DATA STRUCTURES & ALGORITHMS

STACKS AND QUEUES

issntt@nus.edu.sg

Outline

- **Stacks**
  - **What are Stacks?**
  - Using Stacks
  - Implementing Stacks
- Queues (Self Study)

Stack

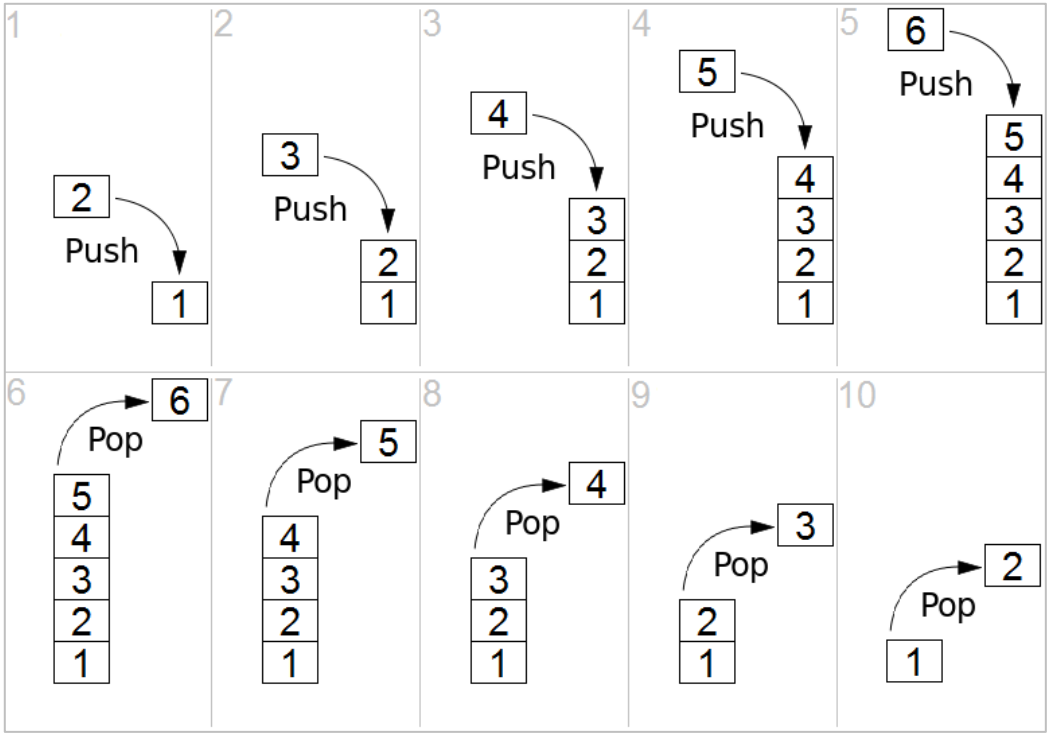
A stack is an ADT that stores a **collections** of items, which can be **inserted** and **removed only** at the **top**

The **last** item **added** is the **first** to be **removed** (**LIFO**: Last In, First Out)

By Foto: Jonn Leffmann, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=67631269

Stack - Main Operations

- ***Push(element)***: **add** an element **to** the **top** of the stack
- ***Pop()***: **remove** the element **at** the **top** of the stack
- ***Peek()***: **look** at the item on the **top** of the stack, but do **not remove** it



Maxtremus, CC0, via Wikimedia Commons

C# Generic `Stack` ADT

Method and Description
Push(T t) Inserts an object at the top of the stack
Peek() Returns the object at the top of the Stack<T> without removing it.
Count Gets the number of elements contained in the Stack<T>.
Clear() Removes all objects from the Stack<T>.
Pop() Removes and returns the object at the top of the Stack<T>.

