VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"GnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

R Abhinav (1BM22CS211)

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 R Abhinav **(1BM22CS211)**, 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.

Code:

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

#define max 10

int stack[max];
int top=-1;

void push(int c){
    if(top==max-1){
        printf("Stack is full");
        return;
    }
    top++;
    stack[top]=c;
}

void pop(){
    if(top==-1){
        printf("Stack is empty");
        return;
    }
    top--;
    printf("The popped elements is:%d",stack[top+1]);
}

void display(){
    if(top==-1){
        printf("Stack is empty");
    }

    for(int i=0;i<=top;i++){
        printf("%d ",stack[i]);
    }
}
```

```
int main(){
    int choice;
    int a;


    while (1)
    {
        printf("\nSelect an option\n1.Push element into
stack\n2.Pop element from stack\n3.Display stack\nChoice:");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1:
                printf("Enter element to be pushed:");
                scanf("%d",&a);
                push(a);
                break;
            case 2:
                pop();
                break;
            default:
                display();
        }
        printf("\n");
    }

    return 0;
}
```

Output:

```
Select an option
1.Push element into stack
2.Pop element from stack
3.Display stack
Choice:1
Enter element to be pushed:1


Select an option
1.Push element into stack
2.Pop element from stack
3.Display stack
Choice:1
Enter element to be pushed:2


Select an option
1.Push element into stack
2.Pop element from stack
3.Display stack
Choice:3
1 2

Select an option
1.Push element into stack
2.Pop element from stack
3.Display stack
Choice:2
The popped elements is:2
```

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<string.h>
#include<ctype.h>

#define size 30

char stack[size];
int top=-1;

void push(char a){
    stack[++top]=a;
}

char pop(){
    return stack[top--];
}

int precedence(char a){
    if(a=='^')
        return(3);
    else if(a=='*' || a=='/')
        return(2);
    else if(a=='+' || a=='-')
        return(1);
    else
        return(0);
}


void main(){
    char infix[size],postfix[size];

    printf("Enter infix expression:");
    scanf("%s",infix);
    int len=strlen(infix);
    int j=0;
```

```
    printf("Char\tStack\tpostfix\n");

    for(int i=0;i<len;i++){
        if(infix[i]=='(')
            push('(');
        else if(infix[i]==')'){
            while(stack[top]!='('){
                postfix[j++]=pop();
            }
            pop();
        }
        else{
            if(isalnum(infix[i])){
                postfix[j++]=infix[i];
            }
            else{
                if(precedence(stack[top])<precedence(infix[i]))
                    push(infix[i]);
                else{
                    while(stack[top]!='('){
                        postfix[j++]=pop();
                    }
                    push(infix[i]);
                }
            }
        }
        stack[top+1]='\0';
        postfix[j]='\0';
        printf("%c\t%s\t%s\n",infix[i],stack,postfix);
    }
    while(top!=-1){
        postfix[j++]=pop();
    }
    postfix[j]='\0';
    printf("Postfix Expression is:%s",postfix);

}
```

Output:

```
Enter infix expression:A+(B*C-(D/E^F)*G)*H
Char    Stack   postfix
A               A
+       +       A
(       +(      A
B       +(      AB
*       +(*     AB
C       +(*     ABC
-       +(-     ABC*
(       +(-(    ABC*
D       +(-(    ABC*D
/       +(-(/   ABC*D
E       +(-(/   ABC*DE
^       +(-(/^  ABC*DE
F       +(-(/^  ABC*DEF
)       +(-     ABC*DEF^/
*       +(-*    ABC*DEF^/
G       +(-*    ABC*DEF^/G
)       +       ABC*DEF^/G*-
*       +*      ABC*DEF^/G*-
H       +*      ABC*DEF^/G*-H
Postfix Expression is:ABC*DEF^/G*-H*+
```

Lab 3:

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.

Code:

```
#include<stdio.h>
#define size 30

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


void insert(int a){
    if(rear==size-1){
        printf("Queue overflow\n");
        return;
    }
    else{
        if(front==-1)
            front=0;
        queue[++rear]=a;
    }
}

void delete(){
    if(front==-1||front>rear){
        printf("Queue Empty\n");
    }
    else{
        front++;
    }
}

void display(){
    if(front==-1){
        printf("Queue Empty\n");
        return;
    }
```

```
    printf("Queue:");
    for(int i=front;i<=rear;i++){
        printf("%d ",queue[i]);
    }
    printf("\n--------------\n");
}


void main(){
    int choice;
    int a;
    while(1){
        printf("Queue
operations:\n1.Insert\n2.Delete\n3.Display\nChoice:");
        scanf("%d",&choice);

        switch (choice)
        {
        case 1:
            printf("Enter Element:");
            scanf("%d",&a);
            insert(a);
            display();
            break;

        case 2:
            delete();
            display();
            break;

        case 3:
            display();
            break;
        }
    }

}
```

Output:

```
Queue operations:
1.Insert
2.Delete
3.Display
Choice:1
Enter Element:3
Queue:3
-------------
Queue operations:
1.Insert
2.Delete
3.Display
Choice:1
Enter Element:4
Queue:3 4
-------------
Queue operations:
1.Insert
2.Delete
3.Display
Choice:2
Queue:4
-------------
Queue operations:
1.Insert
2.Delete
3.Display
Choice:3
Queue:4
```

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

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#define size 5

int q[size], f = 0, r = -1;
int count = 0;

void enqueue(int item)
{
    if (count == size)
    {
        printf("\nQueue full!");
        return;
    }
    q[(++r) % size] = item;
    count++;
}

void dequeue()
{
    if (count == 0)
    {
        printf("\nQueue empty!");
        return;
    }
    f = (f + 1) % size;
    count--;
}

void display()
{
    if (count == 0)
    {
        printf("\nQueue empty!");
        return;
    }
    int front = f;
    for (int i = 0; i < count; i++)
    {
        printf("%d ", q[front]);
        front = (front + 1) % size;
```

```
    }
}

int main()
{
    int ch, item;
    while (1)
    {
        printf("\nSelect choice \n1.Enqueue \n2.Dequeue
\n3.Display\nChoice:");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("\nEnter value to insert: ");
            scanf("%d", &item);
            enqueue(item);
            break;
        case 2:
            dequeue();
            printf("\nItem popped");
            break;
        case 3:
            display();
            break;
        default:
            exit(0);
        }
    }
}
```

Output:

```
Select choice
1.Enqueue
2.Dequeue
3.Display
Choice:1

Enter value to insert: 2

Select choice
1.Enqueue
2.Dequeue
3.Display
Choice:1

Enter value to insert: 4

Select choice
1.Enqueue
2.Dequeue
3.Display
Choice:3
2 4
Select choice
1.Enqueue
2.Dequeue
3.Display
Choice:2

