**STACK**

**EXPRESSION:**

1. **INFIX**
2. **POSTFIX**
3. **PREFIX**

**EXPRESSION CONVERSION :**

1. **INFIX TO POSTFIX**
2. **INFIX TO PREFIX**
3. **POSTFIX TO INFIX**
4. **PREFIX TO INFIX**
5. **POSTFIX TO PREFIX**
6. **PREFIX TO POSTFIX**

**INFIX EXPRESSION:**

**Infix** notation is the most common way of writing expressions. Prefix and Postfix notations are other two ways that are good for machines because they can be parsed and evaluated easily.

**Example:(A+B)**

**POSTFIX EXPRESSION:**

A **postfix expression** is a collection of operators and operands in which the operator is placed after the operands. That means, in a **postfix expression** the operator follows the operands

**Example:AB+**

**PREFIX EXPRESSION:**

An **expression** is called the **prefix expression** if the operator appears in the **expression** before the operands. Simply of the form (operator operand1 operand2). ...

**Example:+AB**

**EXPRESSION CONVERSIONS**

**INFIX TO POSTFIX:**

**WHY?**

To solve infix expression one need to check the whole expression while performing every single operation, it takes more time to evaluate. therefore we go for postfix ,where operator are separated and operator were placed after the operands.

**MANNUAL CONVERSION:**

1. Parenthesis the expression according to operator precedence.
2. Swap the all operator to their corresponding close bracket.
3. Remove the parenthesis

[A+(B\*C)]

[A(BC\*)+]

ABC\*+

**MANNUAL CONVERSION:(USING STACK)**

1. Get the character one by one.
2. If the character is operand copy to the output.
3. If the character is operand push into the stack.
   1. If the operator in top of stack is high prior than the coming operator then pop all the operator and place in output array then push the newly coming operator to the stack.
4. Repeat the step till end of the expression.

**PROGRAM:**

#include<stdio.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

}

main()

{

char exp[20];

char \*e, x;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c",pop());

}

}

**INFIX TO PREFIX:**

**MANNUAL CONVERSION:**

1. Parenthesis the expression according to operator precedence.
2. Swap the all operator to their corresponding close bracket.
3. Remove the parenthesis
   * 1. [A+(B\*C)]
     2. [+A(\*BC)]
     3. +A\*BC

**MANNUAL CONVERSION:(USING STACK)**

Reverse the given prefix expression:

1. Get the character one by one.
2. If the character is operand copy to the output.
3. If the character is operand push into the stack.
   1. If the operator in top of stack is high prior than the coming operator then pop all the operator and place in output array then push the newly coming operator to the stack.
4. Repeat the step till end of the expression.
5. Reverse the output expression.

**PROGRAM:**

# include <stdio.h>

# include <string.h>

# define MAX 20

void infixtoprefix(char infix[20],char prefix[20]);

void reverse(char array[30]);

char pop();

void push(char symbol);

int isOperator(char symbol);

int prcd(symbol);

int top=-1;

char stack[MAX];

main() {

char infix[20],prefix[20],temp;

printf("Enter infix operation: ");

gets(infix);

infixtoprefix(infix,prefix);

reverse(prefix);

puts((prefix));

}

void infixtoprefix(char infix[20],char prefix[20]) {

int i,j=0;

char symbol;

stack[++top]='#';

reverse(infix);

for (i=0;i<strlen(infix);i++) {

symbol=infix[i];

if (isOperator(symbol)==0) {

prefix[j]=symbol;

j++;

} else {

if (symbol==')') {

push(symbol);

} else if(symbol == '(') {

while (stack[top]!=')') {

prefix[j]=pop();

j++;

}

pop();

} else {

if (prcd(stack[top])<=prcd(symbol)) {

push(symbol);

} else {

while(prcd(stack[top])>=prcd(symbol)) {

prefix[j]=pop();

j++;

}

push(symbol);

