Page No. 282
Date: 15/08/24

STACKS

* जब Memory हमारे लिए बनाती है तो उसे implicit stack कहते हैं।
* जब हम अपने लिए बनाते हैं तब उसे explicit stack कहते हैं।

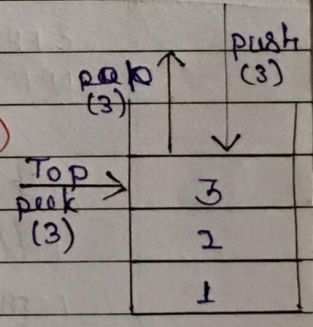
Def.→ Stack refers to a collection or arrangement of items placed one on top of the other.

Ex.→ Stack of plates or books.

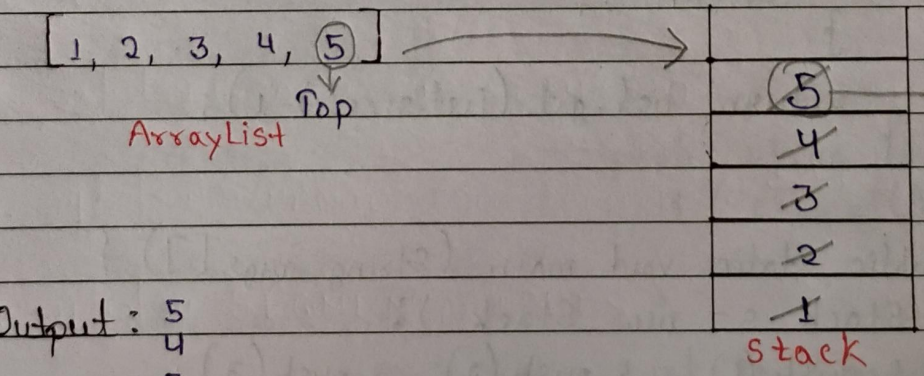
⇒ A Stack is a linear data structure that follows LIFO principle.

* Operation on Stack

i) Push : add an element to the top. O(1)
ii) Pop : remove the element from the top. O(1)
iii) Peek : retrieve the top element without removing it. O(1)



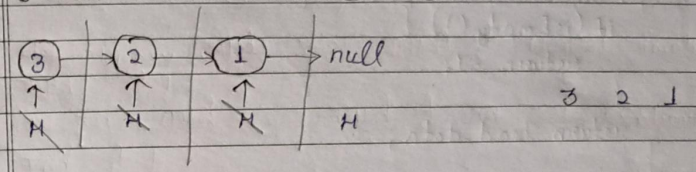
* Stack Using ArrayList



Output : 5
4
3
2
1

⇒
```
public class StackinAL {
  public static class Stack {
    static ArrayList<Integer> list = new ArrayList<>();
    public static boolean isEmpty() { //Empty check
      return list.size() == 0;
    }
    //push
    public static void push (int data) {
      list.add (data);
    }
    //pop
    public static int pop() {
      if (isEmpty()) {
        return -1;
      }
      int top = list.get (list.size()-1);
      list.remove (list.size()-1);
      return top;
    }
    //peek
    public static int peek() {
      if (isEmpty()) {
        return -1;
      }
      return list.get (list.size()-1);
    }
  }
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push (1); s.push (2); s.push (3);
    while (!s.isEmpty()) {
      Sys o (s.peek()); s.pop();
    }
  }
}
```


★ STACK USING LINKED LIST



⇒
```
public class StackLL {
  static class Node {
    int data;
    Node next;
    Node (int data) {
      this.data = data;
      this.next = null;
    }
  }
  static class Stack {
    static Node head = null;
    public static boolean isEmpty() {
      return head == null;
    }
  }
  public static void push (int data) {
    Node newNode = new Node (data);
    if (isEmpty()) {
      head = newNode;
      return;
    }
    newNode.next = head;
    head = newNode;
  }
```

```
public static int pop() {
  if (isEmpty()) {
    return -1;
  }
  int top = head.data;
  head = head.next;
  return top;
}
```

