VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

RASHI K S (1BM23CS263)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by **RASHI K S (1BM23CS263)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Lakshmi Neelima M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:

a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>

#define MAX 5


int stack[MAX];

int top = -1;


void push(int value) {
    if (top == MAX - 1) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        stack[++top] = value;
        printf("Pushed %d into the stack.\n", value);
    }
}


void pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop.\n");
    } else {
        printf("Popped %d from the stack.\n", stack[top--]);
    }
```

```
}

void display() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements are:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nStack Operations Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

```
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to push: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting program.\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
    }
}

return 0;
```

}

Output:

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Stack Underflow! Cannot pop.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 5
Pushed 5 into the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 2
Pushed 2 into the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements are:
2
```

```
5

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 2 from the stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements are:
5

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```

Lab program 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX 100


char stack[MAX];

int top = -1;
```

```
void push(char ch) {
    if (top != MAX - 1) stack[++top] = ch;
}

char pop() {
    return (top != -1) ? stack[top--] : -1;
}

char peek() {
    return (top != -1) ? stack[top] : -1;
}

int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}

int precedence(char op) {
    return (op == '+' || op == '-') ? 1 : (op == '*' || op == '/') ? 2 : 0;
}

void infixToPostfix(char infix[], char postfix[]) {
    int i, j = 0;
    char ch;

    for (i = 0; i < strlen(infix); i++) {
        ch = infix[i];
```

```
        if (isalnum(ch)) postfix[j++] = ch;
        else if (ch == '(') push(ch);
        else if (ch == ')') {
            while (top != -1 && peek() != '(') postfix[j++] = pop();
            pop();
        } else if (isOperator(ch)) {
            while (top != -1 && precedence(peek()) >= precedence(ch)) postfix[j++] =
pop();
            push(ch);
        }
    }

    while (top != -1) postfix[j++] = pop();
    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter an infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

Output:

```
Enter an infix expression: A+B(C*D-E)/F
Postfix expression: ABCD*E-F/+

Process returned 0 (0x0)   execution time : 108.640 s
Press any key to continue.
```

Lab program 3:

3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:
Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

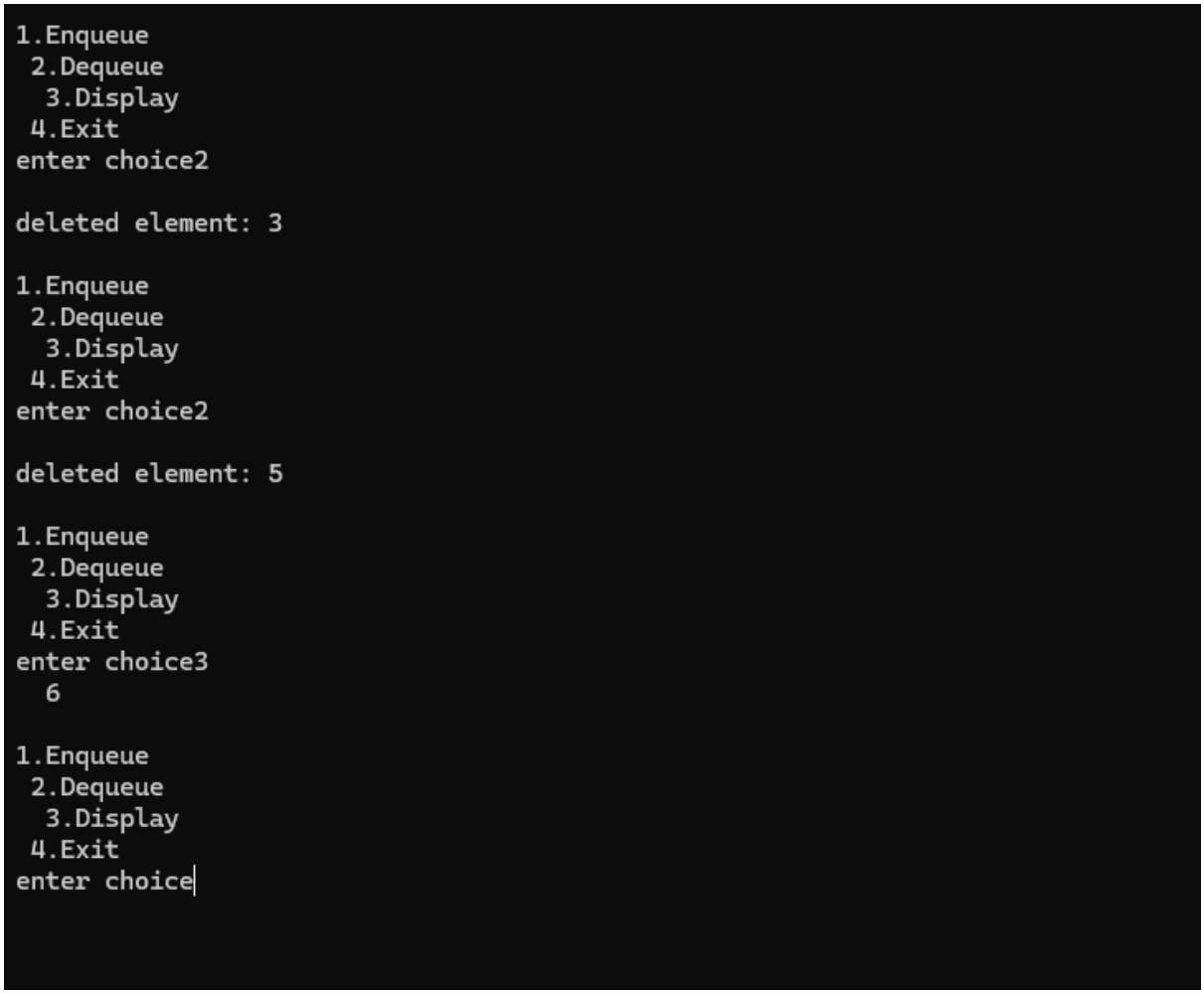
```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 3
int queue[SIZE], rear=-1,front=-1;
int isEmpty()
{
    if(front==-1)
        return 1;
    return 0;
}
int isFull()
{
    if(front==rear+1||front==0&&rear==SIZE-1)
        return 1;
    return 0;
```

```
}
void enqueue(int x)
{
    if(isFull())
        printf("\nqueue overflow");
    else
    {
        if(isEmpty()){
            front=0;
        }
        rear=(rear+1)%SIZE;
        queue[rear]=x;
        printf("inserted:%d",queue[rear]);
    }
}
void dequeue()
{
  if(isEmpty())
        printf("\nqueue underflow");
  else
  {
      int m=queue[front];
      if(front==rear)
        front=rear=-1;
      else
        front=(front+1)%SIZE;
      printf("\ndeleted element: %d",m);
```

```
  }
}
void display()
{
   if(front==-1)
        printf("\n queue is empty");
    else
    { int i;
        for( i=front;i!=rear;i=(i+1)%SIZE)
            printf("  %d  ",queue[i]);
        printf("  %d  ",queue[i]);
}
}
void main()
{
    int x,c;
    while(1)
    {
        printf("\n\n1.Enqueue\n 2.Dequeue\n  3.Display\n 4.Exit ");
        printf("\nenter choice");
        scanf("%d",&c);
        switch(c)
        {
            case 1: printf(" enter element");
                    scanf("%d",&x);
                    enqueue(x);
                    break;
```

```
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
            default : printf("invalid choice try again");
        }
    }
}
```

```
1.Enqueue
 2.Dequeue
  3.Display
 4.Exit
enter choice1
 enter element3
inserted:3

1.Enqueue
 2.Dequeue
  3.Display
 4.Exit
enter choice1
 enter element5
inserted:5

1.Enqueue
 2.Dequeue
  3.Display
 4.Exit
enter choice1
 enter element6
inserted:6

1.Enqueue
 2.Dequeue
  3.Display
 4.Exit
enter choice3
  3    5    6
```



