VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

NAVEEN RAMKUMAR(1BM22CS173)

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
Dec 2023- March 2024

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 NAVEEN RAMKUMAR(1BM22CS173), who is a 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 2023-24. 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. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
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 SIZE 30

int stack[SIZE];

int top=-1;

void push()

{

   if(top==SIZE-1)

      printf("Stack overflow.");

   else

   {

      top=top+1;

      printf("Enter the element that you would like to insert into the stack ");

      scanf("%d",&stack[top]);

      printf("The element inserted successfully.");

   }

}

void pop()

{

   if(top==-1)

      printf("Stack underflow.");

   else

   {
```

```
        printf("The element %d is popped.",stack[top]);
        top=top-1;
    }
}
void display()
{
    int run;
    if(top==-1)
        printf("Stack underflow.");
    else
    {
        printf("-----\n");
        for(run=top;run>=0;run--)
            printf("%d\n",stack[run]);
        printf("-----");
    }
}
void main()
{
    while(1)
    {
        int ch;
        printf("Enter 1 to push, 2 to pop, 3 to display the elements and 4 to exit ");
        scanf("%d",&ch);
        printf("\n");
        if(ch==1)
            push();
```

```
        else if(ch==2)
            pop();
        else if(ch==3)
            display();
        else if(ch==4)
            break;
        else
            printf("Invalid input.");
        printf("\n\n");
    }
}
```

Output:

```
The element inserted successfully.

Enter 1 to push, 2 to pop, 3 to display the elements and 4 to exit 1

Enter the element that you would like to insert into the stack 45
The element inserted successfully.

Enter 1 to push, 2 to pop, 3 to display the elements and 4 to exit 3

-----
45
23
-----

Enter 1 to push, 2 to pop, 3 to display the elements and 4 to exit 2

The element 45 is popped.

Enter 1 to push, 2 to pop, 3 to display the elements and 4 to exit 3

-----
23
-----

Enter 1 to push, 2 to pop, 3 to display the elements and 4 to exit 4


Process returned 4 (0x4)   execution time : 66.879 s
Press any key to continue.
```

Lab program 2:

Write a program 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), / (divide) and ^ (power).

```
#include <stdio.h>
#include <string.h>
#define SIZE 30
char infix[SIZE];
char postfix[SIZE];
char stack[15];
int top=-1;
int toppostfix=-1;
int isbracket=0;
void push(char c)
{
   if(top==SIZE-1)
      printf("Stack overflow.");
   else
   {
      top=top+1;
      stack[top]=c;
   }
}
char pop()
{
   if(top==-1)
      printf("Stack underflow.");
```

```
    else
    {
        top=top-1;
        return(stack[top+1]);
    }
}
int precedence(char c)
{
    if(c=='^')
        return(5);
    else if(c=='/')
        return(4);
    else if(c=='*')
        return(3);
    else if(c=='+')
        return(2);
    else if(c=='-')
        return(1);
}
void main()
{
    int i;
    printf("Enter the infix expression. ");
    scanf("%s",infix);
    for(i=0;i<strlen(infix);i++)
    {
        if((infix[i]>=65 && infix[i]<=90))
```

```
{
   toppostfix=toppostfix+1;
   postfix[toppostfix]=infix[i];
}
else if(infix[i]=='(')
{
   isbracket=1;
   push(infix[i]);
}
else if(top!=-1)
{
   if(precedence(infix[i])<precedence(stack[top]) || infix[i]==')')
   {
      while((isbracket==0 && top!=-1) || (isbracket==1 && stack[top]!='('))
      {
         toppostfix=toppostfix+1;
         postfix[toppostfix]=pop();
      }
      if(infix[i]==')')
      {
         pop();
         isbracket=0;
      }
      else
         push(infix[i]);
   }
   else
```

```
            {
                push(infix[i]);
            }
        }
        else
            push(infix[i]);
    }
    while(top!=-1)
    {
        toppostfix=toppostfix+1;
        postfix[toppostfix]=pop();
    }
    postfix[toppostfix+1]='\0';
    printf("\n\nThe postfix expression is %s",postfix);
}
```

Output:

```
Enter the infix expression. A*B+C*D-E


The postfix expression is AB*CD*+E-
Process returned 37 (0x25)   execution time : 29.555 s
Press any key to continue.
```

Lab program 3(a):

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>
#define MAX 30
int queue[MAX];
int front=-1;
int rear=-1;
void insert(int element)
{
   if(rear==MAX-1)
      printf("Queue overflow");
   else
   {
      rear=rear+1;
      if(front==-1)
         front=front+1;
      queue[rear]=element;
   }
}
void delete()
{
   if(front>rear || (front==-1 && rear==-1))
      printf("Queue underflow");
   else
   {
```

```
        printf("The element popped is %d",queue[front]);
        front=front+1;
    }
}
void display()
{
    int i;
    if(front>rear || (front==-1 && rear==-1))
        printf("Queue underflow");
    else
    {
        for(i=front;i<=rear;i++)
            printf("%d ",queue[i]);
    }
}
void main()
{
    while(1)
    {
        int ch,element;
        printf("Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit ");
        scanf("%d",&ch);
        if(ch==1)
        {
            printf("Enter the element to insert into the queue ");
            scanf("%d",&element);
```

```
            insert(element);
        }
        else if(ch==2)
            delete();
        else if(ch==3)
            display();
        else if(ch==4)
            break;
        else
            printf("Invalid input");
        printf("\n\n");
    }

}
```

Output:

```
Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 1


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 2


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 3


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
1 2 3

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 2
The element popped is 1

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
2 3

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 4

Process returned 4 (0x4)   execution time : 82.458 s
Press any key to continue.
```

Lab program 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>

#define SIZE 5

int queue[SIZE];

int front=-1;

int rear=-1;

void insert(int element)

{

   if((rear+1)%SIZE==front)

      printf("Queue overflow");

   else

   {

      rear=(rear+1)%SIZE;

      if(front==-1)

         front=front+1;

      queue[rear]=element;

   }

}

void delete()

{

   if(front==-1 && rear==-1)

      printf("Queue underflow");

   else

   {

      printf("The element popped is %d",queue[front]);
```

```
        if(front==rear)
            front=rear=-1;
        else
            front=(front+1)%SIZE;
    }
}
void display()
{
    int i;
    if(front==-1 && rear==-1)
        printf("Queue underflow");
    else
    {
        i=front;
        while(1)
        {
            printf("%d ",queue[i]);
            if(i==rear)
                break;
            i=(i+1)%SIZE;
        }
    }
}
void main()
{
    while(1)
    {
```

```
    int ch,element;
    printf("Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit ");
    scanf("%d",&ch);
    if(ch==1)
    {
       printf("Enter the element to insert into the queue ");
       scanf("%d",&element);
       insert(element);
    }
    else if(ch==2)
       delete();
    else if(ch==3)
       display();
    else if(ch==4)
       break;
    else
       printf("Invalid input");
    printf("\n\n");
  }
}
```

Output:

