**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**Anagha D K (1BM23CS032)**

**in partial fulfillment for the award of the degree of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**September 2024-January 2025**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

**(Affiliated To Visvesvaraya Technological University, Belgaum)**

**Department of Computer Science and Engineering**

****

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by Anagha D K **(1BM23CS032)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

**Prof. Basavaraj J** **Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Stack Simulation | 4 |
| 2 | Infix to Postfix  Leetcode: Next Greatest Integer | 7 |
| 3 | Queue implementation  Circular Queue | 17 |
| 4 | Linked List implementation  Leetcode: Recent calls counter | 23 |
| 5 | Singly Linked list creation  Leetcode | 31 |
| 6 | Sort, reverse and concatenate linked lists  Stack and queue implementation using linked list | 43 |
| 7 | Doubly linked list | 60 |
| 8 | Binary search tree with traversal | 68 |
| 9 | Graph traversal:BFS  DFS | 74 |
| 10 | Hashing | 80 |

**Course outcomes:**

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

**Lab program 1:**

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

**a) Push**

**b) Pop**

**c) Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

#include <stdio.h>

#include<stdlib.h>

#define STACK\_SIZE 5

void push(int st[],int \*top)

{

int item;

if(\*top==STACK\_SIZE-1)

printf("Stack overflow\n");

else

{

printf("\nEnter an item :");

scanf("%d",&item);

(\*top)++;

st[\*top]=item;

}

}

void pop(int st[],int \*top)

{

if(\*top==-1)

printf("Stack underflow\n");

else

{

printf("\n%d item was deleted",st[(\*top)--]);

}

}

void display(int st[],int \*top)

{

int i;

if(\*top==-1)

printf("Stack is empty\n");

for(i=0;i<=\*top;i++)

printf("%d\t",st[i]);

}

void main()

{

int st[10],top=-1, c,val\_del;

while(1)

{

printf("\n1. Push\n2. Pop\n3. Display\n");

printf("\nEnter your choice :");

scanf("%d",&c);

switch(c)

{

case 1: push(st,&top);

break;

case 2: pop(st,&top);

break;

case 3: display(st,&top);

break;

default: printf("\nInvalid choice!!!");

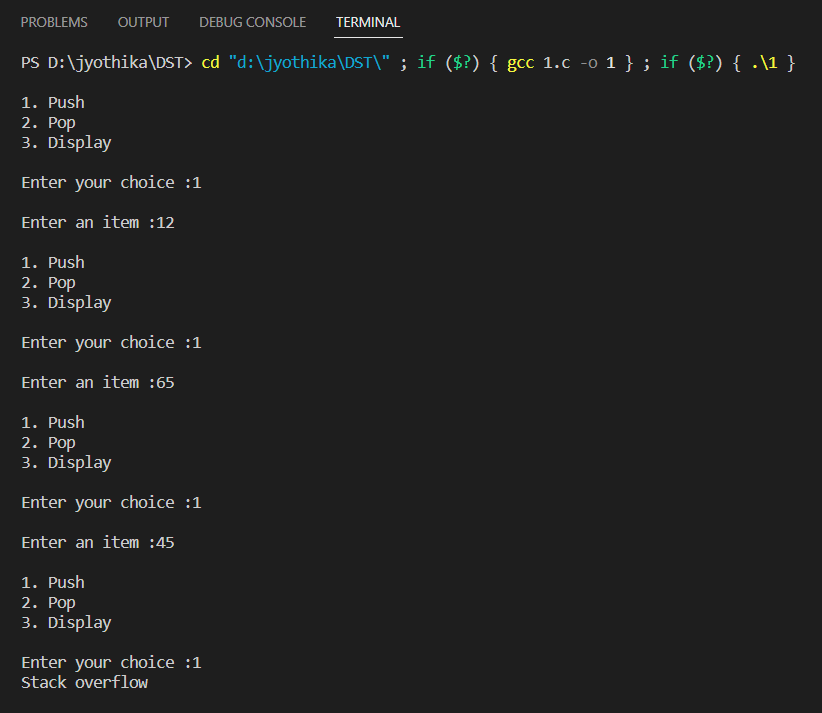
exit(0);