3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert,
Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include<stdio.h>

#include<stdlib.h>

#define SIZE 3

int queue[SIZE], rear=-1,front=-1;

int isEmpty()

{
```

```
    if(front==-1)
        return 1;
    return 0;
}
int isFull()
{
    if(rear==SIZE-1)
        return 1;
    return 0;
}
void enqueue(int x)
{
    if(isFull())
        printf("\nqueue overflow");
    else
    {
        if(isEmpty()){
            front=0;
        }
        rear++;
        queue[rear]=x;
        printf("inserted:%d",queue[rear]);
    }
}
void dequeue()
{
  if(isEmpty())
```

```
        printf("\nqueue underflow");
  else
  {
      int m=queue[front];
      if(front==rear)
        front=rear=-1;
      else
        front=front+1;
      printf("\ndeleted element: %d",m);
  }
}
void display()
{
   if(front==-1)
        printf("\n queue is empty");
    else
    {
        for(int i=front;i<=rear;i++)
            printf("  %d  ",queue[i]);
    }
}
void peek()
{

    if(front==-1)
        printf("\n queue is empty");
    else
```

```
        printf("%d",queue[front]);
}
void main()
{
    int x,c;
    while(1)
    {
        printf("\n\n1.Enqueue\n 2.Dequeue\n 3.peek\n 4.Display\n 5.Exit ");
        printf("\nenter choice");
        scanf("%d",&c);
        switch(c)
        {
            case 1: printf(" enter element");
                    scanf("%d",&x);
                    enqueue(x);
                    break;
            case 2: dequeue();
                    break;
            case 3: peek();
                    break;
            case 4: display();
                    break;
            case 5: exit(0);
            default : printf("invalid choice try again");
        }
    }
}
```

Output:

```
1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice1
 enter element4
inserted:4

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice1
 enter element5
inserted:5

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice1
 enter element7
inserted:7

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice4
```

```
  4    5    7

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice2

deleted element: 4

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice2

deleted element: 5

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice1
 enter element8

queue overflow

1.Enqueue
 2.Dequeue
 3.peek
 4.Display
 5.Exit
enter choice
```

Lab program 4:

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.c) Deletion of first element, specified element and last element in the list.

#include<stdio.h>

