CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

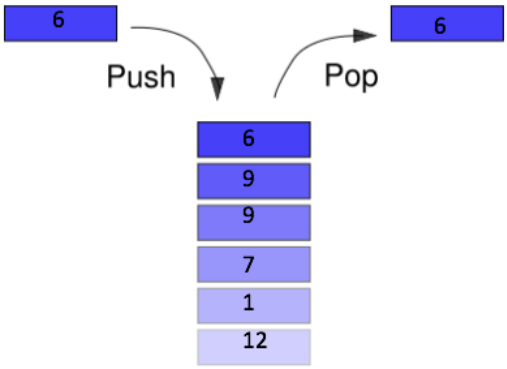
Computing@CIT

LINEAR DATA STRUCTURES AND ALGORITHMS.

ASSIGNMENT 1: DATA STRUCTURES - ADT MyStack

BACKGROUND.

The stack is a fundamental data-structure used extensively in algorithm design and program implementation. At an abstract level it can be described very simply, as it only allows for the addition (pushing) of new elements and removal (popping) of existing elements from the top of the stack. This description can be abbreviated to LIFO, which stands for Last-In-First-Out.



Problem Specification:

In this assignment we are going to specify the ADT MyStack with the following operations:

- **createEmpty():** It creates a new MyStack with no elements and initialises each of the attributes.
- **isEmpty():** It returns whether the stack is empty or not.
- **push(int element):** It places an integer item onto the top of the stack (if there is space for it). If there is no space, it prints on the screen an error message informing that the stack is full.
- **pop():** It removes the top integer from the stack (if the stack is non-empty) and returns that item. If the stack is empty, it prints on the screen an error message informing that the stack is empty and returns -1.
- **print():** It prints the items that are in the stack in a single line (the item on the top of the stack should appear in the left-most position). For the example of the figure above, the printed value on the screen would be: 6 9 9 7 1 12. If the stack is empty, it prints an error message to the screen confirming that the stack is empty.

ASSIGNMENT 1 – HINT 1 **(Week 4)**

A MyStack of int elements: Static Implementation.

BACKGROUND.

The unit on Canvas contains the following files:

- **MyMain.java:** This class tests the functionality of the stack's static implementation.
- **MyStack.java:** This interface specifies the ADT MyStack containing int elements.
- **MyStaticStack.java:** This class implements all operations of MyStack, using a static based implementation based on the following attributes:
    - private int items[];
    - private int numItems;
    - private int maxItems;

EXERCISE.

Implement the class MyStaticStack.java. IMPORTANT: only modify this .java file. Look for the comments: //TO-COMPLETE

ASSIGNMENT 1 – HINT 2 **(Week 5)**

A MyStack of int elements: Dynamic Implementation. Here there is no limit on the size of the stack.

BACKGROUND.

The unit on Canvas contains the following files:

- **MyMain.java:** This class tests the functionality of the stack's dynamic implementation.
- **MyNode.java:** This class models the concept of a single linked node containing an int *info* and a pointer to its next node *next*.
- **MyStack.java:** This interface specifies the ADT MyStack containing int elements. (Same file used last week in part 1)
- **MyDynamicStack.java:** This class implements all operations of MyStack, using a dynamic based implementation based on the following attributes:
    - private MyNode head;

EXERCISE.

Implement the class MyDynamicStack.java. IMPORTANT: only modify this .java file. Look for the comments: //TO-COMPLETE

ASSIGNMENT 1 – HINT 3 **(Week 6)**

A MyStack of generic type <T> elements: Dynamic Double-Linked Implementation.

BACKGROUND.

In this last hint, we extend the ADT MyStack with the capability of adding and removing elements from both front and back. That is why, we do not refer to the top in this hint, but to the head and tail. The head represents the element that is on top of the stack and the tail the element that is on the bottom of the stack. In the following figure, we can see the image where the red color represents the head and the blue color represents the tail. The new extended ADT MyStack contains the following operations:

- **createEmpty():** It creates a new MyStack with no elements.
- **isEmpty():** It returns whether the stack is empty or not.
- **first():** If the stack is non-empty, then it returns its first element (coloured in red in Figure 1). Otherwise, it returns an error message.
- **addByFirst():** Given a new item (coloured in red above in Figure 1), it adds it to the front/head of the stack.
- **removeByFirst():** If the stack is non-empty, then it removes its first element (coloured in red above in Figure 1). Otherwise, it returns an error message.
- **last():** If the stack is non-empty, then it returns its last element (coloured in blue in Figure 1). Otherwise, it returns an error message.
- **addByLast():** Given a new item (coloured in blue above in Figure 1), it adds it to the back/tail of the stack.
- **removeByLast():** If the stack is non-empty, then it removes its last element (coloured in blue below in Figure 1). Otherwise, it returns an error message.

