

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Data Structures using C

Submitted by

SETU MISHRA (1BM22CS250)

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
December-2023 to 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



CERTIFICATE

This is to certify that the Lab work entitled “**Data Structures using C**” carried out by **SETU MISHRA (1BM22CS250)**, who is bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester December-2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Data Structures using C (23CS3PCDST)** work prescribed for the said degree.

Name of the Lab-In charge: Dr.Gauri Kalnoor
(Designation)
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	<p>Write a program to simulate the working of stack using an array with the following : a) Push b) Pop c) Display</p> <p>The program should print appropriate messages for stack overflow, stack underflow</p>	5-8
2	<p>a) 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)</p> <p>b) Demonstration of account creation on LeetCode platform Program - Leetcode platform</p>	9-15
3	<p>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</p> <p>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</p>	16-27
4	<p>4a) WAP to Implement Singly Linked List with following operations</p> <p style="padding-left: 40px;">a) Create a linked list.</p> <p style="padding-left: 40px;">b) Insertion of a node at first position, at any position and at end of list.</p> <p>Display the contents of the linked list.</p> <p>4b) Program - Leetcode platform</p>	28-40
5	<p>5a) WAP to Implement Singly Linked List with following operations</p> <p style="padding-left: 40px;">a) Create a linked list.</p> <p style="padding-left: 40px;">b) Deletion of first element, specified element and last element in the list.</p> <p>Display the contents of the linked list.</p>	41-54

	5b) Program - Leetcode platform	
6	<p>6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.</p> <p>6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.</p>	55-68
7	<p>7a) WAP to Implement doubly link list with primitive operations</p> <ol style="list-style-type: none"> Create a doubly linked list. Insert a new node to the left of the node. Delete the node based on a specific value <p>Display the contents of the list</p> <p>7b) Program - Leetcode platform</p>	69-79
8	<p>8a) Write a program</p> <ol style="list-style-type: none"> To construct a binary Search tree. To traverse the tree using all the methods i.e., in-order, preorder and post order <p>To display the elements in the tree.</p> <p>8b) Program - Leetcode platform</p>	80-85
9	<p>9a) Write a program to traverse a graph using BFS method.</p> <p>9b) Write a program to check whether given graph is connected or not using DFS method.</p>	86-95
10	<p>Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.</p> <p>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.</p> <p>Let the keys in K and addresses in L are integers.</p> <p>Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K)=K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L.</p> <p>Resolve the collision (if any) using linear probing.</p>	96-100

Course Outcome

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyse data structure operations for a given problem.
CO3	CO3 Design and implement operations of linear and nonlinear data structure.
CO4	Conduct practical experiments for demonstrating the operations of different data structures and sorting techniques.

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 n 5
```

```
int stac[n];
```

```
int top=-1;
```

```
void push(){
```

```
    if( top>=n){
```

```
        printf("stack is full, overflow\n");
```

```
        return;
```

```
    }
```

```
    top++;
```

```

    int item;

    printf("enter the number to be inserted\t");

    scanf("%d",&item);

    stac[top]=item;
}

void pop(){
    if( top== -1){
        printf("stack is empty, underflow\n");
        return;
    }

    int data=stac[top];
    printf("removing the %d element\t",top);
    printf("removing %d\n",data);
    top--;
}

void display(){
    int i;
    printf("the given stac is :\n");
    for(i=top;i>=0;i--){
        printf("%d\t",stac[i]);
    }
}

int main()

```

```

{
    int c;
    while(1){
        printf("enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for
        exiting\n");
        printf("enter your choice\t");
        scanf("%d",&c);
        if(c==1){
            push();
        }
        else if(c==2){
            pop();
        }
        else if(c==3){
            display();
        }
        else if(c==4) {
            printf("Exiting!");
            exit(0);
        }else{
            printf("Invalid choice,please enter a valid choice!\n");
        }
    }
    return 0;
}

```

}

```
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
enter the number to be inserted 12
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
enter the number to be inserted 23
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
enter the number to be inserted 34
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
enter the number to be inserted 56
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
enter the number to be inserted 89
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
enter the number to be inserted 34
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      1
stack is full, overflow
```

```
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
removing the 5 element  removing 34
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
removing the 4 element  removing 89
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
removing the 3 element  removing 56
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
removing the 2 element  removing 34
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
removing the 1 element  removing 23
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
removing the 0 element  removing 12
enter 1 for push, 2 for pop and 3 for displaying the stack and 4 for exiting
enter your choice      2
stack is empty, underflow
```


Program 2:

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

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

```
#include<ctype.h>
```

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

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

```
#define MAX 100
```

```
char st[MAX];
```

```
int top = -1;
```

```
void push (char st[], char);
```

```
char pop (char st[]);
```

```
void InfixtoPostfix (char source[], char target[]);
```

```
int getpri (char);
```

```
int
```

```
main ()
```

```
{
```

```
    char infix[100], postfix[100];
```

```
    printf ("Enter any infix expression:\n");
```

```

gets (infix);
strcpy (postfix, "");
InfixtoPostfix (infix, postfix);
printf ("The corresponding postfix expression is:\n");
puts (postfix);

return 0;
}

```

```

void
InfixtoPostfix (char source[], char target[])
{
    int i = 0, j = 0;
    char temp;
    strcpy (target, "");

    while (source[i] != '\0')
    {
        if (source[i] == '(')
        {
            push (st, source[i]);
            i++;
        }
        else if (source[i] == ')')
        {

```

```

while ((top != -1) && (st[top] != '('))
{
    target[j] = pop (st);
    j++;
}
if (top == -1 || st[top] != '(')
{
    printf ("\nIncorrect expression");
    exit (1);
}
temp = pop (st);
i++;
}
else if (isdigit (source[i]) || isalpha (source[i]))
{
    target[j] = source[i];
    j++;
    i++;
}
else if (source[i] == '+' || source[i] == '-' || source[i] == '*'
        || source[i] == '/' || source[i] == '^')
{
    while ((top != -1) && (st[top] != '(')
            && (getpri (st[top]) >= getpri (source[i])))
    {

```

```

        target[j] = pop (st);
        j++;
    }
    push (st, source[i]);
    i++;
}
else
{
    printf ("\nIncorrect element in expression");
    exit (1);
}
}

while ((top != -1) && (st[top] != '('))
{
    target[j] = pop (st);
    j++;
}
target[j] = '\0';
}

int
getpri (char op)
{
    if (op == '^')

```

```

    return 2;
else if ((op == '/') || (op == '*'))
    return 1;
else if ((op == '+') || (op == '-'))
    return 0;
else
    return -1;
}

```

void

push (char st[], char val)