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.stack-1

Outline

- **Stacks**
  - What are Stacks?
  - **Using Stacks**
    - **Reversing words**
  - Implementing Stacks
- Queues

Quiz

What is the output of the following method?

```
public static void UsingStack()
{
   Stack<string> myStack = new Stack<string>();
   myStack.Push("Luffy");
   myStack.Push(myStack.Pop());
   myStack.Push("Zoro");
   myStack.Push("Nami");
   myStack.Push("Usopp");
   myStack.Pop();
   myStack.Push(myStack.Peek());

   while (myStack.Count > 0)
   {
      Console.WriteLine(myStack.Pop());
   }
}
```

```
Nami
Nami
Zoro
Luffy
```

Application: Reversing Words

Given a word, write a method to return the **respective reverse word**

Input: yensid **Output**: disney

Input: leon **Output**: noel

You have just gained a new tool

Now ask yourself if you can use the tool to solve the problem

Reversing words

Push all characters of the given word **to a stack**, then **pop all characters** to form the return string

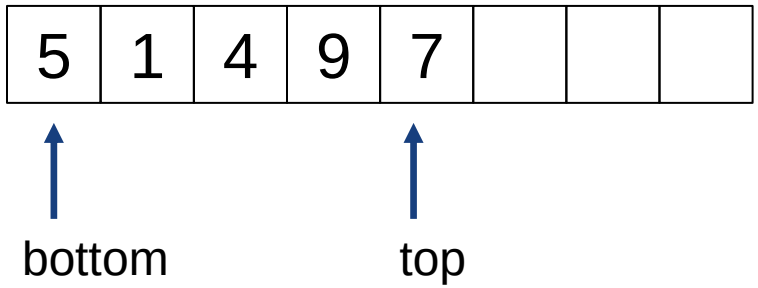
```
public static string Reverse(string word)
{
    Stack<char> myStack = new Stack<char>();
    foreach (char ch in word)
    {
        myStack.Push(ch);
    }

    string retStr = "";
    while (myStack.Count > 0)
    {
        retStr += myStack.Pop();
    }
    return retStr;
}
```

Outline

- **Stacks**
  - What are Stacks?
  - Using Stacks
  - **Implementing Stacks**
    - **Using Arrays**
    - **Using Linked Lists (Self-Study)**
- Queues (Self-Study)
  - What are Queues?
  - Using Queues
  - Implementing Queues

Using Arrays - Ideas

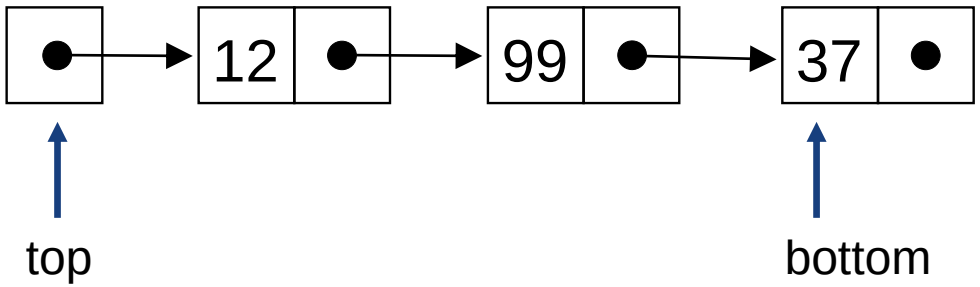
- Keep an array ***data*** to store the data/elements and a variable ***numElements*** to store the number of elements
- The **bottom** of the stack is at ***data[0]***
- The **top** of the stack is at ***data[numElements – 1]***
- ***Push()***: put at ***data[numElements]***
- ***Pop()***: remove from ***data[numElements – 1]***



Using Linked Lists - Ideas

Self study

- Store the items of the stack in a linked list
- The **top** of the stack is the **head node**
- The **bottom** of the stack is the **last node**
- ***Push()***: **add to** the **front** of the list
- ***Pop()***: **remove from** the **front** of the list



Outline

- Stacks
  - What are Stacks?
  - Using Stacks
  - Implementing Stacks
- **Queues (Self-Study)**
  - **What are Queues?**
  - Using Queues
  - Implementing Queues

Queues

Self study

A queue is an ADT that stores a **collections** of items, which can be **inserted** at **one end** and **removed** at **the other end**

The **first** item **added** is the **first** to be **removed** (**FIFO**: First In, First Out)