```
Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 0


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 1


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 2


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 3


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 4


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
0 1 2 3 4

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 2
The element popped is 0

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 2
The element popped is 1

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
2 3 4

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 100


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 1
Enter the element to insert into the queue 200


Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 3
2 3 4 100 200

Enter 1 to insert elements into the queue, 2 to delete from the queue, 3 to display and 4 to exit 4
```

Lab program 4(a)

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.

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head=NULL;
void push()
{
   struct Node *new_node=malloc(sizeof(struct Node));
   int data;
   printf("Enter the data to be entered ");
   scanf("%d",&data);
   (*new_node).data=data;
   (*new_node).next=head;
   head=new_node;

}
void append()
{
   struct Node *new_node=malloc(sizeof(struct Node));
```

```
    int data;
    struct Node *last=head;
    printf("Enter the data to be entered ");
    scanf("%d",&data);
    (*new_node).data=data;
    (*new_node).next=NULL;
    if(head==NULL)
       head=new_node;
    else
    {
       while((*last).next!=NULL)
       {
          last=(*last).next;
       }

       (*last).next=new_node;
    }
}
void insert_at_pos(int pos)
{
    struct Node *new_node=malloc(sizeof(struct Node));
    struct Node *temp=head;
    int data;
    printf("Enter the data to be entered ");
    scanf("%d",&data);
    (*new_node).data=data;
    if(pos==1)
```

```
    {
        (*new_node).next=head;
        head=new_node;
        return;
    }
    int position=1;
    while(1)
    {
        if(position==pos-1)
            break;
        else
        {
            temp=(*temp).next;
            position=position+1;
        }
    }
    (*new_node).next=(*temp).next;
    (*temp).next=new_node;
}
void display()
{
    struct Node *node=head;
    while(1)
    {
        printf("%d ",(*node).data);
        if((*node).next==NULL)
            break;
```

```
        node=(*node).next;
    }
}
void main()
{
    int choice;
    while(1)
    {
        printf("Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. ");
        scanf("%d",&choice);
        if(choice==1)
            push();
        else if(choice==2)
            append();
        else if(choice==3)
        {
            int position;
            printf("Enter the position to insert the node. ");
            scanf("%d",&position);
            insert_at_pos(position);
        }
        else if(choice==4)
            display();
        else if(choice==5)
            break;
        else
```

```
            printf("Invalid input entered.");
        printf("\n\n");
    }
}
```

Output:

```
Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 1
Enter the data to be entered 23


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 1
Enter the data to be entered 45


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 2
Enter the data to be entered 77


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 4
45 23 77

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 3
Enter the position to insert the node. 3
Enter the data to be entered 100


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 4
45 23 100 77

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to display the contents and 5 to exit. 5

Process returned 5 (0x5)   execution time : 114.974 s
Press any key to continue.
```

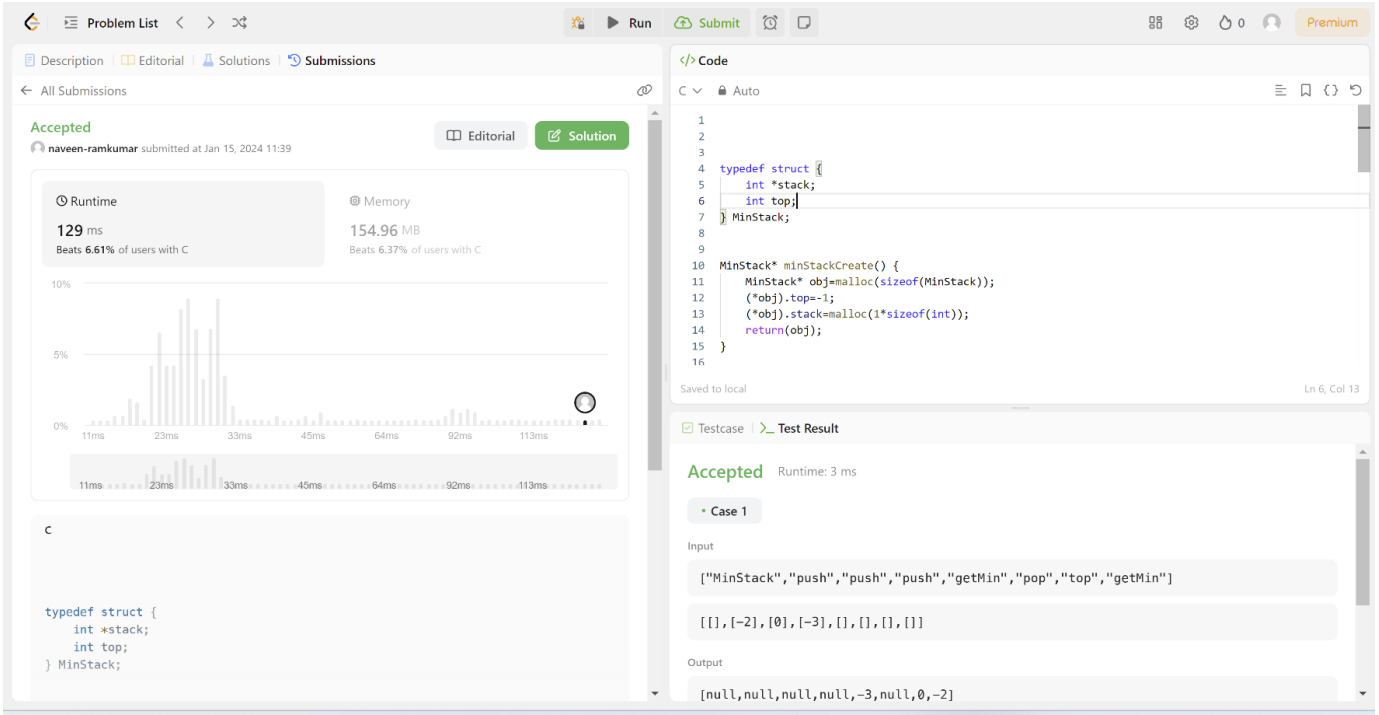
Lab program 4(b)

Demonstration of account creation on LeetCode platform and stack program

```
typedef struct {
    int *stack;
    int top;
} MinStack;


MinStack* minStackCreate() {
    MinStack* obj=malloc(sizeof(MinStack));
    (*obj).top=-1;
    (*obj).stack=malloc(1*sizeof(int));
    return(obj);
}

void minStackPush(MinStack* obj, int val) {
    (*obj).stack=realloc((*obj).stack,(++(*obj).top+1)*sizeof(int));
    *((*obj).stack+(*obj).top)=val;
```

```
}

void minStackPop(MinStack* obj) {
    (*obj).top=(*obj).top-1;
}

int minStackTop(MinStack* obj) {
    return(*((*obj).stack+(*obj).top));
}

int minStackGetMin(MinStack* obj) {
    int i;
    int min=*((*obj).stack+0);
    for(i=1;i<=(*obj).top;i++)
    {
        if(*((*obj).stack+i)<min)
            min=*((*obj).stack+i);
    }
    return(min);
}

void minStackFree(MinStack* obj) {
    free(obj);
}
```

