prefix operations

Contents

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

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

```
For each ith character in infix expression(let it be token)[traverse backwards]
    ➢ if token is operand
        ❖ Append token to output expression
    ➢ Else if token is rparen (closing parenthesis)
        ❖ Push rparen into stack
    ➢ Else if token is lparen (opening parenthesis)
        ❖ Pop all operators from stack until rparen is found
        ❖ Pop and ignore rparen
    ➢ Else if token is operator
        ❖ pop all operators having higher precedence in stack
        ❖ Place token inside stack (push token)
Step2: pop all operators from stack and append to output expression
Step3: reverse output expression and display output expression
```

Infix to prefix (1)

```
#include <stdio.h>
#include <stdlib.h>  #include<string.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 prefix (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;
}
```

```
char *strrev(char *str)
{
if (!str || ! *str)
return str;
int i = strlen(str) - 1,
j = 0;
char ch;
while (i > j)
{
        ch = str[i];
        str[i] = str[j];
        str[j] = ch; i--; j++;
}
return str;
}
```

Infix to prefix (3)

```
int covertInfixToPrefix(char* expression)
{   int i, j;  char output[20]; for(i=0;expression[i]!=‘\0’;i++); i=i-1;
    for (j = -1; i>=0;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]);
       }   //else
   }
   while (!isEmpty())   output[++j] = pop();
   output[++j] = '\0';   strrev(output);
   printf( "%s", output);    }
}
```

Infix to prefix (4)

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

Prefix evaluation Psudocode (Stack used: integer stack)

- Create an empty stack.
- Scan the expression from right to left
- 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.

prefix 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;
}
```

prefix 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');   }
```

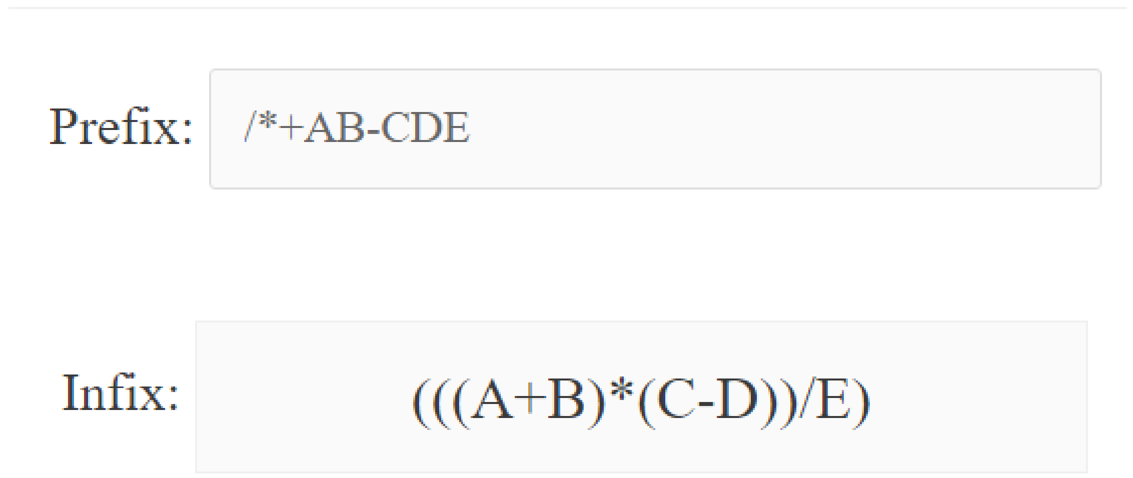
prefix evaluation (3)

```
int evaluate(char* expression) {
   int i = strlen(expression)-1;     char symbol = expression[i];
   int operand1, operand2, result,val;
   while (i>=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)) {
         operand1 = pop();
         operand2 = pop();
         switch(symbol) {    case '+': result = operand1 + operand2; break;
            case '-': result = operand1 - operand2; break;
            case '*': result = operand1 * operand2; break;         case '/': result = operand1 / operand2; break;
         }  //switch
         push(result);
      }     i--;
      symbol = expression[i];
   }
   result = pop();
   return result;
}
```

prefix evaluation (4)

