Data Structures With C Lab

Course Code : 18CSL37

Year : 2020-21

Course Learning Objectives

1.) Demonstrate abstract properties of various data structures such as stacks, queues etc.
2.) Compare different implementations of Datastructures
3.) Demonstrate features of different data structures to solve real world problems.

Course Outcomes.

1.) Demonstrate understanding of structured programming.
2.) Analyze problem statement & able to choose right data structure for implementation.
3.) Develop an ability to construct robust, maintainable programs which satisfy requirements of user.

Term Work - 1

Problem Definition:

Write a C program to merge contents of a file containing USN's of students in a sorted order in to the 3rd file such that 3rd file contains Unique USN's. Program should also display common USN's in both files.

Aim:

The purpose of this TW is to learn the concept of File handling in C. Basic operations using files & implementation of this concept in solving problems

Theory.

In software industry, most of the programs are written to store info. fetched from programs. One such way is to store fetched info in a file. Different operations that can be performed on a file are:

→ Creation of a new file (fopen with attributes)
→ Opening an existing file (fopen)
→ Reading from file (fscanf or fgets)
→ Writing to a file (fprintf)
→ Closing a file (fclose).

Program:

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

int main (int argc, char *argv[]) {
  FILE *f1, *f2, *f3;
   f1 = fopen ("usnfile1.txt", "r");
   f2 = fopen ("usnfile2.txt", "r");
   if (f1 == NULL)
       printf ("\n File 1 cannot be found");
   if (f2 == NULL)
       printf ("\n File 2 cannot be found");
  f3 = fopen ("usnfile3.txt", "w");
 int maxchar = 15;
 char us1[maxchar], usn2[maxchar];
 fgets (usn1, maxchar, f1);
 fgets (usn2, maxchar, f2);
 while (! feof (f1) && ! feof (f2)) {
      if (( strcmp (usn1, usn2)) < 0) {
            fprintf (f3, "\n %s\n", usn1);
             fprintf (f3, "\n %s \n", usn2);
    }
    else {
        fprintf (f3, "\n %s" \n", usn2);

        fprintf (f3, "\n %s \n", usn1);
    }
    fgets (usn1, maxchar, f1);
    fgets (usn2, maxchar, f2);
```

```
if (feof (f2)){
        while (!feof (f1)) {
            fgets (usn1, maxchar, f1);
            fprintf (f3, "%s\n", usn1);
        }
}
else {
    while (!feof (f2)){
        fgets (usn2, maxchar, f2);
        fprintf (usn f3, "\n%s\n", usn2);
    }
}

fclose (f1);
fclose (f2);
fclose (f3);
printf ("Done");

return 0;
}
```

```
Done
Process returned 0 (0x0)   execution time : 1.812 s
Press any key to continue.
```