Output:



Lab program 5(a)

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head=NULL;
void push()
{
   struct Node *new_node=malloc(sizeof(struct Node));
   int data;
   printf("Enter the data to be entered ");
   scanf("%d",&data);
   (*new_node).data=data;
   (*new_node).next=head;
   head=new_node;

}
void append()
{
   struct Node *new_node=malloc(sizeof(struct Node));
```

```
    int data;
    struct Node *last=head;
    printf("Enter the data to be entered ");
    scanf("%d",&data);
    (*new_node).data=data;
    (*new_node).next=NULL;
    if(head==NULL)
       head=new_node;
    else
    {
       while((*last).next!=NULL)
       {
          last=(*last).next;
       }

       (*last).next=new_node;
    }
}
void insert_at_pos(int pos)
{
    struct Node *new_node=malloc(sizeof(struct Node));
    struct Node *temp=head;
    int data;
     printf("Enter the data to be entered ");
    scanf("%d",&data);
    (*new_node).data=data;
    if(pos==1)
```

```
    {
        (*new_node).next=head;
        head=new_node;
        return;
    }
    int position=1;
    while(1)
    {
        if(position==pos-1)
            break;
        else
        {
            temp=(*temp).next;
            position=position+1;
        }
    }
    (*new_node).next=(*temp).next;
    (*temp).next=new_node;
}
void Pop()
{
    if(head==NULL)
        printf("The linked list is empty. You cannot delete from an empty list.");
    else
    {
        struct Node *ptr=head;
        head=(*ptr).next;
```

```
        free(ptr);
    }
}
void End_delete()
{
    if(head==NULL)
        printf("The linked list is empty. You cannot delete from an empty list.");
    else if((*head).next==NULL)
    {
        free(head);
        head=NULL;
    }
    else
    {
        struct Node *ptr1=head;
        struct Node *ptr=(*ptr1).next;
        while((*ptr).next!=NULL)
        {
            ptr1=(*ptr1).next;
            ptr=(*ptr1).next;
        }
        (*ptr1).next=NULL;
        free(ptr);
    }
}
void Delete_at_pos(int pos)
{
```

```
if(head==NULL)
    printf("The linked list is empty. You cannot delete from an empty list.");
else if(pos==1)
{
    struct Node *ptr1=(*head).next;
    free(head);
    head=ptr1;
}
else
{
    int position=2;
    struct Node *ptr1=head;
    struct Node *ptr=(*ptr1).next;
    while(1)
    {
        if(ptr==NULL)
        {
            printf("There are less than required elements in the list.");
            return;
        }
        if(position==pos)
        {
            (*ptr1).next=(*ptr).next;
            free(ptr);
            break;
        }
        position=position+1;
```

```
            ptr1=(*ptr1).next;
            ptr=(*ptr1).next;
        }
    }
}
void display()
{
    struct Node *node=head;
    while(node!=NULL)
    {
        printf("%d ",(*node).data);
        node=(*node).next;
    }
}
void main()
{
    int choice;
    while(1)
    {
        printf("Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from  the middle, 7 to display the contents, and 8 to exit. ");
        scanf("%d",&choice);
        if(choice==1)
            push();
        else if(choice==2)
            append();
        else if(choice==3)
```

```
        {
            int position;
            printf("Enter the position to insert the node. ");
            scanf("%d",&position);
            insert_at_pos(position);
        }
        else if(choice==4)
            Pop();
        else if(choice==5)
            End_delete();
        else if(choice==6)
        {
            int position;
            printf("Enter the position to delete from the list. ");
            scanf("%d", &position);
            Delete_at_pos(position);
        }
        else if(choice==7)
            display();
        else if(choice==8)
            break;
        else
            printf("Invalid input entered.");
        printf("\n\n");
    }
}
```

Output:

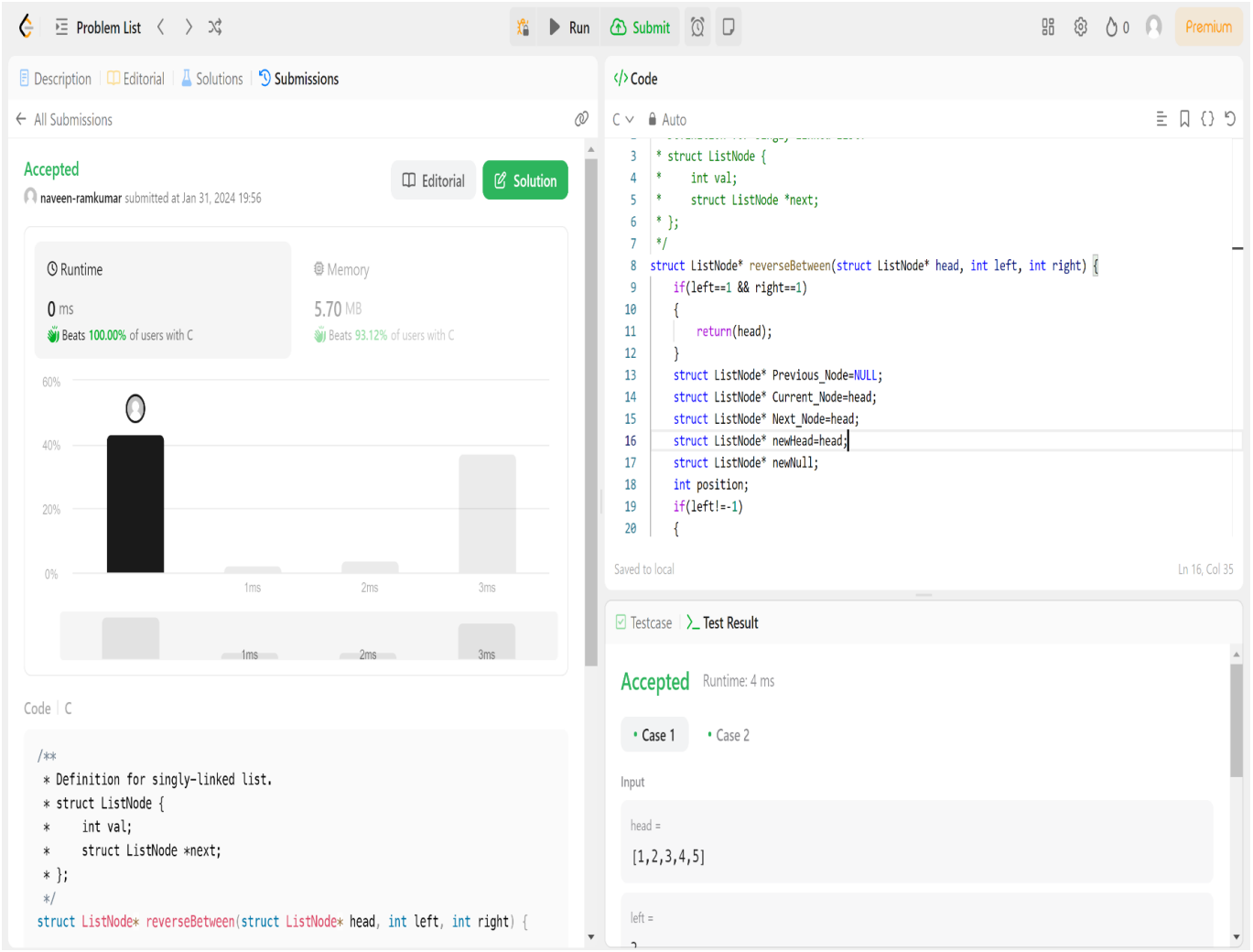
```
Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 7
100 200 300 400 500

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 4


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 7
200 300 400 500

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 5


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 7
200 300 400

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 6
Enter the position to delete from the list. 4
There are less than required elements in the list.