Image by Edgar Curious from Pixabay

Queue – Main Operations

Self study

- ***Enqueue (element)***: **add** an element **to the rear** of the queue
- ***Dequeue***: **remove** an element **from the front** of the queue

C# Generic Queue ADT

Self study

Method and Description
Enqueue(T t) Adds an object to the end of the Queue<T>
Peek() Returns the object at the beginning of the Queue<T> without removing it
Count Gets the number of elements contained in the Queue<T>
Clear() Removes all objects from the Queue<T>
Dequeue() Removes and returns the object at the beginning of the Queue<T>

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.queue-1

Outline

Self study

- Stacks
- **Queues**
  - What are Queues?
  - **Using Queues**
    - **Repeated Key Cipher**
  - Implementing Queues

Quiz

Self study

What is the output if running the following method?

```
public static void UsingQueue()
{
    Queue<string> myQueue = new Queue<string>();
    myQueue.Enqueue("Naruto");
    myQueue.Enqueue("Sasuke");
    myQueue.Enqueue("Madara");
    myQueue.Enqueue("Kakashi");
    myQueue.Enqueue("Itachi");
    myQueue.Dequeue();
    myQueue.Enqueue(myQueue.Peek());
    myQueue.Enqueue(myQueue.Dequeue());

    while (myQueue.Count > 0)
    {
        Console.WriteLine(myQueue.Dequeue());
    }
}
```

```
Madara
Kakashi
Itachi
Sasuke
Sasuke
```

Caesar Cipher

Self study

A type of **substitution cipher** in which each **letter** is **replaced** by a letter some **fixed number** of positions down the alphabet

E.g., a **left shift of 3**

- D is replaced by A
- E is replaced by B
- A is replace by X

THE QUICK BROWN FOX

QEB NRFZH YOLTK CLU

Very **easy** to **break**!

Repeated Key Cipher

Self study

- An improvement: the **shift** of a letter **depends on** the **position** of the letter
- A **repeating key** is a **sequence of integers** that **determine** how much each character is **shift**. For example, **2 5 3 9 1**
  - The **first** character is **shift by 2**, the **second** by **5**, the **next** by **3**…
  - When the key is exhausted, **start over** at the beginning of the key

Repeated Key Cipher

Self study

Following is an example

Message	D	a	t	a		S	t	r	u	c	t	u	r	e	s
Key	5	3	2	7		4	1	5	3	2	7	4	1	5	3
Encoded	I	d	v	h		W	u	w	x	e	a	y	s	j	v

Quiz

Self study

You are given the shift method as follows

```
static char ShiftCharacter(char ch, int position)
{
    if (char.IsUpper(ch))
    {
        return (char)((ch + position - 65) % 26 + 65);
    }
    else
    {
        return (char)((ch + position - 97) % 26 + 97);
    }
}
```

Quiz

Self study

Write a method, **given** a **message** as a string and a **repeating key** as an array of integers, **return** the respective **encoded message**

Input:
- ***Message:*** Data Structures
- ***Key:*** { 10, 20, 7, 3, 1, 6 }

Output: Nuad Tzbojwvxom

Input:
- ***Message:*** Education is the ticket to success
- ***Key:*** { 10, 20, 7, 3, 1, 6 }

Output: Oxbfbzsiu lt zry aldqon ar tamwlvt

Quiz Solution

Self study

Use a **queue** to **load** the **key**, and **dequeue** to **get** the **respective shift position** for each character

```
static string Encode(string message, int[] key) {
    Queue<int> keyQueue = new Queue<int>();
    string encoded = "";

    foreach (char ch in message) {
        LoadKeyWhenEmpty(keyQueue, key);
        if (ch != ' ') {
            int shiftPosition = keyQueue.Dequeue();
            encoded += ShiftCharacter(ch, shiftPosition);
        } else {
            encoded += ch;
        }
    }

    return encoded;
}
```

Quiz Solution

Self study

Load the shift positions into the queue at the **beginning or** whenever there is **no more shift positions**

```
static void LoadKeyWhenEmpty(Queue<int> keyQueue, int[] key)
{
   // At the beginning, the queue is also empty
   if (keyQueue.Count == 0)
   {
      foreach (int shiftPosition in key)
      {
         keyQueue.Enqueue(shiftPosition);
      }
   }
}
```