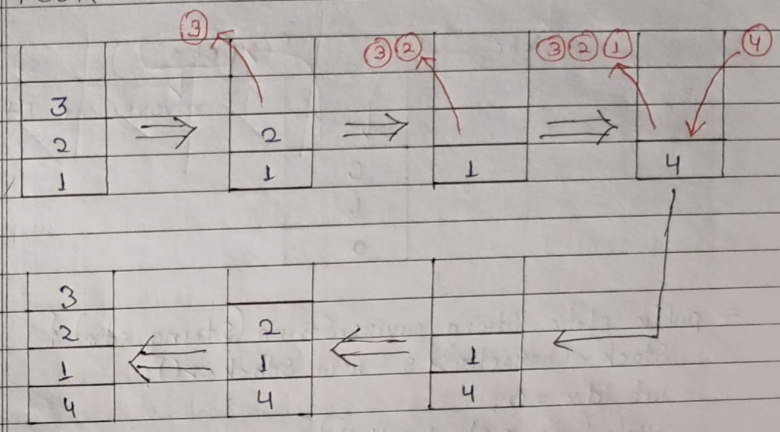
```
public static int peek() {
    if (isEmpty()) {
        return -1;
    }
    return head.data;
}
public static void main (String args[]) {
    Stack s = new Stack();
    s.push(1); s.push(2); s.push(3);
    while (!s.isEmpty()) {
        System.out.println (s.peek());
        s.pop();
    }
}
}
```

★ JAVA COLLECTIONS FRAMEWORK

```
⇒ public class StackJCF {
    public static void main (String args[]) {
        Stack <Integer> s = new Stack <>();
        s.push (1);
        s.push (2);
        s.push (3);
        while (!s.isEmpty()) {
            System.out.println (s.peek());
            s.pop();
        }
    }
}
```

Output : 3
2
1

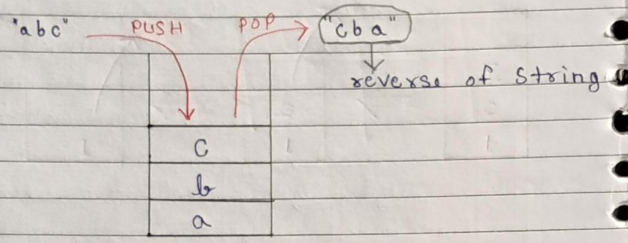
★ PUSH AT BOTTOM OF STACK




```
⇒ public class Stack {
    public static void pushAtBottom (Stack<Integer> s, int data) {
        if (s.isEmpty()) {
            s.push (data);
            return;
        }
        int top = s.pop();
        pushAtBottom (s,data);
        s.push (top);
    }
public static void main (String args []) {
    Stack <Integer> s = new Stack <>();
    s.push (1);
    s.push (2);
    s.push (3);
    pushAtBottom (s, 4);
    while (!s.isEmpty()) {
        System.out.println (s.pop());
    }
}
```

3

★ REVERSE A STRING USING STACK

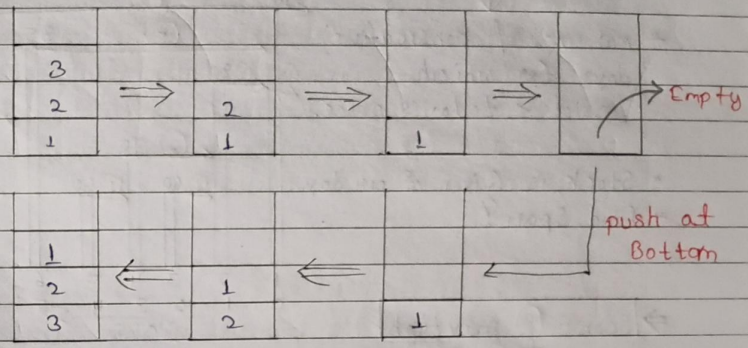


⇒
```
public static String reverseString (String str) {
  Stack <Character> s = new Stack<>();
  int idx = 0;
  While (idx < str.length()){
    s.push (str.charAt (idx));
    idx++;
  }
  StringBuilder result = new StringBuilder ("");
  While (!s.isEmpty()) {
    char curr = s.pop();
    result.append (curr);
  }
  return result.toString();
}

public static void main (String args[]) {
  String str = "abc";
  String result = reverseString (str);
  System.out.println (result);
}
```

