Lab

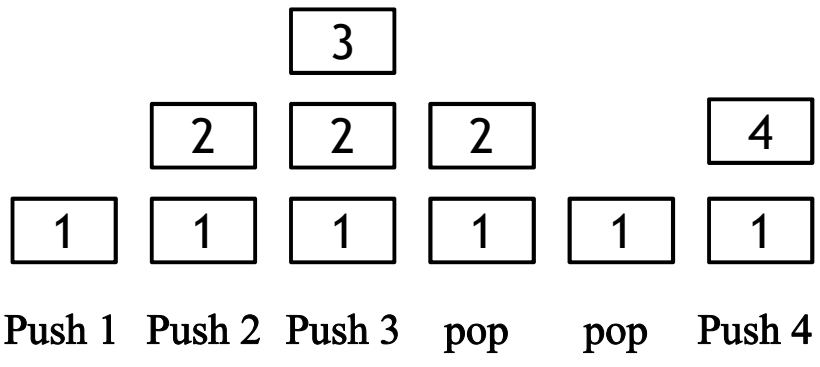
Stacks and Queues

Stacks

Stack

- We define a stack as a list of items that are accessible only from the end of the list, which is called the *top* of the stack.
- A stack is known as a Last-in, First-out **(LIFO)** data structure.

Example of Stack



The Stack Constructor Methods (1)

- There are three ways to instantiate a stack object.

1. The default constructor instantiates an empty stack with an initial capacity of 10 values.
   - Each time the stack reaches full capacity, the capacity is doubled.

The Stack Constructor Methods (2)

- The default constructor is called as follows:

```
Stack myStack = new Stack();
```

- A generic stack is instantiated as follows:

```
Stack<string> myStack = new Stack<string>();
```

The Stack Constructor Methods (3)

- The second Stack constructor method allows you to create a stack object from another collection object. For example, you can pass the constructor as an array and a stack is built from the existing array elements:

```
string[] names = new string[] {"Ahmed", "Khaled", "Mohammed"};
Stack nameStack = new Stack(names);
```


The Stack Constructor Methods (4)

- The third Stack constructor method by specify the initial capacity of the stack.

```
Stack myStack = new Stack(25);
```

Push(), Pop() and Peek

- The primary operations you perform with a stack are Push and Pop.
- Data is added to a stack with the **Push** method.
- Data is removed from the stack with the **Pop** method.
- The **Peek** method return the top value without remove it.

9

Count, Clear() and Contains

- The **Count** method return the number element in the stack.
- The **Clear** method removes all the items from a stack.
- The **Contains** method returns True if the element in the stack; or False otherwise

10

CopyTo()

- The **CopyTo** method copies the contents of a stack into an array.
- The array must be of type Object since that is the data type of all stack objects.
- The method takes two arguments: an array and the starting array index to begin placing stack elements. The elements are copied in LIFO order, as if they were popped from the stack.

ToArray()

- The ToArray method works in a similar manner.
- You cannot specify a starting array index position, and you must create the new array in an assignment statement.

Example Stack (1)

```
using System;
using System.Collections.Generic;
namespace ConsoleApplication1{
class Program{
static void Main(string[] args){
Stack<int> myStack = new Stack<int>();
myStack.Push(25);
myStack.Push(-30);
myStack.Push(33);
```


Example Stack (2)

```
Console.WriteLine("the top is: " +
  myStack.Peek());
  int value = myStack.Peek();
  if(value == 25){
  int del = myStack.Pop();
  Console.Write ("we delete" + del);}
  myStack.Push(100);
  myStack.Push(-200);
  myStack.Push(-3);
```

Example Stack (3)

```
count = myStack.Count;
Console.Write("output of stack : ");
foreach(var i in myStack){
  Console.Write(" " + i);}
Console.Write("\n the first array ");
```


Example Stack (4)

```
int[] FArr = new int[count];
myStack.CopyTo(FArr, 0);
for (int i = 0; i < FArr.Length; i++)
  {
    Console.Write(" " + FArr[i]);
  }
```

Example Stack (5)

```
Console.Write("\n the second array ");
myStack.Push(20);
count = myStack.Count;
int[] editArray = new int[count];
editArray = myStack.ToArray();
for(int i=0; i < count; i++)
    {
      editArray[i] *= 2;
    }
```


Example Stack (6)

```
foreach(var i in editArray)
        {
          Console.Write(" " + i);
        }
      }
    }
}
```

Example Stack (6)

```
The output will be:
the top is: 33
output of stack:
   -3 -200 100 33 -30 25
the first array
   -3 -200 100 33 -30 25
the second array
   40 -6 -400 200 66 -60 50
```


Assignment 5

- Write the following code,

1. Create a stack with double generic
2. First, enter number 25.3
3. Then, enter number 33.5
4. Then, add 88.9

Assignment 5

5. After that, Append -38
6. Copy to Array his name modArray
7. Modify All element in the modArray by multiple by 2
8. Clear All element in the stack
9. Create Stack again by copy the element in the modArray to it.
10. Print All element of the stack.