```
Done
Process returned 0 (0x0)   execution time : 1.812 s
Press any key to continue.
```

References:

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

Term Work - 2

Problem Definition:

Consider a calculator that needs to perform checking correctness of paranthesized arithmetic & convertid the same postfix expression for evaluation. Develop & execute a program in C using suitable DS to perform the same & print both expressions. The input expression consists of a single character operands & binary operators.

Aim:

Aim of this TW is to learn the implementation of Stacks in solving problems.

Theory:

* Stacks : stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO or FIFO

* Mainly following basic operations are performed in Stack
  - → Push : Adds an item in stack. If stack is full then it is said to overflow
  - → Pop : Removes an item from stack. If stack is empty, then stack is said to underflow
  - → Peek : Returns top element of stack.
  - → isEmpty : Returns true if the stack is empty

Program:

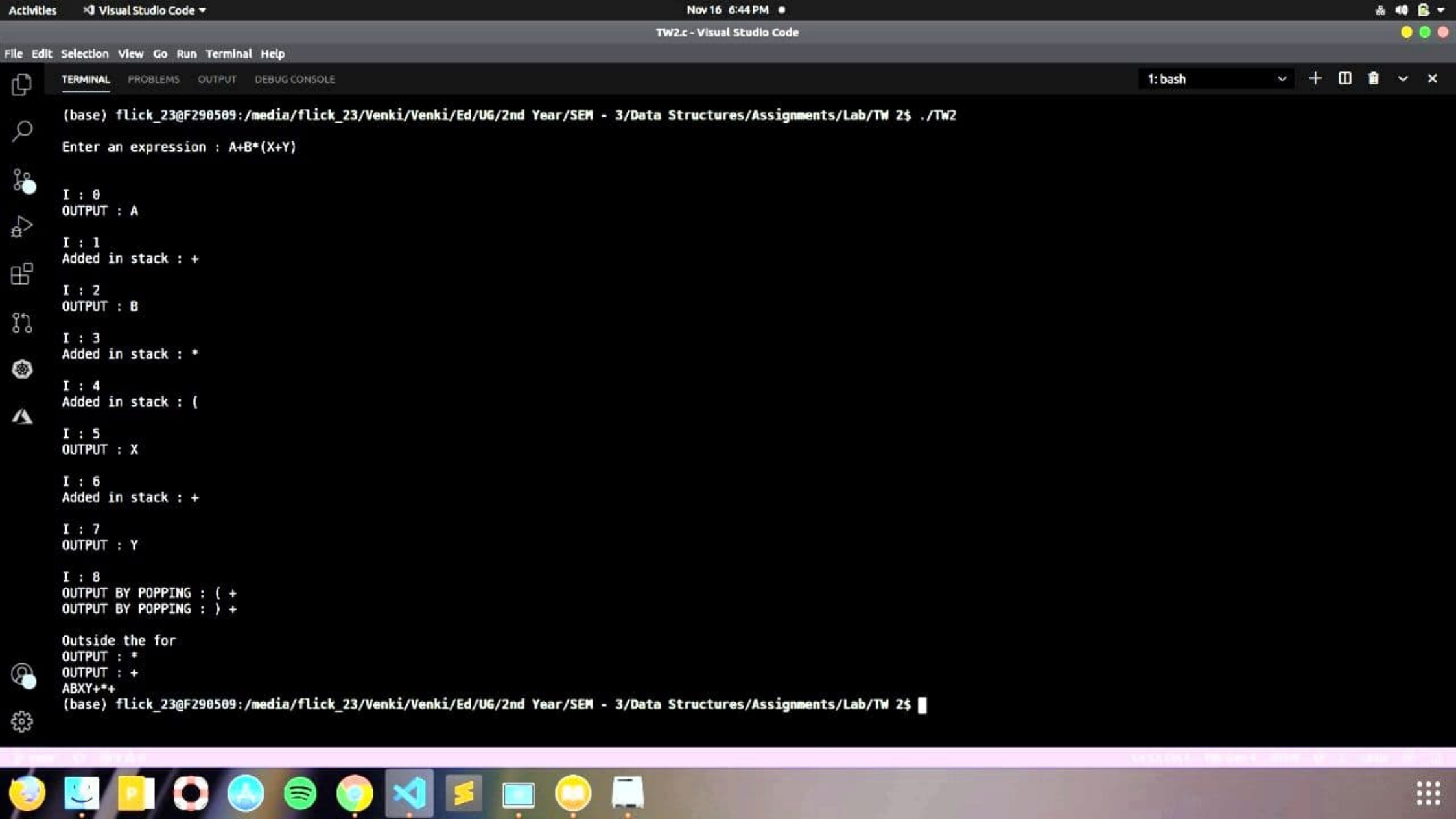
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct Stack {
    int top, capacity;
    int *a;
}
int isEmpty (struct Stack *s) {
    return s->top == -1
}
char peek (struct Stack *s) {
    return s->top s->a[s->top];
}
char pop (struct Stack *s) {
    if (! isEmpty (s))
        return s->a[s->top--];
    return '$';
}
void push (struct Stack *s, char op) {
    s->a[++s->top];
}
int isOperand (char ch) {
    return (ch >='a' && ch<='z') || (ch >='A' && ch <='Z');
}
```

```
int Prec (char ch) {
   switch (ch) {
    case '+':
    case '-':  return 1;
    case '*':
    case '/' : return 2;
    case '^' : return 3;
   }
  return -1;
}
int infixToPostfix (char *exp) {
  struct Stack *s = (struct stack*) malloc (sizeof (struct stack));
  s→top = -1 ;  int i, k=0 ;
  s→capacity = strlen (exp);
  s→a = (int*) malloc (s→capacity * sizeof (int));
  if (!s)
    return -1;
  for (i=0, k=-1 ; i<l; ++i) {
    printf("\n\n"  if (isOperand (exp[i])
                          exp [++k] = exp[i];
    else if (exp[i] == '(')
          push (s, exp[i]);
    else if (exp[i] == ')') {
          while (!isEmpty (s) && peek(s) != '(')
                exp [++k] = pop(s)
          if (!isEmpty (s) && peek(s) != '(')
                return -1;
          else
                pop (s);
    }
```

```
    else {
        while (!isEmpty(s) && Prec(exp[i]) <= Prec(peek(s)))
            exp[++k] = pop(s);
        push(s, exp[i]);
    }
}
while (!isEmpty(s))
    exp[++k] = pop(s);
exp[++k] = '\0';
    printf("\n %s \n", exp);
}
int main() {
    char *exp;
    exp = (char*)malloc(1000*sizeof(char));
    printf("\n Enter an expression : ");
    scanf("%s", exp);
    infixToPostfix(exp);
    return 0;
}

// Written & executed in VS Code, Ubuntu 20.04.1 env.
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE 1: bash