⇒ Output : cba

★ REVERSE A STACK



⇒
```
public static void pushAtBottom {
}
```
→ Pg no. 286

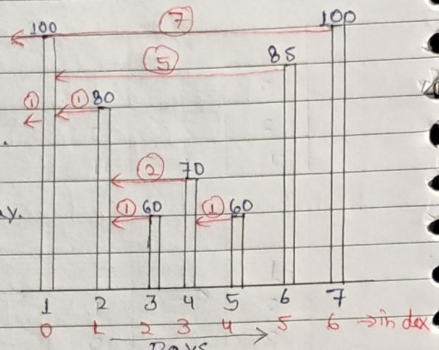
```
public static void reverseStack (Stack<Integer> s) {
  if (s.isEmpty()) {
    return;
  }
  int top = s.pop();
  reverseStack (s);
  pushAtBottom (s, top);
}

public static void printStack (Stack<Integer> s) {
  While (!s.isEmpty()) {
    System.out.println (s.pop());
  }
}

public static void main (String args[]) {
  Stack<Integer> s = new Stack<>();
  s.push (1); s.push(2); s.push(3);
  printStack(s); ↓ reverseStack(s); printStack(s);
}
```
reinitialise again

★ STOCK SPAN PROBLEM

* max no. of consecutive days for which price ≤ today's price.
* Stock is Given of per day.
* Find Span?



⇒ Span = (i) - prevHigh
index

⇒ 1 1 1 2 1 5 6
0 1-0 2-1 Span 4-3 5-0 7-1
3-1 6

* If we imagine as a stack, then at last stack will be empty.

⇒
```
public static void stockSpan (int stocks[], int span[]){
    Stack<Integer> s = new Stack<>();
    span[0] = 1;
    s.push(0);
    for (int i=1; i<stocks.length; i++) {
        int currPrice = stocks[i];
        while (!s.isEmpty() && currPrice > stocks[s.peek()]){
            s.pop();
        }
        if (s.isEmpty()) {
            span[i] = i+1;
        } else {
            int prevHigh = s.peek();
            span[i] = i - prevHigh;
        } s.push(i);
    }
}
```

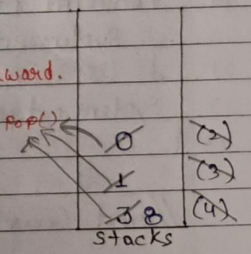


```
public static void main (String args[]) {
    int stocks[] = {100, 80, 60, 70, 60, 85, 100};
    int span[] = new int [stocks.length];
    stockSpan (stocks, span);
    for (int i=0; i<span.length; i++) {
        System.out.print(span[i] + " ");
    }
}
```

★ NEXT GREATER ELEMENT

* The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.
→ arr = [6, 8, 0, 1, 3]
⇒ next Greater = [8, -1, 1, 3, -1]
6<8 | 8<null | 0<1 | 1<3 | 3<null
next greater element of -(6) (8) (0) (1) (3)

* Approach // O(n)
→ arr = [6, 8, 0, 1, 3]
* We will move backward.
* We will make an array of same above length.
→ nextGreater[] = [8, -1, 1, 3, -1, X]
0 1 2 3 4



S1: while (s.empty && stack[top] ≤ arr[i])
s.pop().
S2: if (stack.empty)
nextGreater = -1
else
nextGreater = s.peek()
S3: s.push(element)

* We will perform these steps in above Dry run.

// next Greater right

```
1  public static void main (String args []) { //O(n)
2    int arr [] = {6, 8, 0, 1, 3};
3    Stack <Integer> s = new Stack <>();
4    int nxtGreater [] = new int [arr.length];
5    for (int i=arr.length-1; i>=0; i--) {
6      //step 1 - while loop
7      while (!s.isEmpty() && arr[s.peek()] <= arr [i]) {
8        s.pop();
9      }
10     //step 2 - if-else statement
11     if (s.isEmpty()) {
12       nxtGreater [i] = -1;
13     } else {
14       nxtGreater [i] = arr [s.peek()];
15     }
16     //step 3 - push element is stack
17     s.push (i);
18   }
19   for (int i = 0; i<nxtGreater.length; i++) {
20     System.out.print (nxtGreater [i] +" ");
21   }
22   System.out.println();
23 }
```

// next Greater left - we have to only change line 5 loop.

