Q1. Write a program to convert a given infix expression into a prefix expression using stack.

Program Code:

```
#include <string.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct Stack{
    int top;
    int maxSize;
    int *array;
};
struct Stack *create(int max){
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct
Stack));
    stack->maxSize = max;
    stack->top = -1;
    stack->array = (int *)malloc(stack->maxSize * sizeof(int));
    return stack;
}
int isFull(struct Stack *stack){
    if (stack->top == stack->maxSize - 1){
        printf("Will not be able to push maxSize reached\n");
    }
        return stack->top == stack->maxSize - 1;
}
int isEmpty(struct Stack *stack){
    return stack->top == -1;
}
void push(struct Stack *stack, char item){
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
}
int pop(struct Stack *stack){
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top--];
}
int peek(struct Stack *stack){
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top];
}
```

```
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 '^':
        return 3;
    }
    return -1;
}
int getPostfix(char *expression){
    int i, j;
    struct Stack *stack = create(strlen(expression));
    if (!stack)
        return -1;
    for (i = 0, j = -1; expression[i]; ++i){
        if (checkIfOperand(expression[i]))
            expression[++j] = expression[i];
        else if (expression[i] == '(')
            push(stack, expression[i]);
                else if (expression[i] == ')'){
            while (!isEmpty(stack) && peek(stack) != '(')
                expression[++j] = pop(stack);
            if (!isEmpty(stack) && peek(stack) != '(')
                return -1; // invalid expression
            else
                pop(stack);
        }
        else{
            while (!isEmpty(stack) && precedence(expression[i]) <=
precedence(peek(stack)))
                expression[++j] = pop(stack);
            push(stack, expression[i]);
        }
    }
    while (!isEmpty(stack))
        expression[++j] = pop(stack);
    expression[++j] = '\0';
}
void reverse(char *exp){
```

```
    int size = strlen(exp);
    int j = size, i = 0;
    char temp[size];
    temp[j--] = '\0';
    while (exp[i] != '\0'){
        temp[j] = exp[i];
        j--;
        i++;
    }
    strcpy(exp, temp);
}
void brackets(char *exp){
    int i = 0;
    while (exp[i] != '\0'){
        if (exp[i] == '(')
            exp[i] = ')';
        else if (exp[i] == ')')
            exp[i] = '(';
        i++;
    }
}
void InfixtoPrefix(char *exp){
    int size = strlen(exp);
    reverse(exp);
    brackets(exp);
    getPostfix(exp);
    reverse(exp);
}
int main(){
    printf("The infix is: ");
    char expression[] = "((a/b)+c)-(d+(e*f))";
    printf("%s\n", expression);
    InfixtoPrefix(expression);
    printf("The prefix is: ");
    printf("%s\n", expression);
    return 0;
}
```

Q2. Write a program to convert a given infix expression into a postfix expression using stack.

Program Code:

```
#include <stdio.h>
#include <ctype.h>
char stack[100];
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;
    return 0;
}
int main(){
    char exp[100];
    char *e, x;
    printf("Enter the expression: ");
    scanf("%s", exp);
    printf("\n");
    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());
    }
    return 0;
}
```

Q3. Write a program to evaluate a given postfix expression using stack.

Program Code:

```
#include <stdio.h>
int stack[20];
int top = -1;
void push(int x){
    stack[++top] = x;
}
int pop(){
    return stack[top--];
}
int main(){
    char exp[20];
    char *e;
    int n1, n2, n3, num;
    printf("Enter expression: ");
    scanf("%s", exp);
    e = exp;
    while (*e != '\0'){
        if (isdigit(*e)){
            num = *e - 48;
            push(num);
        }
        else{
            n1 = pop();
            n2 = pop();
            switch (*e){
            case '+':{
                n3 = n1 + n2;
                break;
            }
            case '-':{
                n3 = n2 - n1;
                break;
            }
            case '*':{
```

```
                n3 = n1 * n2;
                break;
            }
            case '/':{
                n3 = n2 / n1;
                break;
            }
            }
            push(n3);
        }
        e++;
    }
    printf("\nResult: %d\n\n", pop());
    return 0;
}
```

Q4. Write a program to evaluate a given prefix expression using stack.

Program Code:

```
#include <stdio.h>
#include <string.h>
int s[50];
int top = 0;
void push(int ch);
int pop();
int main(){
    int a, b, c, i;
    char prefix[50];
    printf("Enter the prefix string in figures: ");
    gets(prefix);
    for (i = strlen(prefix) - 1; i >= 0; i--){
        if (prefix[i] == '+'){
            c = pop() + pop();
            push(c);
        }
        else if (prefix[i] == '-'){
            a = pop();
            b = pop();
            c = b - a;
            push(c);
        }
        else if (prefix[i] == '*'){
            a = pop();
            b = pop();
            c = b * a;
```

```
            push(c);
        }
        else if (prefix[i] == '/'){
            a = pop();
            b = pop();
            c = b / a;
            push(c);
        }
        else{
            push(prefix[i] - 48);
                }
    }
    printf("\nPrefix Expression: %d", pop());
    return 0;
}
void push(int ch){
    top++;
    s[top] = ch;
}
int pop(){
    int ch;
    ch = s[top];
    top = top - 1;
    return (ch);
}
```