Thomas Muamar

Assignment 2 – Stacks

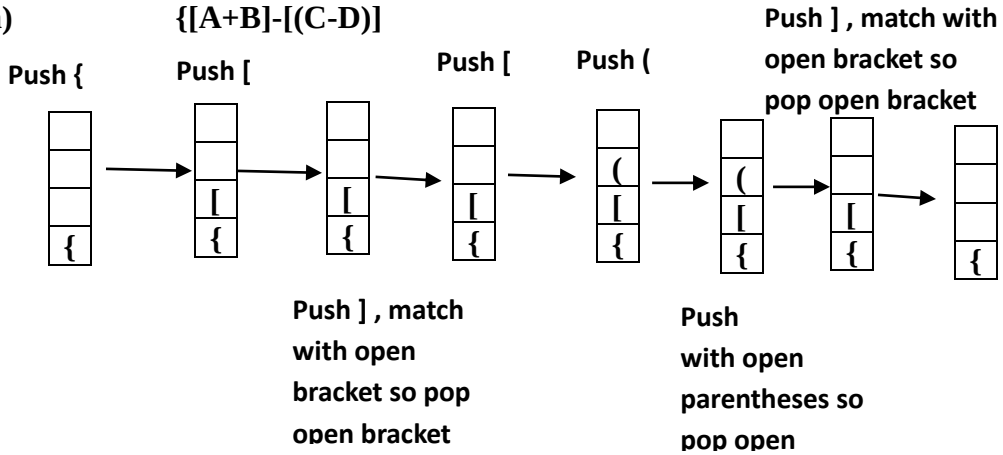
1. a)

First step: initialize another stack which is S_2 =stack()
Second Step: Then check if stack S_1 is empty by using the empty operation if not go to next step. If stack is empty go to Fifth step.
Third Step: Pop S_1 elements from the top and push them into S_2
Fourth Step: Use Empty method to see if it returns S_1 stack empty
Fifth Step: Use peek method to check what the top of the S_2 stack produces
Sixth Step: set variable i to what was returned as the value of the top of S_2
Seventh Step: Pop S_2 elements from the top of stack and Push them into S_1

b)
First step: initialize another stack which is S_2 = stack()
Second Step: Use Empty to check if stack is empty if not go to next step
Third Step: Pop S_1 elements from the top of stack and push to S_2
Fourth Step: Use isEmpty to check if S_1 is empty
Fifth Step: Pop element from top of Stack S_2 and push to S_1
Sixth Step: Pop element from top of Stack S_2 and Push to S_1
Seventh Step: Peek method to check top of S_2 stack and set variable I to that element.
Eighth step: Pop rest of S_2 elements from the top of stack and push them to S_1

2.

a) **{[A+B]-[(C-D)]**



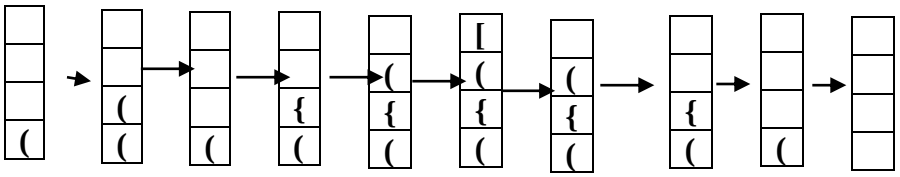
The delimiter at the end of the string are done but the bottom of the stack is left with a {.

b) ((H) * {([J+K])})

1. **Push (**
2. **Push (**
3. **Push), since they match the we pop the (and the top becomes just the (**
4. **Push {**

5. **Push (**
6. **Push [**
7. **Push], match so we pop the open bracket**
8. **Push), match so we pop the (**
9. **Push } pop the { since it's a match**
10. **Push (since it's a match we pop the open bracket.**

We have a empty stack so the equation shows to have a balance with the delimiters.



