Introduction to Stacks

- What is a Stack

- Stack implementation using array.

- Stack implementation using linked list.

- Applications of Stack.
What is a Stack?

- Stack is a data structure in which data is added and removed at only one end called the top.

- To add (**push**) an item to the stack, it must be placed on the top of the stack.

- To remove (**pop**) an item from the stack, it must be removed from the top of the stack too.

- Thus, the last element that is pushed into the stack, is the first element to be popped out of the stack. 
  i.e., Last In First Out (**LIFO**)
An Example of Stack

1. Push(8)
2. Push(2)
3. Pop()
4. Pop()
5. Pop()
public interface Stack extends Container {
    public abstract Object getTop();
    public abstract void push(Object obj);
    public abstract Object pop();
}

• In our implementation, a stack is a container that extends the AbstractContainer class and implements the Stack interface.

• Two implementations:
  – StackAsArray
    • The underlying data structure is an array of Object
  – StackAsLinkedList
    • The underlying data structure is an object of MyLinkedList
StackAsArray – Constructor

• In the StackAsArray implementation that follows, the top of the stack is `array[count – 1]` and the bottom is `array[0]`:

• The constructor’s single parameter, size, specifies the maximum number of items that can be stored in the stack.

• The variable array is initialized to be an array of length size.

```java
public class StackAsArray extends AbstractContainer implements Stack {
    protected Object[] array;

    public StackAsArray(int size) {
        array = new Object[size];
    }

    // ...
```
StackAsArray – purge() Method

• The purpose of the purge method is to remove all the contents of a container.

• To empty the stack, the purge method simply assigns the value null to the first count positions of the array.

```java
public void purge(){
    while (count > 0)
        array[--count] = null;
}
```

Complexity is O(n)
StackAsArray – push() Method

- push() method adds an element at the top the stack.
- It takes as argument an Object to be pushed.
- It first checks if there is room left in the stack. If no room is left, it throws a `ContainerFullException` exception. Otherwise, it puts the object into the array, and then increments count variable by one.

```java
public void push(Object object){
    if (count == array.length)
        throw new ContainerFullException();
    else
        array[count++] = object;
}
```

Complexity is O(1)
StackAsArray – pop() Method

• The pop method removes an item from the stack and returns that item.
• The pop method first checks if the stack is empty. If the stack is empty, it throws a `ContainerEmptyException`. Otherwise, it simply decreases count by one and returns the item found at the top of the stack.

```java
public Object pop(){
    if(count == 0)
        throw new ContainerEmptyException();
    else {
        Object result = array[--count];
        array[count] = null;
        return result;
    }
}
```

Complexity is O(1)
StackAsArray – getTop() Method

• getTop() method first checks if the stack is empty.
• getTop() method is a stack accessor which returns the top item in the stack without removing that item. If the stack is empty, it throws a `ContainerEmptyException`. Otherwise, it returns the top item found at position `count-1`.

```java
public Object getTop(){
    if(count == 0)
        throw new ContainerEmptyException();
    else
        return array[count - 1];
}
```

Complexity is O(1)
public Iterator iterator() {
    return new Iterator() {
        private int position = count-1;
        public boolean hasNext() {
            return position >=0;
        }
        public Object next () {
            if(position < 0)
                throw new NoSuchElementException();
            else
                return array[position--];
        }
    };
}
public class StackAsLinkedList
    extends AbstractContainer
    implements Stack {

    protected MyLinkedList list;

    public StackAsLinkedList()
    {
        list = new MyLinkedList();
    }

    public void purge()
    {
        list.purge();
        count = 0;
    }

    // …

    Complexity is O(1)
public void push(Object obj) {
    list.prepend(obj);
    count++;
}

public Object pop() {
    if (count == 0)
        throw new ContainerEmptyException();
    else {
        Object obj = list.getFirst();
        list.extractFirst();
        count--;
        return obj;
    }
}

public Object getTop() {
    if (count == 0)
        throw new ContainerEmptyException();
    else
        return list.getFirst();
}
public Iterator iterator() {
    return new Iterator() {
        private MyLinkedList.Element position = list.getHead();

        public boolean hasNext() {
            return position != null;
        }

        public Object next() {
            if (position == null)
                throw new NoSuchElementException();
            else {
                Object obj = position.getData();
                position = position.getNext();
                return obj;
            }
        }
    };
}
Applications of Stack

• Some direct applications:
  – Page-visited history in a Web browser
  – Undo sequence in a text editor
  – Chain of method calls in the Java Virtual Machine
  – Evaluating postfix expressions

• Some indirect applications
  – Auxiliary data structure for some algorithms
  – Component of other data structures
Application of Stack - Evaluating Postfix Expression

\[(5+9)*2+6*5\]

- An ordinary arithmetical expression like the above is called infix-expression -- binary operators appear in between their operands.

- The order of operations evaluation is determined by the precedence rules and parenthesis.

- When an evaluation order is desired that is different from that provided by the precedence, parentheses are used to override precedence rules.
Application of Stack - Evaluating Postfix Expression

- Expressions can also be represented using **postfix notation** - where an operator comes after its two operands.

- The advantage of postfix notation is that the order of operation evaluation is unique without the need for precedence rules or parenthesis.

<table>
<thead>
<tr>
<th>Infix</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 / 2</td>
<td>16 2 /</td>
</tr>
<tr>
<td>(2 + 14)* 5</td>
<td>2 14 + 5 *</td>
</tr>
<tr>
<td>2 + 14 * 5</td>
<td>2 14 5 * +</td>
</tr>
<tr>
<td>(6 – 2) * (5 + 4)</td>
<td>6 2 - 5 4 + *</td>
</tr>
</tbody>
</table>
Application of Stack - Evaluating Postfix Expression

- The following algorithm uses a stack to evaluate a postfix expression.

Start with an empty stack
for (each item in the expression) {
    if (the item is a number)
        Push the number onto the stack
    else if (the item is an operator){
        Pop two operands from the stack
        Apply the operator to the operands
        Push the result onto the stack
    }
}
Pop the only one number from the stack – that’s the result of the evaluation
Application of Stack - Evaluating Postfix Expression

- Example: Consider the postfix expression, \( 2 \ 10 \ + \ 9 \ 6 \ - \ / \), which is \((2 + 10) / (9 - 6)\) in infix, the result of which is \(12 / 3 = 4\).

- The following is a trace of the postfix evaluation algorithm for the above.

```
2 \ 10 \ + \ 9 \ 6 \ - \ /
```

```
push 2
push 10
push 2 + 10 = 12
pop 10
pop 2
push 9
push 6
push 9 - 6 = 3
pop 9
pop 6
pop 3
pop 12
push 12 / 3 = 4
pop answer: 4
```