Item popped
Select choice
1.Enqueue
2.Dequeue
3.Display
Choice:3
4
```

Lab 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.

Code:

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

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

void insertAtHead(struct node **head_ref, int new_data) {
    struct node *new_node = (struct node *)malloc(sizeof(struct
node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void insertAtMiddle(struct node *prev_node, int new_data) {
    if (prev_node == NULL) {
        printf("The given previous node cannot be NULL\n");
        return;
    }
    struct node *new_node = (struct node *)malloc(sizeof(struct
node));
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}

void insertAtEnd(struct node **head_ref, int new_data) {
    struct node *new_node = (struct node *)malloc(sizeof(struct
node));
    struct node *last = *head_ref;
    new_node->data = new_data;
```

```
    new_node->next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = new_node;
}

void display(struct node *node) {
    printf("Linked List:");
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    struct node *head = NULL;
    int choice = 0, a,b;
    while (choice != 3) {
        printf("Select an
option\n1.Insert\n2.Display\n3.Exit\nChoice:");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to be inserted:");
                scanf("%d", &a);
                int choice2;
                printf("--------------\n1.Insert at Head\n2.Insert
at end\n3.Insert in the middle\nChoice:");
                scanf("%d", &choice2);
                switch (choice2) {
                    case 1:
                        insertAtHead(&head, a);
                        display(head);
                        break;
                    case 2:
```

```
                        insertAtEnd(&head, a);
                        display(head);
                        break;
                    case 3:
                        printf("Enter the value after which to
insert:");
                        scanf("%d", &b);
                        struct node *temp = head;
                        while (temp != NULL && temp->data != b) {
                            temp = temp->next;
                        }
                        if (temp == NULL) {
                            printf("Element not found in the
list.\n");
                        } else {
                            insertAtMiddle(temp, a);
                            display(head);
                        }
                        break;
                }
                break;
            case 2:
                display(head);
                break;
            default:
                break;
        }
    }

    return 0;
}
```

Output:

```
Select an option
1.Insert
2.Display
3.Exit
Choice:1
Enter the value to be inserted:1
--------------
1.Insert at Head
2.Insert at end
3.Insert in the middle
Choice:1
Linked List:1
Select an option
1.Insert
2.Display
3.Exit
Choice:1
Enter the value to be inserted:3
--------------
1.Insert at Head
2.Insert at end
3.Insert in the middle
Choice:2
Linked List:1 3
Select an option
1.Insert
2.Display
3.Exit
Choice:1
Enter the value to be inserted:2
--------------
1.Insert at Head
2.Insert at end
3.Insert in the middle
Choice:3
Enter the value after which to insert:1
Linked List:1 2 3
```

Program - Leetcode platform:

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

MinStack() initializes the stack object.

void push(int val) pushes the element val onto the stack.

void pop() removes the element on the top of the stack.

int top() gets the top element of the stack.

int getMin() retrieves the minimum element in the stack.

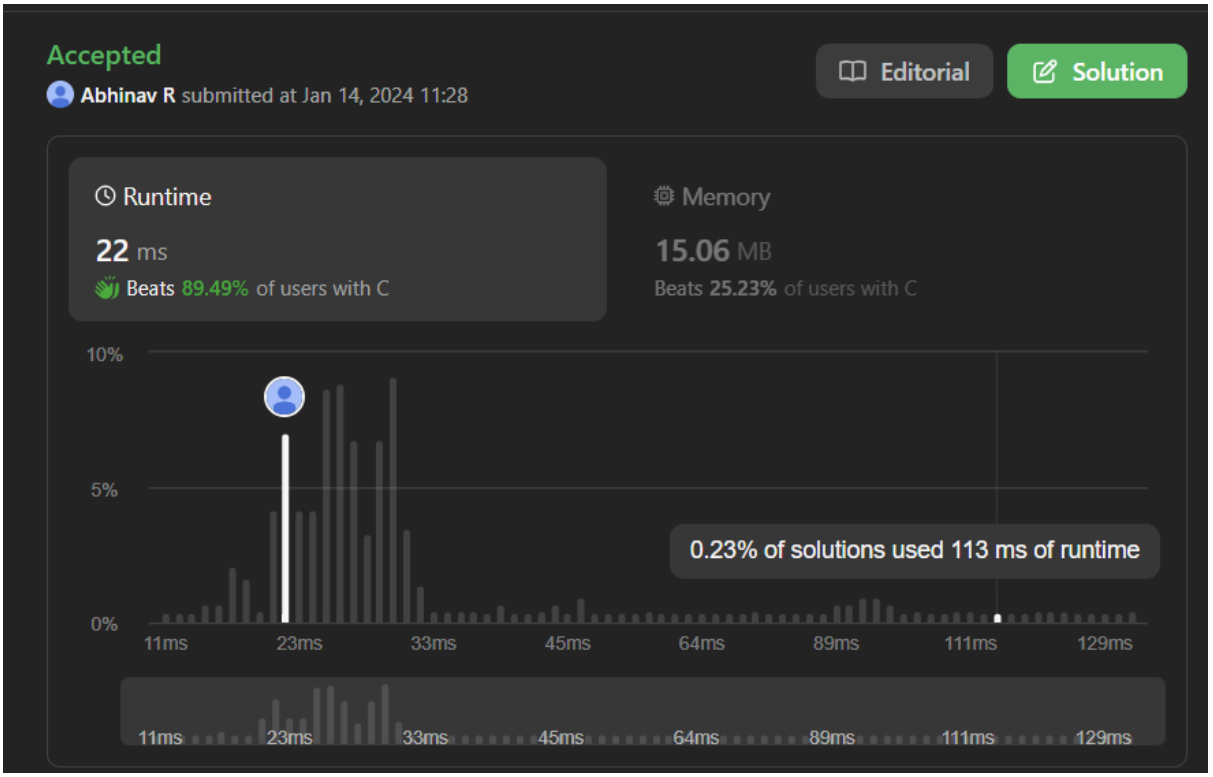
You must implement a solution with O(1) time complexity for each function.