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 6
Enter the position to delete from the list. 2


Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 7
200 400

Enter 1 to insert at the beginning, 2 to append at the end, 3 to insert in the middle, 4 to delete from the beginning, 5 to delete from the end, 6 to delete from the middle, 7 to display the contents, and 8 to exit. 8

Process returned 8 (0x8)   execution time : 133.663 s
Press any key to continue.
```

Lab program 5(b)

Demonstration of LeetCode program on Singly linked list.

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if(left==1 && right==1)
    {
        return(head);
    }
    struct ListNode* Previous_Node=NULL;
    struct ListNode* Current_Node=head;
    struct ListNode* Next_Node=head;
    struct ListNode* newHead=head;
    struct ListNode* newNull;
    int position;
    if(left!=-1)
    {
        for(position=1; position<left; position++)
        {
            if(position==left-1)
            {
                newHead=Current_Node;
            }
            Current_Node=(*Current_Node).next;
        }
    }
    Previous_Node=Current_Node;
    newNull=Current_Node;
    Current_Node=(*Current_Node).next;
    for(position=left+1; position<=right; position++)
    {
        Next_Node=(*Current_Node).next;
        (*Current_Node).next=Previous_Node;
        Previous_Node=Current_Node;
        if(position==right)
        {
            if(left==1)
            {
                head=Current_Node;
            }
            else
                (*newHead).next=Current_Node;
        }
        Current_Node=Next_Node;
    }
    (*newNull).next=Current_Node;
    return(head);
}
```

Output:


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
All Submissions
Accepted
naveen-ramkumar submitted at Jan 31, 2024 19:56
Editorial
Solution
Runtime
0 ms
Beats 100.00% of users with C
Memory
5.70 MB
Beats 93.12% of users with C
60%
40%
20%
0%
1ms
2ms
3ms
Code | C
/**
* Definition for singly-linked list.
* struct ListNode {
*   int val;
*   struct ListNode *next;
* };
*/
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
Code
C
Auto
* struct ListNode {
*   int val;
*   struct ListNode *next;
* };
*/
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
if(left==1 && right==1)
{
return(head);
}
struct ListNode* Previous_Node=NULL;
struct ListNode* Current_Node=head;
struct ListNode* Next_Node=head;
struct ListNode* newHead=head;
struct ListNode* newNull;
int position;
if(left!=-1)
{
Saved to local
Ln 16, Col 35
Testcase
Test Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Input
head =
[1,2,3,4,5]
left =

Lab program 6(a)

WAP to Implement Single Linked 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* head=NULL;
struct Node* head2=NULL;
void sort(struct Node* head)
{
   struct Node* i;
   struct Node* j;
   int temp;
   printf("The linked list before sorting is:\n");
   display(head);
   for(i=head; (*i).next!=NULL; i=(*i).next)
   {
      for(j=(*i).next; (*j).next!=NULL; j=(*j).next)
      {
         if((*j).data<(*i).data)
         {
            temp=(*i).data;
            (*i).data=(*j).data;
```

```
            (*j).data=temp;
        }
    }
    printf("\nThe linked list after sorting is:\n");
    display(head);
}
void reverse(struct Node* head)
{
    struct Node* previous_Node=NULL;
    struct Node* current_Node=head;
    struct Node* next_Node;
    printf("The linked list before reversing is:\n");
    display(head);
    while(current_Node!=NULL)
    {
        next_Node=(*current_Node).next;
        if(next_Node==NULL)
        {
            head=current_Node;
        }
        (*current_Node).next=previous_Node;
        previous_Node=current_Node;
        current_Node=next_Node;
    }
    printf("\nThe linked list after reversing is:\n");
    display(head);
```

```
}
void concatenate(struct Node* head1, struct Node* head2)
{
   printf("The linked list 1 is:\n");
   display(head);
   printf("\nThe linked list 2 is:\n");
   display(head2);
   struct Node* last;
   for(last=head; (*last).next!=NULL; last=(*last).next);
   (*last).next=head2;
   printf("\nThe linked list 1 after concatenation is:\n");
   display(head);
}
void display(struct Node* head)
{
   struct Node* temp;
   for(temp=head; temp!=NULL; temp=(*temp).next)
   {
      printf("%d ", (*temp).data);
   }
}
void main()
{
   struct Node* New_Node;
   int position;
   int data;
   int choice;
```

```
while(1)
{
    head=NULL;
    head2=NULL;
    printf("List 1\n");
    for(position=1; position<=5; position++)
    {
        printf("Enter the data that you wish to enter for position %d. ", 6-position);
        scanf("%d",&data);
        struct Node* New_Node=malloc(sizeof(struct Node));
        (*New_Node).data=data;
        (*New_Node).next=head;
        head=New_Node;
    }
    printf("Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. ");
    scanf("%d", &choice);
    if(choice==1)
        sort(head);
    else if(choice==2)
        reverse(head);
    else if(choice==3)
    {
        printf("List 2\n");
        for(position=1; position<=5; position++)
        {
            printf("Enter the data that you wish to enter for position %d. ", 6-position);
```

```
            scanf("%d",&data);
            struct Node* New_Node=malloc(sizeof(struct Node));
            (*New_Node).data=data;
            (*New_Node).next=head2;
            head2=New_Node;
        }
        concatenate(head, head2);
    }
    else if(choice==4)
        break;
    else
        printf("Invalid input character.");
    printf("\n\n");
  }
}
```

Output:

```
List 1
Enter the data that you wish to enter for position 5. 9
Enter the data that you wish to enter for position 4. 3
Enter the data that you wish to enter for position 3. 1
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 5
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 1
The linked list before sorting is:
5 2 1 3 9
The linked list after sorting is:
1 2 3 5 9

List 1
Enter the data that you wish to enter for position 5. 9
Enter the data that you wish to enter for position 4. 3
Enter the data that you wish to enter for position 3. 1
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 5
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 2
The linked list before reversing is:
5 2 1 3 9
The linked list after reversing is:
9 3 1 2 5

List 1
Enter the data that you wish to enter for position 5. 9
Enter the data that you wish to enter for position 4. 3
Enter the data that you wish to enter for position 3. 1
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 5
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 3
List 2
Enter the data that you wish to enter for position 5. 7
Enter the data that you wish to enter for position 4. 6
Enter the data that you wish to enter for position 3. 3
Enter the data that you wish to enter for position 2. 1
Enter the data that you wish to enter for position 1. 9
The linked list 1 is:
5 2 1 3 9
The linked list 2 is:
9 1 3 6 7
The linked list 1 after concatenation is:
5 2 1 3 9 9 1 3 6 7

