STACKS

CS212: Data Structure

Stacks

- A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle.
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
- Inserting an item is known as “Pushing” onto the stack. “Popping” off the stack is synonymous with removing an item
- Used in Operating system to implement method calls, and in evaluating Expressions.

ADT Stack: Specification

Elements: The elements are of a generic type < Type>. (In a linked implementation an element is placed in a node)

Structure: the elements are linearly arranged, and ordered according to the order of arrival, most recently arrived element is called top.

Domain: the number of elements in the stack is bounded therefore the domain is finite. Type of elements: Stack

جاء البات ما عرفوا يحلوه
سألهم عن المتيدات اللي بالستاك

ADT Stack: Specification

الكوايزمت تعم ايور
المفروض هو اللي بيوها

Operations:

All operations operate on a stack S.

1. Method Push (Type e)
   requires: Stack S is not full. input: Type e.
   results: Element e is added to the stack as its most recently added elements. output: none.
2. Method Pop (Type e)
   requires: Stack S is not empty. input:
   results: the most recently arrived element in S is removed and its value assigned to e. output: Type e.
3. Method Empty (boolean flag)
   input: results: If Stack S is empty then flag is true, otherwise false. output: flag.

ADT Stack: Specification

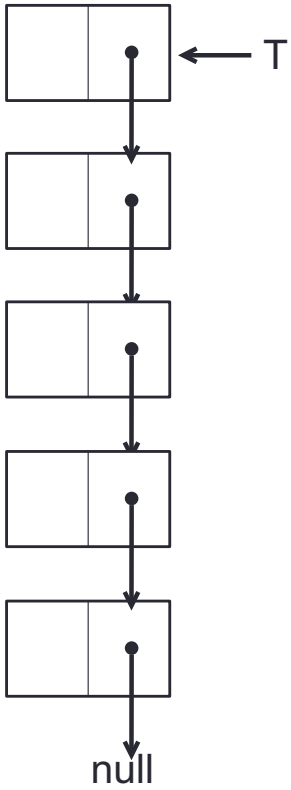
Operations:

4. Method Full (boolean flag).

   requires: input: .

   results: If S is full then Full is true, otherwise Full is false. output: flag.

ADT Stack (Linked-List)



ADT Stack (Linked-List): Element

```
public class Node<T> {
        public T data;
        public Node<T> next;

        public Node () {
                data = null;
                next = null;
        }

        public Node (T val) {
                data = val;
                next = null;
        }

        // Setters/Getters?
}
```

ADT Stack (Linked-List): Implementation

```
public class LinkedStack<T> {
        private Node<T> top;

        /* Creates a new instance of LinkStack */
        public LinkedStack() {
                top = null;
        }
```

ADT Stack (Linked-List): Implementation

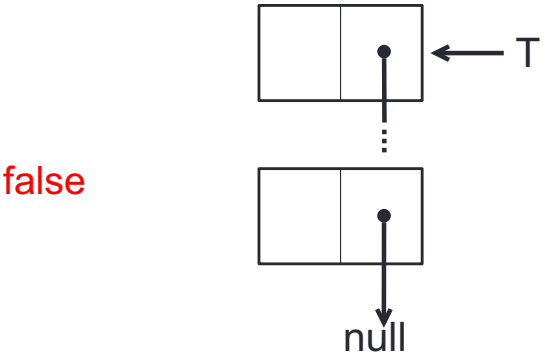
```
public class LinkedStack<T> {
        private Node<T> top;

        /* Creates a new instance of LinkStack */
        public LinkedStack() {
                top = null;
        }
```

ADT Stack (Linked-List): Implementation

```
public boolean empty(){
        return top == null;
}

public boolean full(){
        return false;
}
```

ADT Stack (Linked-List): Implementation

```
public boolean empty(){
        return top == null;
}

public boolean full(){
        return false;
}
```



false

true

ADT Stack (Linked-List): Implementation

```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```

ADT Stack (Linked-List): Implementation

```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```

null ⟵ T

Example #1

ADT Stack (Linked-List): Implementation

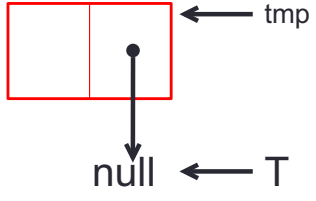
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```

null ← T

Example #1

ADT Stack (Linked-List): Implementation

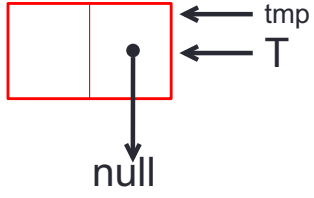
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #1