```

{
    if (top == MAX - 1)
    {
        printf ("\n Stack overflow");
    }
else
    {
        top++;
        st[top] = val;
    }
}

```

char

pop (char st[])

```
{  
    char val = ' ';  
    if (top == -1)  
        printf ("\nStack underflow");  
    else  
    {  
        val = st[top];  
        top--;  
    }  
    return val;  
}
```

```
Enter any infix expression:  
(a+(b*c/d)^e-f  
The corresponding postfix expression is:  
abc*d/e^+f-
```

```
Enter any infix expression:  
(a*(b+c/(d+e)^f))  
The corresponding postfix expression is:  
abcde+f^/+*
```

Program 3:

3)a)3a) WAP to simulate the working of a queue of integers using an array. Provide

the following operations: Insert, Delete, Display

The program should print appropriate messages for queue empty and queue

overflow conditions

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

```
#define MAX_SIZE 5
```

```
int queue[MAX_SIZE];
```

```
int front = -1, rear = -1;
```

```
void insert(int value) {
```

```
    if (rear == MAX_SIZE - 1) {  
        printf("Queue Overflow\n");
```

```
    } else {
```

```
        if (front == -1)
```

```
            front = 0;
```

```
        rear++;
```

```
        queue[rear] = value;
```

```
        printf("%d inserted into the queue\n", value);
```

```
    }
```



```
}
```

```
void delete() {  
    if (front == -1 || front > rear) {  
        printf("Queue Underflow\n");  
    } else {  
        printf("%d deleted from the queue\n", queue[front]);  
        front++;  
    }  
}
```

```
void display() {  
    if (front == -1 || front > rear) {  
        printf("Queue is empty\n");  
    } else {  
        printf("Queue elements: ");  
        for (int i = front; i <= rear; i++)  
            printf("%d ", queue[i]);  
        printf("\n");  
    }  
}
```

```
int main() {  
    int choice, value;
```

```

do {
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &value);
            insert(value);
            break;
        case 2:
            delete();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
    }
}

```

```
    }  
    } while (choice != 4);  
  
    return 0;  
}
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 12
12 inserted into the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 13
13 inserted into the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 14
14 inserted into the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 15
15 inserted into the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 16
16 inserted into the queue
1. Insert
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 16
16 inserted into the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 34
Queue Overflow
1. Insert
```

```
Enter your choice: 2
12 deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
13 deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
14 deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
15 deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
16 deleted from the queue
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue Underflow
```

3)b)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 MAX_SIZE 5
```

```
int queue[MAX_SIZE];
```

```
int front = -1, rear = -1;
```

```
// Function to check if the queue is empty
```

```
int isEmpty() {
```

```
    return front == -1 && rear == -1;
```

```
}
```

```
// Function to check if the queue is full
```

```
int isFull() {
```

```
    return (rear + 1) % MAX_SIZE == front;
```

```
}
```

```
// Function to insert an element into the circular queue
```

```
void insert(int item) {
```

```

if (isFull()) {
    printf("Queue Overflow: Cannot insert element %d\n", item);
    return;
}

if (isEmpty()) {
    front = rear = 0;
} else {
    rear = (rear + 1) % MAX_SIZE;
}

queue[rear] = item;
printf("Element %d inserted successfully\n", item);
}

// Function to delete an element from the circular queue
void delete() {
    if (isEmpty()) {
        printf("Queue Underflow: Cannot delete from an empty queue\n");
        return;
    }

    printf("Element %d deleted successfully\n", queue[front]);

    if (front == rear) {

```



```

        front = rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }
}

// Function to display the elements of the circular queue
void display() {
    if (isEmpty()) {
        printf("Queue is empty\n");
        return;
    }

    printf("Elements in the queue: ");
    int i = front;
    do {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX_SIZE;
    } while (i != (rear + 1) % MAX_SIZE);

    printf("\n");
}

int main() {
    insert(1);

```

```
insert(2);
insert(3);
insert(4);
insert(5);

// Uncomment the line below to test queue overflow condition
// insert(6);

display();

delete();
delete();
delete();

// Uncomment the line below to test queue underflow condition
// delete();

display();

insert(6);
insert(7);

display();

return 0;
```

}

```
Element 1 inserted successfully
Element 2 inserted successfully
Element 3 inserted successfully
Element 4 inserted successfully
Element 5 inserted successfully
Elements in the queue: 1 2 3 4 5
Element 1 deleted successfully
Element 2 deleted successfully
Element 3 deleted successfully
Elements in the queue: 4 5
Element 6 inserted successfully
Element 7 inserted successfully
Elements in the queue: 4 5 6 7
```

```
Element 10 inserted successfully
Element 20 inserted successfully
Element 30 inserted successfully
Element 40 inserted successfully
Element 50 inserted successfully
Elements in the queue: 10 20 30 40 50
Element 10 deleted successfully
Element 20 deleted successfully
Element 30 deleted successfully
Elements in the queue: 40 50
Element 60 inserted successfully
Element 77 inserted successfully
Elements in the queue: 40 50 60 77
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Program 4:

4a) 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.

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

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

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node *next;
```

```
}Node;
```

```
void InsertAtBeginning( Node **head_ref,int new_data);
```

```
void InsertAtEnd( Node **head_ref,int new_data);
```

```
void Insert( Node **prev_node,int new_data,int pos);
```

```
void PrintList(Node * next);
```

```
void InsertAtBeginning( Node **head_ref,int new_data)
```

```
{
```

```
    Node *new_node=(struct Node*)malloc(sizeof( Node));
```

```

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

```

```

void InsertAtEnd(Node **head_ref,int new_data)
{
    Node *new_node=(struct Node*)malloc(sizeof( Node));
    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 Insert(Node **head_ref,int new_data,int pos)
{
    if (*head_ref ==NULL)

```

```

{
    printf("Cannot be NULL\n");
    return;
}
Node *temp = *head_ref;
Node *newNode = ( Node *) malloc (sizeof ( Node));
newNode->data = new_data;
newNode->next = NULL;

while (--pos>0)
{
    temp = temp->next;
}

newNode->next = temp->next;
temp->next = newNode;
}

```

```

void PrintList(Node *node)
{
    while (node!=NULL)
    {
        printf("%d\n",node->data);
        node=node->next;
    }
}

```

```
}
```

```
int main()
```

```
{
```

```
    int ch,new,pos;
```

```
    Node* head=NULL;
```

```
    while(ch!=5)
```

```
    {
```

