Outline

Applications of Stacks (Continued)
  • Infix to Postfix Conversion

INFIX TO POSTFIX CONVERSION

Given Infix Expressions stack can be used to convert the Infix Expressions to Postfix expression by using the following Approach:

Scan the Infix expression left to right
  • If the character x is an operand
    o Output the character into the Postfix Expression
  • If the character x is a left or right parenthesis
    o If the character is “(“
      Push it into the stack
    o if the character is “)”
      Repeatedly pop and output all the operators/characters until “(“ is popped from the stack.
  • If the character x is a is a regular operator
    o Step 1: Check the character y currently at the top of the stack.
    o Step 2: If Stack is empty or y='(' or y is an operator of lower precedence than x, then push x into stack.
    o Step 3: If y is an operator of higher or equal precedence than x, then pop and output y and push x into the stack.

When all characters in infix expression are processed repeatedly pop the character(s) from the stack and output them until the stack is empty.

The following program converts a given Infix expression with variable a to z and operators ‘+’, ‘-‘, ‘*’ and ‘/’ into a Postfix Expression.

```c
#include<stdio.h>
#include<stdlib.h>
#define STACKSIZE 20
typedef struct{
    int top;
    char items[STACKSIZE];
}STACK;
void push(STACK *, char);
char pop(STACK *);
void main()
{
    int i;
    char x, y, E[20] ; /* Assume that Infix Expression E contains single-digit integers/parenthesis/operators*/
    STACK s;
    s.top = -1; /* Initialize the stack */
    printf("Enter the Infix Expression:");
    scanf("%s",E);
    for(i=0;E[i] != '\0';i++){
        x= E[i];
        if(x<='z' && x>='a') /* Consider all lowercase letter operands from a to z */
            printf("%c",x);
        else if(x == '(')
            push(&s ,x);
        else if(x == ')')
            y=pop(&s) ;
        else if(x == '+')
            push(&s,x);
        else if(x == '-')
            push(&s,x);
        else if(x == '*')
            push(&s,x);
        else if(x == '/')
            push(&s,x);
    }
}
```
while(y != '(') {
    printf("%c", y);
    y = pop(&s);
}
else {  
    if(s.top ==-1 || s.items[s.top] == '(')  
        push(&s, x);
    else {
        y = s.items[s.top];  /* y is the top operator in the stack*/
        if( y=='*'|| y=='/') {  /* precedence of y is higher/equal to x*/
            printf("%c", pop(&s));
            push(&s, x);
            }  
        else if ( y=='+' ||  y=='-')  
            if( x=='+' ||  x=='-') {  /* precedence of y is equal to x*/
                printf("%c", pop(&s));
                push(&s, x);
            }  
            else /* precedence of y is less than x*/
                push(&s, x);
    }
}
while(s.top != -1)  
    printf("%c", pop(&s));
}

void push(STACK *sptr, char ps)  /*pushes ps into stack*/
{
    if(sptr->top == STACKSIZE-1)  
        printf("Stack is full\n");
        exit(1); /*exit from the function*/
    else
        sptr->items[++sptr->top] = ps;
}

char pop(STACK *sptr)  
{
    if(sptr->top == -1)  
        printf("Stack is empty\n");
        exit(1); /*exit from the function*/
    else
        return sptr->items[sptr->top--];