**DS L-7**

**INFIX TO PRE FIX**

**A+B IN FIX**

**+AB**

**[(A+B)\*C/(D-E\*F)]^G (A+B), E\*F)**

**[ +AB \*C/ (D- \*EF)]^G (D- \*EF)**

**[ +AB \*C/ -D \*EF]^G +AB \*C**

**[\*+ABC / -D\*EF]^G [\*+ABC / -D\*EF]**

**[/\*+ABC-D\*EF]^G ^**

**^/\*+ABC-D\*EFG**

**[(A+B)\*C/(D-E\*F)]^G convert it into postfix**

**[AB+ \*C / DEF\*-]^G**

**AB+C\*DEF\*-/G^**

**a+b\*c-d –infix to Post fix USING STACK (abc\*+d-)**

|  |  |  |
| --- | --- | --- |
| **Scan Char** | **Stack (operators)** | **Output Postfix** |
| **A** | **#** | **a** |
| **+** | **#+** | **a** |
| **B** | **#+** | **a,b** |
| **\*** | **#+,\*** | **A,b** |
| **C** | **#+,\*** | **A,b,c** |
| **-** | **#-** | **A,b,c,\*,+,** |
| **D** | **#-** | **A,b,c,\*,+,D** |
| **#** | **#** | **A,b,c,\*,+,D-** |

**[(A+B)\*C/(D-E\*F)]^G --🡪 AB+C\*DEF\*-/G^**

**(((A+B)\*C)/(D-E\*F))^G AB+C\*DEF\*-/G^**

|  |  |  |
| --- | --- | --- |
| **Scan Char** | **Stack (operators)** | **Output Postfix** |
| **[** | **#[** |  |
| **(** | **#[,(** |  |
| **A** | **#[,(** | **A** |
| **+** | **#[,(,+** | **A** |
| **B** | **#[,(,+** | **AB** |
| **)** | **#[,** | **AB+** |
| **\*** | **#[,\*** | **AB+** |
| **C** | **#[,\*** | **AB+C** |
| **/** | **#[,/** | **AB+C\*** |
| **(** | **#[,/,(** | **AB+C\*** |
| **D** | **#[,/,(** | **AB+C\*D** |
| **-** | **#[,/,(,-** | **AB+C\*D** |
| **E** | **#[,/,(,-** | **AB+C\*DE** |
| **\*** | **#[,/,(,-,\*,** | **AB+C\*DE** |
| **F** | **#[,/,(,-,\*,** | **AB+C\*DEF** |
| **)** | **#[,/,** | **AB+C\*DEF\*-** |
| **]** | **#** | **AB+C\*DEF\*-/** |
| **^** | **#^** | **AB+C\*DEF\*-/** |
| **G** | **#^** | **AB+C\*DEF\*-/G** |
| **#** | **#** | **AB+C\*DEF\*-/G^** |
|  |  |  |
|  |  |  |
|  |  |  |

**DS L-8**

**Post fix - AB+C\*DEF\*-/G^ convert into**

**Infix (((A+B)\*C)/(D-E\*F))^G**

|  |  |  |
| --- | --- | --- |
| **Scan char** | **stack** | **O/P Infix** |
| **A** | **A** | **A** |
| **B** | **A,B** | **A,B** |
| **+** | **(A+B)** | **A+B** |
| **C** | **(A+B),C** | **A+B,C** |
| **\*** | **((A+B)\*C),** | **(A+B)\*C** |
| **D** | **((A+B)\*C),D** | **(A+B)\*C,D** |
| **E** | **(A+B)\*C,D,E** | **(A+B)\*C,D,E** |
| **F** | **(A+B)\*C,D,E,F** | **(A+B)\*C,D,E,F** |
| **\*** | **((A+B)\*C),D,(E\*F)** | **(A+B)\*C,D,E\*F** |
| **-** | **((A+B)\*C),(D-(E\*F))** | **(A+B)\*C,D-E\*F** |
| **/** | **((A+B)\*C)/(D-(E\*F))** | **((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** | **(((A+B)\*C)/(D-(E\*F)))^G** |
|  |  |  |
|  |  |  |
|  |  |  |