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

```
        printf("1.Insert at beginning\n");
```

```
        printf("2.Insert at a specific position\n");
```

```
        printf("3.Insert at end\n");
```

```
        printf("4.Display linked list\n");
```

```
        printf("5.Exit\n");
```

```
        printf("Enter your choice\n");
```

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

```
        switch(ch)
```

```
        {
```

```
            case 1:
```

```
            {
```

```
                printf("Enter the data you want to insert at beginning\n");
```

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

```
                InsertAtBeginning(&head,new);
```

```
                break;
```

```

}
case 2:
{
printf("Enter the data and position at which you want to insert \n");
scanf("%d%d",&new,&pos);
Insert(&head,new,pos);
break;
}
case 3:
{
printf("Enter the data you want to insert at end\n");
scanf("%d",&new);
InsertAtEnd(&head,new);
break;
}
case 4:
{
printf("Created linked list is:\n");
PrintList(head);
break;
}
case 5:
{
return 0;
break;
}

```



```
    }  
    case 6:  
    {  
        printf("Invalid data!");  
        break;  
    }  
    }  
}  
return 0;  
}
```

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
12
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
23
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
3
Enter the data you want to insert at end
34
Menu
```

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
2
Enter the data and position at which you want to insert
99 1
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
23
99
12
34
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
5

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
1
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
2
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
3
Enter the data you want to insert at end
4
```

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
2
Enter the data and position at which you want to
7
1
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
2
7
1
4
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
5

...Program finished with exit code 0
Press ENTER to exit console.
```

4)b) Program - Leetcode platform

225. Implement Stack using Queues Solved ✓

Easy Topics Companies

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

Implement the `MyStack` class:

- `void push(int x)` Pushes element `x` to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only push to back, peek/pop from front, size and is empty operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

```
1 struct queueNode {
2     int data;
3 };
4
5
6 typedef struct {
7     struct queueNode *arr[101];
8     int front;
9     int rear;
10 } MyStack;
11
12
13 MyStack* myStackCreate() {
14     MyStack *q = (MyStack *)malloc(sizeof(MyStack));
15     q->front = -1;
16     q->rear = -1;
17     return q;
18 }
19
20 void enqueue(MyStack* q, struct queueNode* item) {
21     q->arr[++q->rear] = item;
22     if (q->front == -1) {
```

Accepted Runtime: 0 ms

Case 1

```
void enqueue(MyStack* q, struct queueNode* item) {
    q->arr[++q->rear] = item;
    if (q->front == -1) {
        q->front = 0;
    }
}

struct queueNode* dequeue(MyStack* q) {
    struct queueNode* item = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return item;
}

void myStackPush(MyStack* q, int x) {
    struct queueNode* node = (struct queueNode*) malloc(sizeof(struct queueNode));
    node->data = x;
    enqueue(q, node);
    int size = q->rear - q->front + 1;
    while (size > 1) {
        struct queueNode* front = dequeue(q);
```

```
44     while (size > 1) {
45         struct queueNode* front = dequeue(q);
46         enqueue(q, front);
47         size--;
48     }
49 }
50
51 int myStackPop(MyStack* q) {
52     struct queueNode* front = dequeue(q);
53     int item = front->data;
54     free(front);
55     return item;
56 }
57
58 int myStackTop(MyStack* q) {
59     struct queueNode* front = q->arr[q->front];
60     return front->data;
61 }
62
63 bool myStackEmpty(MyStack* q) {
64     return q->front == -1;
65 }
66
67 void myStackFree(MyStack* q) {
68     while (!myStackEmpty(q)) {
69         myStackPop(q);
70     }
71 }
```

```

bool myStackEmpty(MyStack* q) {
    return q->front == -1;
}

void myStackFree(MyStack* q) {
    while (!myStackEmpty(q)) {
        struct queueNode* front = dequeue(q);
        free(front);
    }
    free(q);
}

```

Problem List < > >

Description Editorial Solutions Submissions

All Submissions

Accepted

Setu submitted at Jan 08, 2024 12:43

Editorial Solution

Runtime

2 ms

Beats 56.19% of users with C

Memory

6.74 MB

Beats 9.06% of users with C

60%

40%

20%

0%

1ms

2ms

1ms

2ms

Code

Saved to local

C v Auto

Testcase Test Result

Accepted Runtime: 0 ms

Case 1

Input

["MyStack", "push", "push", "top", "pop", "empty"]

[[], [1], [2], [], [], []]

Output

[null, null, null, 2, 2, false]

Expected

[null, null, null, 2, 2, false]

Contribute a testcase

Program 5:

5a) 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.

Display the contents of the linked list.

```
//deletion in LL
```

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

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

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node *next;
```

```
}Node;
```

```
void InsertAtBeginning( Node **head_ref,int new_data);
```

```
void DeleteAtBeginning( Node **head_ref);
```

```
void DeleteAtEnd( Node **head_ref);
```

```
void Delete( Node **prev_node,int pos);
```

```
void PrintList(Node * next);
```

```

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

```

```

void DeleteAtBeginning( Node **head_ref)
{
    Node *ptr;
    if(head_ref == NULL)
    {
        printf("\nList is empty");
    }
    else
    {
        ptr = *head_ref;
        *head_ref = ptr->next;
        free(ptr);
        printf("\n Node deleted from the beginning ...");

    }

}

```

```

void DeleteAtEnd(Node **head_ref)
{
    Node *ptr,*ptr1;

    if(*head_ref == NULL)

    {

        printf("\nlist is empty");

    }

    else if((*head_ref)-> next == NULL)

    {

        free(*head_ref);

        *head_ref= NULL;

        printf("\nOnly node of the list deleted ...");

    }

```

```
else

{

ptr = *head_ref;

while(ptr->next != NULL)

{

ptr1 = ptr;

ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

printf("\n Deleted Node from the last ...");

}
```

```

}

void Delete(Node **head_ref, int pos)
{
    Node *temp = *head_ref, *prev;

    if (temp == NULL)
    {
        printf("\nList is empty");
        return;
    }

    if (pos == 0)
    {
        *head_ref = temp->next;
        free(temp);
        printf("\nDeleted node with position %d", pos);
        return;
    }

    for (int i = 0; temp != NULL && i < pos - 1; i++)
    {
        prev = temp;
        temp = temp->next;
    }
}

```

```

    if (temp == NULL)
    {
        printf("\nPosition out of range");
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("\nDeleted node with position %d", pos);
}

void PrintList(Node *node)
{
    while (node!=NULL)
    {
        printf("%d\n",node->data);
        node=node->next;
    }
}

int main()
{
    int ch,new,pos;
    Node* head=NULL;
    while(ch!=6)

