50	
48	3
44	
20	
10	

Size = 5

Top = -1

Push(10) push(20) push(30) push(40)

POP() ---40 top =2 PEEK() --- 11 top = 2 POP() -- 11 top = 1 push(12) top = 2 Push(13) top = 3

POP()--13 top =2 POP() --- 12 top = 1 POP() -- 20 top = 0 pop() – 10 top = -1

Push(60)

POP() pop() pop() top = 1 20 push(44) push(48)

Int S1[5], top1, S2[5],top2, S3[5], top3;

Top = -1;

Struct stack

{

Int items[5];

Int top;

};

S1, S2, S3;

Applications of Stacks:

Conversions & Evaluation of expressions

Function call mechanism

Tree Traversals

Graph Traversals

Conversion & Evaluation of expressions

Infix Expression:

Opnd1 Opr Opnd2

A + B

Postfix Expression (Polished notation):

Opnd1 Opnd2 Opr

A B +

Prefix Expression (Reversed Polished notation) :

Opr Opnd1 Opnd2

+ A B

Conversions :

Infix to Postfix and Prefix

Postfix to Infix and Prefix

Prefix to Infix and Postfix

Infix to Postfix :

1. A + B - C * D / E

-> A + B - **CD*** / E

-> A + B - **CD*E/**

> AB+ - CD*E/

> AB+ CD*E/-

A + (B - C) * (D / E)

A + **BC-** * **DE/**

A + **BC- DE/***

A **BC- DE/*+**

(A – (B + C)) * D ^ (E + F)

> (A - BC+) * D ^ (E + F)

> ABC+- * D ^ (E + F)

> ABC+- * D ^ EF+

> ABC+- * DEF+^

> ABC+-DEF+^*

(((A + B) * C) - (D – E)) ^ (F + G) ^ H

> ((AB+ * C) – (D – E)) ^ (F + G) ^ H

> (AB+C* - DE-) ^ (F + G) ^ H

> AB+C* DE- - ^ (F + G) ^ H

> AB+C* DE- - ^ FG+ ^ H

> AB+C* DE- - ^ FG+H^

> AB+C* DE- - FG+H^^

>

Infix to Prefix:

A + B - C * D / E

> A + B - *CD / E

> A + B - /*CDE

> +AB - /*CDE

> **- +AB/*CDE**

>

A + (B - C) * (D / E)

> A + -BC * /DE

> A + *-BC/DE

> +A*-BC/DE

(A – (B + C)) * D ^ (E + F)

> -A+BC * D ^ (E + F)

> -A+BC * D ^ +EF

> -A+BC * ^D+EF

> *-A+BC ^D+EF

(((A + B) * C) - (D – E)) ^ (F + G) ^ H

> ^ - *+ABC -DE ^+FGH

Postfix to Infix and Prefix

Scan from left to right until an operator is encountered.

When an operator is encountered, apply(infixing or prefixing) on to the two previously available operands. **The most recent one must be operand2.**

Resume scanning and repeat the process until the entire expression is scanned.

A B + C D * E / - 1 2 + 3 2 * 4 / -

> (A + B) C D * E / -

> (A + B) (C * D) E / -

> (A+B) ((C *D) / E) –

> ((A +B) - ((C *D) / E)) ((1+2) – ((3 * 2) /4))

A **B C - D E / * +**

> A (B – C) D E / * +

> A (B – C) (D / E) * +

> A ((B – C) * (D /E)) +

> (A + ((B – C) * (D /E)))

ABC+-DEF+^*

AB+C* DE- - FG+H^^

A B + C D * E / -

> +AB C D * E / -

> +AB *CD E / -

> +AB /*CDE -

> -+AB/*CDE

Prefix to infix and Postfix :

Scan from right to left until an operator is encountered.

When an operator is encountered, apply(infixing or suffixing) on to the two previously available operands. **The most recent one must be operand1.**

Resume scanning and repeat the process until the entire expression is scanned.

- +AB/*CDE

-+AB/ (C* D) E

((A +B) + ((C * D) / E))

AB+ CD*E/ +

ABCDE - + $ * EF * -

> ABC (D-E) + $ * EF * -
> A B (C + (D-E)) $* EF * -
> A (B $ (C + (D-E))) * EF* -
> (A* (B $ (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)

> (A * (B $ (C + DE-))) – (E * F)
> (A * (B $ CDE-+)) – (E * F)
> (A * BCDE-+$) – (E * F)
> A BCDE-+$* - (E*F)
> A BCDE-+$* - EF*
> A BCDE-+$* EF* -

^+ + A - * BCD / + EF + GHI

> ^++ A - * BCD / + EF (G + H) I
> ^++ A - * BCD/ (E + F) (G+H) I
> ^++ A - * BCD ((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
> ((A + ((B*C) – D)) + ((E+F)/ (G+H))) ^ I

Convert the following expressions to other two forms

i)ABCDE - + $ * EF * - ii) ^+ + A - * BCD / + EF * GHI

iii) ++A - * + BCD / + EF * GHI iv) AB + C – BA + C $

v) ((A + B) * C – (D – E)) $ (F + G)

VI) A$B*C – D + E/F/(G+H)