```
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 2$ ./TW2

Enter an expression : A*B-D+C


I : 0
OUTPUT : A

I : 1
Added in stack : *

I : 2
OUTPUT : B

I : 3
OUTPUT BY POPPING : OP *
Added in stack : -

I : 4
OUTPUT : D

I : 5
OUTPUT BY POPPING : OP -
Added in stack : +

I : 6
OUTPUT : C

Outside the for
OUTPUT : +
AB*D-C+
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 2$
```


Activities
Visual Studio Code
Nov 16 6:44 PM
TW2.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
1: bash
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 2$ ./TW2
Enter an expression : A+B*(X+Y)
I : 0
OUTPUT : A
I : 1
Added in stack : +
I : 2
OUTPUT : B
I : 3
Added in stack : *
I : 4
Added in stack : (
I : 5
OUTPUT : X
I : 6
Added in stack : +
I : 7
OUTPUT : Y
I : 8
OUTPUT BY POPPING : (+
OUTPUT BY POPPING :) +
Outside the for
OUTPUT : *
OUTPUT : +
ABXY+*+
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 2$

References

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion.

In this TW, I learnt about stacks, basic operations of stacks & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

Term Work - 3

Problem Definition

A calculator needs to evaluate a postfix expression. Develop & execute a program in C using a suitable data structure to evaluate valid postfix expression. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands & binary arithmetic operators. The arithmetic operators are +, -, * & /.

Aim:

Aim of this TW is to learn the implementation of Stacks in solving problems.

Theory:

* Stacks : stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO or FIFO

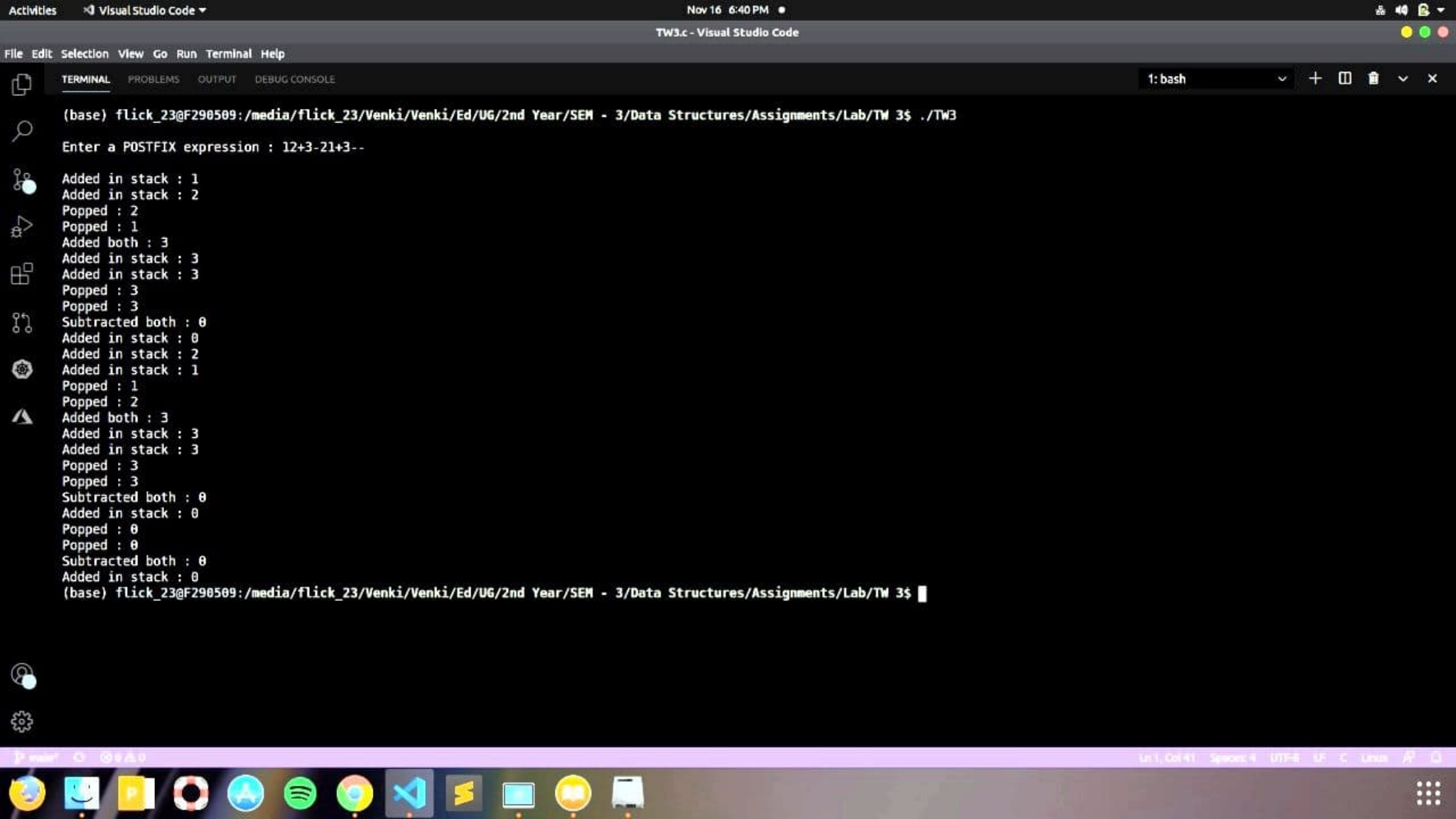
* Mainly following basic operations are performed in Stack
→ Push : Adds an item in stack. If stack is full then it is said to overflow
→ Pop : Removes an item from stack. If stack is empty, then stack is said to underflow
→ Peek : Returns top element of stack.
→ isEmpty : Returns true if the stack is empty

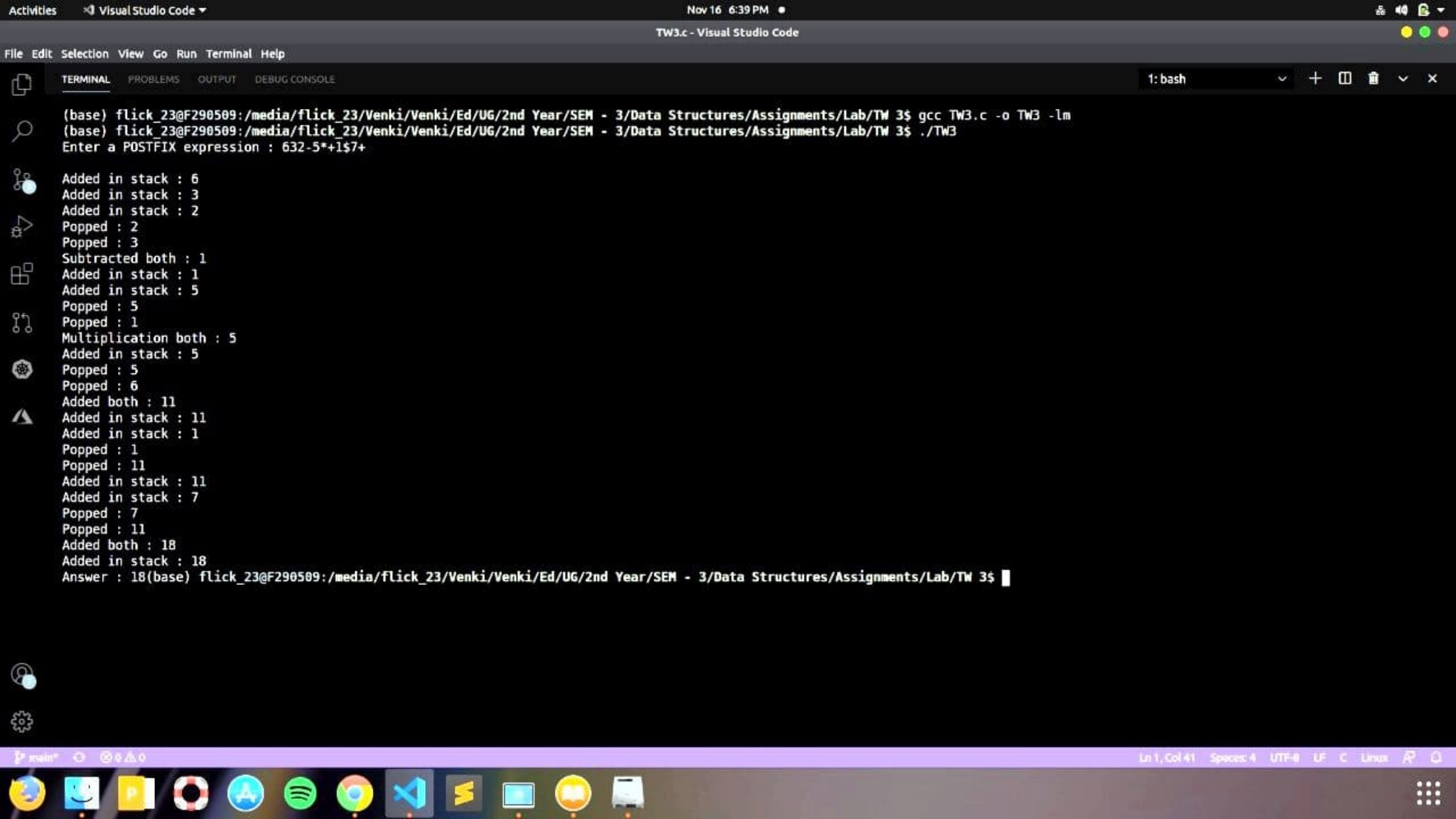
Program:

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

struct stack {
    int capacity, top;
    int *a;
};
int isEmpty(struct stack *s) {
    return s->top == s->capacity;
}
void push(struct stack *s, int op) {
    s->a[++s->top] = op;
}
int pop(struct stack *s) {
    if(!isEmpty(s)){
        return s->a[s->top--];
    }
}
int peek(struct stack *s) {
    if(!isEmpty(s){
        return s->a[s->top];
    }
}
void postFixToInfix(char * exp) {
    int i, op1=0, op2=0, result=0, x=0;
```

```
struct stack *s = (struct stack*)malloc(sizeof(struct stack));
s->top = -1;   s->capacity = strlen(exp);
s->a = (int*)malloc(sizeof(struct int) * s->capacity);
for(i=0; i<s->capacity; i++){
    if(exp[i]>='0' && exp[i]<='9'){
        x = exp[i] - '0';
        push(s, x);
    }
    else{
        op2 = pop(s);
        op1 = pop(s);
        switch(exp[i]){
            case '+': result = op1 + op2;
            break;
            case '-': result = op2 - op1;
            break;
            case '*': result = op2 * op1;
            break;
            case '/': result = op2 / op1;
            break;
        }
        push(s, result);
    }
result = pop(s)
printf("\n Answer : %d", result);
}
```

```
int main()
{
    char exp[100];

    printf("\n Enter a postfix expression: ");
    scanf("%s", exp);

    postFixToInfix(exp);
}
```


