Stacks

CS212: Data Structure

Stacks

- Stack: Last In First Out (LIFO).
  - Used in procedure calls, to compute arithmetic expressions etc.
- Used in Operating system to implement method calls, and in evaluating Expirations.

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

ADT Stack: Specification

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

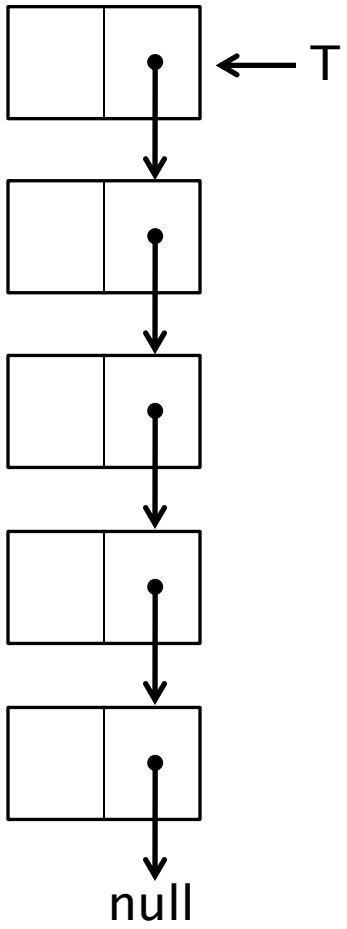
Operations:

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 LinkStack() {
      top = null;
  }
```

ADT Stack (Linked-List): Implementation

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

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

null ⟵ T

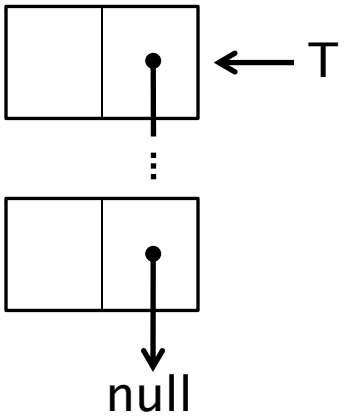
ADT Stack (Linked-List): Implementation

```
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

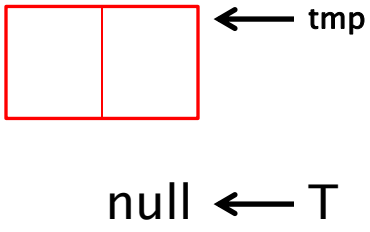
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```

null ⟵ T

Example #1

ADT Stack (Linked-List): Implementation

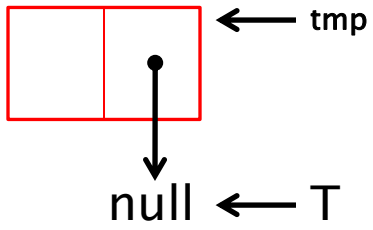
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #1

ADT Stack (Linked-List): Implementation

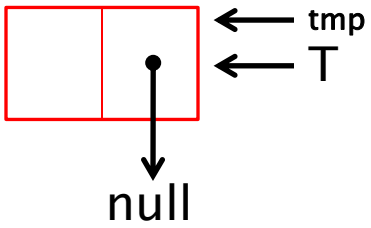
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #1

ADT Stack (Linked-List): Implementation

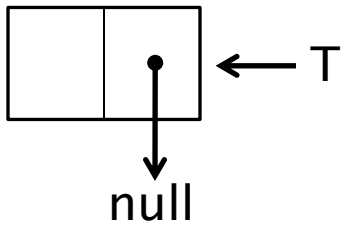
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #1

ADT Stack (Linked-List): Implementation

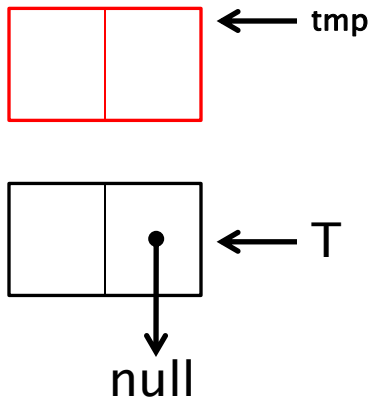
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

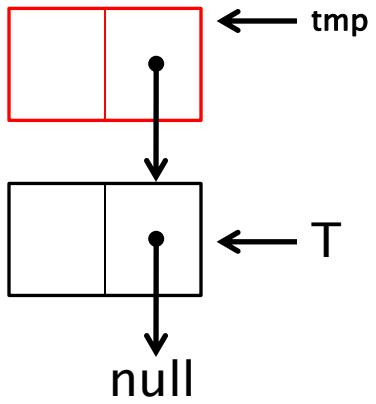
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

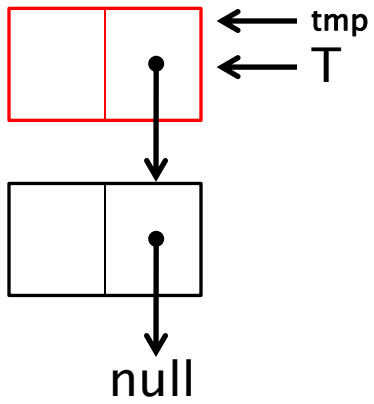
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

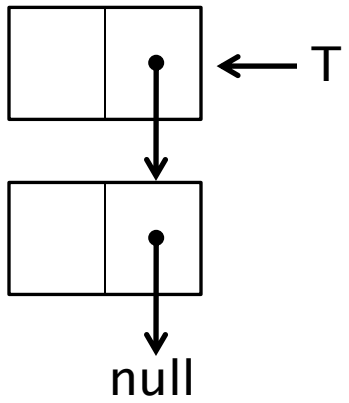
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #2

