Assignment 2 - Stacks

Q1.

a) 1. Given a stack s1, create a new empty stack s2.
   2. While s1 is not empty (can be check by peeking at the top of s1):
      Pop an element from s1 and push it to s2.
   3. Peek the top of the s2 and assign its value to a variable i
   4. While s2 is not empty (can be check by peeking at the top of s2):
      pop the top of s2 and push it to s1.

   Pseudocode:

```
Stack s1, s2

While ( !s1.isEmpty()):
        s2.push(s1.pop());
i = s2.peek()
While (!s2.isEmpty()):
        s1.push(s2.pop())
```

b) 1. Given a stack s1, create a new empty stack s2
   2. While s1 is not empty:
      Pop an element from s1 and push it to s2
   3. After elements in s1 are popped, pop the top element of s2 two times
   4. Peek at the top of the s2 and assign its value to i

   Pseudocode:

```
While (!s1.isEmpty()):
        s2.push(s1.pop())

s2.pop();
s2.pop();

i = s2.peek();
```

Q2.

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

Already Read	Remaining	Delimiters stack (RIght: top)	Comment
{	[A+B]-[(C-D)]	{	Push {
{[	A+B]-[(C-D)]	{[	Push [
{[A+B	]-[(C-D)]	{[	Skip non-delimiters
{[A+B]	-[(C-D)]	{	"]" matches the top of the stack "[", pop [
{[A+B]-	[(C-D)]	{	Skip non-delimiter
{[A+B]-[	(C-D)]	{[	Push [
{[A+B]-[(	C-D)]	{[(	Push (
{[A+B]-[(C-D	)]	{[(	Skip non-delimiter
{[A+B]-[(C-D)	]	{[	Closing parenthesis) matched the open parentheses on the top. Pop (
{[A+B]-[(C-D)]		{	Closing bracket] matched the open bracket on the top. Pop [. There's a curly bracket remaining, error.

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

Already Read	Remaining	Delimiters stack (Right: top)	Comment
(	(H) * {([J+K])})		Push (
((	H) * {([J+K])})	((	Push (
((H	) * {([J+K])})	((	skip

((H)	* {([J+K])})	(	Closing paren matches the top of the stack: Pop (
((H) *	{([J+K])})	(	skip
((H) * {([	J+K])})	({([	Push {, (, [
((H) * {([J+K	])})	({([	skip
((H) * {([J+K]	)})	({(	Closing bracket matches the top: Pop [
((H) * {([J+K])	})	({	Closing parenthesis matches the top: pop (
((H) * {([J+K])}	)	(	Closing curly bracket matches the top: pop {
((H) * {([J+K])})			Closing parenthesis matches the top: pop (The delimiters stack is empty: OK!

Q3.

1. Create a empty stack
2. Read a character from the left of the input string
3. If the character is not ‘C’, push the character to the stack and go back to step 2 and keep reading.
   If the next character is ‘C’, go to step 4.
4. Read a character from the input and pop the top character from the stack, then check if they are the same. If yes, keep reading, popping and checking. If not, return false.
5. If all characters are read,the stack is empty, and no false has been returned in the previous step, return true. Otherwise, return false

Pseudocode:

```
Stack s1 = new stack;
Char ch = the left most character of the input
```

```
// after the loop below, all char before ‘C’ is read and pushed to the stack
while(ch != ‘C’){
        s1.push(ch)
        Ch = read a new character from the input string
}


// after this loop, if the string in the right format, s1 is empty and the all chars are read
While ( not s1.isEmpty && not all characters are read){
        Ch = read a new char from the input
        If (ch != s1.pop) return false
}

Return s1.isEmpty && all chars are read
```

Q4.

1. Create a empty stack
2. Keep reading a next character and push it to the stack while it is not ‘C’,
3. If the next character is ‘C’, read the next character and pop the top character from the stack, and check if they are the same. If yes, keep reading, popping and checking until the next char is ‘D’, all characters in the input string are read, or the stack is empty. If the two characters are different, return false.
4. If the conditions: ‘D’ is read & the stack is empty, are satisfied at the same time, go to step 2 and repeat checking the format of xCy.
   If all characters are read and the stack is empty, return true.