}

//end for else

}

}

//end for else

}

//end for for

while (stack[top]!='#') {

prefix[j]=pop();

j++;

}

prefix[j]='\0';

}

void reverse(char array[30]) // for reverse of the given expression

{

int i,j;

char temp[100];

for (i=strlen(array)-1,j=0;i+1!=0;--i,++j) {

temp[j]=array[i];

}

temp[j]='\0';

strcpy(array,temp);

return array;

}

char pop() {

char a;

a=stack[top];

top--;

return a;

}

void push(char symbol) {

top++;

stack[top]=symbol;

}

int prcd(symbol) // returns the value that helps in the precedence

{

switch(symbol) {

case '+':

case '-':

return 2;

break;

case '\*':

case '/':

return 4;

break;

case '$':

case '^':

return 6;

break;

case '#':

case '(':

case ')':

return 1;

break;

}

}

int isOperator(char symbol) {

switch(symbol) {

case '+':

case '-':

case '\*':

case '/':

case '^':

case '$':

case '&':

case '(':

case ')':

return 1;

break;

default:

return 0;

// returns 0 if the symbol is other than given above

}

}

**POSTFIX TO INFIX:**.

**MANNUAL CONVERSION(USING STACK):**

1. Read the postfix expression.
2. If the char is the operand then push into the stack.
3. When you read the operator pop two char from the stack fix operator in between then push again to stack.
4. Repeat the step till that array is empty.
5. Pop the result from the stack.

**EXAMPLE:**

**AB+>>A+B**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

int top = 10;

struct node

{

char ch;

struct node \*next;

struct node \*prev;

} \*stack[11];

typedef struct node node;

void push(node \*str)

{

if (top <= 0)

printf("Stack is Full ");

else

{

stack[top] = str;

top--;

}

}

node \*pop()

{

node \*exp;

if (top >= 10)

printf("Stack is Empty ");

else

exp = stack[++top];

return exp;

}

void convert(char exp[])

{

node \*op1, \*op2;

node \*temp;

int i;

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

if (exp[i] >= 'a'&& exp[i] <= 'z'|| exp[i] >= 'A' && exp[i] <= 'Z')

{

temp = (node\*)malloc(sizeof(node));

temp->ch = exp[i];

temp->next = NULL;

temp->prev = NULL;

push(temp);

}

else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '\*' || exp[i] == '/' ||

exp[i] == '^')

{

op1 = pop();

op2 = pop();

temp = (node\*)malloc(sizeof(node));

temp->ch = exp[i];

temp->next = op1;

temp->prev = op2;

push(temp);

}

}

void display(node \*temp)

{

if (temp != NULL)

{

display(temp->prev);

printf("%c", temp->ch);

display(temp->next);

}

}

void main()

{

char exp[50];

printf("Enter the postfix expression :");

scanf("%s", exp);

convert(exp);

printf("\nThe Equivalant Infix expression is:");

display(pop());

printf("\n\n");

getch();

}

**PREFIX TO INFIX:**.

**MANNUAL CONVERSION(USING STACK):**

1.Reverse the given the expression.

2.perform the step same as postfix to infix convertor.

**EXAMPLE:**

**+AB>>A+B**

**PROGRAM:**

#include <string.h>

#include <ctype.h>

#include <conio.h>

char opnds[50][80],oprs[50];

int topr=-1,topd=-1;

pushd(char \*opnd)

{

strcpy(opnds[++topd],opnd);

}

char \*popd()

{

return(opnds[topd--]);

}

pushr(char opr)

{

oprs[++topr]=opr;

}

char popr()

{

return(oprs[topr--]);

}

int empty(int t)

{

if( t == 0) return(1);

return(0);

}

void main()

{

char prfx[50],ch,str[50],opnd1[50],opnd2[50],opr[2];

int i=0,k=0,opndcnt=0;

printf("Give an Expression = ");

gets(prfx);

printf(" Given Prefix Expression : %s\n",prfx);

while( (ch=prfx[i++]) != '\0')