ADT Stack (Linked-List): Implementation

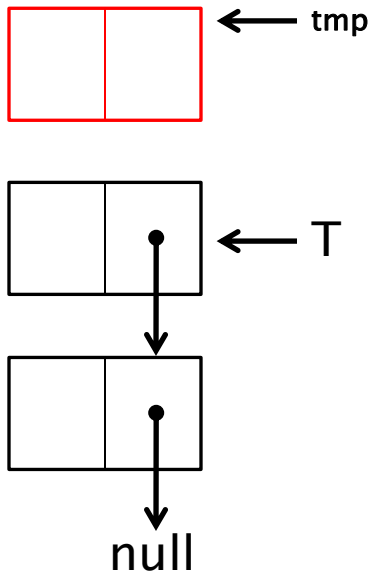
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

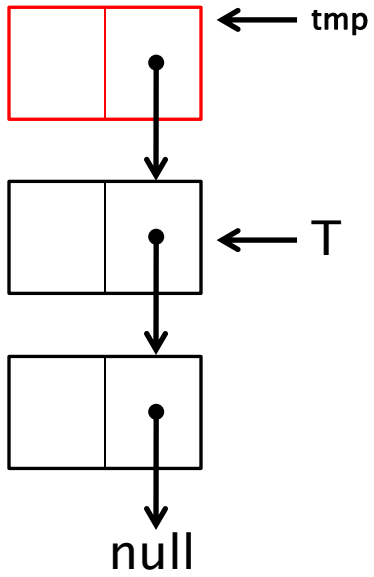
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

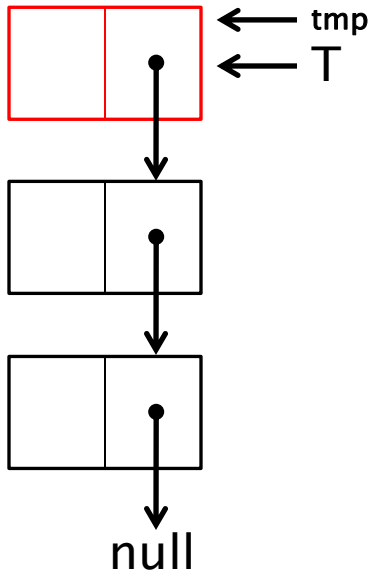
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

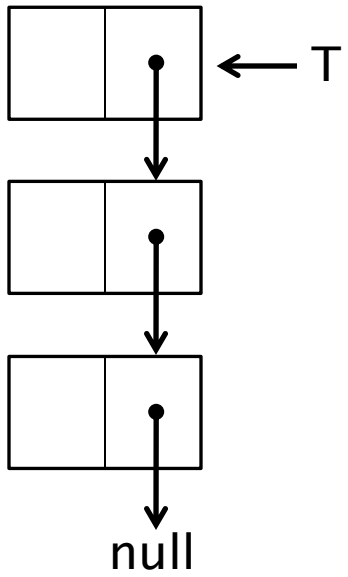
```
public void push(T e){
    Node<T> tmp = new Node(e);
    tmp.next = top;
    top = tmp;
}
```



Example #3

ADT Stack (Linked-List): Implementation

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



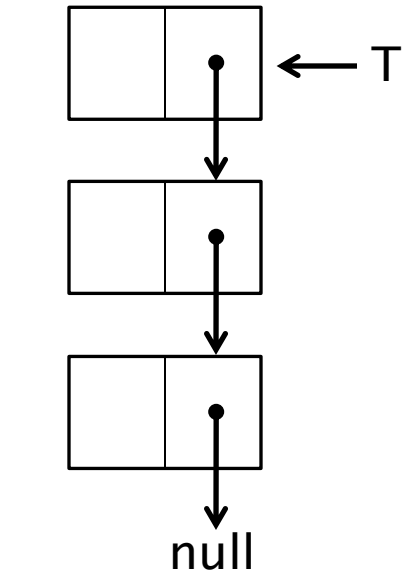
Example #3

ADT Stack (Linked-List): Implementation

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

ADT Stack (Linked-List): Implementation

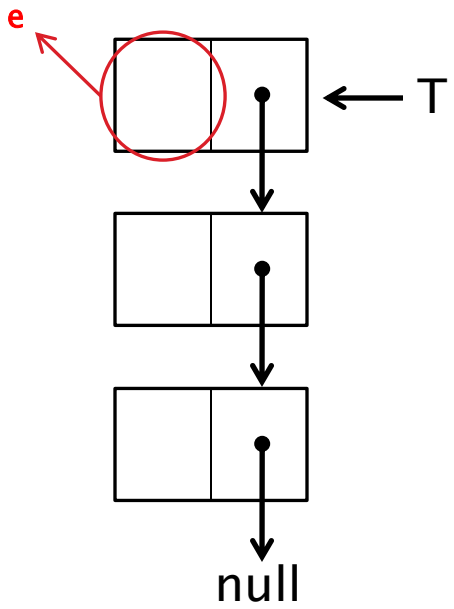
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #1

ADT Stack (Linked-List): Implementation

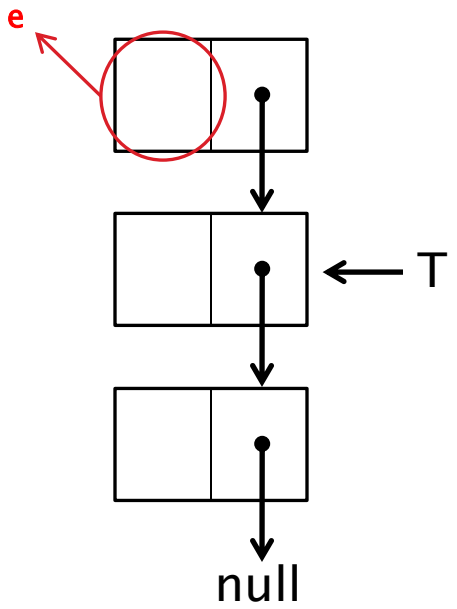
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #1