// next Smaller right - we have to only change line 7 second condition symbol (<=).

// next Smaller left - we have to change both lines 5 & 7 loop & condition.

★ VALID PARENTHESES

Ques. Given a string 's' containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

* An input string is valid if :

i) Open brackets must be closed by same type of brackets.

ii) Open brackets must be closed in correct order.

iii) Every close bracket has a corresponding open bracket of same type.

Ex: → s = "()[]{}" - true → s = "(]" - false

→ s = "()" - true → s = ")(" - false

* Approach

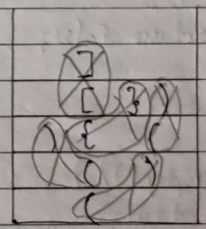
S1: opening bracket
S.push()

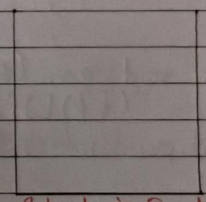
S2: closing bracket
stack.top
pair true;
false;

pop()

S3: Check stack.isEmpty

Ex → ([{[]}()])





Stack is Empty

```
⇒ public static boolean isValid (String str) {
    Stack <Character> s = new Stack<>();
    for (int i=0; i<str.length(); i++) {
      char ch = str.charAt(i);
      if (ch == '(' || ch == '{' || ch == '[') {
        s.push(ch);
      } else {
        return false;
        if (s.isEmpty()) {
          return false;
        }
        if((s.peek() == '(' && ch == ')')
          ||(s.peek() == '{' && ch == '}')
          ||(s.peek() == '[' && ch == ']') ) {
          s.pop();
        } else {
          return false;
        }
      }
    }
    if (s.isEmpty()) {
      return true;
    }else {
      return false;
    }
  }
  public static void main (String args[]) {
    String str = "({})[]";
    System.out.println(isValid(str));
  }
```

⇒ Output: true

★ DUPLICATE PARENTHESES

Qus. Given a balanced expression, find if it contains duplicate parentheses or not. A set of parentheses are duplicate if the same subexpression is surrounded by multiple parentheses.

* Return a true if it contains duplicates else return false.

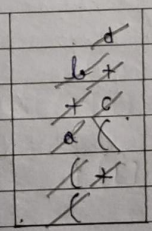
Ex: i) (((a+(b))) + (c+d)) - true
ii) ((((a)+(b)) + c+d)) - true
iii) ((a+b) + (c+d)) - false
iv) (((a+b)) +c) - true

* Approach // O(n)

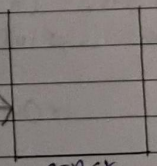
⇒ ((a+b) + (c+d))

S1: opening brackets, operands, operators ——→ push().

S2: int count = 0
while (s.peek() == '(') {
  s->pop()
  count ++i;
} count < 1 ——→ duplicate //true
s.pop() ← '('



→ count = ~~0~~ 1 2 3
→ count = 0 1 2 3
→ count = 0 1



* At last empty.

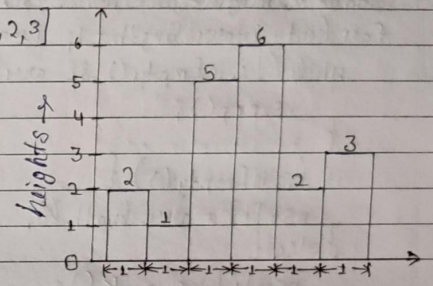
⇒
```
public static boolean isDuplicate(String str) {
    Stack<Character> s = new Stack<>();
    for (int i=0; i<str.length(); i++) {
        char ch = str.charAt(i);
        //closing
        if (ch == ')') {
            int count = 0;
            while (s.peek() != '(') {
                s.pop();
                count++;
            }
            if (count < 1) {
                return true; //duplicate
            } else {
                s.pop(); //opening pair
            }
        } else {
            s.push(ch); //opening
        }
    }
    return false;
}
public static void main (String args[]) {
    //valid String
    String str = "((a+b))"; //true
    String str2 = "(a-b)"; //false
    System.out.println(isDuplicate(str));
}
```