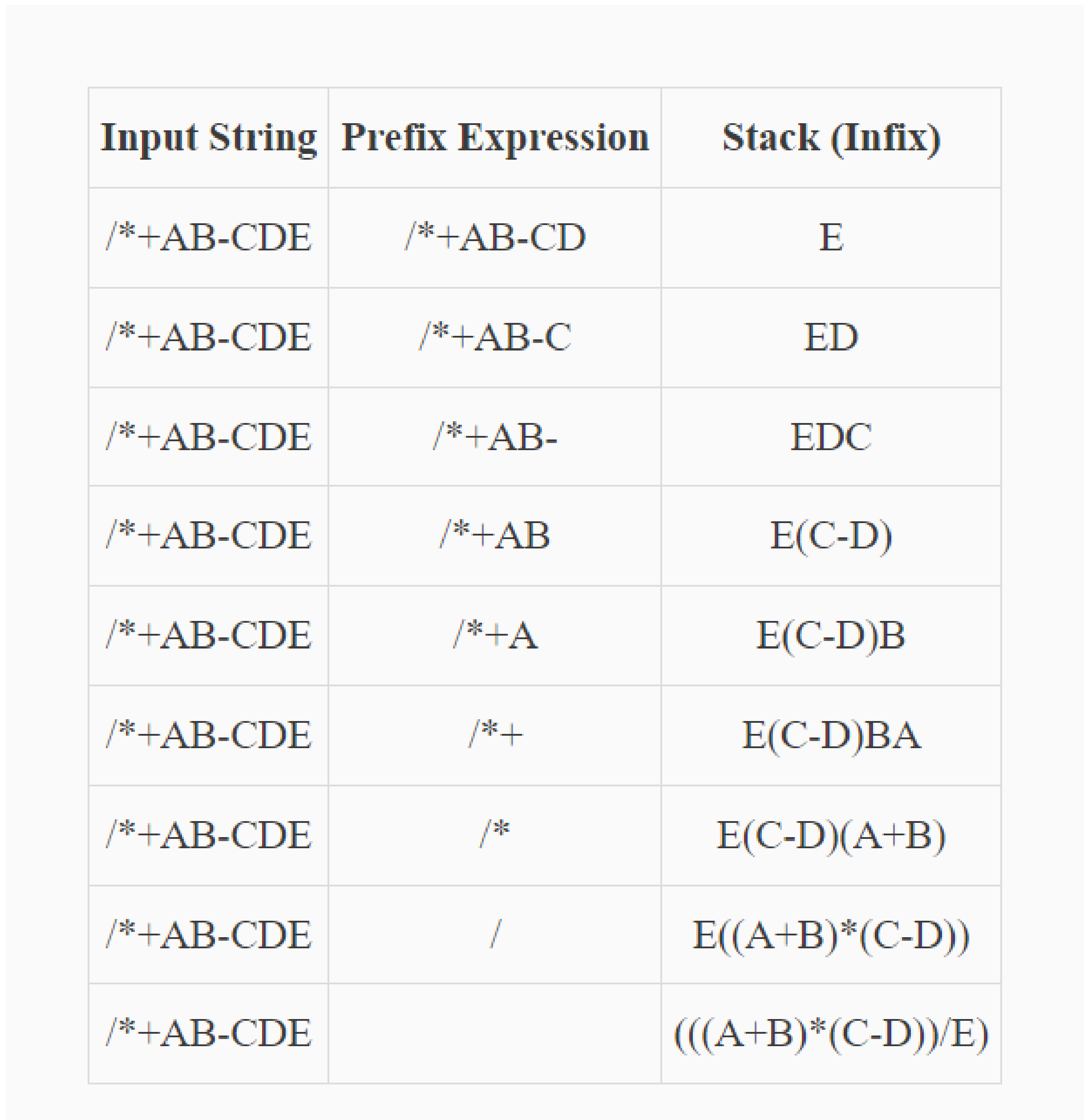
```
int main() {
    char expression[] = “abc+*d-";
    int result = evaluate(expression);
printf("Result= %d\n", result);
return 0;
}
```

Prefix to Fully parenthesized (FP) infix



Prefix: /*+AB-CDE

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



Input String	Prefix Expression	Stack (Infix)
/*+AB-CDE	/*+AB-CD	E
/*+AB-CDE	/*+AB-C	ED
/*+AB-CDE	/*+AB-	EDC
/*+AB-CDE	/*+AB	E(C-D)
/*+AB-CDE	/*+A	E(C-D)B
/*+AB-CDE	/*+	E(C-D)BA
/*+AB-CDE	/*	E(C-D)(A+B)
/*+AB-CDE	/	E((A+B)*(C-D))
/*+AB-CDE		(((A+B)*(C-D))/E)

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

- Create an empty stack.
- Scan the expression from right to left
- If an operand is encountered, it push it onto the stack.
- If an operator is encountered,
    - pop the top two operands from the stack
    - op1=pop and op2=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.

prefix 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;
}
```

prefix 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;
}
```

prefix to FP infix(3)

```
int main(){
        char prefix[MAX],temp[2],op[2]={'(','\0'},cl[2]={')','\0'};
        int i,j=0;
        printf("Enter an prefix expression: ");         gets(prefix);
         i=strlen(prefix)-1
        while(i>=0){
                char exp[MAX]={'\0'},op1[MAX]={'\0'},op2[MAX]={'\0'};
                temp[0]=prefix[i];  temp[1]='\0';
                if(isOperand(temp[0]))       push(temp);
                else if(isOperator(temp[0])){
                        strcpy(op1,pop());                              strcpy(op2,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]);
}
```