Queues

Queues

- A queue is a data structure where data enters at the rear of a list and is removed from the front of the list.
- Queues are used to store items in the order in which they occur.
- Queues are an example of a first-in, first-out (FIFO) data structure.

Example of Queue

A	Enqueue A
A B	Enqueue B
A B C	Enqueue C
B C	Dequeue
C	Dequeue

The Queue Constructor Methods (1)

- There are four ways to instantiate a queue object.

1. The default constructor instantiates an empty queue with an initial capacity of 32 values.
   - Each time the queue reaches full capacity, the capacity is doubled.

The Queue Constructor Methods (2)

- The default constructor is called as follows:

  Queue myQueue = new Queue();

- A generic queue is instantiated as follows:
  Queue<string> myQueue = new Queue<string>();

The Queue Constructor Methods (3)

- The second Queue constructor method allows you to create a Queue object from another collection object. For example, you can pass the constructor as an array and a queue is built from the existing array elements:

```
string[] names = new string[] {"Ahmed", "Khaled", "Mohammed"};
Queue nameQueue = new Queue(names);
```


The Queue Constructor Methods (4)

- The third Queue constructor method by specify the initial capacity of the Queue.

```
Queue myQueue = new Queue(25);
```

- The fourth Queue constructor method by change the growth factor as well. It is the second argument passed to the constructor, as in:

```
Queue myQueue = new Queue(32, 3);
```

Enqueue(), Dequeue() and

- The primary operations you perform with a queue are Enqueue and Dequeue.
- Data is added to a queue with the **Enqueue** method.
- Data is removed from the queue with the **Dequeue** method.
- The **Peek** method return the value of item at the beginning of the queue without remove it.

Example Queue (1)

```
using System;
using System.Collections.Generic;
namespace ConsoleApplication1{
class Program{
static void Main(string[] args){
Queue<int> myQueue = new Queue<int>();
myQueue.Enqueue(25);
myQueue.Enqueue(-30);
myQueue.Enqueue(33);
```

Example Queue (2)

```
Console.WriteLine("the top is: " +
  myQueue.Peek());
  int value = myQueue.Peek();
  if(value == 25){
  int del = myQueue.Dequeue();
  Console.Write ("we delete" + del);}
  myQueue.Enqueue(100);
  myQueue.Enqueue(-200);
  myQueue.Enqueue(-3);
```


Example Queue (3)

```
count = myQueue.Count;
Console.Write("output of Queue : ");
foreach(var i in myQueue){
  Console.Write(" " + i);}
Console.Write("\n the first array ");
```

Example Queue (4)

```
int[] FArr = new int[count];
myQueue.CopyTo(FArr, 0);
for (int i = 0; i < FArr.Length; i++)
  {
    Console.Write(" " + FArr[i]);
  }
```


Example Queue (5)

```
Console.Write("\n the second array ");
myQueue.Enqueue(20);
count = myQueue.Count;
int[] editArray = new int[count];
editArray = myQueue.ToArray();
for(int i=0; i < count; i++)
    {
      editArray[i] *= 2;
    }
```

Example Queue (6)

```
foreach(var i in editArray)
        {
          Console.Write(" " + i);
        }
      }
    }
}
```


Example Queue (6)

```
The output will be:
the top is: 25
we delete 25
output of Queue :
  -30 33 100 -200 -3
The first array
  -30 33 100 -200 -3
The second array
  -60 66 200 -400 -6 40
```

Assignment 6

- Write the following code,

1. Create a queue with int generic
2. First, enter number 36
3. Then, enter number -8
4. Then, add 55

Assignment 6

5. Check if the top is 36 remove it.
6. Copy to Array his name editArray
7. Modify the All element in the editArray by multiple by 10
8. Append all element in the editArray to the queue
9. Print All element of the queue.