ADT Stack (Linked-List): Implementation

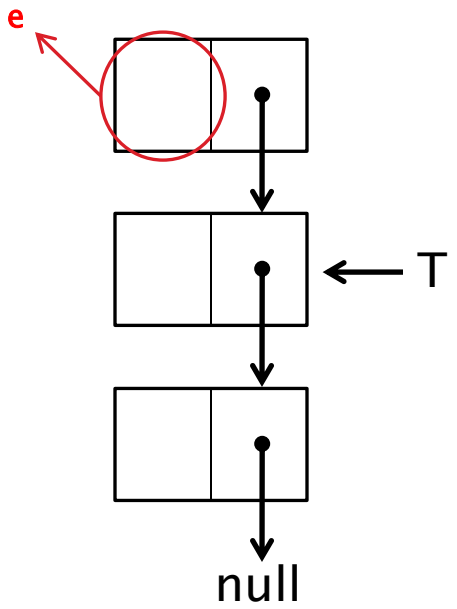
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #1

ADT Stack (Linked-List): Implementation

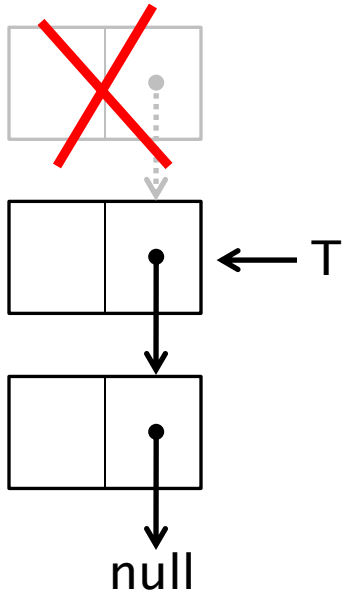
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #1

ADT Stack (Linked-List): Implementation

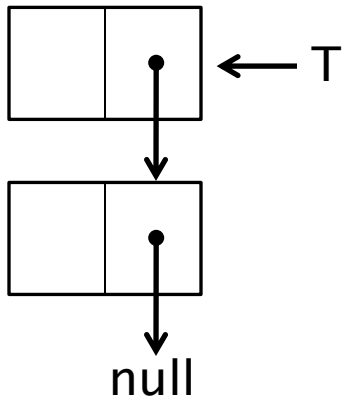
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #1

ADT Stack (Linked-List): Implementation

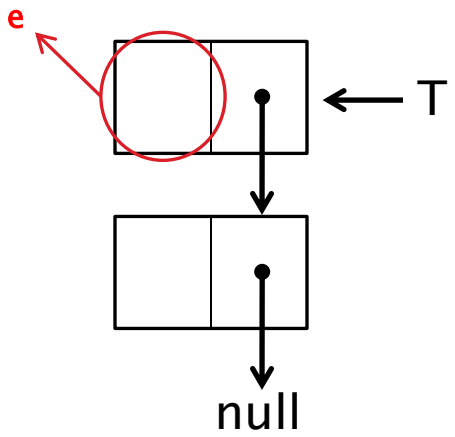
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #2

ADT Stack (Linked-List): Implementation

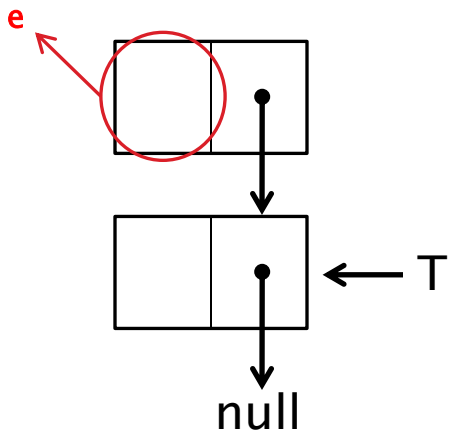
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #2

ADT Stack (Linked-List): Implementation

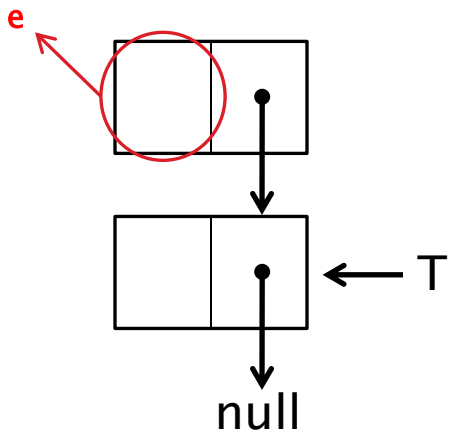
```
public T pop(){
    T e = top.data;
    top = top.next;
    return e;
}
}
```



Example #2

ADT Stack (Linked-List): Implementation

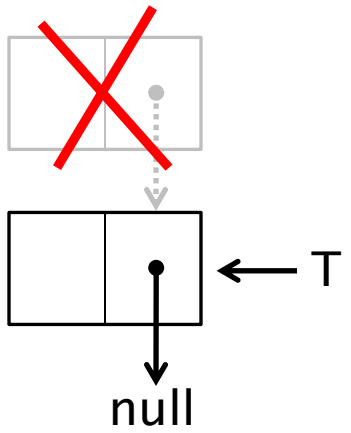
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #2

ADT Stack (Linked-List): Implementation

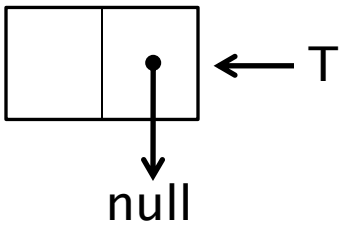
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #2