ADT Stack (Linked-List): Implementation

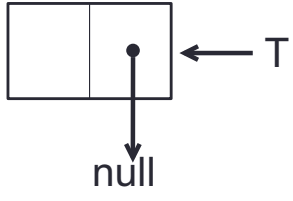
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #1

ADT Stack (Linked-List): Implementation

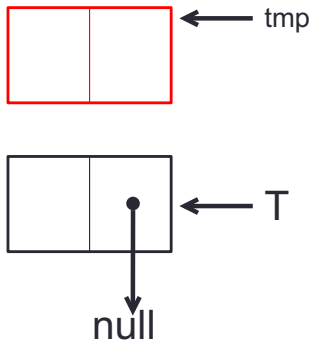
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

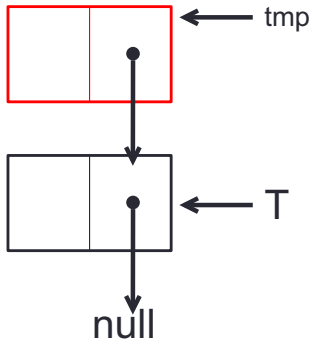
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

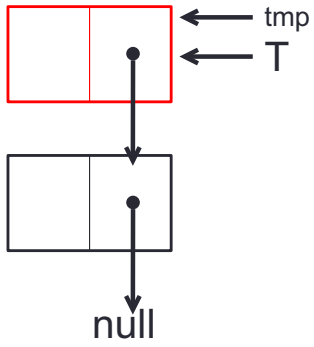
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

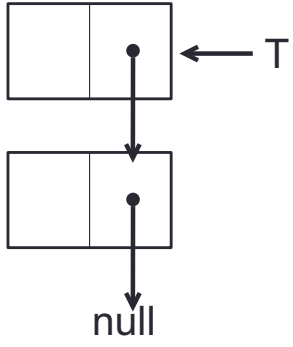
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

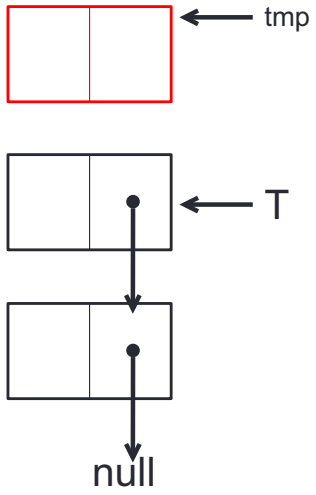
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

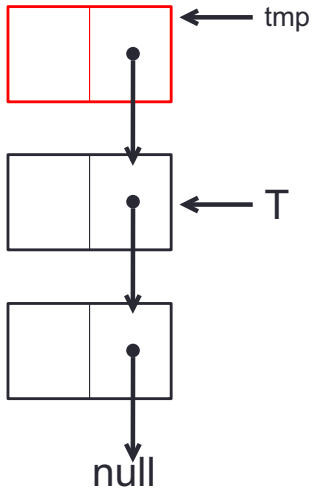
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

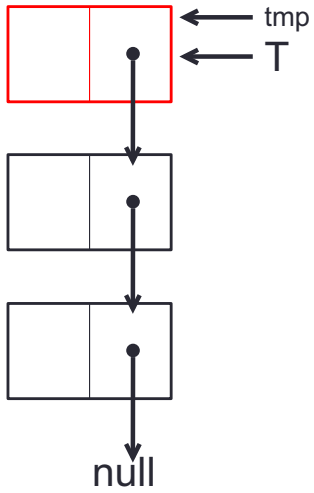
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

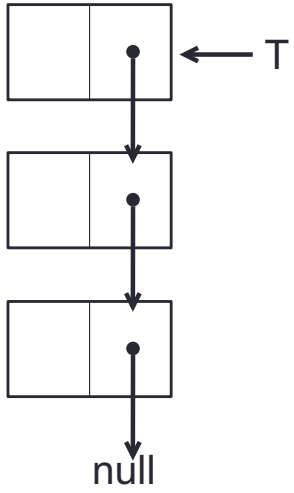
```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

```
public void push(T e){
        Node<T> tmp = new Node(e);
        tmp.next = top;
        top = tmp;
}
```



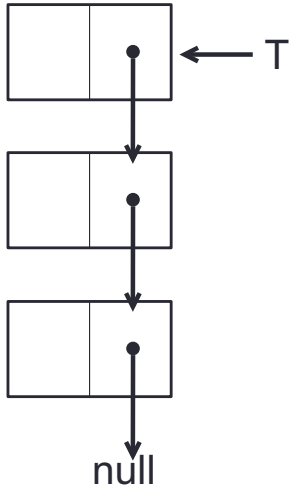
Example #3

ADT Stack (Linked-List): Implementation