**AB+CD-\* OR A,B,+,C,D,-,\***

**6,2,3,+,-,3,8,2,/,+,\*,2,^,3,+**

|  |  |  |
| --- | --- | --- |
| **Scan char** | **stack** | **O/P Infix** |
| **6** | **6** | **6** |
| **2** | **6,2** | **6,2** |
| **3** | **6,2,3** | **6,2,3** |
| **+** | **6,5** | **6,2+3** |
| **-** | **1** | **6-(2+3)** |
| **3** | **0** | **6-(2+3),3** |
| **8** | **1,3,8** | **6-(2+3),3,8** |
| **2** | **1,3,8,2** | **6-(2+3),3,8,2** |
| **/** | **1,3,4** | **6-(2+3),3,8/2** |
| **+** | **1,7** | **6-(2+3),3+8/2** |
| **\*** | **7** | **(6-(2+3))\*(3+8/2)** |
| **2** | **7,2** | **(6-(2+3))\*(3+8/2),2** |
| **^** | **49** | **((6-(2+3))\*(3+8/2))^2** |
| **3** | **49,3** | **((6-(2+3))\*(3+8/2))^2,3** |
| **+** | **52** | **((6-(2+3))\*(3+8/2))^2+3** |
| **#** |  |  |

**Stack Applications**

1.**Well form-ness of Parenthesis/ Balanced parenthesis**

2. Expression conversion : **Infix to Postfix Conversion**

/Infix to prefix conversion

**3. Postfix Evaluation**

**4. Recursion:**

5.Reversal of data : ex: palindrome

**DS L-9**

Queue - Linear DS ( Q init F=R=-1)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | f4 | 5 | 6 | 7 | 8 | R9 |

🡨Delete <- Insert

Front end back / rear end

F ptr R ptr

First element Last element

F=R=-1 Q –Empty

I-2 F=R=0

I-6 F=0 R=1

I-9 F=0 R =2

D F=1 R=2

D F=2 R=2

D F=-1 R=-1 Q –Empty

QInsert(Q, N,F,R, D)

1.[Check for Q Overflow]

If (R==N-1) or (R==N)

Write(“Q is Overflow”)

Exit()

2.[Increase the Rear Pointer]

R++

3. [ Insert the element at the rear end of Q]

Q[R]=D

4.[set Front Pointer if Q was empty]

If(F==-1)

Then F=0

5.[Finish]

Return()

QDelete(Q,F,R,N)

1. [check the Q- underflow cond]

if(F==-1)

write(Queue is underflow)

exit()

2. [Remove the front element of the Q]

data=Q[F]

3. [Check for Q Empty]

If (F==R)

Then F=R=-1

4.[if not Q empty Update the Front Pointer]

F++

5. [finished]

return()

Qdisplay(Q,F,R)

If (f==-1)

Then write(Q is Empty)

else

For(i=F; i<=R; i++)

Write(q[i])

1. Circular Queue

F R

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7R |  | 3F | 8 | 9 | 10 | 4 | 50 | 6 | 8 |
| i0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



F=R=-1 empty CQ



0 G 1 B

3 E 2 D





I – A F=R=0

I-B F=0 R=1

I-D F=0 R=2 I –E F=0 R=3

D F=1 R=3 I –G F=1 R=0

I-H OVERFOLW

DELETE

0 1

3 2





I –G F=1 R=0

D – B F=2 R=0

D- D F=3 R=0

D- E F=0 R=0

D- G F=R=-1

CQInsert(CQ, N,F,R, D)

1.[ Increase the Rear Pointer]

If (R==N-1)

R=0

ELSE

R++

2.[ Check for CQ Overflow]

IF (R==F)

Write(“CQ is Overflow”)

Exit()

3. [ Insert the element at the rear end of CQ]

CQ[R]=D

4.[set Front Pointer if CQ was empty]

If(F==-1)

Then F=0

5.[Finish]

Return()