Activities
Visual Studio Code
Nov 16 6:40 PM
TW3.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
1: bash
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 3$ ./TW3
Enter a POSTFIX expression : 12+3-21+3--
Added in stack : 1
Added in stack : 2
Popped : 2
Popped : 1
Added both : 3
Added in stack : 3
Added in stack : 3
Popped : 3
Popped : 3
Subtracted both : 0
Added in stack : 0
Added in stack : 2
Added in stack : 1
Popped : 1
Popped : 2
Added both : 3
Added in stack : 3
Added in stack : 3
Popped : 3
Popped : 3
Subtracted both : 0
Added in stack : 0
Popped : 0
Popped : 0
Subtracted both : 0
Added in stack : 0
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 3$


Activities
Visual Studio Code
Nov 16 6:39 PM
TW3.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
1: bash
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 3$ gcc TW3.c -o TW3 -lm
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 3$ ./TW3
Enter a POSTFIX expression : 632-5*+1$7+
Added in stack : 6
Added in stack : 3
Added in stack : 2
Popped : 2
Popped : 3
Subtracted both : 1
Added in stack : 1
Added in stack : 5
Popped : 5
Popped : 1
Multiplication both : 5
Added in stack : 5
Popped : 5
Popped : 6
Added both : 11
Added in stack : 11
Added in stack : 1
Popped : 1
Popped : 11
Added in stack : 11
Added in stack : 7
Popped : 7
Popped : 11
Added both : 18
Added in stack : 18
Answer : 18(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 3$
Ln 1, Col 41 Spaces: 4 UTF-8 LF C Linux

References

Books:

* Richard F Gilberg, Behrouz A Fouruzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion.

In this TW, I learnt about stacks, basic operations of stacks & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

TermWork - 4

Problem Definition.

Write a C program to simulate working of Messaging System in which a message is placed in a Queue by a message sender, a message is removed from queue by a message reciever, which can also display contents of Queue.

Aim:

The purpose of this TW is to learn the concept of Queues in C language. Basic operations using queues & implementation of this data structure in solving problems.