Code:

```
typedef struct{
    int array[30000];
    int min[30000];
    int top1;
    int top2;
} MinStack;

MinStack* minStackCreate() {
    MinStack* obj = (MinStack*)malloc(sizeof(MinStack));

    obj->top1=-1;
    obj->top2=-1;
    return obj;
}

void minStackPush(MinStack* obj, int val) {
    obj->array[++obj->top1]=val;
    if(obj->top2==-1){
        obj->min[++obj->top2]=val;
        return;
    }
    int mintop=obj->min[obj->top2];
    if(mintop>val){0
        obj->min[++obj->top2]=val;
        return;
```

```
    }
    else{
        obj->min[++obj->top2]=mintop;
    }
}

void minStackPop(MinStack* obj) {
    obj->top1--;
    obj->top2--;
}

int minStackTop(MinStack* obj) {
    return obj->array[obj->top1];
}

int minStackGetMin(MinStack* obj) {
    return obj->min[obj->top2];
}

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

/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
 * minStackPush(obj, val);

 * minStackPop(obj);

 * int param_3 = minStackTop(obj);

 * int param_4 = minStackGetMin(obj);

 * minStackFree(obj);
*/
```

Output:


Accepted
Abhinav R submitted at Jan 14, 2024 11:28
Editorial
Solution
Runtime
22 ms
Beats 89.49% of users with C
Memory
15.06 MB
Beats 25.23% of users with C
10%
5%
0%
0.23% of solutions used 113 ms of runtime
11ms
23ms
33ms
45ms
64ms
89ms
111ms
129ms

Lab 5:

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.

Code:

#include <stdio.h>

#include <stdlib.h>

```
struct node {
    int data;
    struct node* next;
};

void push(struct node** head, int new_data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));

    new_node->data = new_data;
    new_node->next = *head;
    *head = new_node;
}

void append(struct node** head, int new_data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));

    new_node->data = new_data;
    new_node->next = NULL;
    struct node* last = *head;

    if (*head == NULL)
        *head = new_node;
    else {
        while (last->next != NULL)
            last = last->next;
```

```
        last->next = new_node;
    }
}

void insertafterptr(struct node* prev, int new_data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));

    new_node->data = new_data;
    new_node->next = prev->next;
    prev->next = new_node;
}

void insertafterpos(int position, int data, struct node** head) {
    struct node* prev = *head;

    for (int i = 0; i < position-1; i++) {
        prev = prev->next;
        if (prev == NULL) {
            printf("Position greater than size\n");
            return;
        }
    }

    insertafterptr(prev, data);
}

void insertafterval(int value, int data, struct node** head) {
    struct node* prev = *head;

    while (prev->next != NULL) {
        if (prev->data == value) {
            insertafterptr(prev, data);
            return;
        }
        prev = prev->next;
    }
    printf("The value doesn't exist in the linked list\n");
}

void display(struct node* head) {
```

```
    if (head == NULL) {
        printf("Linked List empty.\n");
        return;
    }
    printf("Linked List:");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

void del_head(struct node **head){
    if(*head==NULL){
        printf("List is empty\n");
        return;
    }
    struct node *temp=(*head)->next;
    free(*head);
    *head=temp;
}

void del_end(struct node *head){
    if(head==NULL){
        printf("List Empty\n");
        return;
    }
    struct node *last=head;

    struct node *prev;
    while(last->next!=NULL){
        prev=last;
        last=last->next;
    }
    free(last);
    prev->next=NULL;

}

void del_mid(struct node *target,struct node *head){
    struct node *temp=target->next;
    struct node *prev=head;
```

```
    while(prev->next!=target)
        prev=prev->next;
    free(target);
    prev->next=temp;
}

void del_val(int val,struct node **head){
    struct node *temp=*head;
    if(temp->data==val){
        del_head(head);
        return;
    }
    while(temp->next!=NULL){
        if(temp->data==val){
            del_mid(temp,*head);
            return;
        }
        temp=temp->next;
    }
    printf("This value does not exist in linked list\n");
}

void del_pos(int val,struct node *head){
    struct node *temp=head;
    for(int i=0;i<val-1;i++){
        temp=temp->next;
        if(temp==NULL){
            printf("Index is more than length of linked list\n");
            return;
        }
    }

    del_mid(temp,head);
}

int main() {
    struct node* head = NULL;
    int choice,val,pos,preval,choice2;
    int choice3;
    append(&head,1);
    append(&head,2);
    append(&head,3);
```

```
    append(&head,4);
    append(&head,5);
    while (1)
    {
        printf("------------------------------\n");
        printf("1.Insert\n2.Delete\n3.Display\nChoice:");
        scanf("%d",&choice);
        printf("------------------------------\n");
        switch (choice)
        {
            case 1:
                printf("Enter Value to insert:");
                scanf("%d",&val);
                printf("1.Insert at Head\n2.Insert at
End\n3.Insert at middle\nChoice:");
                scanf("%d",&choice2);

                switch (choice2)
                {
                    case 1:
                        push(&head,val);
                        printf("%d inserted at head.\n",val);
                        display(head);
                        break;

                    case 2:
                        append(&head,val);
                        printf("%d inserted at end.\n",val);
                        display(head);
                        break;

                    case 3:
                        printf("1.Insert at index\n2.Insert after
value\nChoice:");
                        scanf("%d",&choice3);
                        printf("-+-+-+-+-+-\n");
                        switch (choice3)
                        {
                            case 1:
                                printf("Enter the index:");
                                scanf("%d",&pos);
```

```
                                insertafterpos(pos,val,&head);
                                display(head);
                                printf("-+-+-+-+-+-\n");
                                break;

                            case 2:
                                printf("Enter the Value:");
                                scanf("%d",&preval);

                                insertafterval(preval,val,&head);
                                display(head);
                                break;

                            default:
                                break;

                        }
                    default:
                        break;
                }
                break;
            case 2:
                printf("1.Delete at head\n2.Delete at
tail\n3.Delete in middle\nChoice:");
                scanf("%d",&choice);
                switch (choice)
                {
                case 1:
                    del_head(&head);
                    display(head);
                    break;
                case 2:
                    del_end(head);
                    display(head);
                    break;
                case 3:

                    printf("1.Delete a value\n2.Delete at
index\nChoice:");
                    scanf("%d",&choice2);
                    switch (choice2)
                    {
```

```
                        case 1:
                            printf("Enter Value:");
                            scanf("%d",&val);
                            del_val(val,&head);
                            display(head);
                            break;

                        case 2:
                            printf("Enter Index:");
                            scanf("%d",&pos);
                            del_pos(pos,head);
                            display(head);
                            break;
                    }
                }
                break;
            case 3:
                display(head);
                break;
        }
    }
    return 0;
}
```