* Output → true

★ MAX AREA IN HISTOGRAM

* Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

→ Input = [2,1,5,6,2,3]



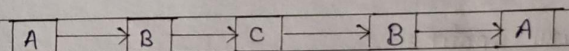
→ Output: Ans = 10.

```
⇒ public static void maxArea (int arr[]) {
    int maxArea = 0;
    int nsr[] = new int [arr.length];
    int nsl[] = new int [arr.length];
    // Next Smaller Right
    Stack<Integer> s = new Stack<>();
    for (int i=arr.length-1; i>=0; i--) {
      while (!s.isEmpty() && arr[s.peek()] >= arr[i]) {
          s.pop();
      }
      if (s.isEmpty()) {
        nsr[i] = arr.length;
      } else {
        nsr[i] = s.peek();
      }
      s.push(i);
    }
    // Next Smaller Left
    s = new Stack<>();
    for (int i=0; i<arr.length; i++) {
      while (!s.isEmpty() && arr[s.peek()] >= arr[i]) {
          s.pop();
      }
      if (s.isEmpty()) {
        nsl[i] = -1;
      } else {
        nsl[i] = s.peek();
      }
      s.push(i);
    }
    // Next Page
```



```
    // Current Area : width = j - i - 1 = nsr[i] - nsl[i] - 1
    for (int i=0; i<arr.length; i++) {
      int height = arr[i];
      int width = nsr[i] - nsl[i] - 1;
      int currArea = height * width;
      maxArea = Math.max(currArea, maxArea);
    }
    System.out.println("Max area in Histogram = " + maxArea);
  }
  public static void main (String args[]) {
    int arr[] = {2, 1, 5, 6, 2, 3}; // Heights in Histogram
    maxArea(arr);
  }
```

Ques 1. Palindrome Linked List — We have a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.



* Input : A → B → C → B → A
* Output : Yes It is Palindrome

Sol ⇒

```
class Node {                    // Time Complexity, → O(n)
  int data;                     // Space Complexity, → O(n)
  Node ptr;
  Node (int d) {
    ptr = null;
    data = d;
  }
}
```

```
public static boolean isPalindrome (Node head) {
    Node slow = head;
    boolean ispalin = true;
    Stack<Integer> stack = new Stack<Integer>();
    while (slow != null) {
        stack.push (slow.data);
        slow = slow.ptr;
    }
    while (head != null) {
        int i = stack.pop();
        if (head.data == i) {
            ispalin = true;
        } else {
            ispalin = false;
            break;
        }
        head = head.ptr;
    }
    return ispalin;
}
public static void main (String args[]) {
    Node one = new Node (1);    Node two = new Node (2);
    Node three = new Node (3);  Node four = new Node (4);
    Node five = new Node (3);   Node six = new Node (2);
    Node seven = new Node (1);
    one.ptr = two;  two.ptr = three;  three.ptr = four;
    four.ptr = five;  five.ptr = six;  six.ptr = seven;
    boolean condition = isPalindrome (one);
    System.out.println ("Palindrome : " + condition);
}
```


Ques2: Simplify Path - We have an absolute path for a file (Unix-style), simplify it. Note that absolute path always begin with '/' (root directory), a dot in path represent current directory & double dot represents parent directory.