Theory:

Like Stack, Queue is a linear structure which follows a particular order in which operations are performed. The order is FIFO. Mainly, the following 4 basic operations are performed on queue:

→ Enqueue : Adds an item to the queue.
→ Dequeue : Removes an item from the queue.
→ Front : Get front item from queue.
→ Rear : Get rear item from queue.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 5
struct msgq {
    char msg[MAX_SIZE][100];
    int rear, front;
};
void initq ( struct msgq *q){
    q→ front = q→ rear = -1;
}
int qfull (struct msgq q){
    return (q.rear == MAX_SIZE -1) ? 1 : 0);
}
int qempty (struct msgq q){
    return ((q.front == -1 && q.rear == -1) || q.front > q.rear) ? 1 : 0);
}
int sender (struct msgq *q, char msg[100]) {
    if (!qfull (*q)) {
        if (q→front == -1) q→ front = 0;
        strcpy (q→msg [++(q → rear)], msg);
        return 1;
    }
    printf ("In QUEUE FULL\n"); EMPTY ");
    return 0;
}
```

```
int receiver (struct msgq *q) {
    if (!qempty (*q)) {
        printf ("Messag = %s", q->msg [q->front]);
        (q->front)++;
        return 1;
    }
printf ("\n QUEUE EMPTY");
return 0;
}
void displayqueue (struct msgq mq){
int main (int argc, char **argv) {
    struct msgq mq;
    int role, flag;
    char msg[100];
    initq (&mq);
    while (1) {
    printf ("\n Select your role: \n1. Sender \n2: Reciever \n3: Exit ");
    scanf ("%d", &role);
    if (role == 1) {
        printf ("\n Enter message: ");
        if (sender (&mq, msg)
            printf ("\n Message Sent ");
        else
            printf ("\n Message is NOT SENT");
    }
    if (role == 2) {
        if (receiver (&mq) {
            printf ("\n Message read successfully!");
        else
            printf ("\n No Messages in queue ");
}
```

```
        if (role == 3)
            break;
    }
}
```

References:

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures : A Pseudo Code Approach with C, Cengage 2007.

E-Resources:

* https://geeksforgeeks.org/

Conclusion:

In this TW, I learnt about queues, basic operations of queues & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE 1: bash

```
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 4$ ./TW4

Select your Role:
 1: Sender
 2: Receiver
 3: Exit
2

Queue is Empty!!!
 No messages in queue SORRY!!!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
1

Enter the message: Yo!!

Message sent Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
2
Message=Yo!!
Message read Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
1

Enter the message: Wassupp??

Message sent Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
2
Message=Wassupp??
Message read Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
3
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 4$
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE 1: bash

```
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 4$ ./TW4

Select your Role:
 1: Sender
 2: Receiver
 3: Exit
1

Enter the message: hey

Message sent Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
2
Message=hey
Message read Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
1

Enter the message: Lets play!

Message sent Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit

Enter the message:
Message sent Successfully!
Select your Role:
 1: Sender
 2: Receiver
 3: Exit
3
(base) flick_23@F290509:/media/flick_23/Venki/Venki/Ed/UG/2nd Year/SEM - 3/Data Structures/Assignments/Lab/TW 4$
```

Term Work - 5

Problem Definition

Consider a supermarket scenario where sales manager wants to search for customer ~~detail~~ details using a customer id. Customer information are stored as a structure & id will be used as hash key. Develop & execute a program in C using suitable DS to implement:

a.) Insertion of new data

b.) Search for customer info using ID

c.) Display records

Aim:

To learn implementation of hashing is solving problems.

Theory:

Hashing is an important data structure which is designed to use a special function called the hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the ~~hands~~ hash function used. Let a hash function H(x) maps the value x at index x%10 in an array eg if the limit of values is [11, 12, ..., 15] it will be stored at post {1, 2, .., 5} in array or hash table

Program:

```
#include <stdio.h>
#include <stdlib.h>
# define size 10

struct customer {
        int custid;
        char custname[30];
        double custphno;
};
struct Record {
    struct customer info;
    int empty;
};
int Hashfx (int key){
        return key % size;
}
int search ( int key , struct record a[]){
    int count, temp, pos;
    temp = Hashfx (key);
    pos = temp;
    for (count = 1 ; count <= size ; count ++){
            if (a[pos].empty == 1)
                    return -1;

    }
    if (a[pos].info).custid == key)
                    return pos;
    pos = (temp + count) % size; }
    return -1;
}
```

```
void INSHT LP(struct customer cust, struct Record ht[]){
    int count, post, temp;
    int key = cust.custid;
    temp = Hashfx(key);
    pos = temp;
    for(count = 1; count <= SIZE; count++){
        if(ht[pos].empty == 1){
            ht[pos].info = cust;
            ht[pos].empty = -1;
            printf("\n Record inserted into Hash Table\n");
            return;
        }
        if((ht[pos].info).custid == key){
            printf("\n Duplicate Record cannot be inserted\n");
        }
        pos = (temp + count) % size;
        printf("\n Hash table is Full\n");
}
void display(struct Record ht[]){
    int count;
    printf("\n Hash Table");
    for(count=0; count<size; count++){
        printf("\n[%d] :\t", count);
    if(ht[count].empty == -1){
    printf("\n Customer ID : %d \t Name : %s\t Phone : %f",
        (ht[count].info).custid, (ht[count].info).custname,
        (ht[count].info).custphno);
    }
```

```
    else {
        printf("\n No hash entry found");
    }
}

int main() {
    int count, key, option;
    struct record ht[size];
    struct customer cust;
    for(count = 0; count < size ; count++) {
        ht[count].empty = 1;
    }
    while(1) {
    printf("\n 1. Insert a record \n 2. Search a Record \n 3.
     Display all Records \n 4. Exit");
    printf("\n Enter your option");
    scanf("%d", &option);
    switch(option) {
      case 1: printf("\n Enter customer ID, name, phone number");
              scanf("%d %s %ld", &cust, &cust.custid, cust.custid,
              cust.custname, cust.custphone);
              INSHT_LP(cust, ht);
      break;
      case 2: printf("\n Enter key to search:");
              scanf("%d", skey);
              count = search(key.ht);
              if(count == -1)
                  printf("\n Record not found");
```

```
        else
          printf("\n Record found at index : %d ", count);
        break;
case 3:  display(lt);
break;
case 4:  printf("\n Program Complete");
          exit(1);
 break;
 }
}
return 0;
}
```

References :

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universe Press 2nd Edition.