List 1
```

```
List 1
Enter the data that you wish to enter for position 5. 4
Enter the data that you wish to enter for position 4. 5
Enter the data that you wish to enter for position 3. 3
Enter the data that you wish to enter for position 2. 2
Enter the data that you wish to enter for position 1. 1
Enter 1 to sort the linked list, 2 to reverse the linked list, 3 to concatenate it with another linked list and 4 to exit. 4

Process returned 4 (0x4)   execution time : 103.886 s
Press any key to continue.
```

Lab Program 6(b)

WAP to Implement Single Linked List to simulate Stack operations

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20
int top=0;
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head=NULL;
void push()
{
   if(top==SIZE)
   {
      printf("Stack overflow. Cannot insert more elements into the stack.");
   }
   else
   {
      struct Node *new_node=malloc(sizeof(struct Node));
      int data;
      struct Node *last=head;
```

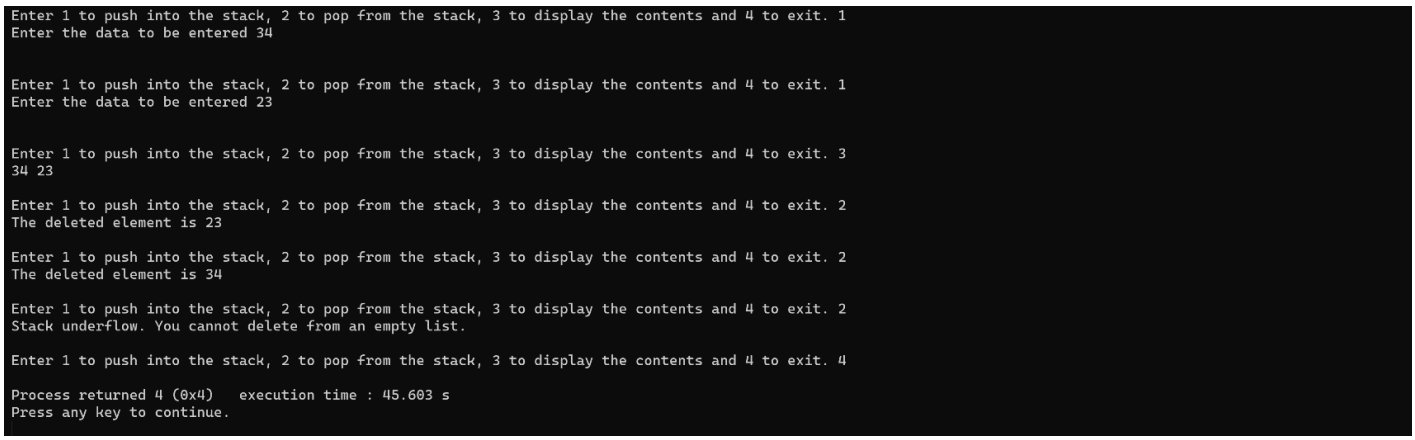
```
        printf("Enter the data to be entered ");
        scanf("%d",&data);
        (*new_node).data=data;
        (*new_node).next=NULL;
        if(head==NULL)
        {
           head=new_node;
        }
        else
        {
           while((*last).next!=NULL)
           {
              last=(*last).next;
           }
           (*last).next=new_node;
        }
        top=top+1;
    }
}
void pop()
{
    if(top==0)
        printf("Stack underflow. You cannot delete from an empty list.");
    else
    {
        int deleted_node;
        if((*head).next==NULL)
```

```
        {
            deleted_node=(*head).data;
            free(head);
            head=NULL;
        }
        else
        {
            struct Node *ptr1=head;
            struct Node *ptr=(*ptr1).next;
            while((*ptr).next!=NULL)
            {
                ptr1=(*ptr1).next;
                ptr=(*ptr1).next;
            }
            (*ptr1).next=NULL;
            deleted_node=(*ptr).data;
            free(ptr);
        }
        top=top-1;
        printf("The deleted element is %d", deleted_node);
    }
}
void display()
{
    if(top==0)
    {
        printf("Stack undeflow. Cannot display the contents of an empty stack.");
```

```
    }
    else
    {
        struct Node *node=head;
        while(node!=NULL)
        {
            printf("%d ",(*node).data);
            node=(*node).next;
        }
    }
}
void main()
{
    while(1)
    {
        printf("Enter 1 to push into the stack, 2 to pop from the stack, 3 to display the contents and 4 to exit. ");
        int ch;
        scanf("%d",&ch);
        if(ch==1)
        {
            push();
        }
        else if(ch==2)
        {
            pop();
        }
```

```
        else if(ch==3)
        {
            display();
        }
        else if(ch==4)
        {
            break;
        }
        else
        {
            printf("Invalid character.");
        }
        printf("\n\n");
    }
}
```

Output:



Lab Program 6(b)

WAP to Implement Single Linked List to simulate Queue Operations.

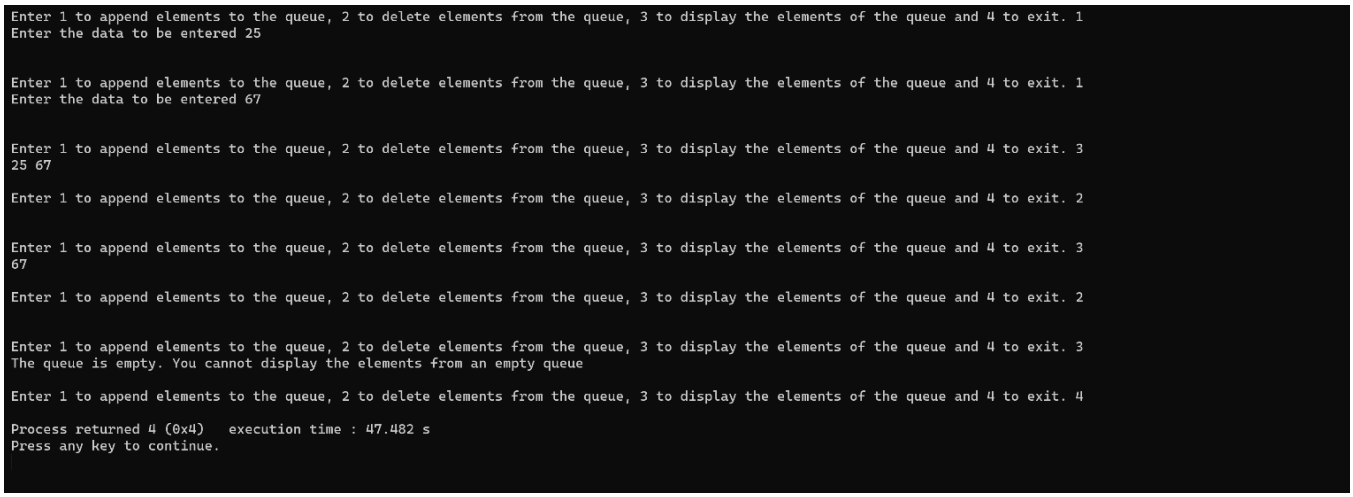
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head=NULL;
int rear=-1;
void append()
{
   if(rear==MAX-1)
   {
      printf("Queue overflow");
   }
   else
   {
      rear=rear+1;
      struct Node *new_node=malloc(sizeof(struct Node));
      int data;
```

