Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology

(An Autonomous Institute affiliated to Savitribai Phule Pune University)

Data Structures Lab

Assignment No: 3

Name	: Gourav Balaji Suram
Roll No	: 39
PRN No	: 12220032

Problem Statement:

Implement a strack for folllowing expression conversion.
a. Infix to Prefix and Postfix.
b. Prefix to Infix.
c. Postfix to Infix.

A1. Infix to Prefix

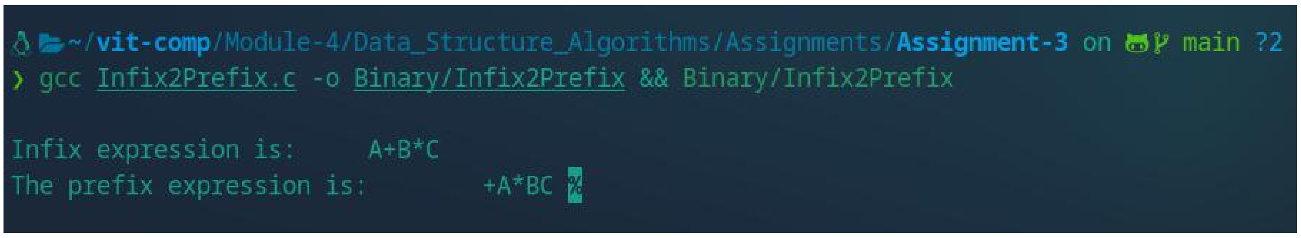
Program :

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#define MAX 6

char stack[MAX];
int top=-1;

int isFull();
int isEmpty();
void push(char);
int isOperator(char);
char pop();
int pre(char);

char infix[MAX],prefix[MAX],postfix[MAX],item,temp;
int i=0,j=0;

char *strrev(char *str)
{
    char *p1, *p2;

    if (! str || ! *str)
        return str;
    for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
    {
        *p1 ^= *p2;
        *p2 ^= *p1;
        *p1 ^= *p2;
    }
    return str;
}


void push(char item){
        if(isFull()){
                printf("Stack overflow!\n");
        }else{
                top++;
                stack[top]=item;
        }
}

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

```
	}
}

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

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

char pop(){
	if(isEmpty()){
		return '\0';
	}
	char ch;
	ch=stack[top];
	top--;
	return ch;
}

int pre(char symbol){
	if(symbol=='^'){
		return 3;
	}else if(symbol=='*'||symbol=='/'){
		return 2;
	}else if(symbol=='+'||symbol=='-'){
		return 1;
	}else{
		return 0;
	}
}

void InToPre(char infix[MAX]){

    strrev(infix);

    while(infix[i]!='\0'){
		item=infix[i];
		if(item==')'){
			push(item);
		}else if(item>='A'&&item<='Z'||item>='a'&&item<='z'){
			prefix[j]=item;
			j++;
		}else if(isOperator(item)){
			temp=pop();
			while(isOperator(temp)==1&&pre(temp)>=pre(item)){
				prefix[j]=temp;
				j++;
```

```
                temp=pop();
            }
            push(temp);
            push(item);
        }else if(item=='('){
            temp=pop();
            while(temp!=')'){
                prefix[j]=temp;
                j++;
                temp=pop();
            }
        }else{
            printf("Invalid Arithmetix expression!\n");
            exit(0);
        }
        i++;
    }
    while(!isEmpty()){
        prefix[j]=pop();
        j++;
    }
    prefix[j]='\0';
    printf("\nThe prefix expression is:\t %s ",strrev(prefix));
}


int main(){

  char infix[] = "A+B*C";
    printf("\nInfix expression is:\t %s ", infix);
  InToPre(infix);
  return 0;
  }
```

Output :



```
~/vit-comp/Module-4/Data_Structure_Algorithms/Assignments/Assignment-3 on  main ?2
❯ gcc Infix2Prefix.c -o Binary/Infix2Prefix && Binary/Infix2Prefix

Infix expression is:     A+B*C
The prefix expression is:        +A*BC %
```

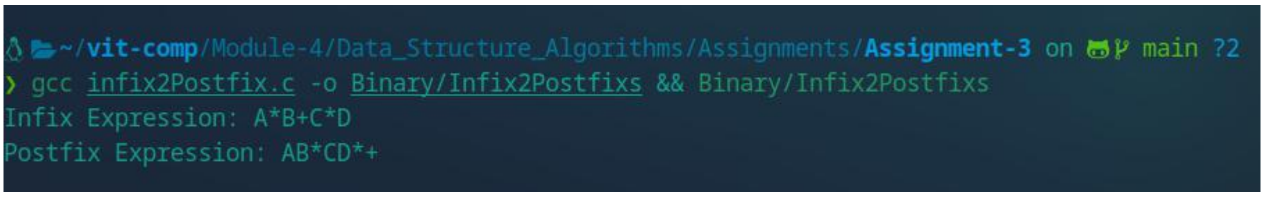
A2. Infix to Postfix

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

typedef struct stack {
  char items[25];
  int top;
} stack;

void push(stack *s, char c) {
  s->items[++s->top] = c;
}

char pop(stack *s) {
  return s->items[s->top--];
}

char peek(stack *s) {
  return s->items[s->top];
}

int is_empty(stack *s) {
  return s->top == -1;
}

int is_operand(char c) {
  return isalpha(c);
```

```
}

int precedence(char c) {
  if (c == '+' || c == '-') {
    return 1;
  } else if (c == '*' || c == '/') {
    return 2;
  } else if (c == '^') {
    return 3;
  }
  return 0;
}
void infix_to_postfix(char infix[], char postfix[]) {
  stack s;
  s.top = -1;