```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```

ADT Stack (Linked-List): Implementation

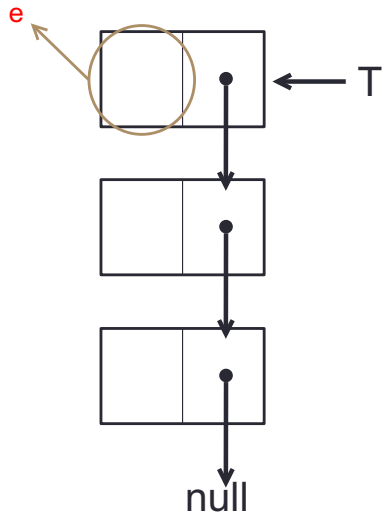
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #1

ADT Stack (Linked-List): Implementation

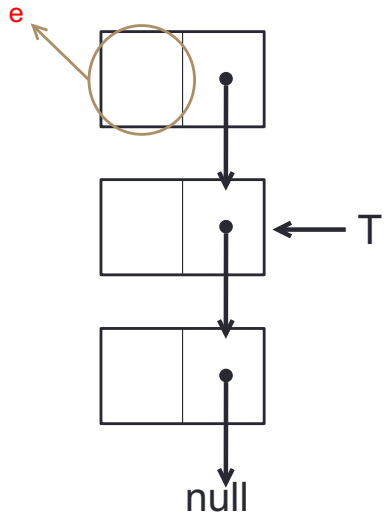
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #1

ADT Stack (Linked-List): Implementation

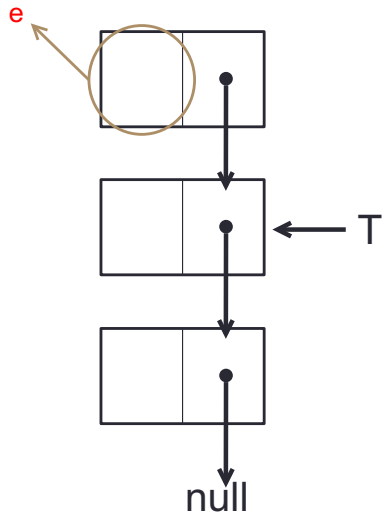
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #1

ADT Stack (Linked-List): Implementation

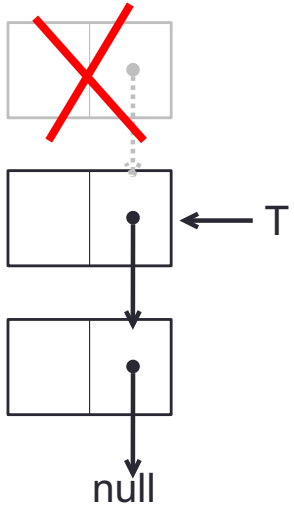
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #1

ADT Stack (Linked-List): Implementation

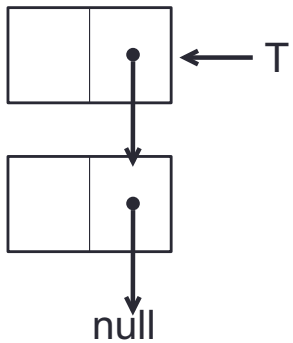
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #1

ADT Stack (Linked-List): Implementation

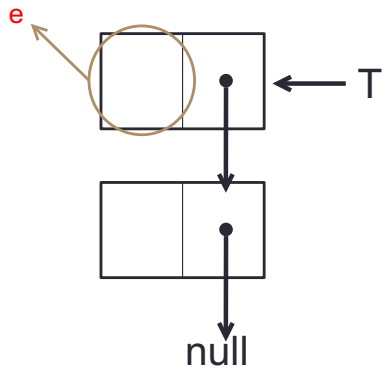
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #2

ADT Stack (Linked-List): Implementation

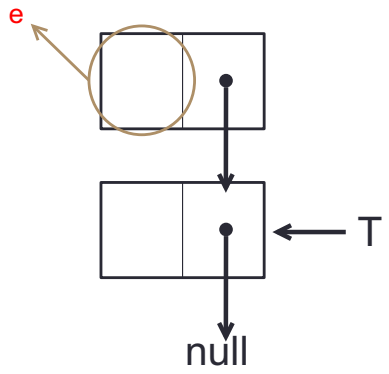
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #2

ADT Stack (Linked-List): Implementation

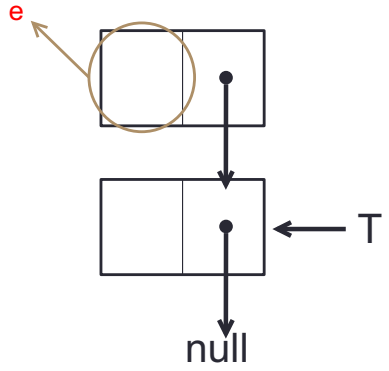
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #2