ADT Stack (Linked-List): Implementation

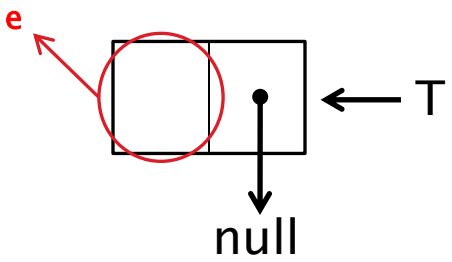
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #3

ADT Stack (Linked-List): Implementation

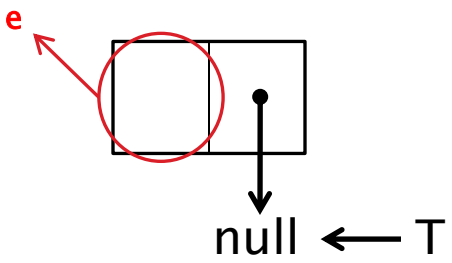
```
public T pop(){
    T e = top.data;
    top = top.next;
    return e;
}
}
```



Example #3

ADT Stack (Linked-List): Implementation

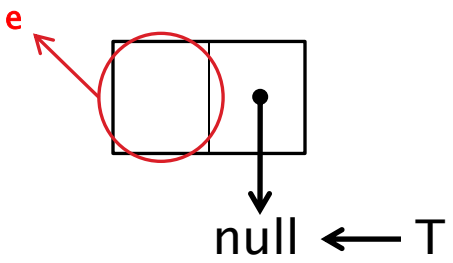
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #3

ADT Stack (Linked-List): Implementation

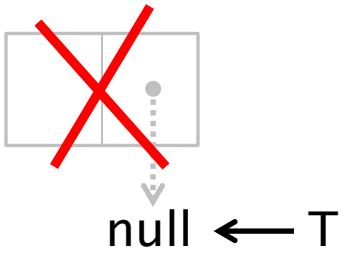
```
    public T pop(){
        T e = top.data;
        top = top.next;
        return e;
    }
}
```



Example #3

ADT Stack (Linked-List): Implementation

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



Example #3

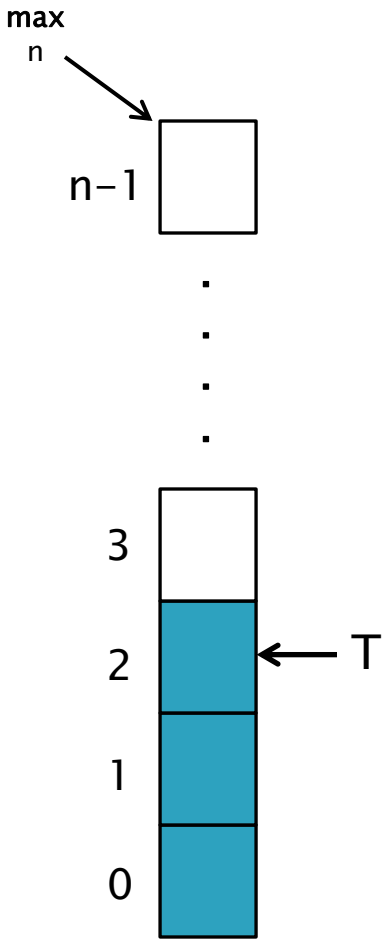
ADT Stack (Linked-List): Implementation

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

null ⟵ T

Example #3

ADT Stack (Array)

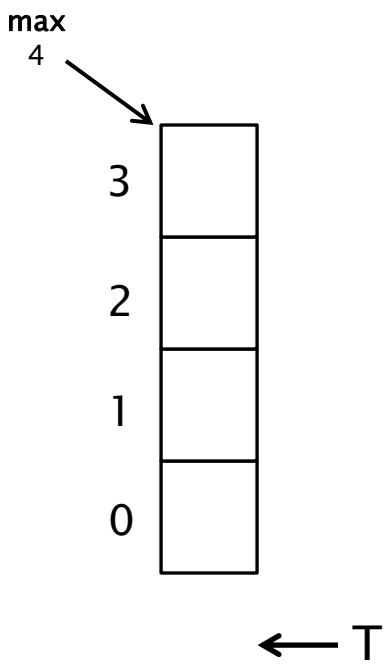


ADT Stack (Array): Representation

```
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];
  }
```