```
#include<stdlib.h>
#include<malloc.h>
struct node
{
   int data;
   struct node *next;
};
struct node *start= NULL;
struct node *create(struct node *start);
struct node *display(struct node *start);
struct node *insert_beg(struct node *start);
struct node *insert_end(struct node *start);
struct node *insert_pos(struct node *start);
struct node *delete_beg(struct node *start);
struct node *delete_end(struct node *start);
struct node *delete_node(struct node *start);
int main()
{
   int ch;
   while(1)
   {
      printf("\n\n1.create \t 2.Display \t 3.Insert beg \t4.Insert end\t5.Insert at pos \t6. delete beg\t7.delete end\t 8.delete node \t 9.exit");
      printf("\nenter your choice:");
```

```
scanf("%d",&ch);
struct node *newnode;
switch(ch)
{
   case 1: start=create(start);
          break;
   case 2: start=display(start);
   break;
   case 3 : start=insert_beg(start);
   break;
   case 4: start=insert_end(start);
   break;
   case 5: start=insert_pos(start);
   break;
   case 6: start=delete_beg(start);
   break;
   case 7: start=delete_end(start);
   break;
   case 8: start=delete_node(start);
   break;
   case 9: exit(0);
    default:printf("invalid");
}
```

```
    }
}
struct node *create(struct node *start)
{
    struct node *newnode, *ptr;
    int num;
    printf("\nenter -1 to end");
    printf("\nenter num:");
    scanf("%d",&num);
    while(num!=-1)
    {
        newnode=(struct node*)malloc(sizeof(struct node*));
        newnode->data=num;
        newnode->next=NULL;
        if(start==NULL)
            start=newnode;
        else
        {
            ptr=start;
            while(ptr->next!=NULL)
                ptr=ptr->next;
            ptr->next=newnode;
```

```
        }
        printf("enter num;");
        scanf("%d",&num);

    }
    return start;

};

struct node *display(struct node *start)
{
    struct node *ptr;
    ptr=start;
    while(ptr->next!=NULL)
    {
        printf("\n%d",ptr->data);
         ptr=ptr->next;
    }
     printf("\n%d",ptr->data);
    return start;

};
```

```
struct node *insert_beg(struct node *start)
{
     struct node *newnode, *ptr;
     int num;
     printf("\nenter num");
    scanf("%d",&num);
    newnode=(struct node*)malloc(sizeof(struct node*));
    newnode->data=num;
    newnode->next=start;
    start=newnode;
    return start;
};
struct node *insert_end(struct node *start)
{
    struct node *newnode, *ptr;
     int num;
     printf("\nenter num:");
    scanf("%d",&num);
    newnode=(struct node*)malloc(sizeof(struct node*));
    newnode->data=num;
    newnode->next=NULL;
    ptr=start;
    while(ptr->next!=NULL)
```

```
        ptr=ptr->next;
    ptr->next=newnode;
    return start;
};
struct node *insert_pos(struct node *start)
{
   struct node *newnode, *ptr,*preptr;
     int num,pos,i=1;

     printf("\nenter num:");
    scanf("%d",&num);
    newnode=(struct node*)malloc(sizeof(struct node*));
    newnode->data=num;
    printf("\nenter the position:");
    scanf("%d",&pos);
     ptr=start;
    while(i!=pos)
    {
          preptr=ptr;
          ptr=ptr->next;
          i++;
    }
    preptr->next=newnode;
```

```
    newnode->next=ptr;
    return start;
};
struct node *delete_beg(struct node *start)
{
    struct node *ptr;
    ptr=start;
    start=start->next;
    free(ptr);
    return start;
};
struct node *delete_end(struct node *start)
{
  struct node *ptr, *preptr;
    ptr=start;
    while(ptr->next!=NULL)
    {
        preptr=ptr;
        ptr=ptr->next;
    }
    preptr->next=NULL;
    free(ptr);
    return start;
```

```
};
struct node *delete_node(struct node *start)
{
   struct node *ptr, *preptr;
   int num;
   printf("\nenter num:");
   scanf("%d",&num);
   ptr=start;
    while(ptr->data!=num)
   {
      preptr=ptr;
      ptr=ptr->next;
   }
   preptr->next=ptr->next;
   free(ptr);
   return start;
};
```