```
        struct Node *last=head;
        printf("Enter the data to be entered ");
        scanf("%d",&data);
        (*new_node).data=data;
        (*new_node).next=NULL;
        if(head==NULL)
           head=new_node;
        else
        {
           while((*last).next!=NULL)
           {
              last=(*last).next;
           }

           (*last).next=new_node;
        }
    }
}
void Pop()
{
    if(head==NULL)
        printf("The queue is empty. You cannot delete from an empty queue");
    else
    {
        struct Node *ptr=head;
        head=(*ptr).next;
        free(ptr);
```

```
    }
}
void display()
{
    if(head==NULL)
        printf("The queue is empty. You cannot display the elements from an empty queue");
    else
    {
        struct Node *node=head;
        while(node!=NULL)
        {
            printf("%d ",(*node).data);
            node=(*node).next;
        }
    }
}
void main()
{
    while(1)
    {
        printf("Enter 1 to append elements to the queue, 2 to delete elements from the queue, 3 to display the elements of the queue and 4 to exit. ");
        int ch;
        scanf("%d", &ch);
        if(ch==1)
        {
            append();
```

```
        }
        else if(ch==2)
        {
            Pop();
        }
        else if(ch==3)
        {
            display();
        }
        else if(ch==4)
        {
            break;
        }
        else
        {
            printf("Invalid character");
        }
        printf("\n\n");
    }
}
```

Output:



Lab program 7(a)

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

Display the contents of the list

```
#include <stdio.h>

#include <stdlib.h>

struct Node

{

   int data;

   struct Node *next;

   struct Node *previous;

};
```

```
struct Node *head=NULL;
void insert(int position)
{
   int pos;
   struct Node *node=head;
   for(pos=1; pos<=position; pos++)
   {
      if(node==NULL && !(head==NULL && position==1))
      {
         printf("The given position is longer than the linked list. Please enter another position.");
         return;
      }
      if(pos==position)
      {
         break;
      }
      node=(*node).next;
   }
   int data;
   printf("Enter the data to be entered in the new node ");
   scanf("%d", &data);
   struct Node *newNode;
   newNode=malloc(sizeof(struct Node));
   (*newNode).data=data;
   (*newNode).next=node;
   if(head==NULL)
```

```
    {
        (*newNode).previous=NULL;
        head=newNode;
    }
    else{
        (*newNode).previous=(*node).previous;
        struct Node *previous;
        previous=(*node).previous;
        (*node).previous=newNode;
        if(previous==NULL)
        {
            head=newNode;
        }
        else
        {
            (*previous).next=newNode;
        }
    }

}
void delete_based_on_a_value(int value)
{
    struct Node *node=head;
    int first_time=1;
    while(1)
    {
        if(node==NULL)
```

```
{
   printf("Cannot delete from an empty list.");
   return;
}
for(node=head; node!=NULL; node=(*node).next)
{
   if((*node).data==value)
   {
      break;
   }
}
if(node==NULL)
{
   if(first_time==1)
   {
      printf("The node with the given value is not found in the linked list.");
   }
   return;
}
else
{
   if((*node).previous==NULL)
   {
      head=(*node).next;
   }
   else
   {
```

```
            (*(*node).previous).next=(*node).next;
        }
        if((*node).next!=NULL)
        {
            (*(*node).next).previous=(*node).previous;
        }
        free(node);
    }
    first_time=0;
  }

}
void display()
{
  if(head==NULL)
  {
    printf("The linked list is empty.");
  }
  else
  {
    struct Node *node;
    for(node=head; node!=NULL; node=(*node).next)
    {
      printf("%d ", (*node).data);
    }
  }
}
```

```
void main()
{
   while(1)
   {
      int ch;
      printf("Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. ");
      scanf("%d", &ch);
      if(ch==1)
      {
         int data, position;
         printf("Enter the position to the left of which you want to enter the data. ");
         scanf("%d", &position);
         insert(position);
      }
      else if(ch==2)
      {
         int value;
         printf("Enter the value for which you want to delete from the linked list. ");
         scanf("%d", &value);
         delete_based_on_a_value(value);
      }
      else if(ch==3)
         display();
      else if(ch==4)
      {
         break;
```

```
        }
        else
        {
            printf("Invalid character");
        }
        printf("\n\n");
    }
}
```

Output:

```
Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 3
The given position is longer than the linked list. Please enter another position.

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 45


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 34


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 3
23 34 45

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 3
Enter the data to be entered in the new node 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 1
Enter the position to the left of which you want to enter the data. 1
Enter the data to be entered in the new node 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 3
23 23 34 23 45

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 2
Enter the value for which you want to delete from the linked list. 23


Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 3
34 45

Enter 1 to insert, 2 to delete an element based on its value, 3 to display the elements of the linked list and 4 to exit. 4