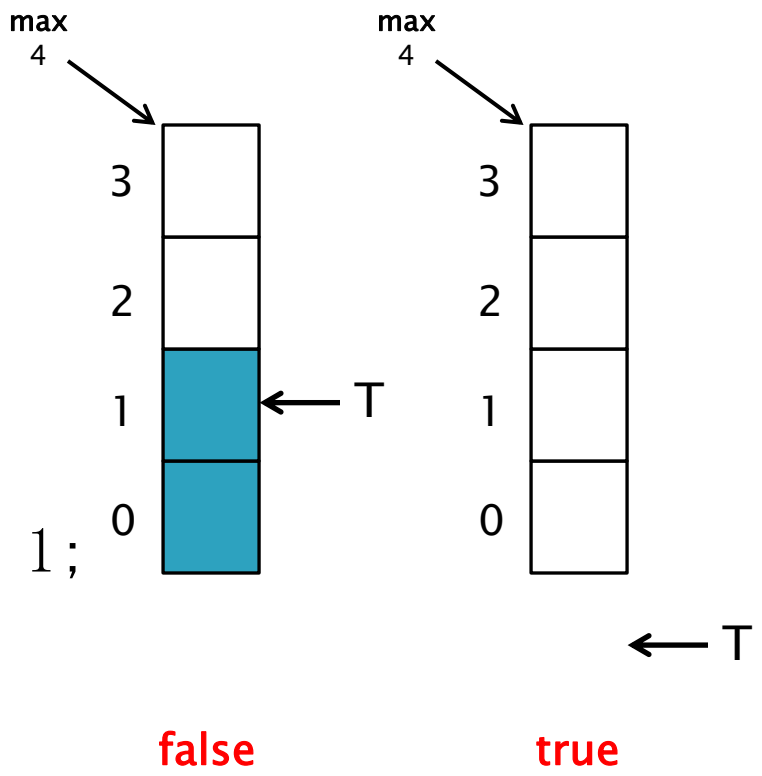


ADT Stack (Array): Implementation

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

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

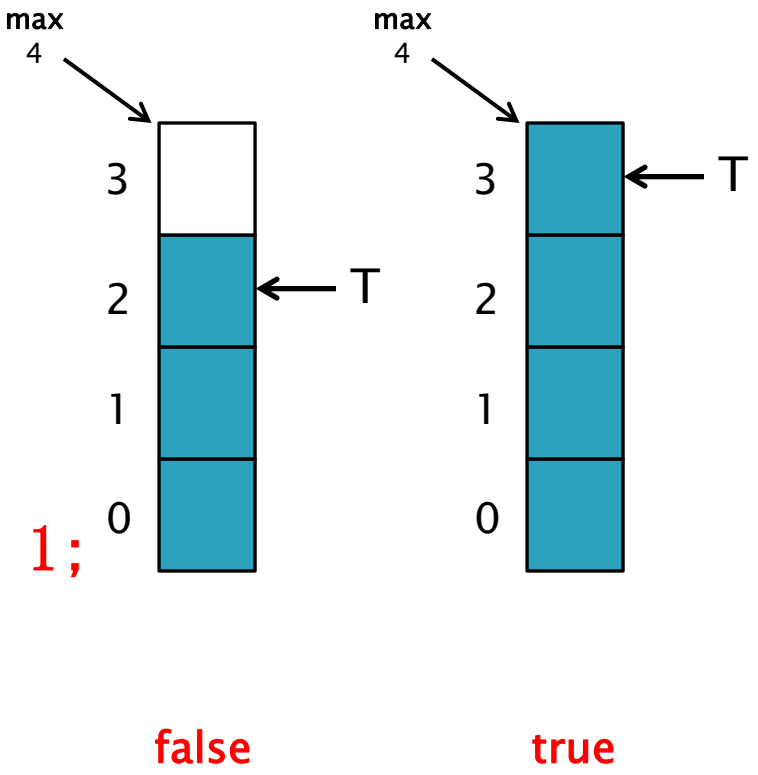
ADT Stack (Array): Implementation

```
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;
}
```

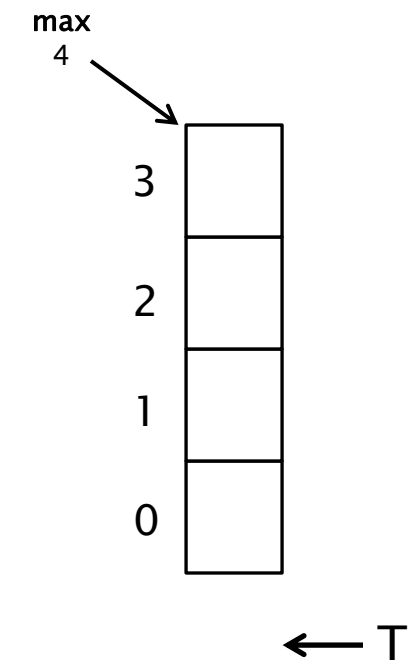


ADT Stack (Array): Implementation

```
    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--];
    }
}
```