ADT Stack (Linked-List): Implementation

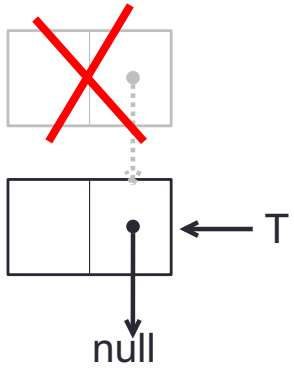
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #2

ADT Stack (Linked-List): Implementation

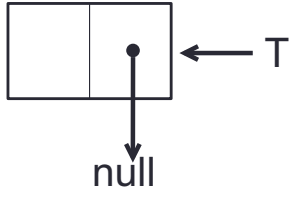
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #2

ADT Stack (Linked-List): Implementation

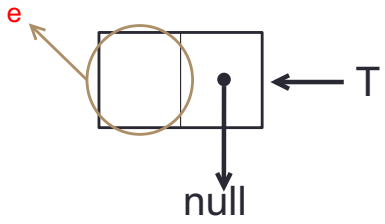
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #3

ADT Stack (Linked-List): Implementation

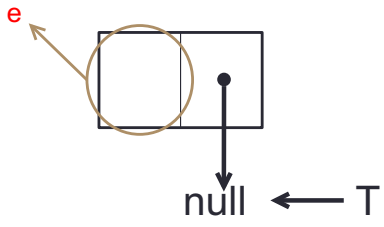
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #3

ADT Stack (Linked-List): Implementation

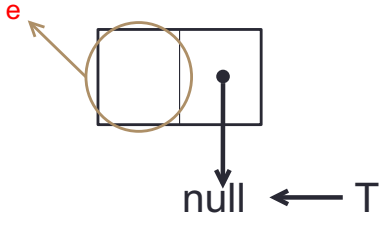
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #3

ADT Stack (Linked-List): Implementation

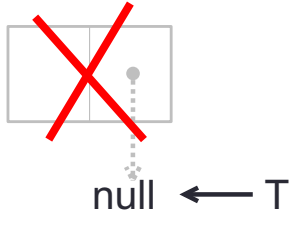
```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #3

ADT Stack (Linked-List): Implementation

```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```



Example #3

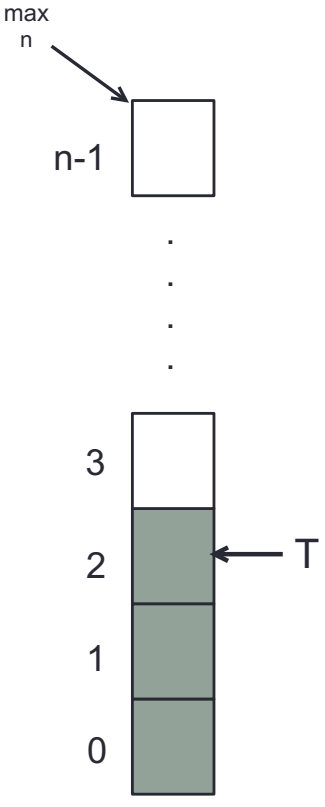
ADT Stack (Linked-List): Implementation

```
        public T pop(){
                T e = top.data;
                top = top.next;
                return e;
        }
}
```

null ⟵ T

Example #3

ADT Stack (Array)

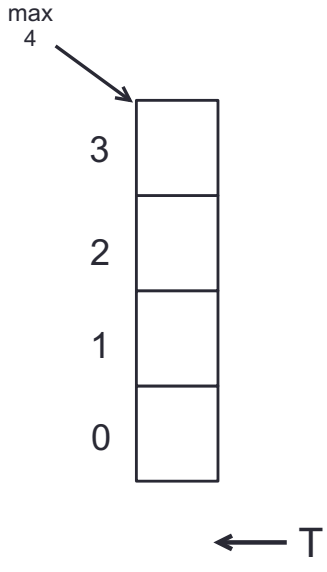


ADT Stack (Array): Representation

```
public class ArrayStack<T> {
        private int maxsize;
        private int top;
        private T[] nodes;

        /** Creates a new instance of ArrayStack */
        public ArrayStack(int n) {
                maxsize = n;
                top = -1;
                nodes = (T[]) new Object[n];
        }
```

ADT Stack (Array): Representation

```
public class ArrayStack<T> {
        private int maxsize;
        private int top;
        private T[] nodes;

        /** Creates a new instance of ArrayStack */
        public ArrayStack(int n) {
                maxsize = n;
                top = -1;
                nodes = (T[]) new Object[n];
        }
```