```
1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:1

enter -1 to end
enter num:1
enter num;3
enter num;6
enter num;8
enter num;-1


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

1
3
6
8

1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:3

enter num9


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

9
1
3
6
8
```

```
1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:4

enter num:2


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

9
1
3
6
8
2

1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:5

enter num:7

enter the position:2


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

9
7
1
3
6
8
2
```

```
1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:5

enter num:7

enter the position:2


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

9
7
1
3
6
8
2

1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:6


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

7
1
3
6
8
2

1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:7


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

7
1
3
6
8

1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:8

enter num:6


1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:2

7
1
3
8

1.create         2.Display       3.Insert beg   4.Insert end    5.Insert at pos         6. delete beg   7.delete end    8.delete node    9.exit
enter your choice:
```

Lab program 5:

6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;
```

```
   struct node *next;
};
struct node *start1=NULL;
struct node *start2=NULL;
struct node *create(struct node *start);
struct node *display(struct node * start);
struct node *sort(struct node *start);
struct node *reverse(struct node *start);
struct node *concat(struct node *start1,struct node *start2);

void main()
{
   int ch,n;
   while(1)
   {
      printf("\n\n1.create\t2.display\t3.sort\t4.reverse\t5.concat\t6.exit\n");
      printf("\nenter choice");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1: printf("\nlist 1");
               start1=create(start1);
               printf("\nlist 2");
```

```
                start2=create(start2);
                break;
        case 2: printf("\nlist 1");
                start1=display(start1);
                printf("\nlist 2");
               start2=display(start2);
               break;
        case 3: start1=sort(start1);
                start2=sort(start2);
                break;
        case 4: start1=reverse(start1);
                start2=reverse(start2);
                break;
        case 5 : start1=concat(start1,start2);
                   break;
        case 6: exit(0);
        default : printf("invalid");
      }
   }
}

struct node *create(struct node *start)
{
```

```
struct node *newnode, *ptr;
int num;
printf("\nenter -1 to end");
printf("\nenter num");
scanf("%d",&num);
while(num!=-1)
{
   newnode=(struct node*)malloc(sizeof(struct node*));
   newnode->data=num;
   newnode->next=NULL;
   if(start==NULL)
      start=newnode;
   else
   {
      ptr=start;
      while(ptr->next!=NULL)
         ptr=ptr->next;
      ptr->next=newnode;
   }
   printf("\nenter num");
   scanf("%d",&num);

}
```

```
    return start;

};

struct node *display(struct node *start)
{
     struct node *ptr;
     ptr=start;
     while(ptr->next!=NULL)
     {
         printf("\n%d",ptr->data);
          ptr=ptr->next;
     }
      printf("\n%d",ptr->data);
     return start;

};

struct node *sort(struct node *start)
{
    struct node *ptr1,*ptr2;
    ptr1=start;
    int temp;
```

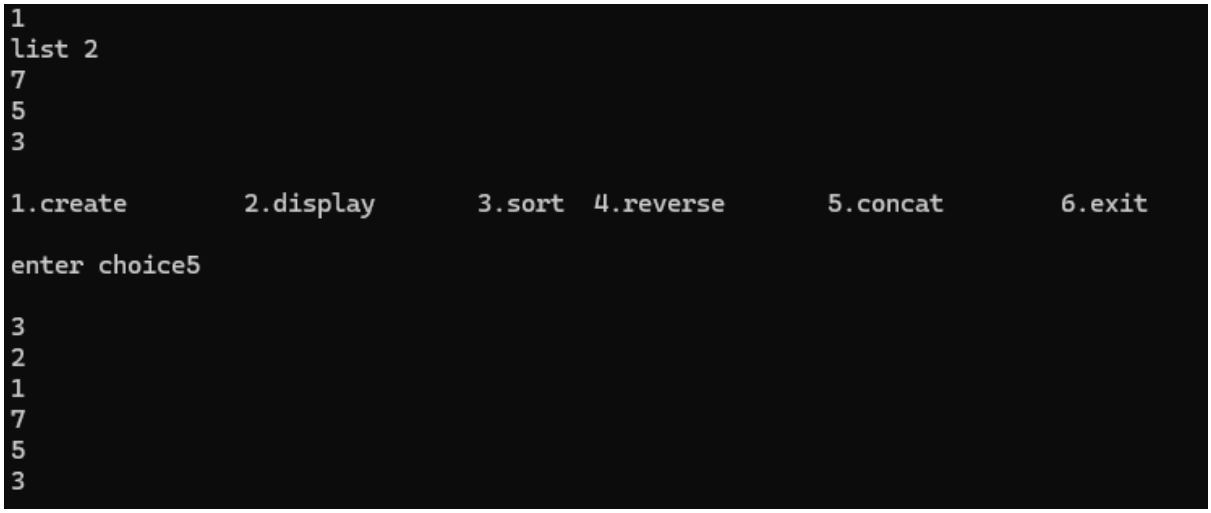
```
    while(ptr1->next!=NULL)
    {
        ptr2=ptr1->next;
        while(ptr2!=NULL)
        {
            if(ptr1->data>ptr2->data)
            {
                temp=ptr1->data;
                ptr1->data=ptr2->data;
                ptr2->data=temp;

            }
            ptr2=ptr2->next;
        }
        ptr1=ptr1->next;
    }
    return start;

}

struct node *reverse(struct node *start)
{
    struct node *prev=NULL;
```

```
    struct node *ptr=start;
    struct node *next=NULL;
    while(ptr!=NULL)
    {
       next=ptr->next;
       ptr->next=prev;
       prev=ptr;
       ptr=next;
    }
    start=prev;
    return start;
}
struct node *concat(struct node *start1, struct node *start2)
 {
     struct node *ptr;
     ptr=start1;
     if(start1==NULL)
     {
        start1=start2;
     }
     else
     {
         while(ptr->next!=NULL)
```

```
        {
            ptr=ptr->next;
        }
        ptr->next=start2;
    }
    start1=display(start1);
}
```