  int i, j;
  int len = strlen(infix);
  for (i = 0, j = 0; i < len; i++) {
    if (is_operand(infix[i])) {
      postfix[j++] = infix[i];
    } else if (infix[i] == '(') {
      push(&s, infix[i]);
    } else if (infix[i] == ')') {
      while (!is_empty(&s) && peek(&s) != '(') {
        postfix[j++] = pop(&s);
      }
      pop(&s);
    } else {
      while (!is_empty(&s) && precedence(peek(&s)) >= precedence(infix[i])) {
        postfix[j++] = pop(&s);
      }
      push(&s, infix[i]);
    }
  }

  while (!is_empty(&s)) {
    postfix[j++] = pop(&s);
  }
  postfix[j] = '\0';
}

int main() {
  char infix[] = "A*B+C*D";
  char postfix[25];

  infix_to_postfix(infix, postfix);
  printf("Infix Expression: %s\n", infix);
  printf("Postfix Expression: %s\n", postfix);

  return 0;
}
```

Output :



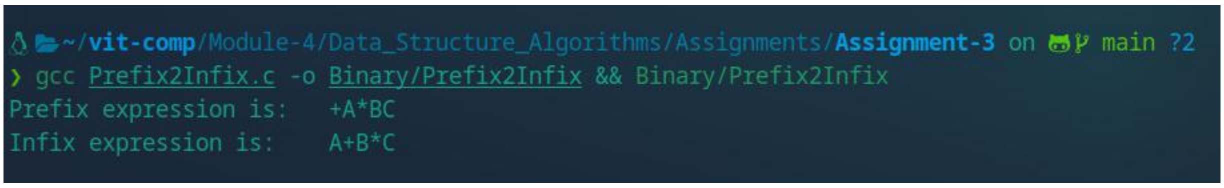
```
~/vit-comp/Module-4/Data_Structure_Algorithms/Assignments/Assignment-3 on main ?2
❯ gcc infix2Postfix.c -o Binary/Infix2Postfixs && Binary/Infix2Postfixs
Infix Expression: A*B+C*D
Postfix Expression: AB*CD*+
```

B. Prefix to Infix

Program :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX 20
char str[MAX], stack[MAX];
int top = -1;

void push(char c)
{
    stack[++top] = c;
}

char pop()
{
    return stack[top--];
}

void PrefixToInfix()
{
    int n, i;
    char a, b, op;
   char str[] = "+A*BC";
    printf("Prefix expression is:\t%s\n", str);

    n = strlen(str);
    for (i = 0; i < MAX; i++)
        stack[i] = '\0';
    printf("Infix expression is:\t");
    for (i = 0; i < n; i++)
    {
        if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/')
        {
            push(str[i]);
        }
        else
        {
            op = pop();
            a = str[i];
            printf("%c%c", a, op);
        }
    }
    printf("%c\n", str[top--]);
}

int main(){
  PrefixToInfix();
  return 0;
}
```

Output :



```
~/vit-comp/Module-4/Data_Structure_Algorithms/Assignments/Assignment-3 on  main ?2
❯ gcc Prefix2Infix.c -o Binary/Prefix2Infix && Binary/Prefix2Infix
Prefix expression is:   +A*BC
Infix expression is:    A+B*C
```

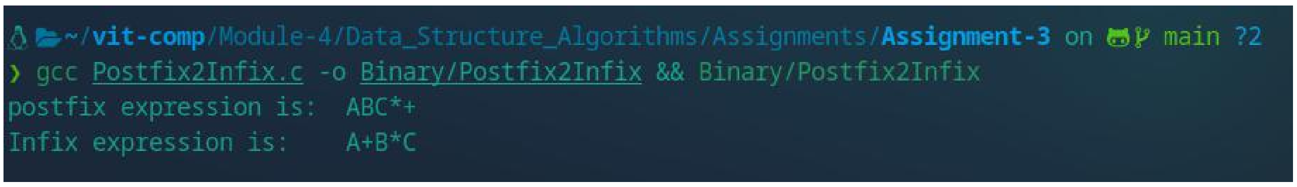
C. Postfix to Infix

Program :

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

void push(char c)
{
    stack[++top] = c;
}

char pop()
{
    return stack[top--];
}

char *strrev(char *str)
{
      char *p1, *p2;

      if (! str || ! *str)
            return str;
      for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
      {
            *p1 ^= *p2;
            *p2 ^= *p1;
            *p1 ^= *p2;
      }
      return str;
}

void PostfixToInfix()
{
    int n, i, j = 0;
    char a, b, op, x[20];
    char str[] = "ABC*+";
    printf("postfix expression is:\t%s\n", str);
    strrev(str);
    n = strlen(str);
    for (i = 0; i < MAX; i++)
        stack[i] = '\0';
    printf("Infix expression is:\t");
    for (i = 0; i < n; i++)
    {
        if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/')
        {
            push(str[i]);
        }
        else
        {
            x[j] = str[i];
            j++;
            x[j] = pop();
            j++;
        }
    }
    x[j] = str[top--];
    strrev(x);
    printf("%s\n", x);
}
```

```
int main(){
  PostfixToInfix();
  return 0;
}
```

Output :



```
~/vit-comp/Module-4/Data_Structure_Algorithms/Assignments/Assignment-3 on  main ?2
❯ gcc Postfix2Infix.c -o Binary/Postfix2Infix && Binary/Postfix2Infix
postfix expression is:  ABC*+
Infix expression is:    A+B*C
```