All structures can add and remove elements along with checking to see if their full or empty, and each structure has its own permitted operation unique to itself |
| Order of Efficiency | Stacks and queues have a order 1 efficiency while a list has an order n efficiency |
| Implementation Options | Stacks, queues, and list are all implemented using arrays and pointers |

# References

*An Introduction to Data Structures with C++.* (2000). Retrieved January 2012, from Oracle

ThinkQuest: http://www.thinkquest.org

Bettle, H. (n.d.). Assignment 1 Description for CPTN278. 7.

Bettle, H. (n.d.). Structure Notes.

*Stack*. (n.d.). Retrieved January 2012, from Wikipedia: http://www.wikipedia.org/