Pseudocode:

```
stack s1 = new stack;
Char ch = the left most character of the input

while (ch!= null){
  while (ch! = 'C' && ch!= null ){
     s1.push(ch);
     ch = read a new character;
  }
  // chars before C are read by here

  // get rid of C
  ch = read a new character

  while (ch! ='D' && ch!= null){
```

```
            // if s1 is empty, means the string before C is shorter, return false
            // if ch and s1.pop() doesn't match, return false
            if (s1.isEmpty || ch != s1.pop()) return false
            ch = read a new character
        }
        //  ch is now D, and all characters before D are checked

        // if there are elements remaining in the stack, means the string after C is shorter
        if (!s1.isEmpty()) return false;

        // get rid of D, and loop back
        ch = read a new character
    }

    return true;
```

Q5.
Assume there are two stacks: One is holding values of the array (s1), the other one is a helper stack (s2).

```
For inserion: insert(s1, s2, value, index)
// pop elements in s1 and push it to s2.
while(!s1.isEmpty()):
    s2.push(s1.pop())

// by now, all elements are stored in the s2 in reverse order.

// push back elements to s1 for index times.
For index times:
    s1.push(s2.pop())

// insert a value at the specific index, and delete the original value at the index by popping.
s1.push(value)
s2.pop()

// push back the remaining
while(!s2.isEmpty()):
    s1.push(s2.pop())


For read: read(s1, s2, index)
```

```
// pop elements in s1 and push it to s2.
while(!s1.isEmpty()):
        s2.push(s1.pop())

// by now, all elements are stored in the s2 in reverse order.

// push back elements to s1 for index times.
For index times:
        s1.push(s2.pop())

// get the value of the top of the stack s2.
Ans = s2.peek()

// push back the remaining
while(!s2.isEmpty()):
        s1.push(s2.pop())
```

Q6.

Two stacks can be kept in the single array s[SPACESIZE] with two integer pointers p1 =0 , p2 = SPACESIZE-1. Stack 1 can be stored from the head to the array, and the stack2 can be stored from the tail of the array.

```
push1(s, p1, value){
        s[p1] = value;
        p1++;
}

push2(s, p2, value){
        s[p2] = value;
        p2--;
}

pop1(s,p1){
        res = s[p1];
        p1--;
        Return res
}

pop2(s,p2){
        res = s[p2];
        p2++;
        Return res
```

}

Q7.

a. (A + B) * (C $ (D-E) + F) -G

Prefix:

Read	Stack (top: left)	Output
-G	-	G
) -G	)-	G
+F)-G	+)-	FG
)+F)-G	)+)-	FG
-E)+F)-G	-)+)-	EFG
D-E)+F)-G	-)+)-	DEFG
(D-E)+F)-G	+)-	-DEFG
$ (D-E) + F) -G	$+)-	-DEFG
C $ (D-E) + F) -G	$+)-	C-DEFG
(C $ (D-E) + F) -G	-	+$C-DEFG
* (C $ (D-E) + F) -G	*-	+$C-DEFG
)* (C $ (D-E) + F) -G	)*-	+$C-DEFG
+ B)* (C $ (D-E) + F) -G	+)*-	B+$C-DEFG
A + B)* (C $ (D-E) + F) -G	+)*-	AB+$C-DEFG
(A + B)* (C $ (D-E) + F) -G	*-	+AB+$C-DEFG
(A + B)* (C $ (D-E) + F) -G		-*+AB+$C-DEFG

Postfix:

Read	Stack (top:right)	Output

(	(	
(A	(	A
(A+B	(+	AB
(A+B)		AB+
(A+B) *	*	AB+
(A+B) * (	*(	AB+
(A+B) * (C $	*($	AB+C
(A+B) * (C $ (	*($(	AB+C
(A+B) * (C $ (D	*($(	AB+CD
(A+B) * (C $ (D - E	*($(-	AB+CDE
(A+B) * (C $ (D - E)	*($	AB+CDE-
(A+B) * (C $ (D - E) +	*(+	AB+CDE-$
(A+B) * (C $ (D - E) + F	*(+	AB+CDE-$F
(A+B) * (C $ (D - E) + F)	*	AB+CDE-$F+
(A+B) * (C $ (D - E) + F)-	-	AB+CDE-$F+*
(A+B) * (C $ (D - E) + F)- G	-	AB+CDE-$F+*G
(A+B) * (C $ (D - E) + F)- G		AB+CDE-$F+*G-

b. A+ (((B-C) * (D-E) + F)/G)$ (H-J)
Prefix:

Read	Stack (top: left)	Output
-J)	-)	J
H-J)	-)	HJ
(H-J)		-HJ
$ (H-J)	$	-HJ
G) $ (H-J)	)$	G-HJ

/G) $ (H-J)	/)$	G-HJ
)/G) $ (H-J)	)/)$	G-HJ
+F)/G) $ (H-J)	+)/)$	FG-HJ
)+F)/G) $ (H-J)	)+)/)$	FG-HJ
D-E)+F)/G) $ (H-J)	-)+)/)$	DEFG-HJ
(D-E)+F)/G) $ (H-J)	+)/)$	-DEFG-HJ
* (D-E)+F)/G) $ (H-J)	*+)/)$	-DEFG-HJ
) * (D-E)+F)/G) $ (H-J)	)*+)/)$	-DEFG-HJ
B-C) * (D-E)+F)/G) $ (H-J)	-)*+)/)$	BC-DEFG-HJ
(B-C) * (D-E)+F)/G) $ (H-J)	*+)/)$	-BC-DEFG-HJ
((B-C) * (D-E)+F)/G) $ (H-J)	/)$	+*-BC-DEFG-HJ
(((B-C) * (D-E)+F)/G) $ (H-J)	$	/+*-BC-DEFG-HJ
+ (((B-C) * (D-E)+F)/G) $ (H-J)	+	$/+*-BC-DEFG-HJ
A + (((B-C) * (D-E)+F)/G) $ (H-J)	+	A$/+*-BC-DEFG-HJ
A + (((B-C) * (D-E)+F)/G) $ (H-J)		+A$/+*-BC-DEFG-HJ