Output:

```
1.create        2.display       3.sort  4.reverse       5.concat        6.exit

enter choice1

list 1
enter -1 to end
enter num2

enter num3

enter num1

enter num-1

list 2
enter -1 to end
enter num5

enter num3

enter num7

enter num-1


1.create        2.display       3.sort  4.reverse       5.concat        6.exit

enter choice2

list 1
2
3
1
list 2
5
3
7

1.create        2.display       3.sort  4.reverse       5.concat        6.exit

enter choice3


1.create        2.display       3.sort  4.reverse       5.concat        6.exit

enter choice2

list 1
1
2
3
list 2
3
5
7

1.create        2.display       3.sort  4.reverse       5.concat        6.exit

enter choice4


1.create        2.display       3.sort  4.reverse       5.concat        6.exit

enter choice2

list 1
3
2
```



Lab program 6:

6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *top=NULL;
void push(int num)
{
 struct node *newnode;
 newnode=(struct node*)malloc(sizeof(struct node*));
 newnode->data=num;
 newnode->next=NULL;
```

```
 if(top==NULL)
 {
    newnode->next=NULL;
    top=newnode;

 }
 else
 {
    newnode->next=top;
    top=newnode;



 }
}

void pop()
{
   if(top==NULL)
   {
      printf("\nStack Underflow");
   }
   else
   {
      printf("deleted:%d", top->data);
      top=top->next;
   }
```

```
}

void display()
{
   if(top==NULL)
   {
      printf("\nStack Empty");
   }
   else
   {
      struct node *ptr;
      ptr=top;
      while(ptr!=NULL)
      {
         printf("%d\t",ptr->data);
         ptr=ptr->next;
      }
   }
}


void main()
{
   int n,c;
   while(1)
   {
```

```
    printf("\n\n1.push\t 2.pop\t 3.display\t 4.exit\n your choice:");
    scanf("%d",&c);
    switch(c)
    {
       case 1: printf("\nenter element:");
            scanf("%d",&n);
            push(n);
            break;
        case 2:pop();
        break;
        case 3: display();
        break;
        case 4: exit(0);
        default: printf("\ninvalid choice");
    }
  }
}
```

Output:

```
1.push   2.pop   3.display       4.exit
 your choice:1

enter element:5


1.push   2.pop   3.display       4.exit
 your choice:1

enter element:8


1.push   2.pop   3.display       4.exit
 your choice:2
deleted:8

1.push   2.pop   3.display       4.exit
 your choice:3
5

1.push   2.pop   3.display       4.exit
 your choice:1

enter element:8


1.push   2.pop   3.display       4.exit
 your choice:3
8       5
```

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;

   struct node *next;

};
```

```
struct node *front=NULL,*rear=NULL;

void insert(int num)
{
   struct node *newnode;
   newnode=(struct node*)malloc(sizeof(struct node*));
   newnode->data=num;
   newnode->next=NULL;
   if(front==NULL)
   {
      front=newnode;
      rear=newnode;
   }
   else
   {
      rear->next=newnode;
      rear=newnode;
   }
}

void Delete()
{
   if(front==NULL)
```

```
    {
        printf("queue Underflow");
    }
    else
    {
        printf("\ndeleted:%d",front->data);
        front=front->next;
    }
}

void display()
{
    if(front==NULL)
    {
        printf("\nqueue Empty");
    }
    else
    {
        struct node *ptr;
        ptr=front;
        while(ptr!=NULL)
        {
            printf("%d  ",ptr->data);
```

```
        ptr=ptr->next;
      }
   }

}

void main()
{
   int n,c;
   while(1)
   {
     printf("\n\n1.insert\t 2.delete\t 3.display\t 4.exit\n your choice:");
     scanf("%d",&c);
     switch(c)
     {
        case 1: printf("\nenter element:");
                scanf("%d",&n);
                insert(n);
                break;
         case 2:Delete();
         break;
         case 3: display();
         break;
```

```
        case 4: exit(0);
        default: printf("\ninvalid choice");
    }
  }
}
```

```
1.insert        2.delete        3.display       4.exit
 your choice:1

enter element:4


1.insert        2.delete        3.display       4.exit
 your choice:1

enter element:6


1.insert        2.delete        3.display       4.exit
 your choice:1

enter element:5


1.insert        2.delete        3.display       4.exit
 your choice:3
4  6  5

1.insert        2.delete        3.display       4.exit
 your choice:2

deleted:4

1.insert        2.delete        3.display       4.exit
 your choice:3
6  5
```

Lab program 7:

WAP to Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *prev, *next;
};
struct node *start=NULL;



struct node *create(struct node *start)
{
   struct node *ptr, *newnode;
   int num;
   printf("\nenter -1 to end\nenter num:");
   scanf("%d",&num);
   while(num!=-1)
   {
      newnode=(struct node*)malloc(sizeof(struct node*));
      newnode->data=num;
      if(start==NULL)
      {
       newnode->next=NULL;
       newnode->prev=NULL;
       start=newnode;
      }
```