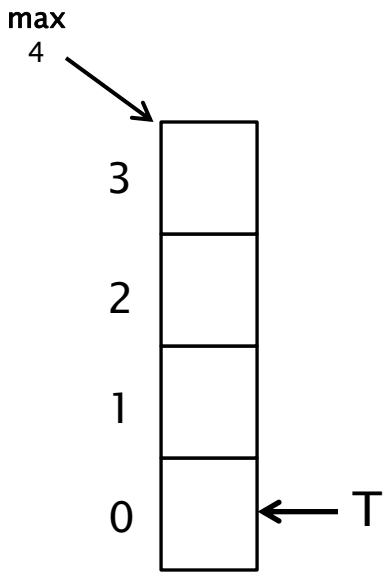


Example #1

ADT Stack (Array): Implementation

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

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

S1 ← ++top
nodes[S1] = e

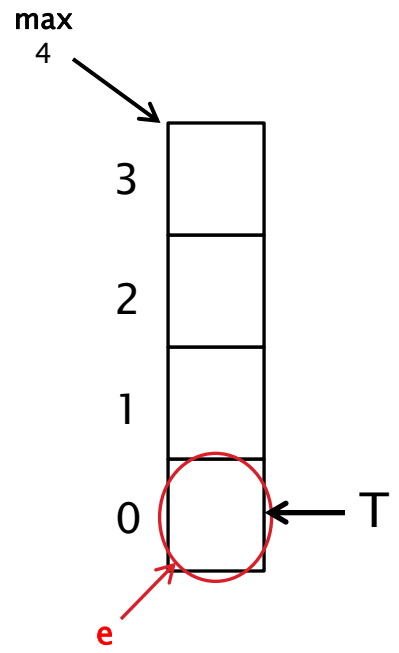


Example #1

ADT Stack (Array): Implementation

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

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

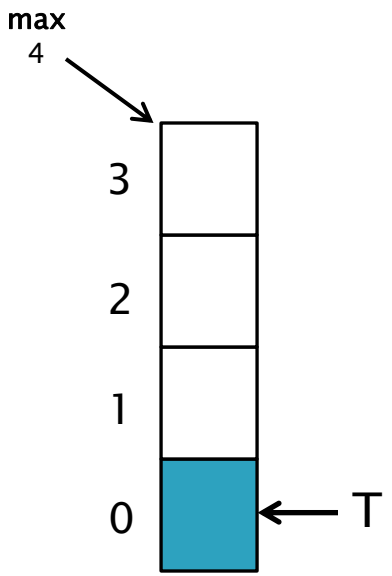
S1 ← ++top
nodes[S1] = e



Example #1

ADT Stack (Array): Implementation

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

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

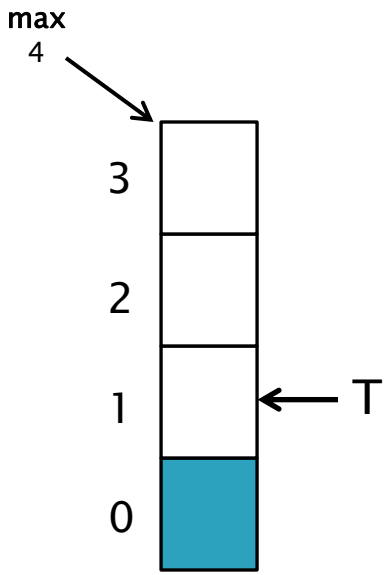


Example #2

ADT Stack (Array): Implementation

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

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

S1 ← ++top
nodes[S1] = e

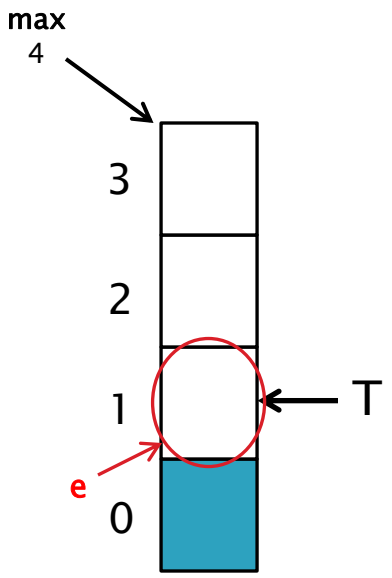


Example #2

ADT Stack (Array): Implementation

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

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

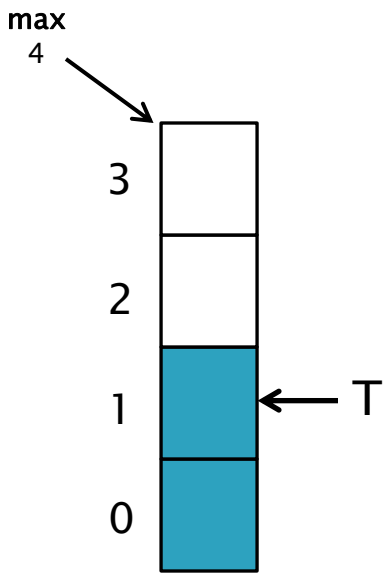
S1 ← ++top
nodes[S1] = e



Example #2

ADT Stack (Array): Implementation

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