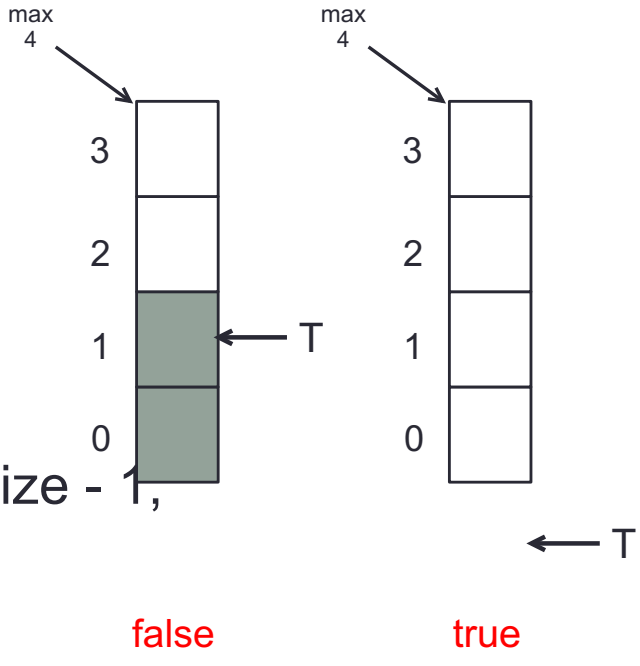


ADT Stack (Array): Implementation

```
public boolean empty(){
        return top == -1;
}

public boolean full(){
        return top == maxsize - 1;
}
```

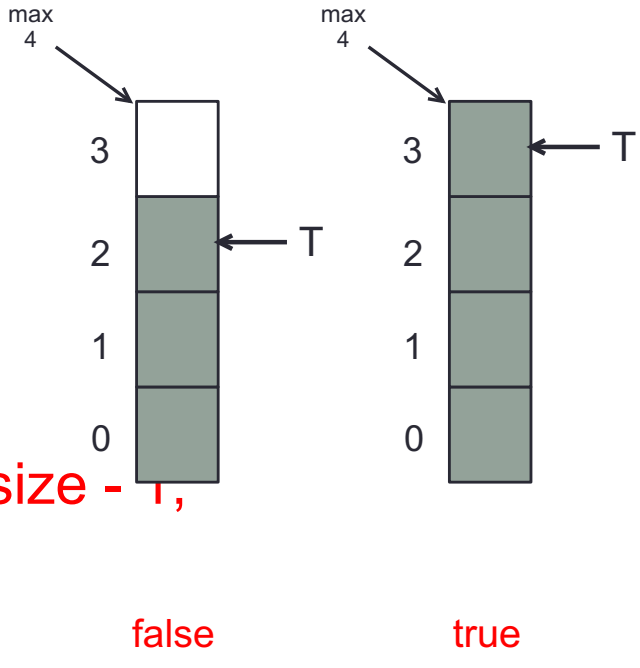
ADT Stack (Array): Implementation

```
public boolean empty(){
        return top == -1;
}

public boolean full(){
        return top == maxsize - 1;
}
```



ADT Stack (Array): Implementation

```
public boolean empty(){
        return top == -1;
}

public boolean full(){
        return top == maxsize - 1;
}
```