```

```

{
printf("Menu\n");
printf("1.Create a linked list\n");
printf("2.Delete at beginning\n");
printf("3.Delete at a specific position\n");
printf("4..Delete at end\n");
printf("5..Display linked list\n");
printf("6..Exit\n");
printf("Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
    case 1:
    {
        printf("Enter the data you want to insert at beginning\n");
        scanf("%d",&new);
        InsertAtBeginning(&head,new);
        break;
    }
    case 2:
    {
        DeleteAtBeginning(&head);
        break;
    }
    case 3:

```

```

{
    printf("Enter the position at which you want to delete \n");
    scanf("%d",&pos);
    Delete(&head,pos);
    break;
}
case 4:
{
    DeleteAtEnd(&head);
    break;
}
case 5:
{
    printf("Created linked list is:\n");
    PrintList(head);
    break;
}
case 6:
{
    return 0;
    break;
}
default:
{
    printf("Invalid data!");

```



```
        break;
    }
}
}
return 0;
}
```

```

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
22
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
45
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
45
22
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
56
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
4

```

```

Deleted Node from the last ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
56
45
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
2
Node deleted from the beginning ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
45
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
6
...Program finished with exit code 0
Press ENTER to exit console.

```

End of document

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
12
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
23
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
34
Menu
1.Create a linked list
2.Delete at beginning
```

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
34
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
34
23
12
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
3
Enter the position at which you want to delete
1
```

5)b) Program - Leetcode platform

Problem List < > 🔍

Description Editorial Solutions Submissions

725. Split Linked List in Parts Solved

Medium Topics Companies Hint

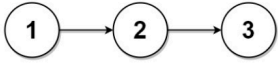
Given the `head` of a singly linked list and an integer `k`, split the linked list into `k` consecutive linked list parts.

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.

Example 1:



Input: head = [1,2,3], k = 5
Output: [[1], [2], [3], null, null]

3.9K 49 ☆ 📌 🔄

</> Code

C Auto

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8 /**
9  * Note: The returned array must be malloced, assume caller calls free().
10 */
11 struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize){
12     /**
13      * Input:
14      * head,
15      * k
16      */
17     /**
18      * Algorithm:
19      * Length(list) = 50, k = 7
20      * 50 / 7 = 7 ... 1
21      * First 6 parts should contain (7 + 1) nodes
22      */
23 }
```

Saved to local Ln 78, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Description Editorial Solutions Submissions

</> Code

C Auto

```
9  * Note: The returned array must be malloced, assume caller calls free().
10 */
11 struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize){
12     /**
13      * Input:
14      * head,
15      * k
16      */
17     /**
18      * Algorithm:
19      * Length(list) = 50, k = 7
20      * 50 / 7 = 7 ... 1
21      * First 6 parts should contain (7 + 1) nodes
22      */
23 }
```

Restored from local Upgrade to Cloud Saving Ln 12, Col 5

Testcase Test Result

Code

C

Auto

```
32
33
34     prev->next = NULL;
35
36 }
37
38 if (base) {
39     for (int i = part; i < k; i++) {
40
41         ans[i] = head;
42         part++;
43
44         for (int j = 0; j < base; j++) {
45             prev = head;
46             head = head->next;
47         }
48
49         prev->next = NULL;
50     }
51 }
52
53 /*
54  * Output:
55  * *returnSize
56  * Return an array of the k parts.
57  */
58 *returnSize = k;
59
60 Saved to local
```

Ln 12, Col 5

Testcase Test Result

Code

C

Auto

```
42     part++;
43
44     for (int j = 0; j < base; j++) {
45         prev = head;
46         head = head->next;
47     }
48
49     prev->next = NULL;
50 }
51
52
53 /*
54  * Output:
55  * *returnSize
56  * Return an array of the k parts.
57  */
58 *returnSize = k;
59
60 return ans;
61
62
63
64 Saved to local
```

Problem List

Run

Submit

Premium

Description

Editorial

Solutions

Submissions

All Submissions

Accepted

Setu submitted at Mar 03, 2024 00:27

Editorial

Solution

Runtime

6 ms

Beats 37.43% of users with C

Memory

6.54 MB

Beats 50.88% of users with C

Runtime (ms)	Percentage of Solutions
1	~22%
2	~10%
3	~10%
4	5.85%
5	~8%
6	~25%
7	~5%

Code

C

Auto

```
42     part++;
43
44     for (int j = 0; j < base; j++) {
45         prev = head;
46         head = head->next;
47     }
48
49     prev->next = NULL;
50 }
51
52
53 /*
54  * Output:
55  * *returnSize
56  * Return an array of the k parts.
57  */
58 *returnSize = k;
59
60 return ans;
61
62
63
64 Saved to local
```

Ln 12, Col 5

Testcase Test Result

Program 6:

6a) WAP to Implement Single Link List with following operations: Sort the linked

list, Reverse the linked list, Concatenation of two linked lists.

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

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

```
struct Node {
```

```
    int data;
```

```
    struct Node *next;
```

```
};
```

```
// Function to create a new node
```

```
struct Node* createNode(int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    if (newNode == NULL) {
```

```
        printf("Memory allocation failed\n");
```

```
        exit(1);
```

```
    }
```

```
    newNode->data = data;
```

```
    newNode->next = NULL;
```

```
    return newNode;
```

```
}
```

// Function to insert a node at the end of the linked list

```
void insertEnd(struct Node **head, int data) {  
    struct Node *newNode = createNode(data);  
    if (*head == NULL) {  
        *head = newNode;  
        return;  
    }  
    struct Node *temp = *head;  
    while (temp->next != NULL) {  
        temp = temp->next;  
    }  
    temp->next = newNode;  
}
```

// Function to display the linked list

```
void display(struct Node *head) {  
    if (head == NULL) {  
        printf("Linked list is empty\n");  
        return;  
    }  
    struct Node *temp = head;
```



```

while (temp != NULL) {
    printf("%d -> ", temp->data);
    temp = temp->next;
}
printf("NULL\n");
}

// Function to sort the linked list in ascending order
void sortLinkedList(struct Node **head) {
    if (*head == NULL || (*head)->next == NULL)
        return;

    struct Node *current = *head;
    struct Node *index = NULL;
    int temp;

    while (current != NULL) {
        index = current->next;
        while (index != NULL) {
            if (current->data > index->data) {
                temp = current->data;
                current->data = index->data;
            }
        }
    }
}

```