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

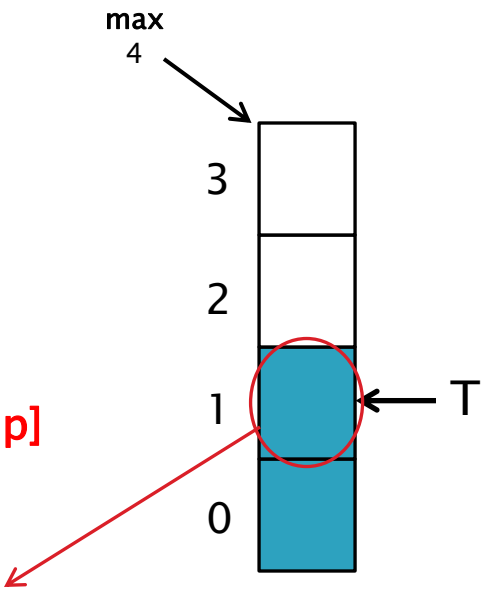


Example #3

ADT Stack (Array): Implementation

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

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

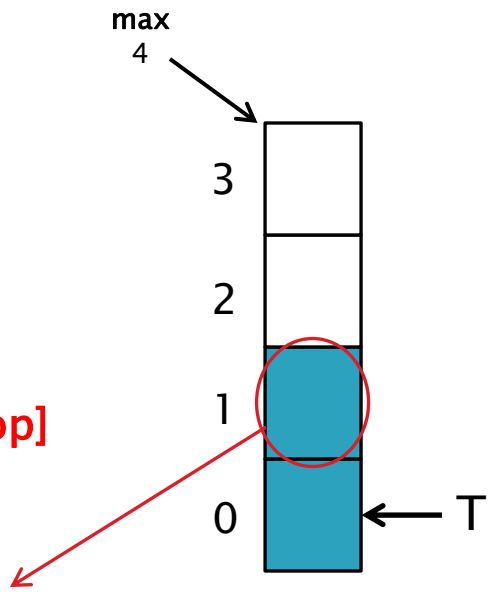
S1 ← nodes[top]
top--
return S1



Example #3

ADT Stack (Array): Implementation

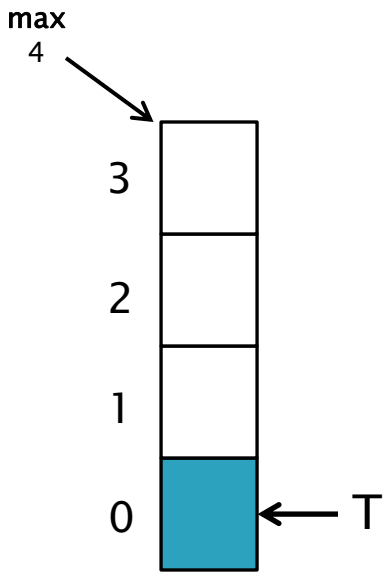
```
    public void push(T e){
        nodes[++top] = e;
    }

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



Example #3

ADT Stack (Array): Implementation

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

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

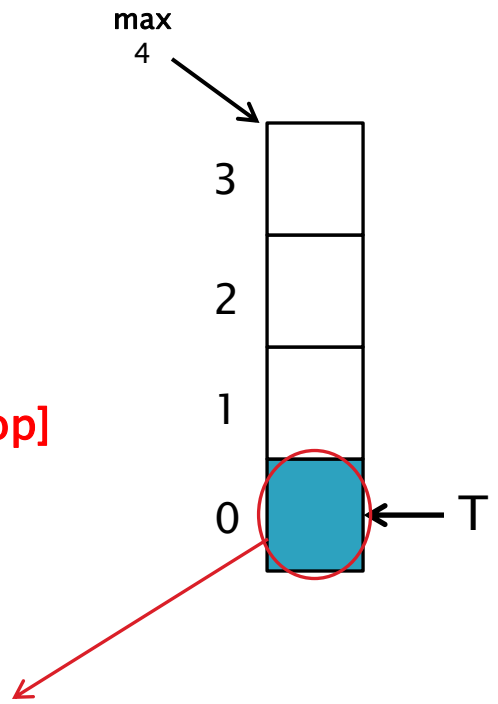


Example #4

ADT Stack (Array): Implementation

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

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

S1 ← nodes[top]
top--
return S1

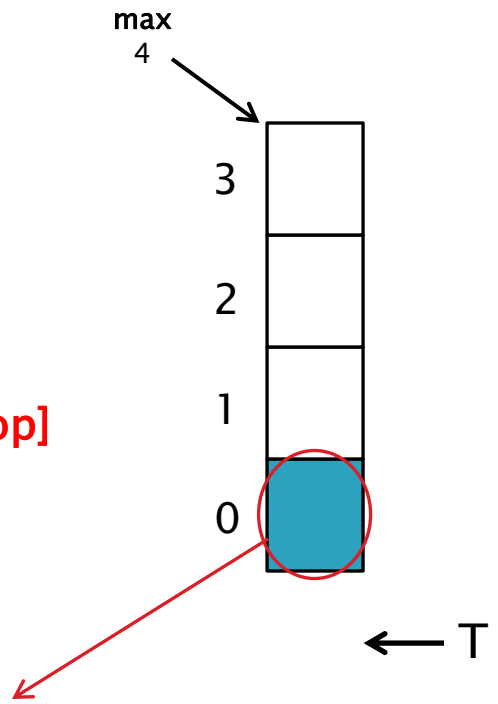


Example #4

ADT Stack (Array): Implementation

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

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

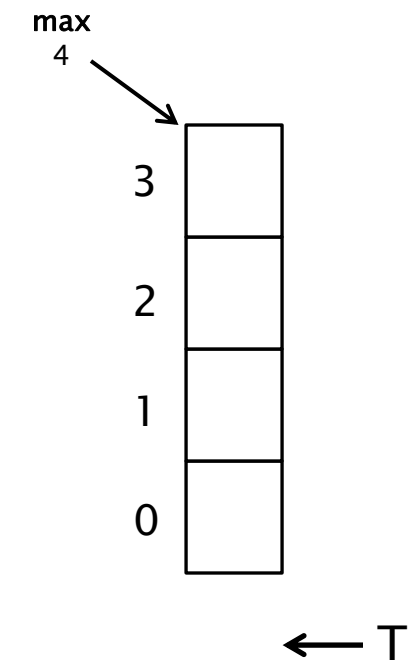
S1 ← nodes[top]
top--
return S1



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.**