```
        else
        {
           ptr=start;
           while(ptr->next!=NULL)
              ptr=ptr->next;
           newnode->prev=ptr;
           newnode->next=ptr->next;
           ptr->next=newnode;
        }
         printf("\nenter num:");
        scanf("%d",&num);
    }
    return start;
};

struct node *display(struct node *start)
{
    struct node *ptr;
    ptr=start;
    while(ptr!=NULL)
    {
        printf("%d ", ptr->data);
        ptr=ptr->next;
    }
    return start;
};
```

```
struct node *insert_before(struct node *start)
{
   struct node *newnode, *ptr;
   int num, val;
   printf("\nenter num:");
   scanf("%d", &num);
   printf("\n enter the val before you want to insert:");
   scanf("%d",&val);
  newnode=(struct node*)malloc(sizeof(struct node*));
   newnode->data=num;
   if(start->data==val)
   {
      ptr=start;
      newnode->next=start;
      newnode->prev=NULL;
      start->prev=newnode;
      start=newnode;
   }
   else
   {
      ptr=start;
      while(ptr->data!=val)
      {
         ptr=ptr->next;
      }
```

```
        newnode->next=ptr;
        newnode->prev=ptr->prev;
        ptr->prev->next=newnode;
        ptr->prev=newnode;
    }
    return start;
};

struct node *delete_node(struct node *start)
{
    int val;
    struct node *ptr;
    printf("\n enter the val:");
    scanf("%d", &val);
    ptr=start;
    if(ptr->data==val){

        start=start->next;
        start->prev=NULL;
        free(ptr);
    }
    else{

        while(ptr->data!=val)
            ptr=ptr->next;
        if(ptr->next==NULL)
```

```
        {
          ptr->prev->next=NULL;
           free(ptr);
        }
        else{
              ptr->next->prev=ptr->prev;
              ptr->prev->next=ptr->next;
              free(ptr);
           }
    }
    return start;
};

void main()
{
    int ch;
    while(1)
    {
        printf("\n\n1.create \t 2.Display \t 3.Insert_before \t4.delete_node\t5.exit");
        printf("\nenter your choice");
        scanf("%d",&ch);
        struct node *newnode;
        switch(ch)
        {
            case 1: start=create(start);
                    break;
```

```
        case 2: start=display(start);
        break;
        case 3 : start=insert_before(start);
        break;
        case 4: start=delete_node(start);
        break;
        case 5:  exit(0);
        default:printf("invalid");
      }

   }
}
```

Output:

```
1.create         2.Display       3.Insert_before        4.delete_node   5.exit
enter your choice1

enter -1 to end
enter num:3

enter num:6

enter num:7

enter num:-1


1.create         2.Display       3.Insert_before        4.delete_node   5.exit
enter your choice2
3 6 7

1.create         2.Display       3.Insert_before        4.delete_node   5.exit
enter your choice3

enter num:3

 enter the val before you want to insert:7


1.create         2.Display       3.Insert_before        4.delete_node   5.exit
enter your choice2
3 6 3 7

1.create         2.Display       3.Insert_before        4.delete_node   5.exit
enter your choice4

 enter the val:3

1.create         2.Display       3.Insert_before        4.delete_node   5.exit
enter your choice2
6 3 7
```

Lab program 8:

Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.

```
#include<stdio.h>

#include<stdlib.h>

typedef struct bst

{

   int data;

   struct bst *left, *right;
```

```
}node;

node *create()
{
   node *temp;
   temp=(node *)malloc(sizeof(node*));
   printf("enter data");
   scanf("%d", &temp->data);
   temp->left=NULL;
   temp->right=NULL;
   return temp;
}

node *insert(node *root, node *temp)
{
   if(temp->data<root->data)
   {
     if(root->left!=NULL)
     insert(root->left, temp);
     else
       root->left=temp;

   }
   if(temp->data>root->data)
   {
      if(root->right!=NULL)
```

```
            insert(root->right, temp);
        else
            root->right=temp;
    }
}




void inorder( node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d  ", root->data);
        inorder(root->right);
    }

}

void preorder( node *root)
{
    if(root!=NULL)
    {
        printf("%d  ", root->data);
        preorder(root->left);
        preorder(root->right);
```

```
    }

}

void postorder( node *root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d  ", root->data);
    }

}


void main()
{

    char ch;
    node *root=NULL, *temp;
    do
    {
        temp=create();
        if(root==NULL)
            root=temp;
```

```
        else
            insert(root, temp);
        printf("\nyou want to enter more number(y/n)");
        getchar();
        scanf("%c",&ch);
    }while(ch=='y'||ch=='Y');
    printf("\n Preorder:");
    preorder(root);
     printf("\n Inorder:");
    inorder(root);
     printf("\n Postorder:");
    postorder(root);

}
```

Output:

```
enter data6

you want to enter more number(y/n)y
enter data4

you want to enter more number(y/n)y
enter data9

you want to enter more number(y/n)y
enter data3

you want to enter more number(y/n)n

 Preorder:6  4  3  9
 Inorder:3  4  6  9
 Postorder:3  4  9  6
Process returned 3 (0x3)   execution time : 29.410 s
Press any key to continue.
```

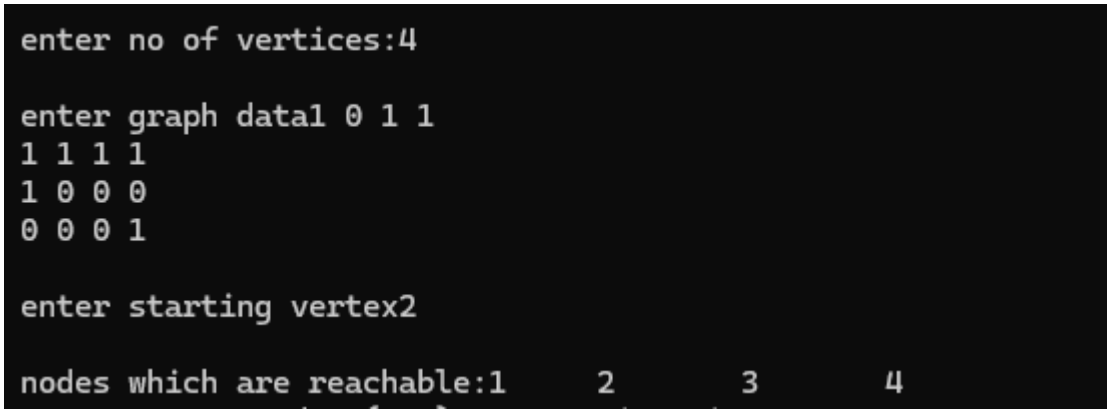
Lab program 9:

9a) Write a program to traverse a graph using BFS method.

```
#include<stdio.h>
#include<stdlib.h>
int a[20][20],q[20], visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
   for(i=1;i<=n;i++)
   {
      if(a[v][i]&& !visited[i])
         q[++r]=i;
   }
   if(f<=r)
   {
      visited[q[f]]=1;
      bfs(q[f++]);
   }
}

void main()
{
   int v;
   printf("\nenter no of vertices:");
```

```
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        q[i]=0;
        visited[i]=0;
    }
    printf("\nenter graph data");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
            scanf("%d", &a[i][j]);
    }
    printf("\nenter starting vertex");
    scanf("%d",&v);
    bfs(v);
    printf("\nnodes which are reachable:");
    for(i=1;i<=n;i++)
    {
        if(visited[i])
            printf("%d\t", i);

    }
}
```

Output:



9b) Write a program to check whether given graph is connected or not using DFS method.

```
#include<stdio.h>
#include<stdlib.h>
int a[20][20],s[20],n;
void dfs(int v)//v=vertex
{
   s[v]=1;
   int i;
   for(i=1;i<=n;i++)
   {
   if(a[v][i]&&!s[i])
   {
      printf("\n%d->%d",v,i);
      dfs(i);
   }
   }
}
```

```
void main()
{
   int i,j,count=0;
   printf("\n enter the no. of vertices:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      s[i]=0;
      for(j=1;j<=n;j++)
         a[i][j]=0;
   }

   printf("\n Enter the Adjacent matrix:\n");
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
         scanf("%d",&a[i][j]);
   }
   dfs(1);
   printf("\n");

   for(int i=1;i<=n;i++)
   {
      if(s[i])
         count++;
```

```
    }
    if(count==n)
        printf("Graph is Connected");
    else
        printf("Graph is disconnected");

}
```

Output:

```
 enter the no. of vertices:4

 Enter the Adjacent matrix:
0 0 0 1
1 1 0 1
1 0 1 0
1 0 0 1

1->4
Graph is disconnected
Process returned 21 (0x15)   execution time : 28.764 s
Press any key to continue.
```

Lab program 10:

Given a File of N employee records with a set K of Keys(4 digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers.

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

#define TABLE_SIZE 100 // Size of the hash table (m memory locations)
```

```
// Structure to represent an employee record
typedef struct {
   int key;    // 4-digit key
   char name[30];
   int age;
} Employee;

// Hash table array
Employee* hashTable[TABLE_SIZE];

// Initialize the hash table
void initializeHashTable() {
   for (int i = 0; i < TABLE_SIZE; i++) {
      hashTable[i] = NULL;
   }
}

// Hash function to compute memory address
int hashFunction(int key) {
   return key % TABLE_SIZE;
}

// Insert an employee record into the hash table
void insert(Employee* emp) {
   int index = hashFunction(emp->key);
```

```
    // Linear probing to handle collisions
    while (hashTable[index] != NULL) {
        index = (index + 1) % TABLE_SIZE; // Go to the next index
    }

    hashTable[index] = emp;
}

// Search for an employee record by key
Employee* search(int key) {
    int index = hashFunction(key);

    // Search using linear probing
    while (hashTable[index] != NULL) {
        if (hashTable[index]->key == key) {
            return hashTable[index];
        }
        index = (index + 1) % TABLE_SIZE;
    }

    return NULL; // Record not found
}

// Display the hash table
void displayHashTable() {
```

```
    printf("\nHash Table:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != NULL) {
            printf("Index %d: Key: %d, Name: %s, Age: %d\n", i, hashTable[i]->key, hashTable[i]->name, hashTable[i]->age);
        }
    }
}

// Main function
int main() {
    initializeHashTable();

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

    for (int i = 0; i < n; i++) {
        Employee* emp = (Employee*)malloc(sizeof(Employee));

        printf("Enter key (4-digit), name, and age for employee %d: ", i + 1);
        scanf("%d %s %d", &emp->key, emp->name, &emp->age);

        insert(emp);
    }
```

```
    displayHashTable();

    int searchKey;
    printf("\nEnter a key to search for: ");
    scanf("%d", &searchKey);

    Employee* result = search(searchKey);
    if (result != NULL) {
        printf("Employee found: Key: %d, Name: %s, Age: %d\n", result->key, result->name,
result->age);
    } else {
        printf("Employee with key %d not found.\n", searchKey);
    }

    return 0;
}
```