```

        index->data = temp;
    }
    index = index->next;
}
current = current->next;
}
}

```

```

// Function to reverse the linked list
void reverseLinkedList(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;
}

```

```
// Function to concatenate two linked lists
void concatenateLinkedLists(struct Node **head1, struct Node *head2)
{
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    struct Node *temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}
```

```
int main() {
    struct Node *list1 = NULL;
    struct Node *list2 = NULL;
    int n, data;

    printf("Enter the number of elements for the first linked list: ");
    scanf("%d", &n);
    printf("Enter the elements for the first linked list:\n");
```

```
for (int i = 0; i < n; i++) {  
    scanf("%d", &data);  
    insertEnd(&list1, data);  
}
```

```
printf("Enter the number of elements for the second linked list: ");  
scanf("%d", &n);  
printf("Enter the elements for the second linked list:\n");  
for (int i = 0; i < n; i++) {  
    scanf("%d", &data);  
    insertEnd(&list2, data);  
}
```

```
printf("First linked list: ");  
display(list1);  
printf("Second linked list: ");  
display(list2);
```

```
// Sorting first linked list  
sortLinkedList(&list1);  
printf("First linked list after sorting: ");  
display(list1);
```

```
// Reversing second linked list
reverseLinkedList(&list2);
printf("Second linked list after reversing: ");
display(list2);

// Concatenating two linked lists
concatenateLinkedLists(&list1, list2);
printf("Concatenated linked list: ");
display(list1);

return 0;
}
```

```
Enter the number of elements for the first linked list: 4
Enter the elements for the first linked list:
1
2
3
4
Enter the number of elements for the second linked list: 4
Enter the elements for the second linked list:
5
6
7
8
First linked list: 1 -> 2 -> 3 -> 4 -> NULL
Second linked list: 5 -> 6 -> 7 -> 8 -> NULL
First linked list after sorting: 1 -> 2 -> 3 -> 4 -> NULL
Second linked list after reversing: 8 -> 7 -> 6 -> 5 -> NULL
Concatenated linked list: 1 -> 2 -> 3 -> 4 -> 8 -> 7 -> 6 -> 5 -> NULL

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter the number of elements for the first linked list: 3
Enter the elements for the first linked list:
44
55
66
Enter the number of elements for the second linked list: 3
Enter the elements for the second linked list:
77
88
99
First linked list: 44 -> 55 -> 66 -> NULL
Second linked list: 77 -> 88 -> 99 -> NULL
First linked list after sorting: 44 -> 55 -> 66 -> NULL
Second linked list after reversing: 99 -> 88 -> 77 -> NULL
Concatenated linked list: 44 -> 55 -> 66 -> 99 -> 88 -> 77 -> NULL

...Program finished with exit code 0
Press ENTER to exit console.
```

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

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

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

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

```
struct Node* createNode(int data) {  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    if (newNode == NULL) {  
        printf("Memory allocation failed\n");  
        exit(1);  
    }  
    newNode->data = data;  
    newNode->next = NULL;  
    return newNode;  
}
```

```
void push(struct Node **top, int data) {  
    struct Node *newNode = createNode(data);  
    newNode->next = *top;  
    *top = newNode;
```

```
}
```

```
int pop(struct Node **top) {  
    if (*top == NULL) {  
        printf("Stack Underflow\n");  
        exit(1);  
    }  
    struct Node *temp = *top;  
    int data = temp->data;  
    *top = (*top)->next;  
    free(temp);  
    return data;  
}
```

```
void enqueue(struct Node **front, struct Node **rear, int data) {  
    struct Node *newNode = createNode(data);  
    if (*rear == NULL) {  
        *front = *rear = newNode;  
    } else {  
        (*rear)->next = newNode;  
        *rear = newNode;  
    }  
}
```

```
int dequeue(struct Node **front, struct Node **rear) {
```



```

if (*front == NULL) {
    printf("Queue Underflow\n");
    exit(1);
}
struct Node *temp = *front;
int data = temp->data;
if (*front == *rear) {
    *front = *rear = NULL;
} else {
    *front = (*front)->next;
}
free(temp);
return data;
}

```

```

void displayStack(struct Node *top) {
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements: ");
    while (top != NULL) {
        printf("%d ", top->data);
        top = top->next;
    }
}

```

```

    printf("\n");
}

void displayQueue(struct Node *front) {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    while (front != NULL) {
        printf("%d ", front->data);
        front = front->next;
    }
    printf("\n");
}

```

```

int main() {
    struct Node *stackTop = NULL;
    struct Node *queueFront = NULL;
    struct Node *queueRear = NULL;

    // Pushing elements onto the stack
    push(&stackTop, 10);
    push(&stackTop, 20);
    push(&stackTop, 30);
}

```

```

printf("After pushing onto stack:\n");
displayStack(stackTop);

// Popping elements from the stack
printf("Popped element from stack: %d\n", pop(&stackTop));
printf("After popping from stack:\n");
displayStack(stackTop);

// Enqueueing elements into the queue
enqueue(&queueFront, &queueRear, 10);
enqueue(&queueFront, &queueRear, 20);
enqueue(&queueFront, &queueRear, 30);
printf("After enqueueing into queue:\n");
displayQueue(queueFront);

// Dequeueing elements from the queue
printf("Dequeued element from queue: %d\n", dequeue(&queueFront,
&queueRear));
printf("After dequeuing from queue:\n");
displayQueue(queueFront);

return 0;
}

```

```
After pushing onto stack:  
Stack elements: 30 20 10  
Popped element from stack: 30  
After popping from stack:  
Stack elements: 20 10  
After enqueueing into queue:  
Queue elements: 10 20 30  
Dequeued element from queue: 10  
After dequeuing from queue:  
Queue elements: 20 30  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

```
After pushing onto stack:  
Stack elements: 33 22 11  
Popped element from stack: 33  
After popping from stack:  
Stack elements: 22 11  
After enqueueing into queue:  
Queue elements: 11 22 33  
Dequeued element from queue: 11  
After dequeuing from queue:  
Queue elements: 22 33  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Program 7:

7a) 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 *prev;  
    struct Node *next;  
};
```

```
// Function to create a new node
```

```
struct Node* createNode(int data) {  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    if (newNode == NULL) {  
        printf("Memory allocation failed\n");  
        exit(1);  
    }  
    newNode->data = data;  
    newNode->prev = NULL;
```

```

    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node to the left of the given node
void insertLeft(struct Node **head, struct Node *givenNode, int data) {
    struct Node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    if (givenNode == NULL) {
        printf("Given node is NULL\n");
        exit(1);
    }
    if (givenNode->prev != NULL) {
        givenNode->prev->next = newNode;
        newNode->prev = givenNode->prev;
    }
    givenNode->prev = newNode;
    newNode->next = givenNode;
    if (givenNode == *head) {
        *head = newNode;
    }
}