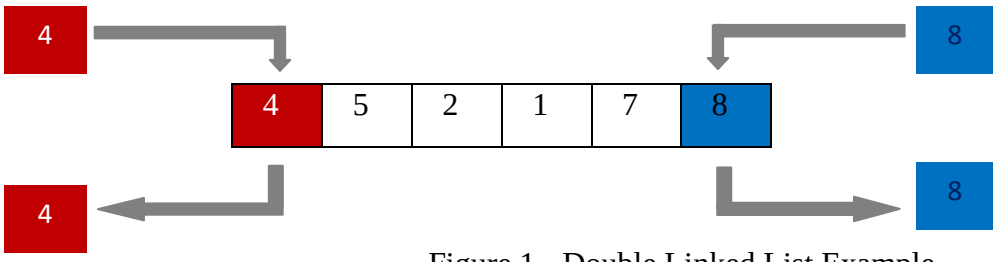
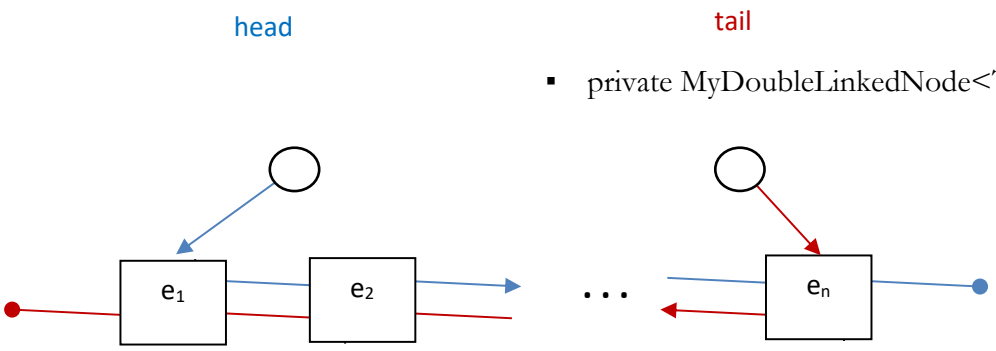


Figure 1 - Double Linked List Example

The folder **/src** contains the following files:

- **MyMain.java:** This class tests the functionality of the stack's dynamic implementation.
- **MyDoubleLinkedNode.java:** This class models the concept of a double linked node containing an element of type <T> *info*, a pointer to its next node *left* and a pointer to its previous node *right*.
- **MyStack.java:** This interface specifies the ADT MyStack containing elements of type <T>.
- **MyDoubleDynamicStack.java:** This class implements all operations of MyStack, using a dynamic based implementation based on the following attributes:
    - private MyDoubleLinkedNode<T> head;



    - private MyDoubleLinkedNode<T> tail;

EXERCISE.

Implement the class MyDoubleDynamicStack.java. IMPORTANT: only modify this .java file. Look for the comments: //TO-COMPLETE

MARK BREAKDOWN.

Assignment 1: 25 marks.

- Hint 1: 7.5 marks
  - public MyStaticStack(int m){...} 1.5 marks
  - public boolean isEmpty( ){...} 1.5 marks
  - public int pop( ){...} 1.5 marks
  - public void push(int element){...} 1.5 marks
  - public void print( ){...} 1.5 marks

- Hint 2 ➹ 7.5 marks.
  - public MyDynamicStack( ){...} 1.5 marks
  - public boolean isEmpty( ){...} 1.5 marks
  - public int pop( ){...} 1.5 marks
  - public void push(int element){...} 1.5 marks
  - public void print( ){...} 1.5 marks

- Hint 3 ➹ 10 marks.
  - public MyDoubleDynamicStack( ){...} 1 marks
  - public boolean isEmpty( ){...} 1 marks
  - public int first( ){...} 1 marks
  - public void addByFirst(int element){...} 1.5 marks
  - public void removeByFirst( ){...} 1.5 marks
  - public int last( ){...} 1 marks
  - public void addByLast(int element){...} 1.5 marks
  - public void removeByLast( ){...} 1.5 marks

To evaluate each function I will run it over some tests. The results are based on the performance of your code over these tests.

For each function, there are 5 possible scenarios:

A. The function passes all the test ➹ 100% of marks.
B. The function does not pass all tests by small details ➹ 70% of marks.
C. The function does not pass all tests by big details, but it compile and does not generate an exception (make the program crash) ➹ 35% of marks.
D. The function was barely attempted, it does not compile or it generates and exception (makes the program crash) ➹ 15% of marks.
E. The function was not attempted ➹ 0% of marks.

SUBMISSION DETAILS.

Deadline.

Sunday 8th of November, 11:59pm.

Submission Details.

Please submit the following on Canvas:

- Hint 1 File “MyStaticStack.java”.
- Hint 2 File “MyDynamicStack.java”.
- Hint 3 File “MyDynamicDoubleStack.java”.

Lab Demo.

A brief individual interview about the assignment will take place on our lab session on week 7 (Monday 9th - Friday 13th of November).
The demo is mandatory for the assignment to be evaluated.