Output:

```
Enter the number of employees: 3
Enter key (4-digit), name, and age for employee 1: 1234 rashi 19
Enter key (4-digit), name, and age for employee 2: 5678
sneha 13
Enter key (4-digit), name, and age for employee 3: 9012 harsh 34

Hash Table:
Index 12: Key: 9012, Name: harsh, Age: 34
Index 34: Key: 1234, Name: rashi, Age: 19
Index 78: Key: 5678, Name: sneha, Age: 13

Enter a key to search for: 1234
Employee found: Key: 1234, Name: rashi, Age: 19
```

Leetcode programs:

1. **Game of two stacks**

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'twoStacks' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER maxSum
 *  2. INTEGER_ARRAY a
 *  3. INTEGER_ARRAY b
 */
```

```
int twoStacks(int maxSum, int a_count, int* a, int b_count, int*
b) {
int count1=0,sum=0,j=0,count2=0;

  while(count1<a_count && sum+a[count1]<=maxSum)
  {
     sum+=a[count1];
     count1++;
  }
count2=count1;
while(j<b_count && sum+b[j]<=maxSum)
{
   sum+=b[j];
   j++;
}
count2+=j;

while(j<b_count)
{
   sum+=b[j++];
   while(count1>0&& sum>maxSum)
   {
      count1--;
      sum-=a[count1];
   }
   if(sum<=maxSum && count1+j> count2)
   {
      count2=count1+j;
   }

}
```

```
return count2;

}


int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int g = parse_int(ltrim(rtrim(readline())));

    for (int g_itr = 0; g_itr < g; g_itr++) {
        char** first_multiple_input = split_string(rtrim(readline()));

        int n = parse_int(*(first_multiple_input + 0));

        int m = parse_int(*(first_multiple_input + 1));

        int maxSum = parse_int(*(first_multiple_input + 2));

        char** a_temp = split_string(rtrim(readline()));

        int* a = malloc(n * sizeof(int));

        for (int i = 0; i < n; i++) {
            int a_item = parse_int(*(a_temp + i));

            *(a + i) = a_item;
        }

        char** b_temp = split_string(rtrim(readline()));
```

```
        int* b = malloc(m * sizeof(int));

        for (int i = 0; i < m; i++) {
            int b_item = parse_int(*(b_temp + i));

            *(b + i) = b_item;
        }

        int result = twoStacks(maxSum, n, a, m, b);

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
```

```
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1]
== '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);
```

```
        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }
```

```
    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
```

```
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

2. Finding majority element

```
int majorityElement(int* nums, int numsSize) {
  int el=nums[0];
 int count=0;
 for(int i=0;i<numsSize;i++)
 {
    if(nums[i]==el)
    count++;
    else
    {
        count--;
        if(count<0){
        el=nums[i];
        count=0;
        }

    }
}

return el;
}
```

3.move zeros

```
void moveZeroes(int* nums, int numsSize){
   int j=0;
      for(int i=0;i<numsSize;i++){
         if(nums[i]!=0){
            int temp=nums[i];
            nums[i]=nums[j];
            nums[j]=temp;
            j++;
         }
   }
}
```

4.palindrome linked list

```
bool isPalindrome( struct ListNode* head) {
   struct ListNode * top=NULL;
  struct ListNode * ptr=head;

    struct ListNode *temp=NULL;

  while(ptr!=NULL)
  {       temp=(struct ListNode*)malloc(sizeof(struct ListNode ));
         temp->val=ptr->val;
         temp->next=top;
         top=temp;

      ptr=ptr->next;
  }

  ptr=head;
  int flag=1;

  while(ptr!=NULL && top!=NULL)

  {

    if(ptr->val!=top->val)
    {
         flag=0;
         break;
```

```
    }

    ptr=ptr->next;
    temp=top;
     top=top->next;
    free(temp);

  }
  if (flag==0)
  return false;
  else
  return true;
  }
```

5.Finding tree path sum

```
bool hasPathSum(struct TreeNode* root, int targetSum) {
    if ( root == NULL ) return false;

   if ( root->left == NULL && root->right == NULL && root->val==targetSum)
      return true;

   if ( hasPathSum(root->left, targetSum - root->val) )
      return true;

   if ( hasPathSum(root->right, targetSum - root->val) )
      return true;

   return false;

}
```