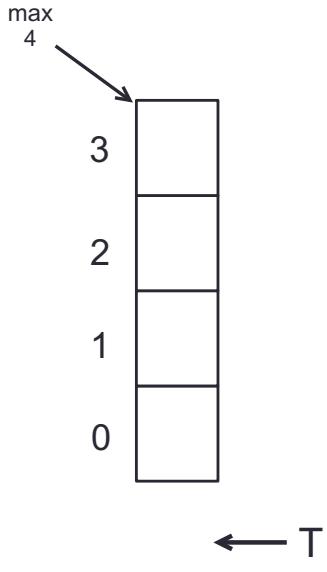


ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

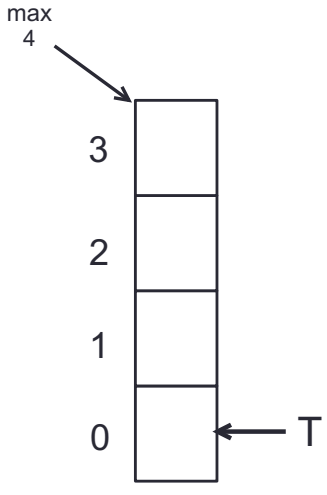


Example #1

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

S1 ← ++top
nodes[S1] = e

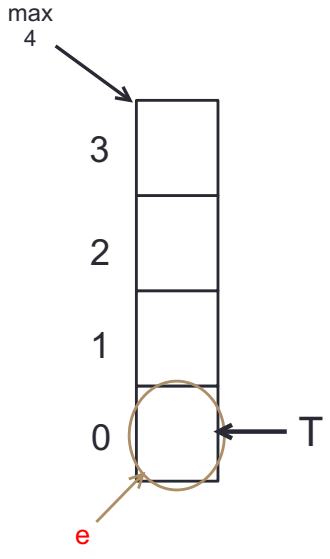


Example #1

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

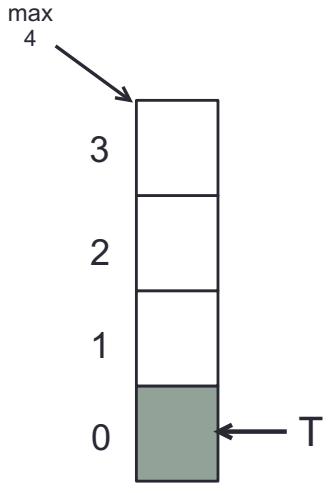
S1 ← ++top
nodes[S1] = e



Example #1

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

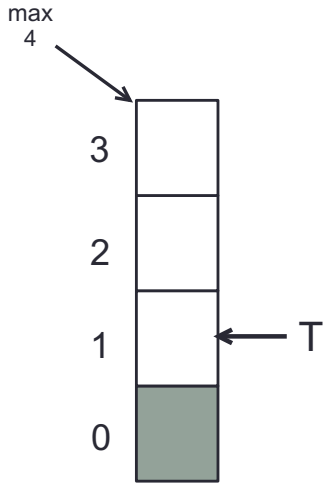


Example #2

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

S1 ← ++top
nodes[S1] = e

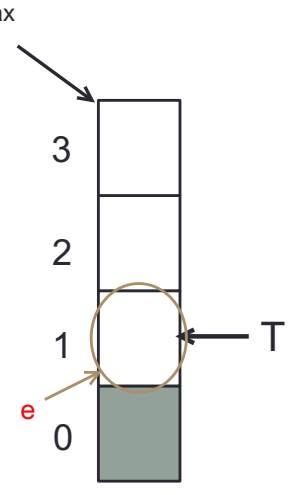


Example #2

ADT Stack (Array): Implementation

```
public void push(T e){
        nodes[++top] = e;
}

public T pop(){
        return nodes[top--];
}
}
```