Output:

```
PS C:\Users\rabhi\OneDrive\Desktop\C> cd
) { .\linklist }
-----------------------------
1.Insert
2.Delete
3.Display
Choice:3
-----------------------------
Linked List:1 2 3 4 5
-----------------------------
1.Insert
2.Delete
3.Display
Choice:2
-----------------------------
1.Delete at head
2.Delete at tail
3.Delete in middle
Choice:1
Linked List:2 3 4 5
-----------------------------
1.Insert
2.Delete
3.Display
Choice:2
-----------------------------
1.Delete at head
2.Delete at tail
3.Delete in middle
Choice:2
Linked List:2 3 4
```

```
-----------------------------
1.Delete at head
2.Delete at tail
3.Delete in middle
Choice:2
Linked List:2 3 4
-----------------------------
1.Insert
2.Delete
3.Display
Choice:2
-----------------------------
1.Delete at head
2.Delete at tail
3.Delete in middle
Choice:3
1.Delete a value
2.Delete at index
Choice:1
Enter Value:3
Linked List:2 4
-----------------------------
```

Program - Leetcode platform

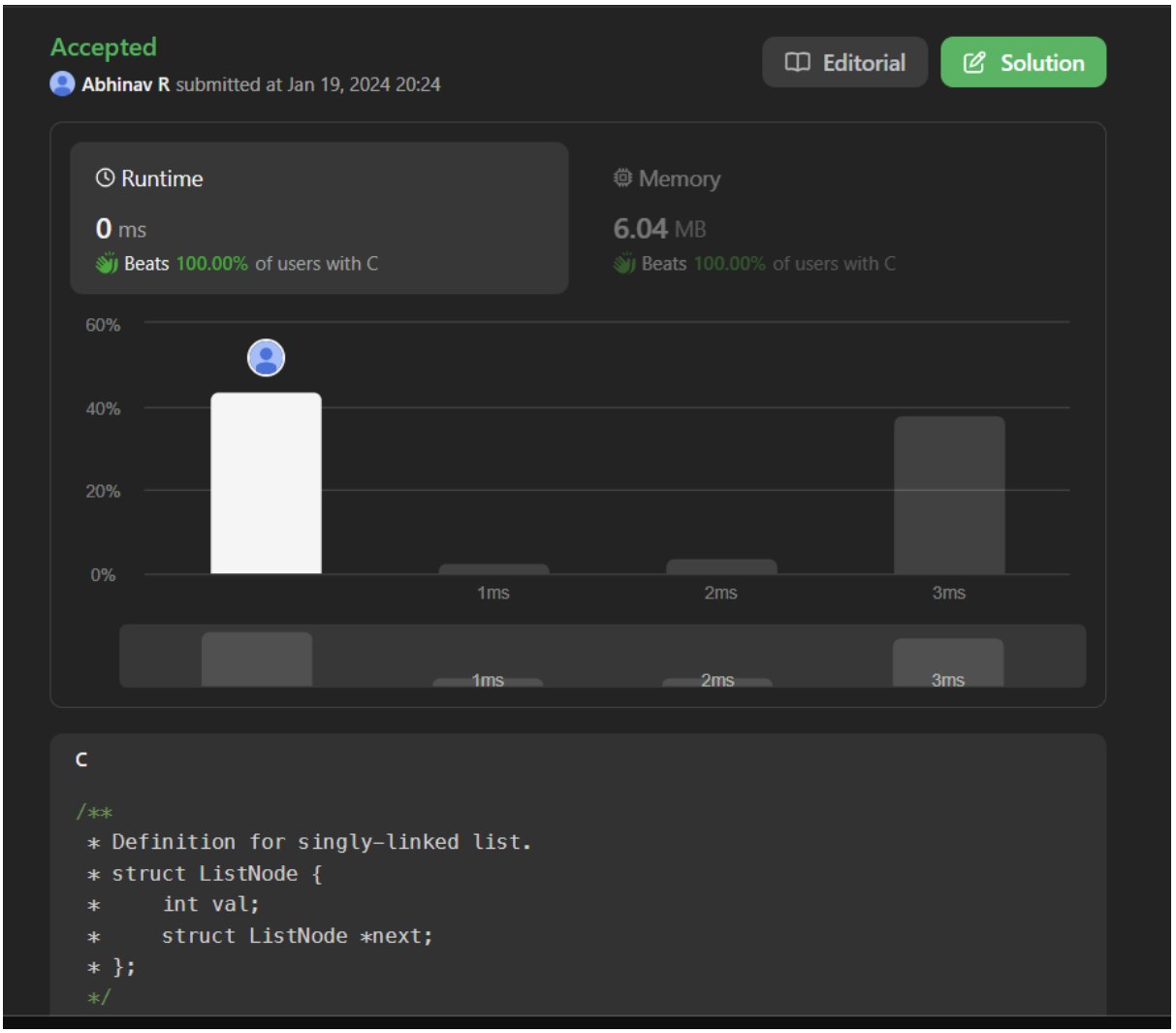
Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Code:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */

void append(struct ListNode** head,int val){
    struct ListNode *new_node=(struct ListNode*)malloc(sizeof(struct
ListNode));
    new_node->val=val;
    new_node->next=NULL;
    struct ListNode *prev=*head;
    if(prev==NULL){
        *head=new_node;
        return;
    }
    while(prev->next!=NULL)
        prev=prev->next;
    prev->next=new_node;
}

void mid(struct ListNode *prev,int val){

    struct ListNode* new_node=(struct ListNode*)malloc(sizeof(struct
ListNode));
    new_node->val=val;
    new_node->next=prev->next;
    prev->next=new_node;
}

void push(struct ListNode **head,int value){
    struct ListNode *new_node=(struct ListNode*)malloc(sizeof(struct
ListNode));

    new_node->val=value;
    new_node->next=(*head);
    *head=new_node;
}
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
```

```
    struct ListNode* newhead=NULL;
    int i=1;
    struct ListNode* cur=head;
    struct ListNode* prev=newhead;
    while(cur!=NULL)
    {

        if(left<=i && right>=i)
        {
            if(prev==NULL){
                append(&newhead,cur->val);
                prev=newhead;
                cur=cur->next;
                i++;
                continue;
            }
            else if(left==1){
                push(&newhead,cur->val);
            }

            else
                mid(prev,cur->val);
        }
        else
        {
            append(&newhead,cur->val);
            prev = (prev == NULL) ? newhead : prev->next;
        }