Process returned 4 (0x4)   execution time : 92.693 s
```

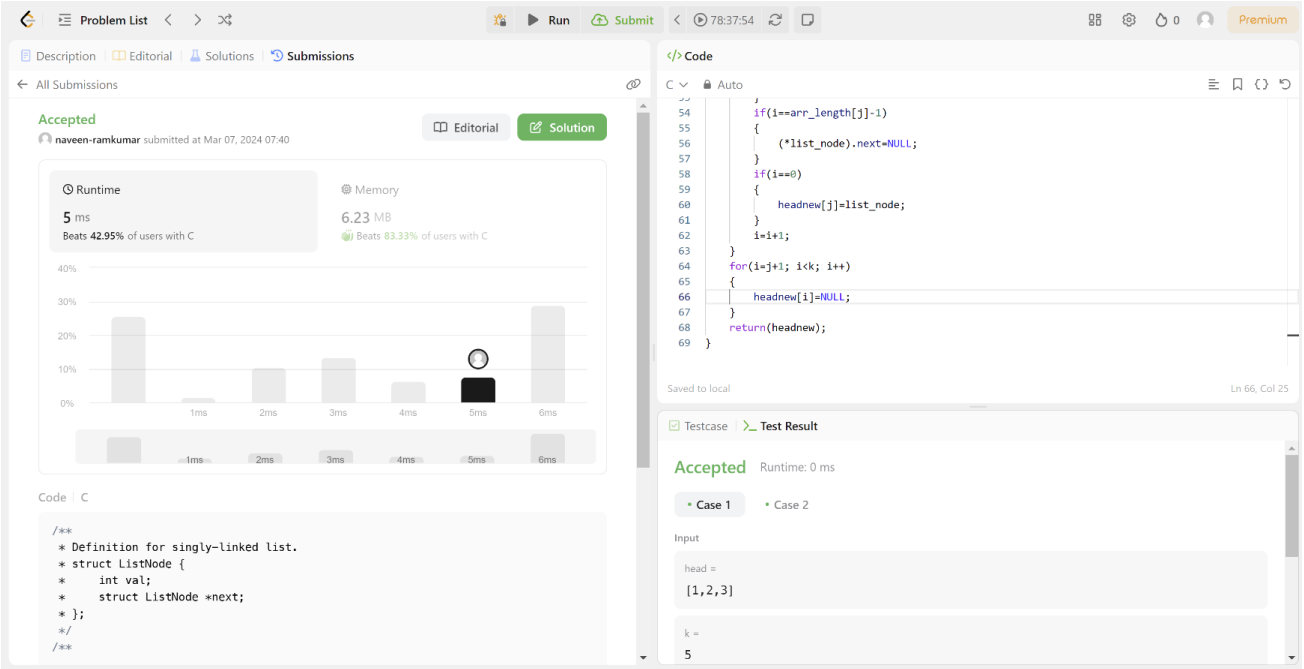
Lab program 7(b)

Demonstration of LeetCode program on Singly linked list

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
   *returnSize=k;
   struct ListNode** headnew;
   headnew=malloc(k*sizeof(struct ListNode*));
   int i;
   if(head==NULL)
   {
      for(i=0; i<k; i++)
      {
         headnew[i]=NULL;
      }
      return(headnew);
   }
   int list_length=0;
   struct ListNode* list_node;
   for(list_node=head; list_node!=NULL; list_node=(*list_node).next)
   {
      list_length=list_length+1;
   }
   int arr_length[k];
   for(i=0; i<k; i++)
   {
      if(i<list_length-(list_length/k)*k)
      {
         arr_length[i]=(list_length/k)+1;
      }
      else
      {
         arr_length[i]=(list_length/k);
      }
   }
   list_node=head;
   struct ListNode *array;
   int j=0;
   i=0;
   for(list_node=head; list_node!=NULL; list_node=array)
   {
      array=(*list_node).next;
      if(i==arr_length[j])
      {
         i=0;
         j=j+1;
      }
```

```
        if(i==arr_length[j]-1)
        {
           (*list_node).next=NULL;
        }
        if(i==0)
        {
           headnew[j]=list_node;
        }
        i=i+1;
    }
    for(i=j+1; i<k; i++)
    {
       headnew[i]=NULL;
    }
    return(headnew);
}
```

Output:



Lab program 8(a)

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 postorder
c. To display the elements in the tree.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *left;
   struct Node *right;
};
struct Node *root=NULL;
void inorder(struct Node *root)
{
   if(root!=NULL)
   {
      inorder((*root).left);
      printf("%d ", (*root).data);
      inorder((*root).right);
   }
   else
   {
      return;
   }
```

```
}
void preorder(struct Node *root)
{
   if(root!=NULL)
   {
      printf("%d ", (*root).data);
      preorder((*root).left);
      preorder((*root).right);
   }
   else
   {
      return;
   }
}
void postorder(struct Node *root)
{
   if(root!=NULL)
   {
      postorder((*root).left);
      postorder((*root).right);
      printf("%d ", (*root).data);
   }
   else
   {
      return;
   }
}
```

```
void insert(int value)
{
   struct Node *temp=root;
   struct Node *new_node=malloc(sizeof(struct Node));
   while(temp!=NULL)
   {
      if(value<(*temp).data)
      {
         if((*temp).left==NULL)
         {
            (*temp).left=new_node;
            break;
         }
         else
         {
            temp=(*temp).left;
         }
      }
      else
      {
         if((*temp).right==NULL)
         {
            (*temp).right=new_node;
            break;
         }
         else
         {
```

```
            temp=(*temp).right;
        }
    }
}
(*new_node).data=value;
(*new_node).left=NULL;
(*new_node).right=NULL;
if(root==NULL)
{
    root=new_node;
}
}
void main()
{
  while(1)
  {
    printf("Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. ");
    int choice;
    scanf("%d", &choice);
    if(choice==1)
    {
      printf("Enter the value to insert. ");
      int value;
      scanf("%d", &value);
      insert(value);
    }
```

```
        else if(choice==2)
        {
            preorder(root);
        }
        else if(choice==3)
        {
            inorder(root);
        }
        else if(choice==4)
        {
            postorder(root);
        }
        else if(choice==5)
        {
            break;
        }
        else
        {
            printf("Invalid character.");
        }
        printf("\n\n");
    }
}
```

Output:

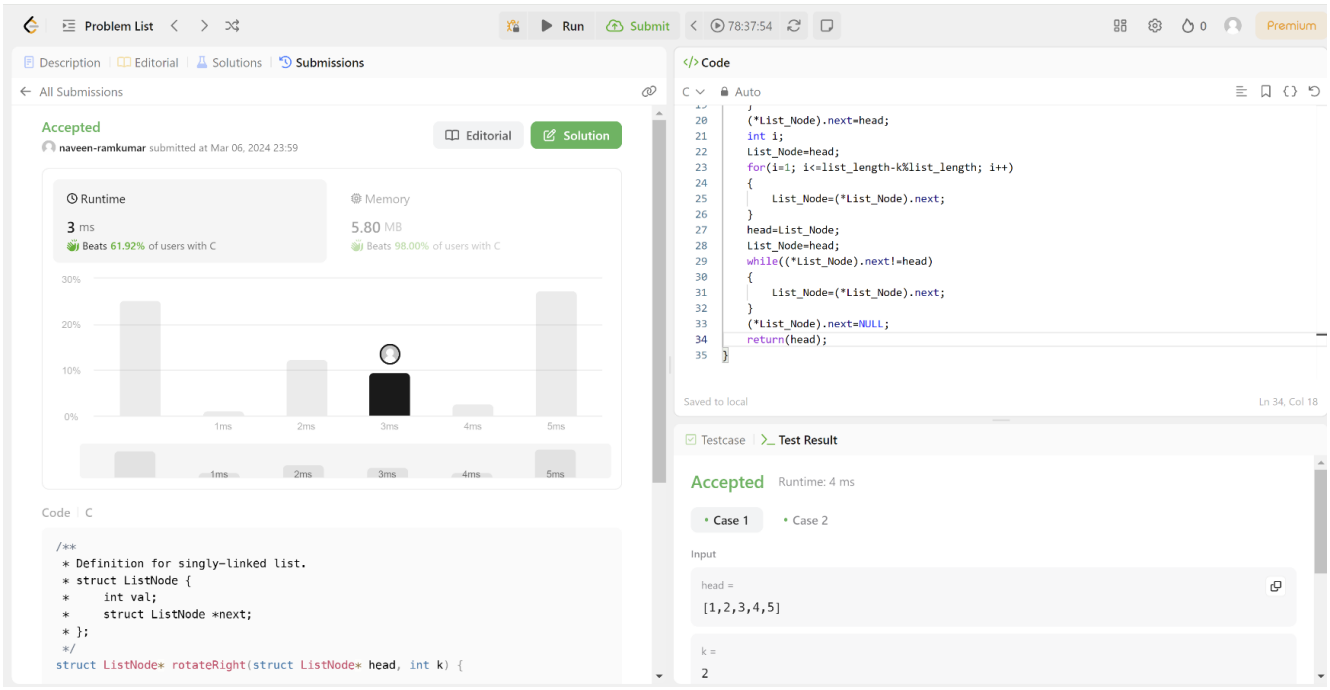
```
Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 100