3. **Input character string would be equal to ABABBACABBABA**

Initialize stack(stack)
While pushes string into stack until C.
whileInput cannot equal c
stack.push(Input)
input = nextinput
Initialize a second while loop.
While input cannot equal the endInput then pop(stack).
If (input != pop(stack)
Then exit and string is not xCy
Else if (Input = nextinput)
Then String is xCy
End second while loop
End first while loop

4. **Taking my approach with three and adding on a extra stack.**

Initialize stack(stack1, stack2)
While input cannot equal D then
Stack.Push((Input)

Do {
If input cannot equal endInput
Exit It is not in form
Else if (Input = pop(stack))
Stack2.Push(input);
}while input cannot equal D
While the input != D then if the input at the end of the string is equal to the initial input then it is xCy
If the input is equal to pop the stack2 then push it into stack1.
Repeat do while loop to process next portion of string.
End while loop

5.

Inserting in such an array:
First step: initialize stack1 and stack2. We are assuming that Stack1 contains values while stack2 is empty.
Second step: Use isEmpty method to depict whether stack is empty if not go to next step.
Third Step: If(!stack1.empty) then pop stack1 elements from top of stack and push into stack2 stack stack2.push(stack1.pop()).
Fourth step: stack1.push(value) – allows for the insertion of the element to the first stack.
Fifth step: If(!stack2.empty) since this value is false I then Pop stack2 back into stack1.
Sixth step: stack1.push(stack2.pop())
End

In order to read values from such an array:
First step: initialize stack 1 and stack2
Second step: Use peek operation to display top of the operation in stack1 if empty will return null.
Third step: stack2.push(stack1.pop()) to push first operation from stack 1 to stack2.
Fourth step: in order to read what is on top of stack1 now you use peek operation again if you don't want to remove a value from the stack peek operation works the best to read the value.

6.

```
Class Stacks{
//initializing variables
int = x
int spaceSize = x;
int topS1;
int topS2;
int[] s;
   public stacks(){
s = new int [spaceSize];

Boolean emptyS1(){
If(topS1 == -1)
Return true;
Else
Return false;

Boolean emptyS2(){
If(topS2 == spaceSize)
Return true;
Else
Return false;

public void push1(int 5)
if(topS1 > 0){
s[topS1++] = 5
}else
System.out.print("overflow");

public void push2(int 5)
```

```
if(topS2 < spaceSize - 1){
s[topS2—] = 5
}else
System.out.print("overflow");

Public void pop1()
If(emptyS1() == true)
System.out.print("s1 underflow");
Else{
Return s[topS1—];

Public void pop2()
If(emptyS2() == true)
System.out.print("s2 underflow");
Else{
Return s[topS2++];
```

7.

a. (A+B)*(C$(D-E)+F)-G
   Prefix:-*+AB+$C-DEFG
   Postfix: AB+CDE-$F+*G-
b. A+(((B-C)*(D-E)+F)/G)$(H-J)
   Prefix: +A$/+*-BC-DEFG-HJ
   Postfix: ABC-DE-*F+G/HJ-$+

8.

a. ++A-*$BCD/+EF*GHI
   A+(B$C*D-((E+F)/(G*H)))+I
b. +-$ABC*D**EFG
   (A$B)-C)+D*(E*F*G)
c. AB-C+DEF-+$
   (A-B+C)$(D+(E-F))
d. ABCDE-+$*EF*-
   (A*B$(C+(D-E)))-(E*F)

9.

a. AB+C-BA+C$-
   ((A+B)-C)-(B+A)$C
   ((1+2)-3) – (1+2)^3
   = -27
b. ABC+*CBA-+*
   A*(B+C)*(C+(B-A))
   =1*(2+3)*(3+(2-1))
   =5*4

= 20

10.

Method:

Infixtoprefix

1. **Read the infix string**
2. **Initialize prefix as a string**
3. **Initialize a stack.**
4. **If the expression has an Operand then push into prefix string**
5. **If(expression==')') then push the expression on to the stack**
6. **If the input is a (use peek method. While(stack.peek() != ')') then pop value of that stack and set it to the top. Top = stack.pop().**
7. **Else if(stack.empty){**
   Stack.push(expression)
8. **Now if precedence of the next expression is > that of the peek/top of the stack then stack.push(expression)**
9. **Else discard the top into the top= stack.pop() and then push the next expression.**