postfix operations

Contents

- Pseudocode and C code to perform the following
  1. Infix to Postfix conversion
  2. Postfix Evaluation
  3. Postfix to infix conversion

Infix to postfix Pseudocode (Stack used: operator/char stack)

Step1: For each ***ith character*** in infix expression (let it be ***token***)

- if token is operand
  - Append token to output expression
- Else if token is lparen (opening parenthesis)
  - Push lparen into stack
- Else if token is rparen (closing parenthesis)
  - Pop all operators from stack until lparen is found
  - Pop and ignore lparen
- **Else if token is operator**
  - **pop all operators having higher/equal precedence in stack**
  - **Place token inside stack (push token)**

Step2: pop all operators from stack and append to output expression

Step3: Display output expression

Infix to postfix (1)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
char stk[20];
int top = -1;

int isEmpty()  {      return top == -1;  }
int isFull()  {      return (top == MAX – 1);  }
char peek()  {      return stk[top];  }

char pop()
{      if(isEmpty())  return -1;
    char ch = stk[top];
    top--;
    return(ch);
}
```

Infix to postfix (2)

```
void push(char oper)
{
   if(isFull())  printf("Stack Full!!!!");
   else{         top++;    stk[top] = oper;      }
}

int checkIfOperand(char ch)
{      return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');   }

int precedence(char ch)
{
   switch (ch)
   {       case '+':
           case '-':          return 1;
   case '*':
   case '/':          return 2;
   case '^':
case ‘%’:       return 3;
    }
   return -1;
}
```

Infix to postfix (3)

```
int covertInfixToPostfix(char* expression)
{   int i, j;  char output[20];
   for (i = 0, j = -1; expression[i]; ++i)
   {   if (checkIfOperand(expression[i]))                    output[++j] = expression[i];
      else if (expression[i] == '(')                  push(expression[i]);
      else if (expression[i] == ')')
      {   while (!isEmpty() && peek() != '(') output[++j] = pop();
         if (!isEmpty() && peek() != '(')    return -1;
         else                                pop();
      }
      else
      {   while (!isEmpty() &&
precedence(expression[i]) <= precedence(peek()))
            output[++j] = pop();
         push(expression[i]);
      }
   }
   while (!isEmpty())   output[++j] = pop();
   output[++j] = '\0';
   printf( "%s", output);
```

Infix to postfix (4)

```
int main()
{
char expression[] = "((x+(y*z))-w)";
    covertInfixToPostfix(expression);
    return 0;
}
```

Postfix evaluation Psudocode (Stack used: integer stack)

- Create an empty stack.
- Scan the expression from left to right.
- If an operand is encountered, it push it’s numeric value onto the stack.
- If an operator is encountered,
  - pop the top two operands from the stack, perform the operation, and push the result back onto the stack.
- After that, it Continue scanning the expression until all tokens have been processed.
- When the expression has been fully scanned, the result will be the top element of the stack.

postfix evaluation (1)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
int stack[MAX_SIZE];
int top = -1;

void push(int item) {
    if (top >= MAX_SIZE - 1) {  printf("Stack Overflow\n");  }
    top++;
    stack[top] = item;
}

int pop() {
    if (top < 0) {  printf("Stack Underflow\n");          return -1;      }
    int item = stack[top];
    top--;
    return item;
}
```

postfix evaluation (2)

```
int is_operator(char symbol) {
    if (symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/')
        return 1;
return 0;
}
```

```
int checkIfOperand(char ch)
{       return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');    }
```

postfix evaluation (3)

```
int evaluate(char* expression) {
   int i = 0;      char symbol = expression[i];
   int operand1, operand2, result,val;
   while (symbol != '\0') {
      if (symbol >= '0' && symbol <= '9') {  int num = symbol - '0';            push(num);         }
      else if(checkIfOperand(symbol))
               { printf("Enter value of %c",symbol);  scanf("%d",&val); push(val); }
      else if (is_operator(symbol)) {
         operand2 = pop();
         operand1 = pop();
         switch(symbol) {    case '+': result = operand1 + operand2; break;
            case '-': result = operand1 - operand2; break;
            case '*': result = operand1 * operand2; break;         case '/': result = operand1 / operand2; break;
         }
         push(result);
      }
       i++;
      symbol = expression[i];
   }
   result = pop();
   return result;
}
```

postfix evaluation (4)

```
int main() {
    char expression[] = “abc+*d-";
    int result = evaluate(expression);
printf("Result= %d\n", result);
return 0;
}
```

Postfix to Fully parenthesized (FP) infix

```
Input : abc++
Output : (a + (b + c))

Input  : ab*c+
Output : ((a*b)+c)
```

Input String	Postfix Expression	Stack (Infix)
ab*cd+/	b*cd+/	a
ab*cd+/	*cd+/	ab
ab*cd+/	cd+/	(a*b)
ab*cd+/	d+/	(a*b)c
ab*cd+/	+/	(a*b)cd
ab*cd+/	/	(a*b)(c+d)
ab*cd+/		((a*b)/(c+d))

Postfix to Fully parenthesized (FP) infix Pseudocode (Stack used: String stack)

- Create an empty stack.
- Scan the expression from left to right.
- If an operand is encountered, it push it onto the stack.
- If an operator is encountered,
  - pop the top two operands from the stack
  - op2=pop and op1=pop and push("(op1 operator op2)")
- After that, it continue scanning the expression until all tokens have been processed.
- When the expression has been fully scanned, the result will be the top element of the stack.

postfix to FP infix(1)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 20
char stack[MAX][MAX];
int top=-1;
void push(char *item){
        if(isFull()) printf("Overflow detected!\n");
        else{
                top++;
                strcpy(stack[top],item);
        }
}

int isFull(){       if(top==MAX-1) return 1;
        else   return 0;
}

int isEmpty(){     if(top==-1) return 1;
        else return 0;
}
```

postfix to FP infix(2)

```
int isOperator(char sym){
	if(sym=='+'||sym=='-'||sym=='*'||sym=='/'||sym=='^') return 1;
	else return 0;
}

char *pop(){
	if(isEmpty()) exit(0);
	return stack[top--];
}

int isOperand(char sym){
	if(sym>='A'&&sym<='Z'||sym>='a'&&sym<='z') return 1;
	else return 0;
}
```

postfix to FP infix(3)

```
int main(){
        char postfix[MAX],temp[2],op[2]={'(','\0'},cl[2]={')','\0'};
        int i=0,j=0;
        printf("Enter an postfix expression: ");
        gets(postfix);
        while(postfix[i]!='\0'){
                char exp[MAX]={'\0'},op1[MAX]={'\0'},op2[MAX]={'\0'};
                temp[0]=postfix[i];
                temp[1]='\0';
                if(isOperand(temp[0]))          push(temp);
                else if(isOperator(temp[0])){
                        strcpy(op2,pop());                      strcpy(op1,pop());
                        strcat(exp,op);                         strcat(exp,op1);
                        strcat(exp,temp);                       strcat(exp,op2);
                        strcat(exp,cl);                         push(exp);
                }
                else{       printf("Invalid Arithmetic expression!\n"); exit(0); }
                i++;
        }
        printf("The infix expression is: ");
        puts(stack[0]);
}
```