Postfix:

Read	Stack (top: right)	Output
A +	+	A
A + (((	+(((	A
A + (((B- C	+(((-	ABC
A + (((B- C)	+((	ABC-
A + (((B- C) *	+((*	ABC-

A + (((B- C) * (	+((*(	ABC-
A + (((B- C) * (D-E	+((*(-	ABC-DE
A + (((B- C) * (D-E)	+((*	ABC-DE-
A + (((B- C) * (D-E)+	+((+	ABC-DE-*
A + (((B- C) * (D-E)+F	+((+	ABC-DE-*F
A + (((B- C) * (D-E)+F)	+(	ABC-DE-*F+
A + (((B- C) * (D-E)+F)/G	+(/	ABC-DE-*F+G
A + (((B- C) * (D-E)+F)/G)	+	ABC-DE-*F+G/
A + (((B- C) * (D-E)+F)/G) $	+$	ABC-DE-*F+G/
A + (((B- C) * (D-E)+F)/G) $(	+$(	ABC-DE-*F+G/
A + (((B- C) * (D-E)+F)/G) $(H-J	+$(-	ABC-DE-*F+G/HJ
A + (((B- C) * (D-E)+F)/G) $(H-J)	+$	ABC-DE-*F+G/HJ-
A + (((B- C) * (D-E)+F)/G) $(H-J)		ABC-DE-*F+G/HJ-$+

Q8.

a. ++A-*$BCD/+EF*GHI

   = ++A-*(B$C)D/(E+F)(G*H)I
   = ++A- [(B$C)*D][(E+F)/(G*H)]I
   = ++A{[(B$C)*D]-[(E+F)/(G*H)]}I
   = +(A + {[(B$C)*D]-[(E+F)/(G*H)]})I
   = A + B$C*D - (E+F)/(G*H) + I

b. +-$ABC*D**EFG

   = +- (A$B) C*D * (E*F) G
   = + (A$B -C)*D (E*F*G)
   = + (A$B -C) (D*E*F*G)
   = A$B -C + D*E*F*G

c. AB-C+DEF-+$

   = (A-B) C + D (E-F) +$
   = (A-B + C) (D +(E-F)) $
   = (A - B + C) $ (D + E -F)

d. ABCDE-+$*EF*-

   = ABC (D-E) +$* (E*F)-
   = AB(C + D - E) $* (E*F) -
   = A (B $(C+D-E)) * (E*F)-
   = (A * B $ (C+D-E)) (E*F) -
   = A * B $ (C+D-E) - (E*F)

Q9:

a. AB+C - BA + C $ -

   => 1 2 + 3 - 2 1 + 3 $ -

   Read from the left:

Read	Stack	operation
1 2 +	1, 2	+
1 2 +	3	
1 2 + 3 -	3, 3	-
1 2 + 3 -	0	
1 2 + 3 - 2 1	0, 2, 1	
1 2 + 3 - 2 1 +	0, 3	+
1 2 + 3 - 2 1 + 3	0, 3, 3	
1 2 + 3 - 2 1 + 3 $	0, 27	$
1 2 + 3 - 2 1 + 3 $ -	-27	-

⇒ Ans: -27

b. ABC+*CBA-+*

=> 1 2 3 + * 3 2 1 - + *

Read	Stack	Ops
1 2 3	1,2,3	
1 2 3 +	1, 5	+
1 2 3 + *	5	*
1 2 3 + * 3 2 1	5, 3, 2,1	
1 2 3 + * 3 2 1 -	5, 3, 1	-
1 2 3 + * 3 2 1 - +	5, 4	+
1 2 3 + * 3 2 1 - + *	20	*

⇒ Ans: 20

Q10:
Pseudocode:

```
infix2prefix(input) {
  String output;
  Stack s;

  for ch in input{ // starting from right
     if (ch is operand){
        prepend ch to output;
     } else {
        if (s.isEmpty() || ch == ')' ) {
           s.push(ch);
        } else if (ch == "("){
           while(s.peek() != ')'){
              prepend s.pop() to output
           }
           s.pop()
        } else if (ch.precedence >= s.peek().precedence){
                s.push(ch);
            } else {
                while (ch.precedence < s.peek().precedence){
                    prepend s.pop() to output
                }
                s.push(ch)
            }

        }
    }

    while (!s.isEmpty()){
        prepend s.pop to output
    }

    return output
}
```