```

// Function to delete a node based on a specific value

```
void deleteNode(struct Node **head, int key) {  
    if (*head == NULL) {  
        printf("List is empty\n");  
        return;  
    }  
    struct Node *temp = *head;  
    while (temp != NULL && temp->data != key) {  
        temp = temp->next;  
    }  
    if (temp == NULL) {  
        printf("Node with key %d not found\n", key);  
        return;  
    }  
    if (temp->prev != NULL) {  
        temp->prev->next = temp->next;  
    }  
    if (temp->next != NULL) {  
        temp->next->prev = temp->prev;  
    }  
    if (temp == *head) {  
        *head = temp->next;  
    }  
    free(temp);  
}
```

```
}
```

```
// Function to display the contents of the list
```

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

```
    if (head == NULL) {
```

```
        printf("List is empty\n");
```

```
        return;
```

```
    }
```

```
    printf("List contents: ");
```

```
    while (head != NULL) {
```

```
        printf("%d ", head->data);
```

```
        head = head->next;
```

```
    }
```

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

```
}
```

```
int main() {
```

```
    struct Node *head = NULL;
```

```
    int choice, value, key;
```

```
    do {
```

```
        printf("\n1. Create a doubly linked list\n");
```

```
        printf("2. Insert a new node to the left of the given node\n");
```

```
        printf("3. Delete the node based on a specific value\n");
```

```
        printf("4. Display the contents of the list\n");
```



```

printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the value for the first node: ");
        scanf("%d", &value);
        head = createNode(value);
        break;
    case 2:
        printf("Enter the value to be inserted: ");
        scanf("%d", &value);
        printf("Enter the value of the given node: ");
        scanf("%d", &key);
        insertLeft(&head, head, value);
        break;
    case 3:
        printf("Enter the value to be deleted: ");
        scanf("%d", &key);
        deleteNode(&head, key);
        break;
    case 4:
        display(head);
        break;
}

```

```
    case 5:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice\n");
    }
} while (choice != 5);

return 0;
}
```

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit

Enter your choice: 1

Enter the value for the first node: 22

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit

Enter your choice: 2

Enter the value to be inserted: 34

Enter the value of the given node: 22

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit

Enter your choice: 4

List contents: 34 22

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit

Enter your choice: 2

Enter the value to be inserted: 56

Enter the value of the given node: 34

```
1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
List contents: 56 34 22
```

```
1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 5
Exiting...
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

```
1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 1
Enter the value for the first node: 23

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 2
Enter the value to be inserted: 45
Enter the value of the given node: 23

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
List contents: 45 23

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 3
Enter the value to be deleted: 45
```

```
1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
List contents: 23

1. Create a doubly linked list
2. Insert a new node to the left of the given node
3. Delete the node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 5
Exiting...

...Program finished with exit code 0
Press ENTER to exit console.
```

7b) Program - Leetcode platform

Description | Editor | Solutions | Submissions
</> Code

All Submissions
C v Auto

Accepted

Setu submitted at Mar 04, 2024 00:04

Runtime
 3 ms
 Beats 52.84% of users with C

Memory
 5.77 MB
 Beats 74.29% of users with C

```

1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     struct TreeNode *left;
6  *     struct TreeNode *right;
7  * };
8  */
9  struct TreeNode* invertTree(struct TreeNode* root){
10
11     // Base case...
12     if(root == NULL){
13         return root;
14     }
15
16     invertTree(root->left);
17     invertTree(root->right);
18
19     struct TreeNode* curr = root->left;
20     root->left = root->right;
21     root->right = curr;
22     return root;
23 }
```

Code | C

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
```

Saved to local
Ln 23, Col 2

Testcase
Test Result

Program 8:

8a) 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

To display the elements in the tree.

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

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

```
struct TreeNode {
```

```
    int data;
```

```
    struct TreeNode* left;
```

```
    struct TreeNode* right;
```

```
};
```

```
struct TreeNode* createNode(int data) {
```

```
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct  
TreeNode));
```

```
    newNode->data = data;
```

```
    newNode->left = NULL;
```

```
    newNode->right = NULL;
```

```
    return newNode;
```

```
}
```

```
struct TreeNode* insertNode(struct TreeNode* root, int data) {
```

```
    if (root == NULL) {
```

```
        root = createNode(data);
```



```

    } else if (data <= root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}

```

```

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

```

```

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

```

```

void postorderTraversal(struct TreeNode* root) {

```

```

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

```

```

void display(struct TreeNode* root) {
    printf("Inorder traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Preorder traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Postorder traversal: ");
    postorderTraversal(root);
    printf("\n");
}

```

```

int main() {
    struct TreeNode* root = NULL;
    root = insertNode(root, 10);
    root = insertNode(root, 5);
}

```

```
root = insertNode(root, 15);  
root = insertNode(root, 3);  
root = insertNode(root, 7);  
root = insertNode(root, 12);  
root = insertNode(root, 18);  
display(root);  
  
return 0;  
}
```

