Nithin S
221IT085

IT206 Lab Assignment

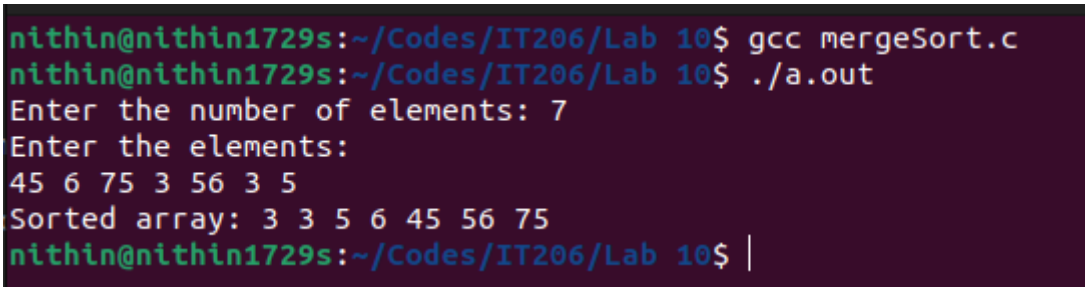
Q1) Implement merge sort

CODE

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

void merge(int A[], int l, int mid, int h)
{
    int i = l, j = mid + 1, k = l;
    int B[100];
    while (i <= mid && j <= h)
    {
        if (A[i] < A[j])
            B[k++] = A[i++];
        else
            B[k++] = A[j++];
    }
    for (; i <= mid; i++)
        B[k++] = A[i];
    for (; j <= h; j++)
        B[k++] = A[j];
    for (i = l; i <= h; i++)
        A[i] = B[i];
}

void mergeSort(int A[], int l, int h)
{
    int mid;
```

```
    if (l < h)
    {
        mid = (l + h) / 2;
        mergeSort(A, l, mid);
        mergeSort(A, mid + 1, h);
        merge(A, l, mid, h);
    }
}

int main()
{
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int A[n], i;
    printf("Enter the elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &A[i]);

    mergeSort(A, 0, n - 1);

    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", A[i]);
    printf("\n");

    return 0;
}
```

OUTPUT



Q2) Infix to PostFix

CODE

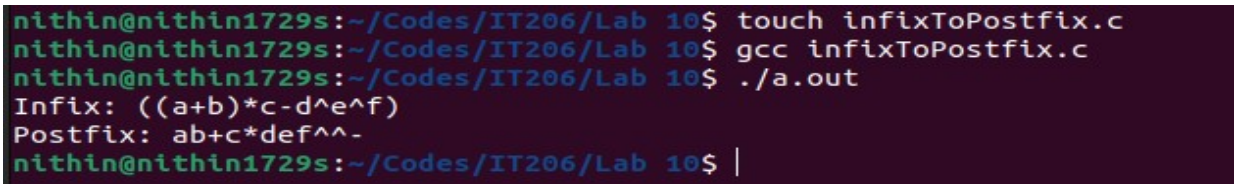
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Node
{
    char data;
    struct Node *next;
};

struct Node *stk1 = NULL;

void push(struct Node **head, char x)
{
    struct Node *temp = (struct Node
*)malloc(sizeof(struct Node));
    temp->data = x;
    temp->next = *head;
    *head = temp;
}

char pop(struct Node **head)
{
    char x;
    struct Node *p = *head;
    (*head) = (*head)->next;
    x = p->data;
    free(p);
    return x;
}

int isEmpty(struct Node *head)
{
    if (head == NULL)
        return 1;
    else
        return 0;
```

```
}

char stackTop(struct Node *head)
{
    char x = '\0';
    if (head == NULL)
        return x;
    return (head->data);
}

int outStackprecedence(char x)
{
    if (x == '\0') return -1;
    if (x == '+' || x == '-') return 1;
    else if (x == '*' || x == '/') return 3;
    else if (x == '^') return 6;
    else if (x == '(') return 7;
    else if (x == ')') return 0;
    else return -1;
}

int inStackprecedence(char x)
{
    if (x == '\0') return -1;
    if (x == '+' || x == '-') return 2;
    else if (x == '*' || x == '/') return 4;
    else if (x == '^') return 5;
    else if (x == '(') return 0;
    else return -1;
}

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

char *infixToPostfix(char *infix)
{
    int i = 0, j = 0;
```

```
    int len = strlen(infix);
    char *postfix = (char *)malloc(sizeof(char) *
(len + 1));

    while (infix[i] != '\0')
    {
        if (isOperand(infix[i]))
        {
            postfix[j++] = infix[i++];
        }
        else
        {
            if (outStackprecedence(infix[i]) >
inStackprecedence(stackTop(stk1)))
            {
                push(&stk1, infix[i++]);
            }
            else if (outStackprecedence(infix[i]) ==
inStackprecedence(stackTop(stk1)))
            {
                pop(&stk1);
                i++;
            }
            else
            {
                postfix[j++] = pop(&stk1);
            }
        }
    }
    while (!isEmpty(stk1))
    {
        postfix[j++] = pop(&stk1);
    }
    postfix[j] = '\0';

    return postfix;
}

int main()
{
    char infix[] = "((a+b)*c-d^e^f)";
```

```
    printf("Infix: %s \n", infix);
    char *postfix = infixToPostfix(infix);
    printf("Postfix: %s \n", postfix);
    free(postfix);
    return 0;
}
```

OUTPUT



Q3) Postfix Evaluation

CODE

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

struct Stack
{
    int data;
    struct Stack *next;
};

struct Stack *stk1 = NULL;

void push(struct Stack **head, int x)
{
```

```
    struct Stack *temp = (struct Stack
*)malloc(sizeof(struct Stack));
    temp->data = x;
    temp->next = *head;
    *head = temp;
}

int pop(struct Stack **head)
{
    int x;
    struct Stack *p = *head;
    (*head) = (*head)->next;
    x = p->data;
    free(p);
    return x;
}

int isEmpty(struct Stack *head)
{
    if (head == NULL)
        return 1;
    else
        return 0;
}

int stackTop(struct Stack *head)
{
    int x = 0;
    if (head == NULL)
        return x;
    return (head->data);
}

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

```
int eval(char *postfix)
{
    int i, x1, x2, r;
    for (i = 0; postfix[i] != '\0'; i++)
    {
        if (isOperand(postfix[i]))
            push(&stk1, postfix[i] - '0'); // Convert
character to integer
        else
        {
            x2 = pop(&stk1);
            x1 = pop(&stk1);
            switch (postfix[i])
            {
            case '+':
                r = x1 + x2;
                break;
            case '-':
                r = x1 - x2;
                break;
            case '*':
                r = x1 * x2;
                break;
            case '/':
                if (x2 != 0) // Check for division by
zero
                    r = x1 / x2;
                else
                {
                    printf("Error: Division by zero\
n");
                    exit(1);
                }
                break;
            }
            push(&stk1, r);
        }
    }
    return pop(&stk1);
}
```

```
int main()
{
    char *postfix = "35*62/+4-";
    printf("%s  = ",postfix);
    int res = eval(postfix);
    printf("%d ", res);
    return 0;
}
```

OUTPUT