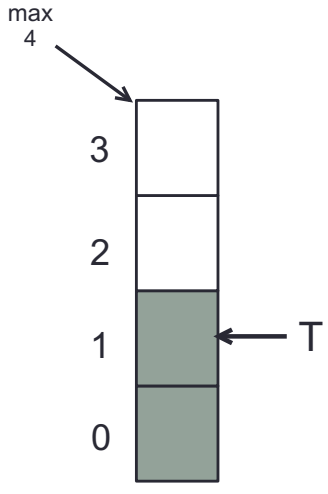
S1 ← ++top
nodes[S1] = e



Example #2

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

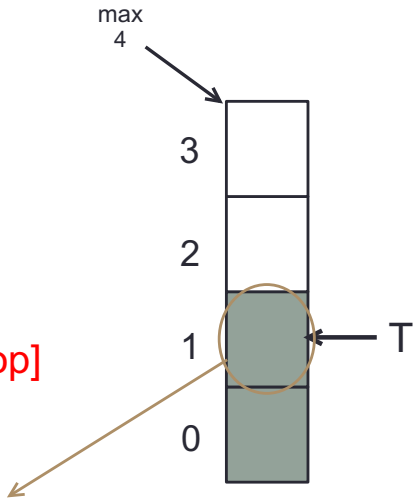


Example #3

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

S1 ← nodes[top]
top--
return S1

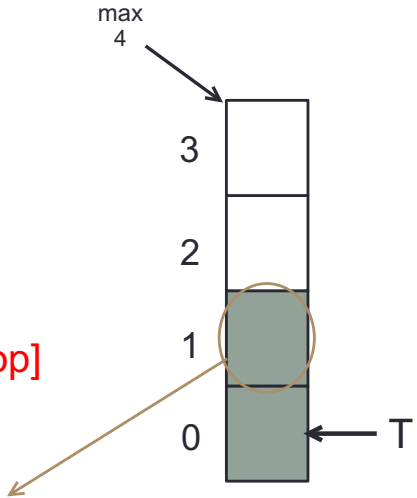


Example #3

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

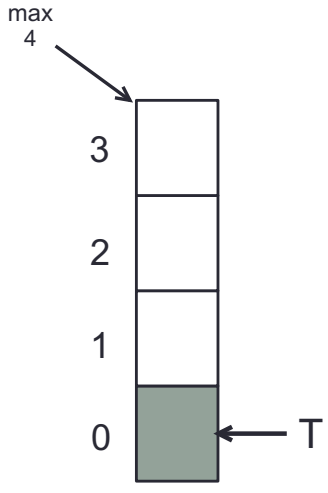
S1 ← nodes[top]
top--
return S1



Example #3

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

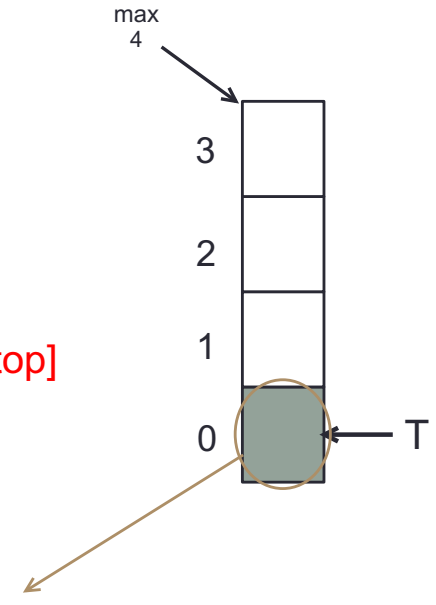


Example #4

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

S1 ← nodes[top]
top--
return S1

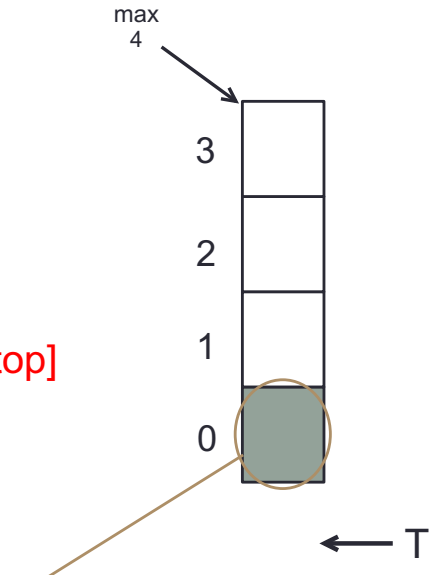


Example #4

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```

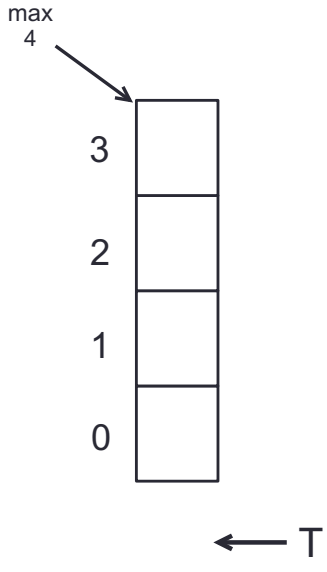
S1 ← nodes[top]
top--
return S1



Example #4

ADT Stack (Array): Implementation

```
        public void push(T e){
                nodes[++top] = e;
        }

        public T pop(){
                return nodes[top--];
        }
}
```