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 20


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 10


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 30


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 200


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 150


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 1
Enter the value to insert. 300


Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 2
100 20 10 30 200 150 300

Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 3
10 20 30 100 150 200 300

Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 4
10 30 20 150 300 200 100

Enter 1 to insert the elements into the tree, 2 to do preorder traversal, 3 to do inorder traversal, 4 to do postorder traversal and 5 to exit. 5

Process returned 5 (0x5)   execution time : 70.125 s
Press any key to continue.
```

Lab program 8(b)

Leet code on Linked list

```
struct ListNode* rotateRight(struct ListNode* head, int k) {
    if(head==NULL)
    {
        return(head);
    }
    struct ListNode* List_Node=head;
    int list_length=1;
    while((*List_Node).next!=NULL)
    {
        list_length=list_length+1;
        List_Node=(*List_Node).next;
    }
    (*List_Node).next=head;
    int i;
    List_Node=head;
    for(i=1; i<=list_length-k%list_length; i++)
    {
        List_Node=(*List_Node).next;
    }
    head=List_Node;
    List_Node=head;
    while((*List_Node).next!=head)
    {
        List_Node=(*List_Node).next;
    }
    (*List_Node).next=NULL;
    return(head);
}
```

Output:


Problem List
Run
Submit
78:37:54
Premium
Description
Editorial
Solutions
Submissions
All Submissions
Accepted
naveen-ramkumar submitted at Mar 06, 2024 23:59
Editorial
Solution
Runtime
3 ms
Beats 61.92% of users with C
Memory
5.80 MB
Beats 98.00% of users with C
30%
20%
10%
0%
1ms
2ms
3ms
4ms
5ms
Code
C
/**
* Definition for singly-linked list.
* struct ListNode {
*   int val;
*   struct ListNode *next;
* };
*/
struct ListNode* rotateRight(struct ListNode* head, int k) {
Code
C
Auto
(*List_Node).next=head;
int i;
List_Node=head;
for(i=1; i<=list_length-k%list_length; i++)
{
List_Node=(*List_Node).next;
}
head=List_Node;
List_Node=head;
while((*List_Node).next!=head)
{
List_Node=(*List_Node).next;
}
(*List_Node).next=NULL;
return(head);
}
Saved to local
Ln 34, Col 18
Testcase
Test Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Input
head =
[1,2,3,4,5]
k =
2

Lab program 9

Write a program to traverse a graph using BFS method.
Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>
int adjacency_matrix[20][20];
int visited[20];
int n;
void depth_first_search(int v)
{
   int i;
   visited[v]=1;
   for(i=0; i<7; i++)
        if(adjacency_matrix[v][i] && !visited[i])
        {
           printf("\n %d->%d",v,i);
           depth_first_search(i);
        }
}
int main()
{
   int i, j, count=0;
   printf("\n Enter number of vertices:");
   scanf("%d", &n);
```

```
    for(i=0; i<n; i++)
    {
        visited[i]=0;
        for(j=0; j<n; j++)
            adjacency_matrix[i][j]=0;
    }
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            scanf("%d", &adjacency_matrix[i][j]);
    }
    depth_first_search(0);
    printf("\n");
    for(i=0; i<n; i++)
    {
        if(visited[i])
            count++;
    }
    if(count==n)
        printf("Graph is connected");
    else
        printf("Graph is not connected");
    return 0;
}
```

Output:

```
 Enter number of vertices:7
Enter the adjacency matrix:
0 1 0 1 0 0 0
1 0 1 1 0 1 0
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0

 0->1
 1->2
 2->3
 3->4
 4->6
 2->5
Graph is connected
Process returned 0 (0x0)   execution time : 50.338 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.

Design and develop a Program in C
that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.

Resolve the collision (if any)
using linear probing.

```
#include <stdio.h>

#define TABLE_SIZE 7

void insert_into_hash_table(int hash_table[], int key)

{

   int hashkey;

   hashkey=key%TABLE_SIZE;

   int i=0;

   int isalreadypresent=0;

   while(i<TABLE_SIZE)

   {

      if(hash_table[hashkey]==key)

      {
```

```
            isalreadypresent=1;
            break;
        }
        if(hash_table[hashkey]==-1)
        {
            hash_table[hashkey]=key;
            printf("The key %d is inserted in position %d of the hash table.\n", key, hashkey);
            break;
        }
        else
        {
            hashkey=(hashkey+1)%TABLE_SIZE;
            i=i+1;
        }
    }
    if(i==TABLE_SIZE)
    {
        printf("The hash table is full.\n");
    }
    else if(isalreadypresent==1)
    {
        printf("The given element already exists in the table.");
    }
}
void search_the_hash_table(int hash_table[], int key)
{
    int hashkey;
```

```
    hashkey=key%TABLE_SIZE;
    int i=0;
    while(i<TABLE_SIZE)
    {
        if(hash_table[hashkey]==key)
        {
            printf("The key %d is found in position %d in the hash table.\n", key, hashkey);
            break;
        }
        else
        {
            hashkey=(hashkey+1)%TABLE_SIZE;
            i=i+1;
        }
    }
    if(i==TABLE_SIZE)
    {
        printf("The element is not found.\n");
    }
}
void main()
{
    int hash_table[TABLE_SIZE];
    int i;
    for(i=0; i<TABLE_SIZE; i++)
    {
        hash_table[i]=-1;
```

```
    }
    while(1)
    {
        int key;
        int choice;
        printf("Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. ");
        scanf("%d", &choice);
        if(choice==1)
        {
            printf("Enter a key to enter into the hash table ");
            scanf("%d", &key);
            insert_into_hash_table(hash_table, key);
        }
        else if(choice==2)
        {
            printf("Enter a key to search in the hash table ");
            scanf("%d", &key);
            search_the_hash_table(hash_table, key);
        }
        else if(choice==3)
        {
            break;
        }
        else
        {
            printf("Invalid character.");
```

```
        }
        printf("\n\n");
    }
}
```

Output:

```
Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 700
The key 700 is inserted in position 0 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 50
The key 50 is inserted in position 1 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 85
The key 85 is inserted in position 2 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 92
The key 92 is inserted in position 3 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 73
The key 73 is inserted in position 4 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 176
Invalid character.

Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 76
The key 76 is inserted in position 6 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 1
Enter a key to enter into the hash table 101
The key 101 is inserted in position 5 of the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 2
Enter a key to search in the hash table 92
The key 92 is found in position 3 in the hash table.


Enter 1 to enter a key into the hash table and 2 to search for a key in the hash table and 3 to exit. 3

Process returned 3 (0x3)   execution time : 130.615 s
```