E-Resources :

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

```
1. Insert a Record
2. Search a Record
3. Display All Records
4. Exit

Enter Your Option:2

Enter the Key to Search:2

Record Found at Index pos:2

1. Insert a Record
2. Search a Record
3. Display All Records
4. Exit

Enter Your Option:1

Enter Customer id, name, ph:3 Ximi 986564346

Record Inserted into Hash Table

1. Insert a Record
2. Search a Record
3. Display All Records
4. Exit

Enter Your Option:3

Hash Table
[0]:    Customer - ID: 0     Name: minho     Phone: 765645474.000000
[1]:    Customer - ID: 1     Name: hazel     Phone: 767825626.000000
[2]:    Customer - ID: 2     Name: Olivia    Phone: 98764356.000000
[3]:    Customer - ID: 3     Name: Ximi      Phone: 986564346.000000
[4]:    No Hash Entry
[5]:    No Hash Entry
[6]:    No Hash Entry
[7]:    No Hash Entry
[8]:    No Hash Entry
[9]:    No Hash Entry
1. Insert a Record
2. Search a Record
3. Display All Records
4. Exit

Enter Your Option:4

Process returned 1 (0x1)   execution time : 122.016 s
Press any key to continue.
```

Term-Work 6

Problem Statement

Consider a warehouse where the items have to be arranged in ascending order. Development & execute a program in C using suitable data structures to implement warehouse such that items can be traced easily.

Aim.

To learn the implementation of linked lists in solving problems.

Theory.

A linked list is a sequence of data structures which are connected together via links. Linked lists is a sequence of links which contains items. Each link contains a connection to another link. It is the second most used data structure

Basic operations of linked lists are insertion, deletion, display, delete.

Program:

```
#include <stdio.h>
struct node {
    int data;
    struct node *next;
};
void display (struct node * head) {
    struct node* temp;
    temp = head;
    while (temp != NULL) {
        printf ("%d", temp->data);
        temp = temp->next;
    }
}
struct node* add (struct node* head, int data) {
    struct node* newnode, *prev, *curr;
    newnode->data = value;
    newnode->next = NULL;
    if (newnode == NULL)
        printf (" Could not allocate memory ");
    else {
        if (head == NULL)
            head = NULL
        else {
            curr = head->next;
            prev = head;
            while (curr != NULL && newnode->data != curr->data)
                prev = prev->next;
                curr = curr->next;
        }
```

```
            prev->next = newnode;
            newnode->next = curr;
        }
    }
}
 return head;
}
void main(){
int choice, value;
struct node *head = NULL;
 while (choice != 3) {
   printf("\n1.Add \n2.Display \n3.EXIT");
   printf("\n Enter choice");
   scanf("%d", &choice);
   switch (choice) {
    case 1: printf("\n Enter value: ");
            scanf("%d", &value);
            head = add(head, value);
   break;
   case 2: if(head == NULL) {
               printf("\n list is empty"); }
            display(head);
             break;
 case 3:    exit(1);
 break;
  }
 }
}
```

References:

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving technique & programming paradigms.

```
Enter 1.Add 2.Display 3.Exit
Enter choice:2
List is empty
Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:7

Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:4

Enter 1.Add 2.Display 3.Exit
Enter choice:2
4  7
Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:5

Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:12

Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:7

Enter 1.Add 2.Display 3.Exit
Enter choice:2
4  5  7  7  12
Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:0

Enter 1.Add 2.Display 3.Exit
Enter choice:1
Enter value:3

Enter 1.Add 2.Display 3.Exit
Enter choice:2
0  3  4  5  7  7  12
Enter 1.Add 2.Display 3.Exit
Enter choice:3

Process returned 1 (0x1)   execution time : 51.828 s
Press any key to continue.
```

Term Work - 07

Problem Definition

Consider a polynomial addition for two polynomials. Develop & execute a program in C using suitable DS to implement the same.

Theory

A polynomial may be represented using array (or) structure. A structure may be defined such that it contains 2 parts:

1) One is coefficient
2) Second is corresponding exponent.

The basic idea of polynomial addition is to add coefficient parts of the polynomials having same exponent

The terms are arranged in ascending order of the exponent of terms in 'x'.