```
begin                          {
        S1                             S1
        S2                             {
        begin                                  S2
                S3                             S3
                begin                  }
                ....                   S4
                end            }
        end
end
```

1. Balancing Symbols

- Delimiters must be balanced. [()] is legal but [(]) illegal.
- A stack can be used to check if the delimiters are balanced.
  - 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).
  - 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).

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*+

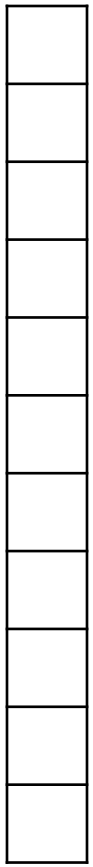
  →Advantage: no brackets are needed and a stack can be used to compute the expression.

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.

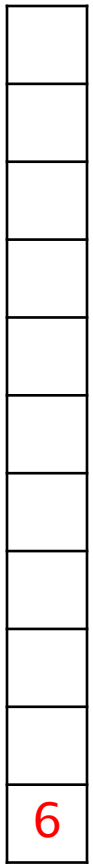
2. Postfix Expressions

- 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.



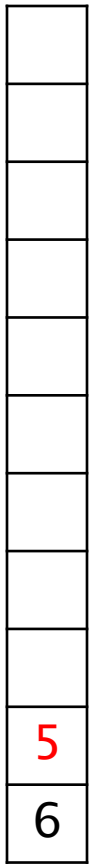
2. Postfix Expressions

- 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.



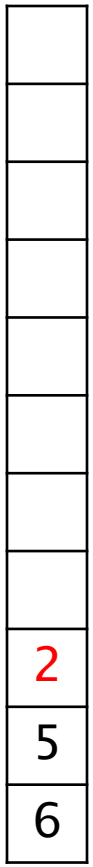
2. Postfix Expressions

- 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.



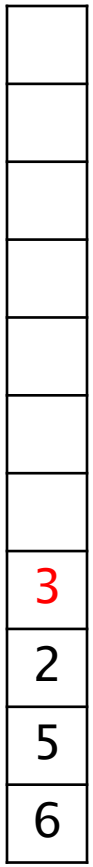
2. Postfix Expressions

- 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.



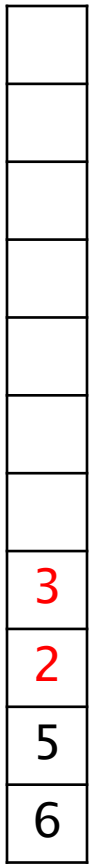
2. Postfix Expressions

- 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

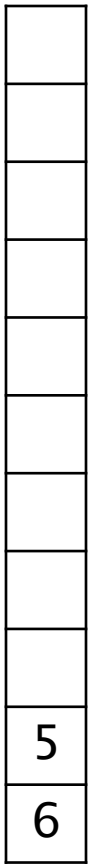
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

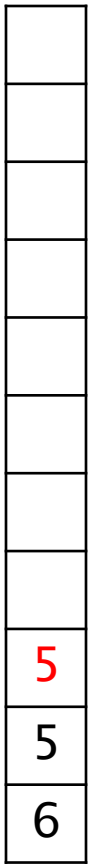
2 + 3 = 5



2. Postfix Expressions

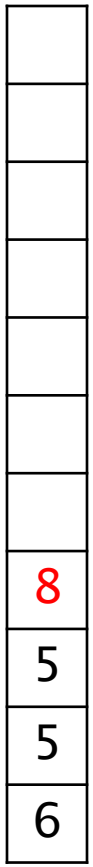
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

2 + 3 = 5



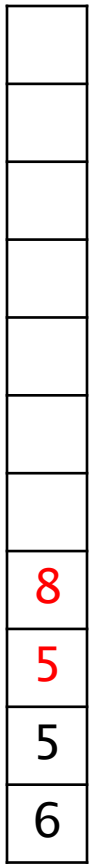
2. Postfix Expressions

- 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

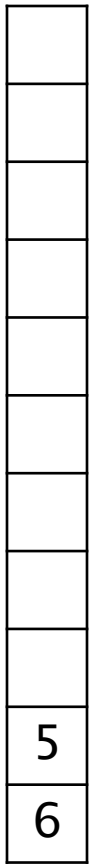
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

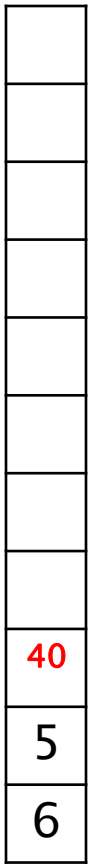
5 * 8 = 40



2. Postfix Expressions

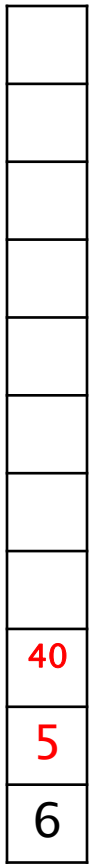
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

5 * 8 = 40



2. Postfix Expressions

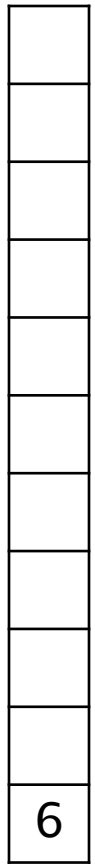
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

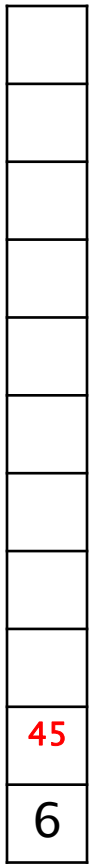
5 + 40 = 45



2. Postfix Expressions

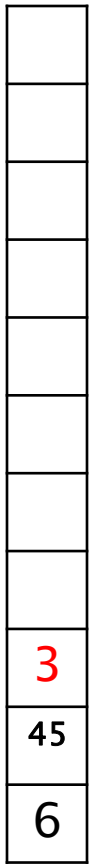
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

5 + 40 = 45



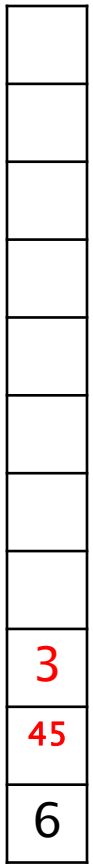
2. Postfix Expressions

- 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

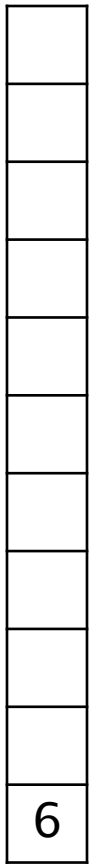
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

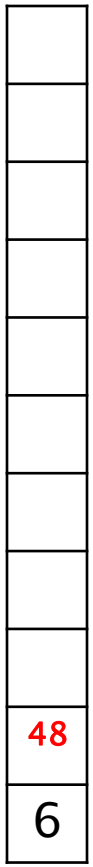
45 + 3 = 48



2. Postfix Expressions

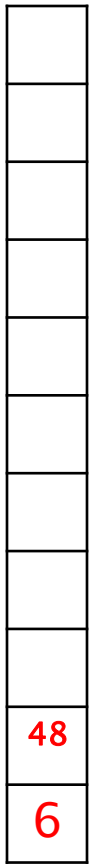
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

45 + 3 = 48



2. Postfix Expressions

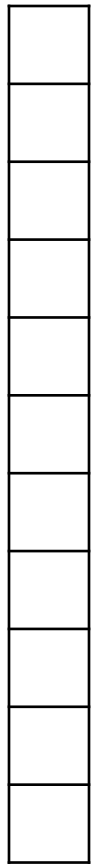
- Example:
  - infix: 6*(5+((2+3)*8)+3)
  - postfix: 6 5 2 3 + 8 * + 3 + *.
- Algorithm to compute postfix expression:
  - Read the postfix expression left to right.
  - When a number is read push it on the stack.
  - When an operator is read:
    - pop two numbers from the stack
    - carry out the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

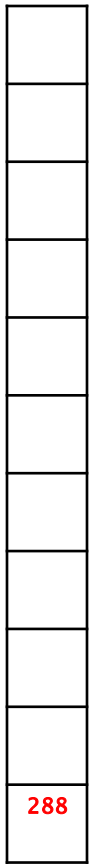
6 * 48 = 288



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!

result