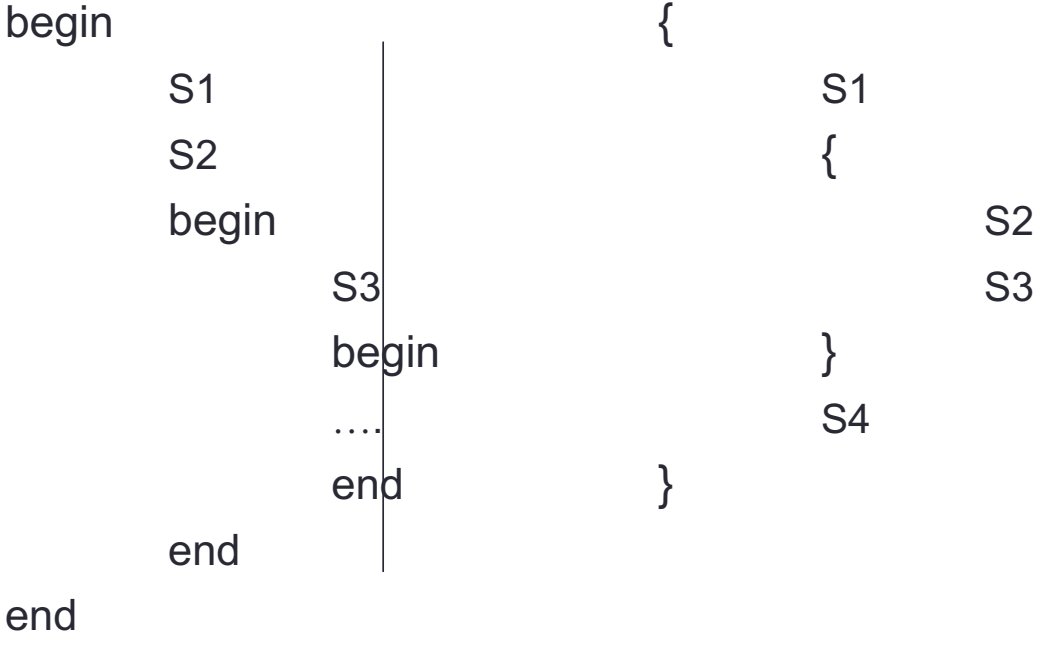
Example #4

Applications of Stacks

- Some applications of stacks are:
  - Balancing symbols.
  - Computing or evaluating postfix expressions.
  - Converting expressions from infix to postfix.

1. Balancing Symbols

- Expressions: mathematical (a + ((b-c)*d)) or programs have delimiters.



1. Balancing Symbols

- Delimiters must be balanced.
- One of the common use of the stacks is to parse certain kinds of expressions or string text.
- Write a program that verifies the delimiters in a line of text or expression typed by the user.
  - a*(b+c) //This expression is right
  - b/[a*(b+c)] //This expression is right
  - {a*(b+c]} //This expression is wrong

1. Balancing Symbols

- Read characters from the start of the expression to the end.
  - If the token is a starting delimiter, then push on to the stack.
  - If the token is a closing delimiter, then pop from the stack.
    - If symbol from this pop operation matches the closing delimiter, then we carry on.
    - If not, or the stack was empty, then we have unbalanced symbols (report an error). قوس زايد
- If stack is empty at the end of expression, we have balanced symbols.
- If not (stack is not empty), then we have unbalanced symbols (report an error). قوس ناقص

1. Balancing Symbols

- Input : expression
- Output: True if and only if delimiters are balanced
- Let S be empty Stack
- Let n be number of characters
- for i=0? n-1
  - If expression[i] is an opening delimiter, then
    - S.push(expression[i]).
  - else If expression[i] is a closing delimiter, then
    - If the S is empty
      - return false unbalanced symbols
    - symbol=S.pop().
    - If symbol does not matches the closing delimiter
      - return false unbalanced symbols.
- If S is empty
  - return true balanced symbols.
- else
  - return false unbalanced symbols

2. Postfix Expressions

- Evaluating Postfix Expressions:
  - Infix expression: 4.99*1.06+5.99+6.99*1.06
  - Value 18.69 correct parenthesis used.
  - Value 19.37 incorrect no parenthesis used.
  - In postfix form, above expression becomes:

    4.99 1.06 * 5.99 + 6.99 1.06*+

  Advantage: no brackets are needed and a stack can be used to compute the expression.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

Know how to ↑ we don't have to convert to this form, but she taught us how to do it anyway.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2 + 3 = 5

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2 + 3 = 5

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

5 * 8 = 40

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

5 * 8 = 40

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

5 + 40 = 45

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

5 + 40 = 45

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

45 + 3 = 48

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

45 + 3 = 48

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

6 * 48 = 288

2. Postfix Expressions

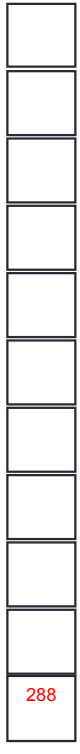
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

6 * 48 = 288

2. Postfix Expressions

- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the operation on them
    - push the result back on the stack.

End!



متى هالألغوريذم تستوي واروره؟
لازم نفكر فيها لأنه ممكن تجي
as Programming assignment

Wow

- لكنه إذا صار عندي
عملية زيادة؟

Converting from infix to postfix

Assume the infix expression is a string of tokens delimited by spaces. The operator tokens are *, /, +, and -, along with the left and right parentheses, (with). The operand tokens are the single-character identifiers A, B, C, and so on.

The following steps will produce a string of tokens in postfix order.

1. Create an empty stack called opstack for keeping operators. Create an empty list for output.
2. Scan the token list from left to right.
   - If the token is an operand, append it to the end of the output list.
   - If the token is a left parenthesis, push it on the opstack.
   - If the token is a right parenthesis, pop the opstack until the corresponding left parenthesis is removed. Append each operator to the end of the output list.
   - If the token is an operator, *, /, +, or -, push it on the opstack. However, first remove any operators already on the opstack that have higher or equal precedence and append them to the output list.
3. When the input expression has been completely processed, check the opstack. Any operators still on the stack can be removed and appended to the end of the output list.