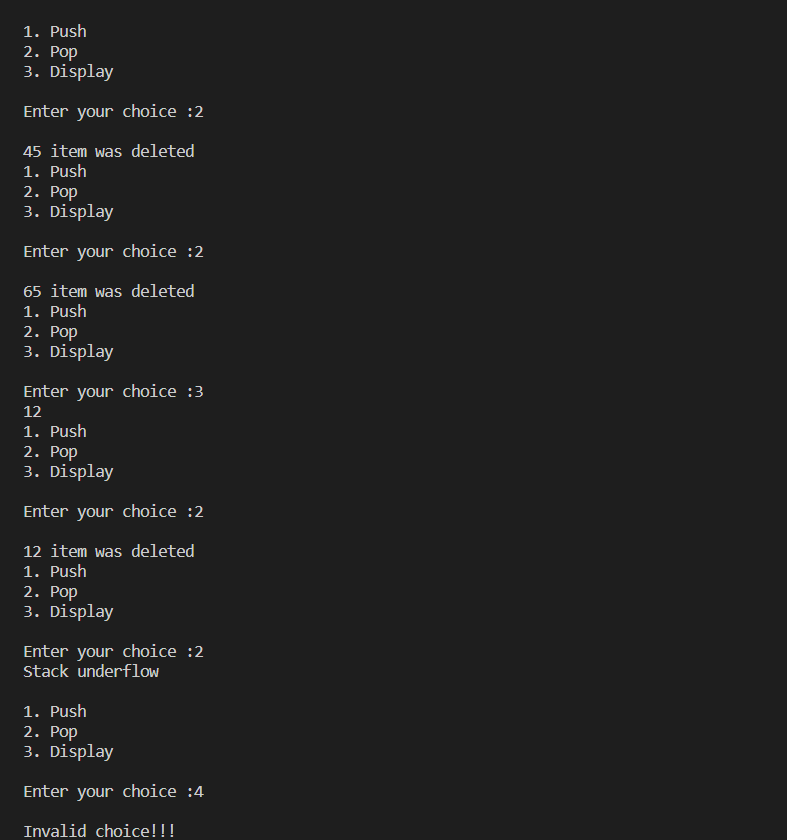
}

}

}

**Output:**





**Lab program 2**

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

**Code:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int ind=0,pos=0,top=-1,len;

char infix[20],postfix[20],stack[20],symbol,temp;

void intopost();

void push(char);

char pop();

int prec(char);

void main()

{

printf("Enter the infix expression: ");

scanf("%s",infix);

intopost();

printf("infix expression is: %s",infix);

printf("postfix expression is: %s",postfix);

}

void intopost()

{

len=strlen(infix);

push('#');

while(ind<len)

{

symbol=infix[ind];

switch (symbol)

{

case '(': push(symbol);

break;

case ')': temp=pop();

while(temp!='(')

{

postfix[pos++]=temp;

temp=pop();

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^': while(prec(stack[top])>=prec(symbol))

{

temp=pop();

postfix[pos++]=temp;

}

push(symbol);

break;

default: postfix[pos++]=symbol;

break;

}

ind++;

}

while(top>0)

{

temp=pop();

postfix[pos++]=temp;

}

}

void push(char symbol)

{

top=top+1;

stack[top]=symbol;

}

char pop()

{

char sym;

sym=stack[top];

top=top-1;

return sym;

}

int prec(char symbol)

{

int p;

switch(symbol)

{

case '^':p=3;

break;

case '\*':

case '/':p=2;

break;

case '+':

case '-':p=1;

break;

case '(':p=0;

break;

case '#':p=-1;

break;

}

return p;

}

A black background with white text

Description automatically generated

**LEETCODE next greater integer:**

The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return *an array*ans*of length*nums1.length*such that*ans[i]*is the next greater element as described above.*

Code:

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int\* nextGreaterElement(int\* nums1, int nums1Size, int\* nums2, int nums2Size,

int\* returnSize)

{

int\* ans=malloc(nums1Size\*sizeof(int));

for (int i = 0; i < nums1Size; i++)

{

ans[i]=-1;

for (int j = 0; j < nums2Size; j++)

{

if (nums1[i] == nums2[j])

{

for (int k=j+1; k < nums2Size; k++)

{

if (nums2[k] > nums1[i])

{

ans[i] = nums2[k];

break;

}

}

break;

}

}

}

\*returnSize = nums1Size;

return ans;

}

**Lab program 3**

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

**a) Insert**

**b) Delete**

**c) Display**

**Code:**

#include<stdio.h>

#include<stdlib.h>

# define MAX 3

int queue[20],rear=-1,front=-1;

void insert();

int del();

void display();

void main()

{

int ch,temp;

while(1)

{

printf("Enter your choice:\n 1:Insert\n2:Delete\n3:Display\n4:Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1: insert();

break;

case 2: if(rear==-1 && front==-1)