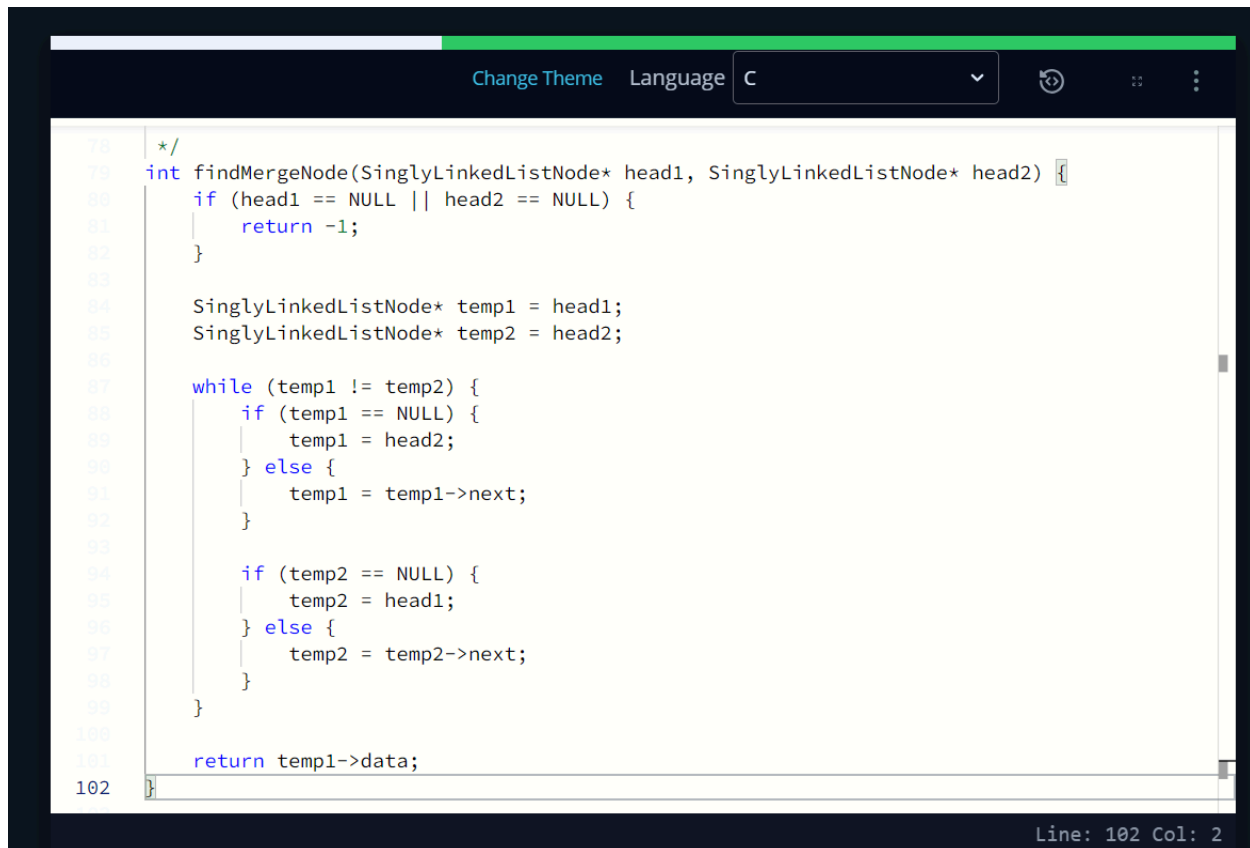
```
Inorder traversal: 3 5 7 10 12 15 18  
Preorder traversal: 10 5 3 7 15 12 18  
Postorder traversal: 3 7 5 12 18 15 10
```

```
...Program finished with exit code 0  
Press ENTER to exit console.
```

```
Inorder traversal: 7 14 15 17 18 57 71  
Preorder traversal: 14 7 57 17 15 18 71  
Postorder traversal: 7 15 18 17 71 57 14
```

```
...Program finished with exit code 0  
Press ENTER to exit console.
```

8)b) Program - Leetcode platform



```
78  */
79  int findMergeNode(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {
80      if (head1 == NULL || head2 == NULL) {
81          return -1;
82      }
83
84      SinglyLinkedListNode* temp1 = head1;
85      SinglyLinkedListNode* temp2 = head2;
86
87      while (temp1 != temp2) {
88          if (temp1 == NULL) {
89              temp1 = head2;
90          } else {
91              temp1 = temp1->next;
92          }
93
94          if (temp2 == NULL) {
95              temp2 = head1;
96          } else {
97              temp2 = temp2->next;
98          }
99      }
100
101      return temp1->data;
102 }
```

Line: 102 Col: 2



You have earned 5.00 points!

You are now 20 points away from the 1st star for your problem solving badge.

33%

10/30

Congratulations

You solved this challenge. Would you like to challenge your friends?



Next Challenge

✓ Test case 0

✓ Test case 1

✓ Test case 2

✓ Test case 3

✓ Test case 4

✓ Test case 5

✓ Test case 6

Compiler Message

Success

Input (stdin)

Download

```
1 1
2 1
3 3
4 1
5 2
6 3
7 1
8 1
```

Program 9:

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

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

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

```
#define MAX_VERTICES 100
```

```
// Structure to represent a node in the graph
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
// Structure to represent the graph
```

```
struct Graph {
```

```
    int numVertices;
```

```
    struct Node* adjLists[MAX_VERTICES];
```

```
    int visited[MAX_VERTICES];
```

```
};
```

```
// Function to create a new graph
```

```
struct Graph* createGraph(int vertices) {
```

```
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
```

```
    graph->numVertices = vertices;
```

```

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = dest;
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Since it's an undirected graph, add an edge from dest to src as well
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = src;
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to perform BFS traversal

```

```

void BFS(struct Graph* graph, int startVertex) {
    // Create a queue for BFS
    int queue[MAX_VERTICES];
    int front = -1, rear = -1;

    // Mark the current node as visited and enqueue it
    graph->visited[startVertex] = 1;
    queue[++rear] = startVertex;

    while (front != rear) {
        // Dequeue a vertex from queue and print it
        int currentVertex = queue[++front];
        printf("%d ", currentVertex);

        // Get all adjacent vertices of the dequeued vertex currentVertex
        // If an adjacent vertex has not been visited, then mark it visited and enqueue
        it
        struct Node* temp = graph->adjLists[currentVertex];
        while (temp) {
            int adjVertex = temp->data;
            if (!graph->visited[adjVertex]) {
                graph->visited[adjVertex] = 1;
                queue[++rear] = adjVertex;
            }
            temp = temp->next;
        }
    }
}

```



```

    }
}
}

```

```

int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices and edges: ");
    scanf("%d %d", &numVertices, &numEdges);

    struct Graph* graph = createGraph(numVertices);

    printf("Enter the edges (src dest):\n");
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    printf("BFS traversal starting from vertex %d: ", startVertex);
    BFS(graph, startVertex);
    printf("\n");
}

```

```
    return 0;  
}
```

```
Enter the number of vertices and edges: 4  
3  
Enter the edges (src dest):  
3  
4  
5  
6  
7  
8  
Enter the starting vertex for BFS: 3  
BFS traversal starting from vertex 3: 3 4  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

```
Enter the number of vertices and edges: 3  
2  
Enter the edges (src dest):  
1  
2  
3  
4  
Enter the starting vertex for BFS: 2  
BFS traversal starting from vertex 2: 2 1  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

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

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

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

```
#define MAX_VERTICES 100
```

```
// Structure to represent a node in the graph
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
// Structure to represent the graph
```

```
struct Graph {
```

```
    int numVertices;
```

```
    struct Node* adjLists[MAX_VERTICES];
```

```
    int visited[MAX_VERTICES];
```

```
};
```

```
// Function to create a new graph
```

```
struct Graph* createGraph(int vertices) {
```

```
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
```

```
    graph->numVertices = vertices;
```

```

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = dest;
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Since it's an undirected graph, add an edge from dest to src as well
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = src;
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to perform DFS traversal
void DFS(struct Graph* graph, int vertex) {