        i++;
        cur=cur->next;
    }

    return newhead;
}

void display(struct ListNode* head) {
    if (head == NULL) {
        printf("Linked List empty.\n");
        return;
    }
    printf("Linked List:");
    while (head != NULL) {
        printf("%d ", head->val);
        head = head->next;
    }
    printf("\n");
}
```

Output:


Accepted
Abhinav R submitted at Jan 19, 2024 20:24
Editorial
Solution
Runtime
0 ms
Beats 100.00% of users with C
Memory
6.04 MB
Beats 100.00% of users with C
60%
40%
20%
0%
1ms
2ms
3ms
1ms
2ms
3ms
C
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
 *   struct ListNode *next;
 * };
 */

Lab 6

a) WAP to Implement Single Link List with following operations:

- **Sort the linked list**
- **Reverse the linked list**
- **Concatenation of two linked lists.**

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

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

void append(struct node **head, int new_data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));

    new_node->data = new_data;
    new_node->next = NULL;
    struct node *last = *head;

    if (*head == NULL)
        *head = new_node;
    else
    {
        while (last->next != NULL)
            last = last->next;

        last->next = new_node;
    }
}


void display(struct node *head)
{
    if (head == NULL)
    {
        printf("Linked List empty.\n");
        return;
    }
    printf("Linked List:");
    while (head != NULL)
```

```
    {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}


void bubble_sort(struct node *head)
{
    struct node *prev;
    struct node *cur;
    int nex;
    int flag = 1;
    int flag2 = 1;

    while (flag)
    {
        prev = head;
        while (prev != NULL && prev->next != NULL)
        {
            cur = prev->next;

            if (cur->data < prev->data)
            {
                nex = cur->data;
                cur->data = prev->data;
                prev->data = nex;
            }

            prev = prev->next;
        }

        int max = 0;
        prev = head;

        while (prev != NULL)
        {
            if (max > prev->data)
            {
                flag2 = 0;
                break;
            }
            max = prev->data;
            prev = prev->next;
        }

        if (flag2)
```

```
            flag = 0;
        else
            flag2 = 1;
    }
}

void reverse(struct node **head)
{
    struct node *prev = NULL;
    struct node *current = *head;
    struct node *next = NULL;

    while (current != NULL)
    {
        next = current->next;
        current->next = prev;


        prev = current;
        current = next;
    }

    *head = prev;
}

void concat(struct node *head1,struct node *head2){
    struct node *prev=head2;

    while(prev!=NULL){
        append(&head1,prev->data);
        prev=prev->next;
    }
}

int main()
{
    struct node *head=NULL;
    append(&head,5);
    append(&head,2);
    append(&head,3);
    append(&head,4);
    append(&head,1);
    append(&head,6);

    display(head);

    bubble_sort(head);
    display(head);
```

```
    reverse(&head);
    display(head);

    struct node *head2=NULL;
    append(&head2,76);
    append(&head2,43);
    append(&head2,34);

    concat(head,head2);
    display(head);
    return 0;
}
```

Output:

```
PS C:\Users\rabhi\OneDrive\Desktop\C> cd "c:\Users
Linked List:5 2 3 4 1 6
Linked List:1 2 3 4 5 6
Linked List:6 5 4 3 2 1
Linked List:6 5 4 3 2 1 76 43 34
PS C:\Users\rabhi\OneDrive\Desktop\C\C project>
```

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

i) Stack

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

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


void append(struct node** head, int new_data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));

    new_node->data = new_data;
    new_node->next = NULL;
    struct node* last = *head;

    if (*head == NULL)
        *head = new_node;
```

```
    else {
        while (last->next != NULL)
            last = last->next;

        last->next = new_node;
    }
}

void display(struct node* head) {
    if (head == NULL) {
        printf("Linked List empty.\n");
        return;
    }
    printf("Stack:");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}


void del_end(struct node *head){
    if(head==NULL){
        printf("List Empty\n");
        return;
    }
    struct node *last=head;

    struct node *prev;
    while(last->next!=NULL){
        prev=last;
        last=last->next;
    }
    free(last);
    prev->next=NULL;

}


int main() {
    struct node* head = NULL;

    int choice,a;


    while(choice<4){
        printf("1.Push\n2.Pop\n3.Display\nChoice:");
```

```
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            printf("Enter value:");
            scanf("%d",&a);
            append(&head,a);
            display(head);
            break;
        case 2:
            del_end(head);
            display(head);
            break;
        case 3:
            display(head);
        default:
            break;
        }
    }

    return 0;
}
```

Output:

```
1.Push
2.Pop
3.Display
Choice:1
Enter value:1
Stack:1
1.Push
2.Pop
3.Display
Choice:1
Enter value:2
Stack:1 2
1.Push
2.Pop
3.Display
Choice:2
Stack:1
1.Push
2.Pop
3.Display
Choice:3
Stack:1
1.Push
2.Pop
3.Display
Choice:
```

ii)Queue

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

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

void append(struct node** head, int new_data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));

    new_node->data = new_data;
    new_node->next = NULL;
    struct node* last = *head;

    if (*head == NULL)
        *head = new_node;
    else {
        while (last->next != NULL)
            last = last->next;

        last->next = new_node;
    }
}


void display(struct node* head) {
    if (head == NULL) {
        printf("Linked List empty.\n");
        return;
    }
    printf("Queue:");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

void del_head(struct node **head){
    if(*head==NULL){
        printf("List is empty\n");
        return;
    }
    struct node *temp=(*head)->next;
    free(*head);
    *head=temp;
```

```
}


int main() {
    struct node* head = NULL;

    int choice,a;


    while(choice<4){
        printf("1.Push\n2.Pop\n3.Display\nChoice:");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            printf("Enter value:");
            scanf("%d",&a);
            append(&head,a);
            display(head);
            break;
        case 2:
            del_head(&head);
            display(head);
            break;
        case 3:
            display(head);
        default:
            break;
        }
    }
    return 0;
}
```

Output:

```
1.Push
2.Pop
3.Display
Choice:1
Enter value:1
Queue:1
1.Push
2.Pop
3.Display
Choice:1
Enter value:2
Queue:1 2
1.Push
2.Pop
3.Display
Choice:2
Queue:2
1.Push
2.Pop
3.Display
Choice:3
Queue:2
```

