Algorithms for Stack operations:

1. PUSH

Procedure:

1. The push operation in stack refers to addition

of an element to the stack.

2. As an element gets added we need to first check

if at all the stack is full or not

3. If the stack is not full then we can add our element

onto the stack

4. Once the element is added it reflects a variation in the

size of the stack

To check if stack is full

void isfull(int Stack[],int top)

{

if(top==(n-1))/\* As we are implementing stacks using arrays\*/

printf("\n Stack is full or overflow condition");

else

return 0;

}

/\* C Procedure for PUSH operation\*/

void push(int Stack[],int item)

{

if(top==(n-1))

printf("\n overflow");

else

{

top=top+1;

stack[top]=item;

}

POP operation:

1. To delete an element from the stack check if the stack

contains elements or its empty.

isempty(int top)

{

if(top==-1)

return 1;

else

return 0;

}

2. Once an element is deleteed from stack the size of the

stack varies.

3. Display the popped element.

/\* C Procedure for POP operation\*/

void pop(int stack[])

{

if(top==-1)

printf("underflow");

else

{

item=stack[top];

top=top-1;

printf("the popped element is:%d",item);

}

}

PEEP operation

1. It refers to extractiong a required element from

the given position.

2. To do this we need to first check if the specified

position is a valid one or not.(top-pos+1>=0)

void peep(int stack[],int pos)

{

int item;

if(top-pos+1<0)

printf("invalid position");

else

{

item=stack[pos-1];

printf("the peeped element is:%d",item);

}

}

Traversal:

It refers to a process of visiting all the elements of

stack once.

void traversal(int stack[])

{

int i;

if(top==-1)

printf("stack is empty");

else

{

for(i=top;i>=0;i--)

{

Printf(%d",stack[i]);

}

}

}

Evaluation of postfix expression:

So far we have seen the conversion of infix to postfix

now if at all we are given a postfix expression now lets

look into the procedure of evaluating it using stacks.

Procedure:

1. Add a ) which acts like a sentinel to determine end of expr.

2. Scan the expression in Postfix notation(P) from L->R.

3. If an operand encounters then PUSH it onto stack.

4. Else if an operator encounters then pop 2 consecutive

items from the stack i.e.top(A),next to top(B) elements.

5. Then evaluate the operation based on operator occured on

A,B ex:A\*B and PUSH the result back onto the stack.

6. Repeat the steps 3,4 &5 until the sentinel ) occurs which

determines end of expr.

C Procedure to evaluate postfix expression.

void evaluate\_postfix(int stack[],char P[])

{

int i,A,B;

for(i=0;p[i]!='\0';i++)

{

if(isalpha(p[i])

{

top=top+1;

printf("enter the value of %c",p[i]);

scanf("%d",&stack[top]);

}

else

{

switch(p[i])

{

case '/':

A=stack[top];

top--;

B=stack[top];

stack[top]=B/A;

break;

case '\*':

A=stack[top];

top--;

B=stack[top];

stack[top]=B\*A;

break;

case '%':

A=stack[top];

top--;

B=stack[top];

stack[top]=B%A;

break;

case '+':

A=stack[top];

top--;

B=stack[top];

stack[top]=B+A;

break;

case '-':

A=stack[top];

top--;

B=stack[top];

stack[top]=B-A;

break;

case '^':

A=stack[top];

top--;

B=stack[top];

stack[top]=pow(B,A);

break;

}

}

}

printf("result=%d",stack[top]);

}

Evaluation of a prefix Expression

When we are given a prefix expression (P) we can evaluate

it and generate the result using the following procedure.

Procedure:

1. Reverse the given prefix expression (P).

2. Add a ) which acts like a sentinel to determine end of expr.

3. Scan the expression in Postfix notation(P) from L->R.

4. If an operand encounters then PUSH it onto stack.

5. Else if an operator encounters then pop 2 consecutive

items from the stack i.e.top(A),next to top(B) elements.

6. Then evaluate the operation based on operator occured on

A,B ex:A\*B and PUSH the result back onto the stack.

7. Repeat the steps 4,5 &6 until the sentinel ) occurs which

determines end of expr.

Examples:

p-> +,-,\*,2,2,/,16,8,5

C Procedure for evaluating prefix Expression:

void evaluate\_prefix(int stack[],char p[])

{

int i,A,B;

strrev(p);

for(i=0;p[i]!='\0';i++)

{

if(isalpha(p[i])

{

top=top+1;

printf("enter the value of %c",p[i]);

scanf("%d",&stack[top]);

}

else

{

switch(p[i])

{

case '/':

A=stack[top];

top--;

B=stack[top];

stack[top]=A/B;

break;

case '\*':

A=stack[top];

top--;

B=stack[top];

stack[top]=A\*B;

break;

case '%':

A=stack[top];

top--;

B=stack[top];

stack[top]=A%B;

break;

case '+':

A=stack[top];

top--;

B=stack[top];

stack[top]=A+B;

break;

case '-':

A=stack[top];

top--;

B=stack[top];

stack[top]=A-B;

break;

case '^':

A=stack[top];

top--;

B=stack[top];

stack[top]=pow(A,B);

break;

}

}

}

printf("result=%d",stack[top]);

}

Conversion of postfix expression to infix expression:

Follow the same procedure that we have done for evaluation

of postfix expression.

Examples:

AB+C\*DEF-/-

AB-DE+F\*/

Conversion of Prefix expression to infix expression

Follow the same procedure of evaluation of prefix to

obtain the infix expression.

Examples:

+-\*AB/CDE