Program:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
        int coeff;
        int exp;
        struct node* next;
} node;
typedef struct {
        int count;
        node* head;
} poly;

node* getnode(int c, int e){
        node *temp = (node*) malloc(sizeof(node));
        temp→coeff = c;
        temp→exp = e;
        temp→next = NULL;
        return temp;
}
void addterm(poly *p, int c, int e){
   node* temp, *newnode;
   newnode = getnode(c, e);
   if(p→head == NULL) {
           p→head = newnode;
           p→count++;
           return;
   }
   temp = p→head;
```

```
while (temp->next != NULL) {
        temp = temp->next;
        temp->next = newnode;
        p->count++;
}
void display (poly p) {
    node *temp = p->head;
    while (temp != NULL) {
            printf ("%d x ^ %d -> ", temp->coeff, temp->exp);
            temp = temp->next;
    }
    printf ("Total terms : %d\n", p->count);
}
void polyadd (poly *p1, poly *p2, poly *p3) {
        int c1, c2;
        node *temp, *t1, *t2;
        c1 = p1->count;
        c2 = p2->count;
        t1 = p1->head;
        t2 = p2->head;
        while (c1 != 0 && c2 != 0) {
            temp = (node *) malloc (sizeof (node));
            if (t1->exp == t2->exp) {
                addterm (p3, t1->coeff + t2->coeff, t1->exp);
            c1--;
            c2--;
            t1 = t1->next;
            t2 = t2->next;
        }
```

```
    else if (t1->exp > t2->exp) {
        addterm (p3, t1->coeff, t1->exp);
        c1--;
        t1 = t1->next;
    }
    else {
        addterm (p3, t2->coeff, t2->exp);
        c2--;
        t2 = t2->next;
    }
}
while (c2!=0) {
    addterm (p3, t2->coeff, t2->exp);
    c2--;
    t2 = t2->next;
}
while (c1!=0) {
    addterm (p3, t1->coeff, t1->exp);
    c1--;
    t1 = t1->next;
}
int main() {
    poly p1, p2, p3;
    p1->head = NULL;
    p1->count = 0;
    addterm (&p1, 3, 14);
    addterm (&p1, 2, 8);
    addterm (&p1, 1, 0);
    printf("Polynomial 1: \n");
    display (p1);
```

```
p2->head = NULL;
p2->count = 0;
addTerm(&p2, 100, 45);
addTerm(&p2, 8, 14);
addTerm(&p2, 4, 6);
addTerm(&p2, 1, 5);
printf("Polynomial 2: ");
display(p2);
p3->head = NULL;
p3->count = 0;
~~addTerm~~ //p3 = p1 + p2;
polyadd(&p1, &p2, &p3);
printf("Resultant polynomial: ");
display(p3);
return 0;
}
```

References

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

```
3x^14 -> 2x^8 -> 1x^0 -> NULL
Total terms: 3
3x^14 -> 2x^6 -> 1x^0 -> NULL
Total terms: 3
11x^14 -> 2x^8 -> 2x^6 -> 2x^0 -> NULL
Total terms: 4
Process returned 0 (0x0)   execution time : 0.075 s
Press any key to continue.
```

Term Work - 8

Problem Definition

Develop & execute a program in C to:

a) To count number of non-terminal nodes.
b) To count number of terminal nodes.
c) To count nodes with degree 2.
d) To count total number of nodes.

Theory:

A Binary Search Tree (BST) is a tree in which all the nodes follow mentioned properties:

1) The value of key of the left sub-tree is less than value of its parent node's key.
2) The value of key of right sub-tree is greater than (or) equal to value of its parent nodes key.

Thus, BST divides all its sub-trees into two segments: the left subtree & right sub-tree

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int data;
        struct node* left, *right;
};
int getleafCount (struct node* node) {
    if (node = NULL)
            return 0;
    if (node → left != NULL && node → right != NULL)
            return 1;
    return getleafCount (node → left) + getleafCount (node → right);
}
int nonleafnode (struct node * root) {
    if (root == NULL || (root → leaft == NULL && root → right == NULL))
            return 0;
    return 1+nonleafnodes (root → left) + nonleafnodes (root → right);
}
int countTwo (struct node* node) {
    if (node == NULL)
            return 0;
    if (node → left != NULL && node → right != NULL)
            return 1;
    return countTwo (node → left) + countTwo (node → right);
}
```

```
struct node* newNode (int data) {
    struct node* node = (struct node*)malloc (sizeof (struct node));
    node → data = data;
    node → left = NULL;
    node → right = NULL;
    return (node);
}
int main () {
  struct node *root = newNode (1);
  root → left = newNode (2);
  root → right = newNode (3);
  root → left → left = newNode (4);
  root → left → right = newNode (5);
  printf ("Leaf count of tree is %d \n", getLeafCount (root));
  printf ("Non leaf count of tree is %d \n", nonLeafNode (root));
  printf ("Nodes with degree 2 is %d \n", countTwo (root));
  printf ("Total count of nodes is %d \n", countNodes (root));

  getchar ();
  return 0;
}
```

References

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.
* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

```
Leaf count of the tree is 3
non Leaf count of the tree is 2
nodes with degree 2 is 1
Total count of nodes is 5


Process returned 0 (0x0)   execution time : 1960.964 s
Press any key to continue.
```

Term Work - 9

Problem Definition

Write a program in C suitable datastructure to create a binary tree for an expression. The tree traversals in some proper method should result in conversion of original expression into prefix, infix & postfix forms.

Display the original expression along with 3 different forms also

Theory:

In a preorder traversal the sequence is node → left → right since node is visited before left sub-tree it is known as preorder traversal.

In an inorder traversal the sequence is left → node → right. Since coe will visit nodes "in order" from left to right it is known as inorder traversals.

In a post order traversal the sequence is left → right → node. Since left subtree is visited before right subtree it is known as post order traversal.