```

```

graph->visited[vertex] = 1;

struct Node* temp = graph->adjLists[vertex];
while (temp != NULL) {
    int adjVertex = temp->data;
    if (!graph->visited[adjVertex]) {
        DFS(graph, adjVertex);
    }
    temp = temp->next;
}
}

// Function to check if the graph is connected
int isConnected(struct Graph* graph) {
    // Perform DFS traversal starting from vertex 0
    DFS(graph, 0);

    // Check if all vertices are visited after DFS traversal
    for (int i = 0; i < graph->numVertices; i++) {
        if (!graph->visited[i]) {
            return 0; // Graph is not connected
        }
    }
    return 1; // Graph is connected
}

```

```

int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices and edges: ");
    scanf("%d %d", &numVertices, &numEdges);

    struct Graph* graph = createGraph(numVertices);

    printf("Enter the edges (src dest):\n");
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    if (isConnected(graph)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}

```

Enter the number of vertices and edges: 3

2

Enter the edges (src dest):

1

2

3

4

The graph is not connected.

...Program finished with exit code 0

Press ENTER to exit console.

Enter the number of vertices and edges: 4

3

Enter the edges (src dest):

4

5

6

7

8

9

The graph is not connected.

...Program finished with exit code 0

Press ENTER to exit console.

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 $H: K \rightarrow L$ 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>
```

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

```
#define M 100 // number of memory locations (m)
```

```
#define N 1000 // number of employee records (N)
```

```
#define K 10000 // number of keys (K)
```

```
typedef struct {
```

```
    int id;
```

```
    char name[50];
```



```

    float salary;
} Employee;

typedef struct Node {
    int key;
    Employee emp;
    struct Node *next;
} Node;

Node *HT[M]; // Hash Table (HT) of m memory locations

// Hash function to map key K to address space L
int hash(int key) {
    return key % M;
}

// Function to insert employee record into Hash Table
void insert(Employee emp) {
    int key = emp.id;
    int index = hash(key);
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->key = key;
    newNode->emp = emp;
    newNode->next = NULL;

```

```

if (HT[index] == NULL) {
    HT[index] = newNode;
} else {
    Node *temp = HT[index];
    while (temp->next != NULL) {
        if (temp->key == key) {
            printf("Collision detected! Updating employee record...\n");
            temp->emp = emp;
            free(newNode);
            return;
        }
        temp = temp->next;
    }
    if (temp->key == key) {
        printf("Collision detected! Updating employee record...\n");
        temp->emp = emp;
        free(newNode);
        return;
    }
    temp->next = newNode;
}
}

```

// Function to search employee record using key K

```
void search(int key) {
```

```

int index = hash(key);

Node *temp = HT[index];

while (temp != NULL) {

    if (temp->key == key) {

        printf("Employee Record Found:\n");

        printf("ID: %d\n", temp->emp.id);

        printf("Name: %s\n", temp->emp.name);

        printf("Salary: %.2f\n", temp->emp.salary);

        return;

    }

    temp = temp->next;

}

printf("Employee Record Not Found!\n");

}

```

```

int main() {

    // Insert employee records into Hash Table

    for (int i = 0; i < N; i++) {

        Employee emp;

        emp.id = i + 1;

        sprintf(emp.name, "Employee %d", i + 1);

        emp.salary = (float)(i + 1) * 1000;

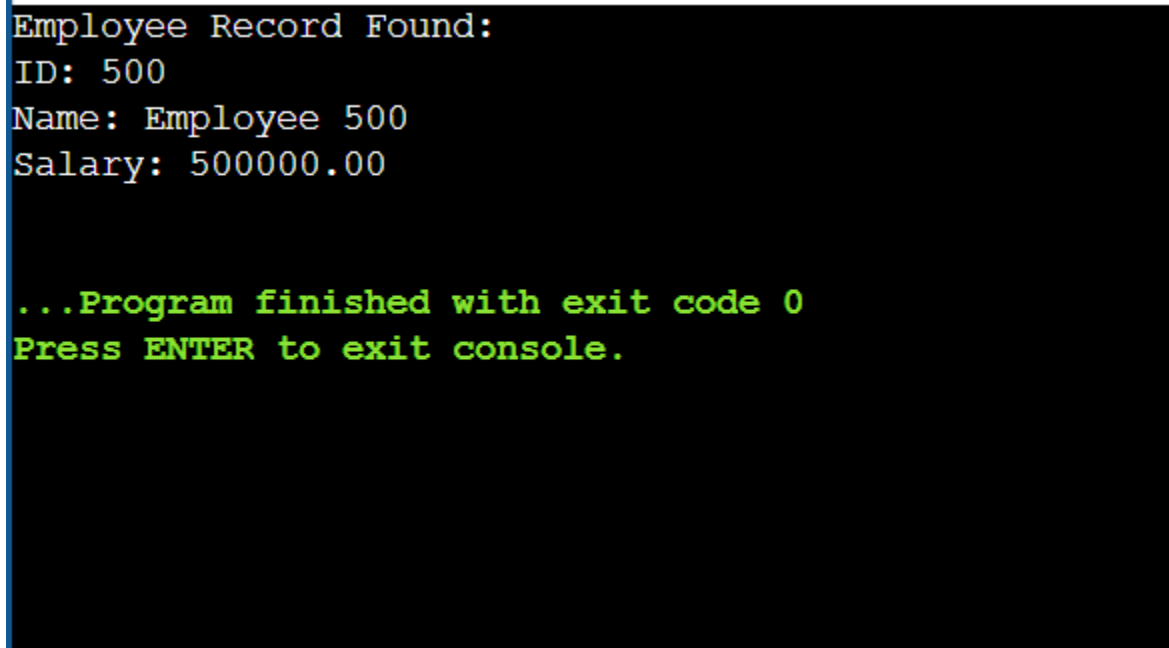
        insert(emp);

    }

}

```

```
// Search employee record using key K  
  
int key = 500;  
  
search(key);  
  
return 0;  
}
```



A screenshot of a console window with a black background and white text. The text displays the output of a program search function. It shows 'Employee Record Found:' followed by 'ID: 500', 'Name: Employee 500', and 'Salary: 500000.00'. Below this, in green text, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.'.

```
Employee Record Found:  
ID: 500  
Name: Employee 500  
Salary: 500000.00  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```