Lab 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

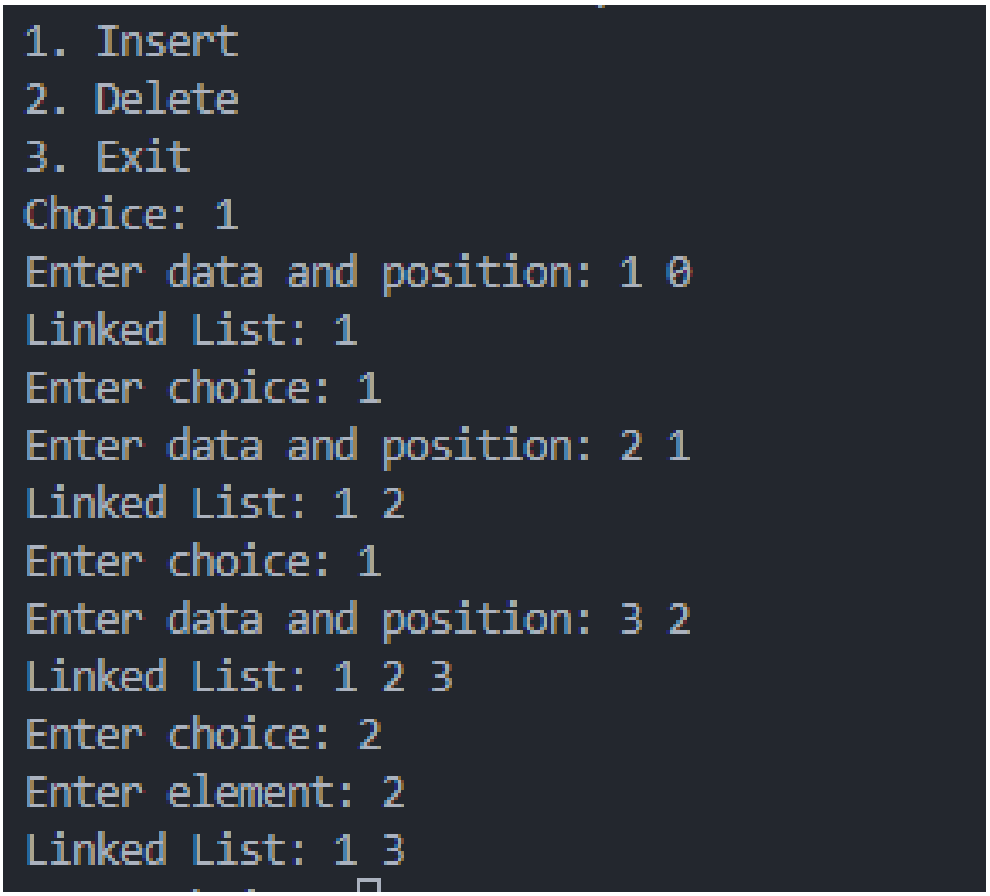
Code:

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

typedef struct Node {
    int data;
    struct Node *next;
    struct Node *prev;
} Node;

Node* head = NULL;
int count = 0;

void insert(int data, int position) {
    if (position == 0) {
        Node* newNode = malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = head;
        newNode->prev = NULL;
        if (head != NULL) head->prev = newNode;
        head = newNode;
        count++;
        return;
    } else if (position == count) {
        Node* newNode = malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = NULL;
        Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
        count++;
        return;
    } else if (position > count || position < 0) {
        printf("Unable to insert at the given position\n");
        return;
    } else {
        Node* temp = head;
```

```
        for (int i = 0; i < position - 1; i++)
            temp = temp->next;
        Node* newNode = malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = temp->next;
        newNode->prev = temp;
        temp->next->prev = newNode;
        temp->next = newNode;
        count++;
        return;
    }
}

void delete(int element) {
    int position = 0;
    Node* temp = head;
    if (head == NULL) {
        printf("List is empty, cannot delete\n");
        return;
    }
    for (; position < count; temp = temp->next, position++)
        if (temp->data == element) break;
    if (temp == NULL) {
        printf("Element does not exist in the list\n");
        return;
    }
    if (position == 0) {
        Node* temp = head;
        temp = temp->next;
        temp->prev = NULL;
        free(head);
        head = temp;
        count--;
        return;
    } else if (position == count - 1) {
        Node* temp = head;
        for (int i = 1; i < count - 1; i++)
            temp = temp->next;
        Node* temp1 = temp->next;
        temp->next = NULL;
        free(temp1);
        count--;
        return;
    } else if (position > count || position < 0) {
        printf("Unable to delete at the given position\n");
        return;
    } else {
        Node* temp = head;
```

```
        for (int i = 0; i < position; i++)
            temp = temp->next;
        temp->next->prev = temp->prev;
        temp->prev->next = temp->next;
        free(temp);
        count--;
        return;
    }
}

void display() {
    Node* temp = head;
    printf("Linked List: ");
    while (temp->next != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("%d ", temp->data);
    printf("\n");
}

int main() {
    int data, choice, pos;
    printf("1. Insert\n2. Delete\n3. Exit\nChoice: ");
    scanf("%d", &choice);
    while (choice != 3) {
        if (choice == 1) {
            printf("Enter data and position: ");
            scanf("%d%d", &data, &pos);
            insert(data, pos);
            printf("Count: %d\n", count);
        } else if (choice == 2) {
            printf("Enter element: ");
            scanf("%d", &pos);
            delete(pos);
            printf("Count: %d\n", count);
        }
        display();
        printf("Enter choice: ");
        scanf("%d", &choice);
    }