{

if(isalnum(ch))

{

str[0]=ch; str[1]='\0';

pushd(str); opndcnt++;

if(opndcnt >= 2)

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,"(");

strcat(str,opnd1);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

strcat(str,opnd2);

strcat(str,")");

pushd(str);

opndcnt-=1;

}

}

else

{

pushr(ch);

if(opndcnt==1)opndcnt=0; /\* operator followed by single operand\*/

}

}

if(!empty(topd))

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,"(");

strcat(str,opnd1);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

strcat(str,opnd2);

strcat(str,")");

pushd(str);

}

printf(" Infix Expression: ");

puts(opnds[topd]);

getch();

}

**POSTFIX TO PREFIX:**.

**MANNUAL CONVERSION(USING STACK):**

1. *Read the Postfix expression from left to right*
2. *If the symbol is an operand, then push it onto the Stack.*
3. *If the symbol is an operator, then pop two operands from the Stack  
   Create a string by concatenating the two operands and the operator before them.****string = operator + operand2 + operand1*** *And push the resultant string back to Stack*
4. *4.Repeat the above steps until end of Prefix expression*

***EXAMPLE:***

***AB+>>+AB***

**PROGRAM:**

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

char str[MAX], stack[MAX];

int top = -1;

char pop()

{

      return stack[top--];

}

void push(char ch)

{

      stack[++top] = ch;

}

void postfix\_to\_prefix(char expression[])

{

      int count, length;

      length = strlen(expression);

      printf("\nPrefix Expression:\t");

      for(count = length - 1; count >= 0; count--)

      {

            printf("%c", expression[count]);

      }

}

int main()

{

      char postfix\_expression[35];

      printf("\nEnter Postfix Expression:\t");

      scanf("%s", postfix\_expression);

      postfix\_to\_prefix(postfix\_expression);

      printf("\n");

      return 0;

}

**PREFIX TO POSTFIX:**.

**MANNUAL CONVERSION(USING STACK):**

1. Read the Prefix expression in reverse order (from right to left)
   * 1. If the symbol is an operand, then push it onto the Stack
2. If the symbol is an operator, then pop two operands from the Stack  
   Create a string by concatenating the two operands and the operator after them.  
   **string = operand1 + operand2 + operator**  
   And push the resultant string back to Stack
3. Repeat the above steps until end of Prefix expression

**EXAMPLE:**

**+AB>>AB+**

**PROGRAM:**

#include <string.h>

#include <ctype.h>

char opnds[50][80],oprs[50];

int topr=-1,topd=-1;

pushd(char \*opnd)

{

strcpy(opnds[++topd],opnd);

}

char \*popd()

{

return(opnds[topd--]);

}

pushr(char opr)

{

oprs[++topr]=opr;

}

char popr()

{

return(oprs[topr--]);

}

int empty(int t)

{

if( t == 0) return(1);

return(0);

}

main()

{

char prfx[50],ch,str[50],opnd1[50],opnd2[50],opr[2];

int i=0,k=0,opndcnt=0;

gets(prfx);

printf(" Given Prefix Expression : %s\n",prfx);

while( (ch=prfx[i++]) != '\0')

{

if(isalnum(ch))

{

str[0]=ch; str[1]='\0';

pushd(str); opndcnt++;

if(opndcnt >= 2)

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,opnd1);

strcat(str,opnd2);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

pushd(str);

opndcnt-=1;

}

}

else

{

pushr(ch);

if(opndcnt==1)opndcnt=0; /\* operator followed by single operand\*/

}

}

if(!empty(topd))

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,opnd1);

strcat(str,opnd2);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

pushd(str);

}

printf(" Postfix Expression: ");

puts(opnds[topd]);

}

**REF:GEEKSFORGEEKS**