Outline

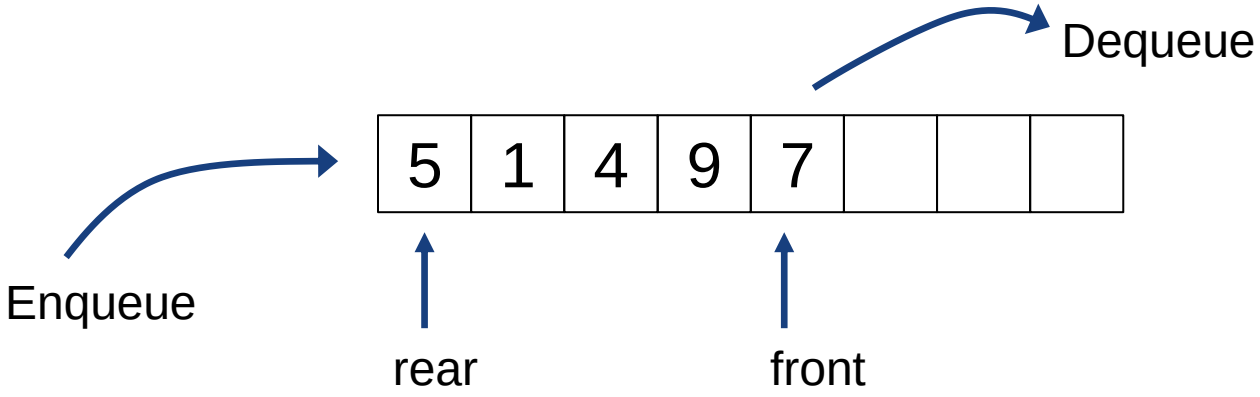
Self study

- **Stacks**
  - What are Stacks?
  - Using Stacks
  - Implementing Stacks
- **Queues**
  - What are Queues?
  - Using Queues
  - **Implementing Queues**
    - **Using Arrays**
    - **Using Linked Lists**

Using arrays - Ideas

Self study

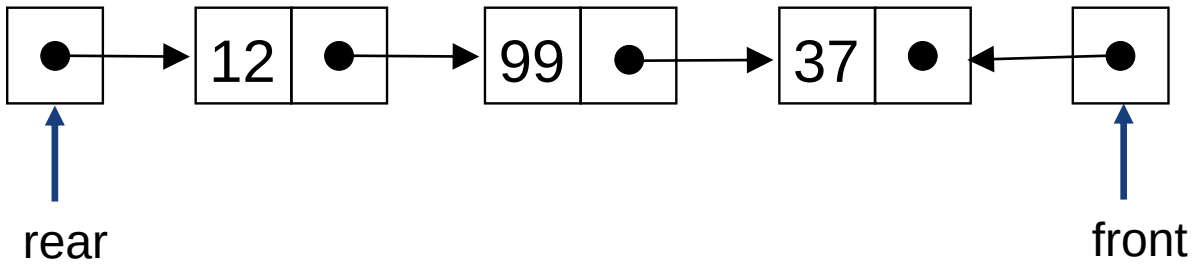
- Keep an array ***data*** to keep the data and a variable ***numElements*** to keep the number of elements
- ***Enqueue*** at ***data[0]*** and shift the rest of items in array down to make room
- ***Dequeue*** from ***data[numElements-1]***



Using Linked Lists – Ideas

Self study

- Keep references to two nodes: the **front** the **node at one end** and the **rear** to the **node at the other end**
- ***Enqueue*** by **adding** to the **rear** of the list
- ***Dequeue*** by **removing** from the **front** of the list



Readings

- Data structures and abstractions with Java, 4ed – Chapter 5, Stacks, *Frank M.Carrano and Timothy M. Henry*
- Data structures and abstractions with Java, 4ed – Chapter 10, section Queues, *Frank M.Carrano and Timothy M. Henry*
- C# Data Structures and Algorithms - Chapter 3, Stacks and Queues, section Stacks and section Queues*, by Marcin Jamro (2018)*