* Input: /apnacollege/
* Output: /apnacollege

* Input: /a/..
* Output: /

Sol =>

```
import java.io.*;                    // Time Complexity, O(n)
import java.util.*;                  // Space Complexity, O(1)
public static String simplify (String A) {
    Stack<String> st = new Stack<String>();
    String res = "";
    res += "/";
    int len_A = A.length();
    for (int i=0; i<len_A; i++) {
        String dir = "";
        while (i<len_A && A.charAt(i) == '/') {
            i++;
        }
        while (i<len_A && A.charAt(i) != '/') {
            dir += A.charAt(i);
            i++;
        }
        if (dir.equals("..") == true) {
            if (!st.empty()) {
                st.pop();
            }
        } else if (dir.equals(".") == true) {
            continue;
        } else if (dir.length() != 0) {
            st.push(dir);
        }
    }
```

```
Stack<String> st1 = new Stack<String>();
while (!st.empty()) {
    st1.push(st.pop());
}
while (!st1.empty()) {
    if (st1.size() != 1) {
        res += (st1.pop() + "/");
    } else {
        res += st1.pop();
    }
}
return res;
}
public static void main (String args[]) {
    String str = new String("/a/./b/../../c/");
    String res = simplify(str);
    System.out.println(res);
}
```

Ques 3. Decode a String - We have an encoded string s & the task is to decode it. The pattern in which the strings are encoded is as follows:

* Input - 2[cv]
* Output - cvcv
* Input - 3[b2[v]]L
* Output - bvvbvvbvv

Sol ⇒

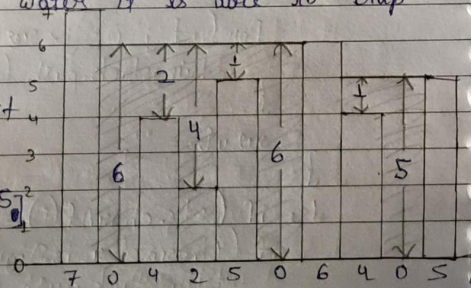
```
import java.util.Stack;
public static String decode(String str) {
    Stack<Integer> integerstack = new Stack<>();
    Stack<Character> stringstack = new Stack<>();
    String temp = "", result = "";
    for (int i = 0; i < str.length(); i++) {
        int count = 0;
```


Q There are many errors so study the code of pc.

```
if (Character.isDigit(str.charAt(i))) {
    while (Character.isDigit(str.charAt(i))) {
        count = count * 10 + str.charAt(i) - '0';
        i++;
    }
    i--;
    integerstack.push(count);
} else if (str.charAt(i) == ']') {
    temp = "";
    count = 0;
    if (!integerstack.isEmpty()) {
        count = integerstack.peek();
        integerstack.pop();
    }
    while (!stringstack.isEmpty() && stringstack.peek() != '[') {
        temp = stringstack.peek() + temp;
        stringstack.pop();
    }
    if (!stringstack.empty() && stringstack.peek() == '[') {
        stringstack.pop();
    }
    for (int j = 0; j < count; j++) {
        result = result + temp;
    }
    for (int j = 0; j < result.length(); j++) {
        stringstack.push(result.charAt(j));
    }
    result = "";
}
else if (str.charAt(i) == ']') {
    if (Character.isDigit(str.charAt(i-1))) {
        stringstack.push(str.charAt(i));
```

```
        else {
            stringstack.push(str.charAt(i));
            integerstack.push(1);
        }
    }
    else {
        stringstack.push(str.charAt(i));
    }
}
while (!stringstack.isEmpty()) {
    result = stringstack.peek() + result;
    stringstack.pop();
}
return result;                    // TC, → O(n)
}                                 // SC, → O(n)
public static void main (String args[]) {
    String str = "3[b2[ca]]";
    System.out.println(decode(str));
}
```

Ques 4. Trapping Rainwater - We have an array of N non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.



* NOTE: We have solve it using stack.
* Input: [7 0 4 2 5 0 6 4 0 5]
* Output: 25

Sol:

```
import java.io.*;
import java.util.*;
public static int maxWater (int[] height) {
    Stack<Integer> stack = new Stack<>();
    int n = height.length;
    int ans = 0;
    for (int i = 0; i<n; i++) {
        while((!stack.isEmpty()) && (height[stack.peek()] < height[i])) {
            int pop_height = height[stack.peek()];
            stack.pop();
            if (stack.isEmpty()) {
                break;
            }
            int distance = i - stack.peek() - 1;
            int min_height = Math.min(height[stack.peek()],
                                      height[i]) - pop_height;
            ans += distance * min_height;
        }
        stack.push(i);
    }
    return ans;
}
public static void main (String[] args) {
    int arr[] = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
    System.out.print (maxWater (arr));
}
```