    return 0;
}
```

Output:



Leetcode

Given the head of a singly linked list and an integer k, split the linked list into k consecutive linked list parts.

The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.

The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.

Return an array of the k parts.

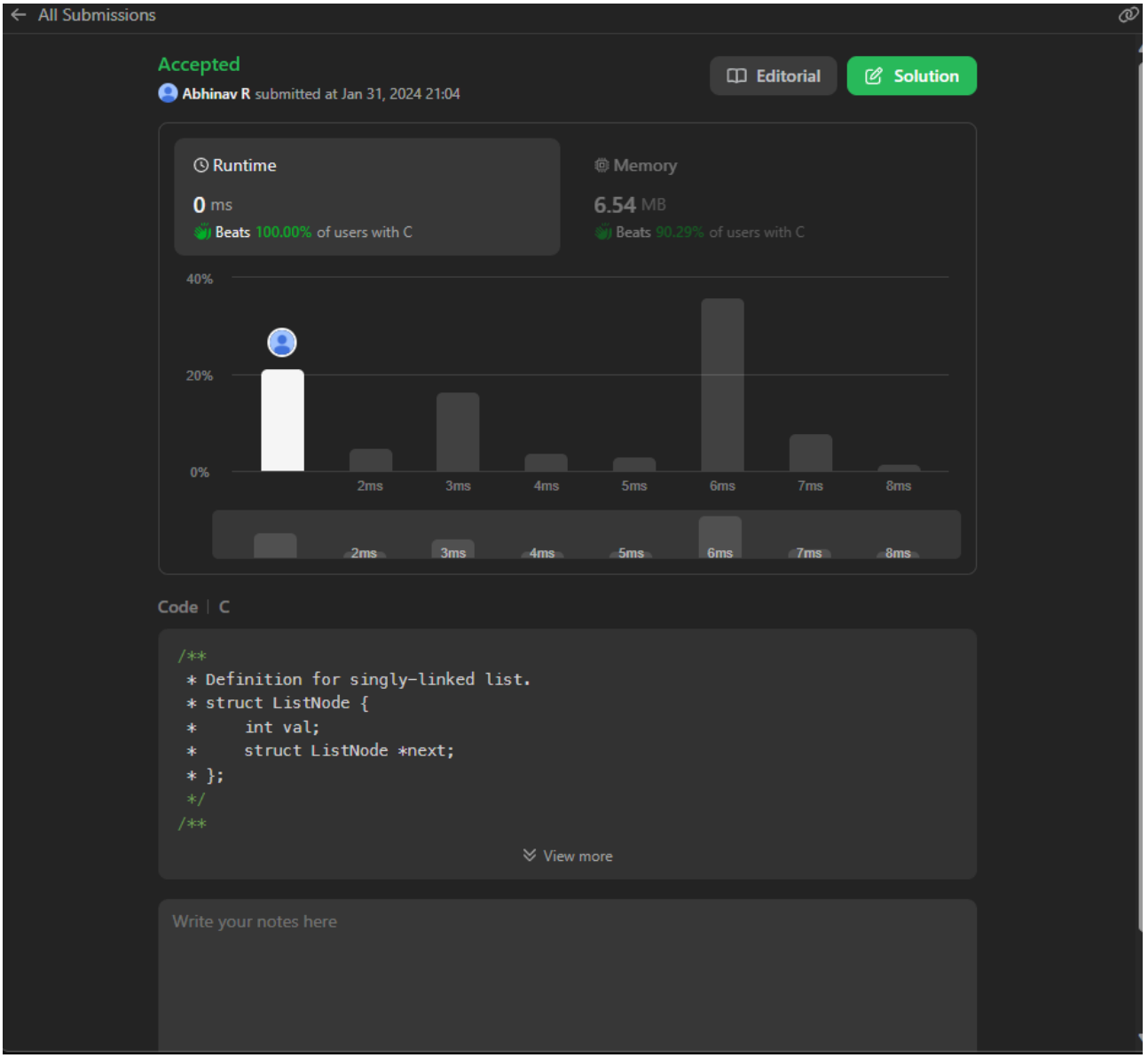
Code:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
```

```
 */
void append(struct ListNode **head, int val){
    struct ListNode* new=(struct ListNode*)malloc(sizeof(struct ListNode));
    struct ListNode* prev=*head;
    new->val=val;
    new->next=NULL;
    if(*head==NULL)
        *head=new;

    else{
        while(prev->next!=NULL)
            prev=prev->next;
        prev->next=new;
    }
}

int length(struct ListNode *head){
    struct ListNode *prev=head;
    int len=0;
    while(prev!=NULL){
        prev=prev->next;
        len++;
    }
    return len;
}

struct ListNode** splitListToParts(struct ListNode* head, int k, int*
returnSize) {
    struct ListNode** heads=(struct ListNode**)malloc(sizeof(struct
ListNode*)*k);
    int len=length(head);
    struct ListNode* prev=head;
    for(int i=0;i<k;i++){
        struct ListNode* nhead=NULL;
        heads[i]=nhead;
    }

    int common=len/k;
    int extra=len%k;

    int iter;
    int i=0;
    while(prev!=NULL){
        for(int j=0;j<common+((extra>0)?1:0);j++){
            append(&heads[i],prev->val);
            prev=prev->next;
        }
        i++;
```

```
        extra--;
    }
    *returnSize=k;
    return heads;
}
```

Output:



Lab 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.

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

typedef struct bst{
    int val;
    struct bst *left;
    struct bst *right;
}node;

node *newNode(int val){
    node *newN=(node*)malloc(sizeof(node));

    newN->left=newN->right=NULL;
    newN->val=val;

    return newN;
}

void insert(node *root,node *temp){
    if(root->val>temp->val){
        if(root->left!=NULL)
            insert(root->left,temp);
        else
            root->left=temp;
    }

    if(root->val<temp->val){
        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->val);
        inorder(root->right);
    }
```

```
}

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

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

}

void main(){
    int choice,val;
    node *root=NULL,*temp;
    while(choice<3){
        printf("1.Insert\n2.Display\nChoice:");
        scanf("%d",&choice);

        switch (choice)
        {
        case 1:
            printf("Enter value to insert:");
            scanf("%d",&val);
            temp=newNode(val);
            if(root==NULL)
                root=temp;
            else
                insert(root,temp);
            break;
        case 2:
            printf("preorder: ");
            preorder(root);
            printf("\n");
            printf("postorder:");
            postorder(root);
            printf("\n");
            printf("inorder:");
            inorder(root);
            printf("\n");
            break;
```

```
        default:
            break;
        }
        printf("---------------\n");
    }

}
```

Output:

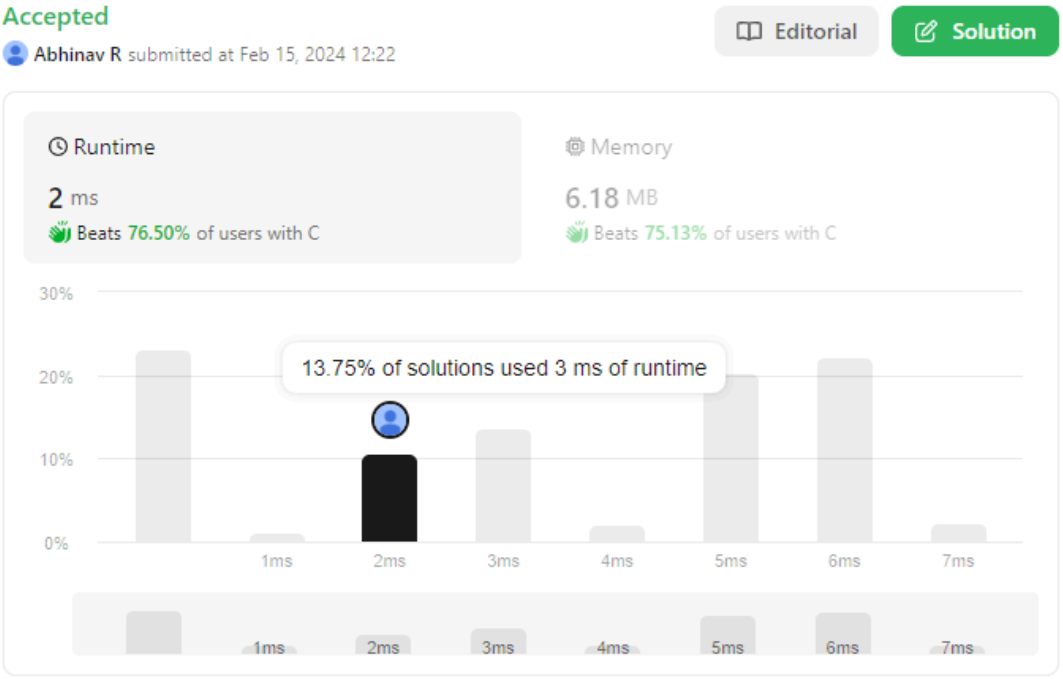
```
PS C:\Users\bmsce\Desktop\1BM22CS211\DS>
1.Insert
2.Display
Choice:1
Enter value to insert:50
---------------
1.Insert
2.Display
Choice:1
Enter value to insert:70
---------------
1.Insert
2.Display
Choice:1
Enter value to insert:60
---------------
1.Insert
2.Display
Choice:1
Enter value to insert:20
---------------
1.Insert
2.Display
Choice:1
Enter value to insert:90
---------------
1.Insert
2.Display
Choice:1
Enter value to insert:10
---------------
1.Insert
2.Display
Choice:1
Enter value to insert:40
---------------
1.Insert
2.Display
Choice:2
preorder: 50 20 10 40 70 60 90
postorder:10 40 20 60 90 70 50
inorder:10 20 40 50 60 70 90
---------------
```

Leetcode
Given the head of a linked list, rotate the list to the right by k places.

Code:

```
struct ListNode* rotateRight(struct ListNode* head, int k) {
    if(head==NULL)
        return head;
    int length=1;
    struct ListNode *temp=head,*temp2;

    while(temp->next!=NULL){
        temp=temp->next;
        length++;
    }
    int rotate = length-(k%length);
    if(rotate==0)
        return head;
    temp->next=head;
    temp=head;
    while(rotate!=1){
        rotate--;
        temp=temp->next;
    }
    temp2=temp->next;
    head=temp2;
    temp->next=NULL;

    return head;

}
```

Output:



Lab 9

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

Code:

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

#define size 7

void push(int a);
int pop();
void display();
void bfs(int graph[][7]);

int fpos = -1, rpos = -1;
int queue[size];

int main(){
   int adj_matrix[7][7] = {
        {0, 1, 0, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 1, 1},
        {0, 1, 0, 1, 1, 1, 0},
        {1, 1, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 1},
```

```
        {0, 1, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 1, 0, 0},
    };
    for(int i = 0; i < 7; i++) queue[i] = NULL;
    // display();
    bfs(adj_matrix);
    return 0;

}

void bfs(int graph[][7]){
    int visited[7];
    for(int i = 0; i < 7; i++) visited[i] = 0;
    push(0); visited[0]= 1;
    while (fpos != size){
        for(int i = 0; i < 7; i++){
            if(graph[queue[fpos]][i] == 1 && visited[i] != 1){
                push(i);
                visited[i] = 1;
                // break;
            }
        }
        printf("%d ", pop());
        // printf("%d\n", new_node);
    }
}

void push(int a){
    if (fpos == -1 && rpos == -1){
        queue[++rpos] = a;
        fpos++;
        return;
    }
    else if (rpos == size-1){
        printf("Queue overflow condition\n");
        return;
    }
    else{
        queue[++rpos] = a;
        return;
    }
}

int pop(){
    if (fpos == -1){
        printf("Queue Underflow condition\n");
    }
    int n = queue[fpos];
```

```
    queue[fpos] = (int) NULL;
    fpos++;
    return n;
}

void display(){
    printf("Queue: ");
    for(int i = 0; i < size; i++)
        printf("%d ", queue[i]);
    printf("\n");
}
```

Output:

```
BFS: 0 1 3 2 5 6 4
PS C:\Users\rabhi\OneDrive\Desktop\C\C project>
```

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

Code:

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

#define size 7
int pos = -1;
int stack[size];

void push(int a);
int pop();
void display();
void dfs(int graph[][7]);

int main(){
    int adj_matrix[7][7] = {
        {0, 1, 0, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 1, 1},
        {0, 1, 0, 1, 1, 1, 0},
        {1, 1, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 1},
        {0, 1, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 1, 0, 0},
    };
    for(int i = 0; i < 7; i++) stack[i] = NULL;
    // display();
    dfs(adj_matrix);
    return 0;


}

void dfs(int graph[][7]){
    int visited[7];
    for (int i = 0; i < 7; i++) visited[i] = 0;
    push(0); visited[0] = 1; printf("0 ");
    // printf("%d ", pos);
    // return;
    // display();
    while(pos != -1){
        bool new_node = false;
        for(int i = 0; i < 7; i++){
            // printf("%d ", graph[stack[pos]][i]);
            if(graph[stack[pos]][i] == 1 && visited[i] != 1){
                new_node = true;
                // printf("Current top: %d\n", i);
```

```
                push(i);
                // display();
                visited[i] = 1; printf("%d ", i);
                break;
            }
        }
        // printf("%d\n", new_node);
        if (!new_node) pop();
    }

}

void push(int a){
    if (pos == size-1){
        printf("Stack Overflow condition");
        return;
    }
    stack[++pos] = a;
}

int pop(){
    if (pos == -1){
        printf("Stack Underflow condition");
        return (int) NULL;
    }
    return stack[pos--];
}

void display(){
    for(int i = 0; i < size; i++){
        printf("%d ", stack[i]);
    }
    printf("\n");
}
```

Output:

```
DFS: 0 1 2 3 4 6 5
PS C:\Users\rabhi\OneDrive\Desktop\C\C project>
```

Lab 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 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.

Code:

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

int h[TABLE_SIZE]={NULL};

void insert()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
    {

     index=(hkey+i)%TABLE_SIZE;

     if(h[index] == NULL)
     {
        h[index]=key;
         break;
     }

    }

    if(i == TABLE_SIZE)

     printf("\nelement cannot be inserted\n");
}
void search()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
```

```
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
    index=(hkey+i)%TABLE_SIZE;
    if(h[index]==key)
    {
      printf("value is found at index %d",index);
      break;
    }
  }
  if(i == TABLE_SIZE)
    printf("\n value is not found\n");
}
void display()
{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE_SIZE; i++)

  printf("\nat index %d \t value =  %d",i,h[i]);

}
main()
{
    int opt,i;
    while(1)
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
    }
}
```

Output:

```
Press
1. Insert
2. Display
3. Search
4.Exit
1
enter a value to insert into hash table
2

Press
1. Insert
2. Display
3. Search
4.Exit
1
enter a value to insert into hash table
3

Press
1. Insert
2. Display
3. Search
4.Exit
1
enter a value to insert into hash table
6
```

```
Press
1. Insert
2. Display
3. Search
4.Exit
2

elements in the hash table are

at index 0       value =  0
at index 1       value =  0
at index 2       value =  2
at index 3       value =  3
at index 4       value =  0
at index 5       value =  0
at index 6       value =  6
at index 7       value =  0
at index 8       value =  0
at index 9       value =  0
Press
1. Insert
2. Display
3. Search
4.Exit
3

enter search element
2
value is found at index 2
```