Program:

```
#include <stdio.h>
#include <iostream.h>
struct node {
    struct node *left, *right;
    char data;
};
char expression[256], operatorstack[128];
int operatorstacktop = -1;
struct node* createNode(char value) {
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
void pushInOutputstack(struct node* newNode) {
    struct node* popfromoutputstack();
    return outputstack[outputstacktop--];
}
char popfromOperatorStack() {
    return operatorstack[operatorStacktop--];
}
```

```
void printErrorMessage(){
    printf("\n Invalid Expression, Expression should have single characters"
    printf("or digit operator operand & + - */ as operators \n");
}
int precedence(char operation){
    switch(operation){
        case 'C' : return 0;
        case '*' :
        case '+' :
        case '/' :
        case '-' : return 2;
    }
}
struct node* infixToBinaryTree(char *expression){
    int i=0;
    while(expression[i]!='\0'){
        if(expression[i]==" ")
            pushIntoOutputStack(createNode(expression[i]));
        else if(expression[i]==')'){
            int j = operatorStackTop;
            while(operatorStack[j]!='(' && j>=0){
                buildexpression();
                j--;
            }
        }
    char temp = popFromOutputstack();
```

```
}
else if(expression[i]=='+'||expression[i]=='-'||expression[i]=='*'){
    while(precendence(operatorstack[operatorstackTop])>=precedence(expression[i]))
        buildExpression();
    pushIntoOperatorstack(expression[i]);
}
else{
    printErrorMessage();
    exit(1);
}
i++;
}
void inorder(struct node *root){
    if(root!=NULL){
        inorder(root->left);
        printf("%c",root->data);
        inorder(root->right);
    }
}
void postorder(struct node *root){
    if(root!=NULL){
        postorder(root->left);
        postorder(root->right);
        printf("%c",root->data);
    }
}
int main(){
printf("\n Expression should have single character or digit as operand");
printf("\n and + - * / operators \n Enter valid arithmetic operator");
scanf("%[^\n]",expression);
struct node *ast = infixTobinaryTree(expression);
printf("\n Prefix = ");
preorder(ast); printf("\n Infix = "); inorder(ast); printf("\n Postfix"); postorder(
}
```

References

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

```
Expression should have single character or digit as operand and + - * / operators
Enter valid arithmetic expression : (2+3*(4/2)-1)

Original expression = (2+3*(4/2)-1)

Prefix = - - + 2 * 3 / 4 2 1
Infix = 2 + 3 * 4 / 2 - 1
Postfix = 2 3 4 2 / * + 1 -
rocess returned 0 (0x0)   execution time : 120.288 s
ress any key to continue.
```

TermWork - 10

Problem Statement :

Develop & execute a program with C using suitable DS to perform searching a data item in an ordered & list of items in both directions & implementation of following operations :

a) Create a DLL by adding each node at start

b) Insert a new node at end of list.

c) Display contents of list.

Consider an integer number as a data item.

Theory

DLL is a complex type of linked list in which a node contains a pointer to previous as well as the node in sequence.

Therefore, in a DLL a node consists of 3 parts:

Node data, pointer to next node in sequence, pointer to previous node.

The previous part of node & next part of last node will always contain null indicating end in each direction.

Program:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node {
        int data;
        struct node *prev,*next;
} Node;
typedef struct {
      int count;
      NODE *head;
      NODE *rear;
}List;
List *createList(){
            LIST *lptr = (LIST *)malloc(sizeof(LIST));
            lptr->count =0
            lptr->head = lptr->rear = NULL;
            return lptr;
}
Node *getNode(int e){
        Node *t = (Node*)malloc(sizeof(Node));
        t->data = e;
        t->next = t->prev = NULL;
        return t;
}
void insertFront(List *lp, int ele){
    Node * newNode = getNode(ele);
    if(lp->count==0){
            lp->head = lp->rear = newNode;
    }
    return;
}
```

```
void insertRear (List *lp, int a){
    Node * newnode;
    newnode = getNode (a);
    if (lp→ head ==NULL){
        insertFront(lp,a);
        return;
    }
    lp→ rear = newnode;
    lp→ count ++;
    lp→ rear = newnode;
    return;
}
void displayList (List lp){
    Node *temp;
    if (lp.count ==0){
        printf("\n List is empty");
        return;
    }
    temp = lp.head;
    printf("NULL");
    while (temp !=NULL){
        printf(" /\\"; temp→data);
}
int main(){
    List *pl = createList();
    int ch, inp, dir;
    while(1){
    printf("\n Insert Front \n 2. Insert Rear\n 3. Display \n 4. Search \n 5 Exit ");
    printf("Enter your choice :");
    scanf("%d", &ch);
    switch(ch){
```

```
case 1:
    printf("Enter the element to be inserted: ");
    scanf("%d", &inp);
    insertfront(pl, inp);
    break;
case 2:
    printf("Enter element to be inserted: ");
    scanf("%d", &inp);
    insertrear(pl, inp);
    break;
case 3: displaylist(*pl);
    break;
case 4: printf("Enter element to be searched:");
    scanf("%d", &inp);
    printf("Enter direction: ");
    scanf("%d", &dir);
    int temp = searchlist(*pl, inp, dir);
    if (temp)
        printf("Search element found at posit %d", temp);
    else
        printf("Search element not found");
    break;
case 5:
    printf("Exiting");
    exit(0);
    break;
default:
    printf("Invalid Input");
  }
}
return 0;
}
```

References:

Books:

* Richard F Gilberg, Behrouz A Fourouzan, Data Structures: A Pseudo Code Approach with C, Cengage 2007.

* Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universe Press 2nd Edition.

E-Resources:

* https://geeksforgeeks.org/

Conclusion

In this TW, we learnt about files, basic operations of files & their implementation to solve problems. We also learned basic problem solving techniques & programming paradigms.

```
1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 1
Enter the element to be inserted: 23

1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 1
Enter the element to be inserted: 20

1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 2
Enter the element to be inserted: 22

1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 2
Enter the element to be inserted: 31

1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 3
NULL<- 20 -> <- 23 -> <- 22 -> <- 31 -> NULL
Node Count: 4

1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 4
Enter the element to be searched: 23
Enter the Direction:(1: From Begining 2: From End)1
Search Element found at position 2
1.Insert Front
2.Insert Rear
3.Display List
4.Search
5.ExitEnter your choice: 5
```