{

printf("Underflow\n");

}

else

{

temp=del();

printf("Deleted element is %d\n",temp);

}

break;

case 3: display();

break;

case 4: exit(0);

break;

default: printf("Invalid Choice\n");

break;

}

}

}

void insert()

{

int ele;

if(rear==MAX-1)

{

printf("Overflow");

return;

}

printf("Enter the element to be inserted:\n");

scanf("%d",&ele);

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

{

rear=0;

front=0;

}

else

{

rear++;

}

queue[rear]=ele;

return;

}

int del()

{

int t;

t=queue[front];

if(rear==front)

{

rear=-1;

front=-1;

}

else

{

front++;

}

return t;

}

void display()

{

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

{

printf("No elements to display\n");

return;

}

printf("Content is:\n");

for(int i=front;i<=rear;i++)

{

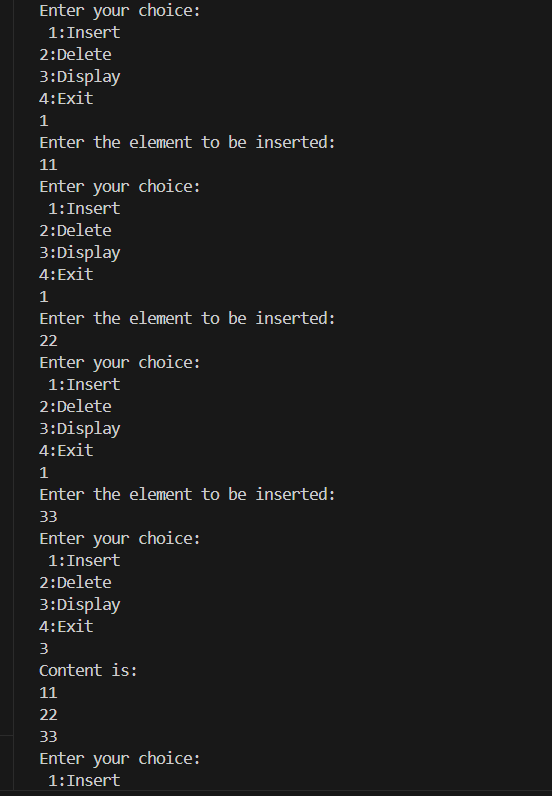
printf("%d\n",queue[i]);

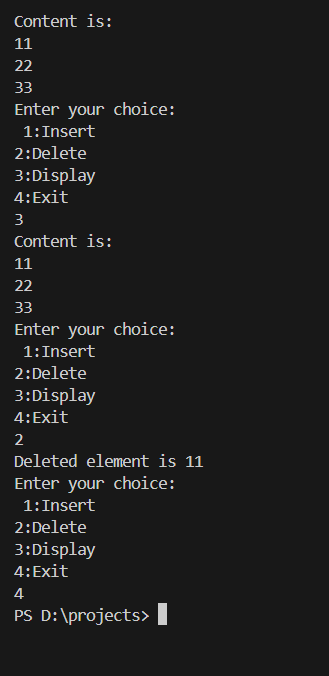
}

return;

}

Output:





**LAB program 3**

**Circular Queue implementation**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

void insert();

int del();

void display();

int front=-1,rear=-1,queue[20];

void main()

{

int ch,t;

while(1)

{

printf("Enter your choice: \n1:Insert\n2:Delete\n3:Display\n4:Exit");

scanf("\n%d",&ch);

switch(ch)

{

case 1:insert();

break;

case 2:

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

{

printf("Underflow\n");

}

else

{

t=del();

printf("Deleted element is %d",t);

}

break;

case 3:display();

break;

case 4:exit(0);

break;

default: printf("Invalid Choice\n");

break;

}

}

}

void insert()

{

int no;

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

{

printf("Circular Queue is full");

return;

}

printf("Enter the element to be inserted: ");

scanf("%d",&no);

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

{

rear=0;

front=0;

}

else

{

rear=((rear+1)%MAX);

}

queue[rear]=no;

return;

}

int del()

{

int temp;

temp=queue[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

front=((front+1)%MAX);

}

return temp;

}

void display()

{

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

{

printf("No elements to display");

return;

}

printf("Content is \n");

if(front<=rear)

{

for(int i=front;i<=rear;i++)

{

printf("%d\n",queue[i]);

}

}

else

{

for(int i=front;i<=MAX-1;i++)

{

printf("%d\n",queue[i]);

}

for(int i=0;i<=rear;i++)

{

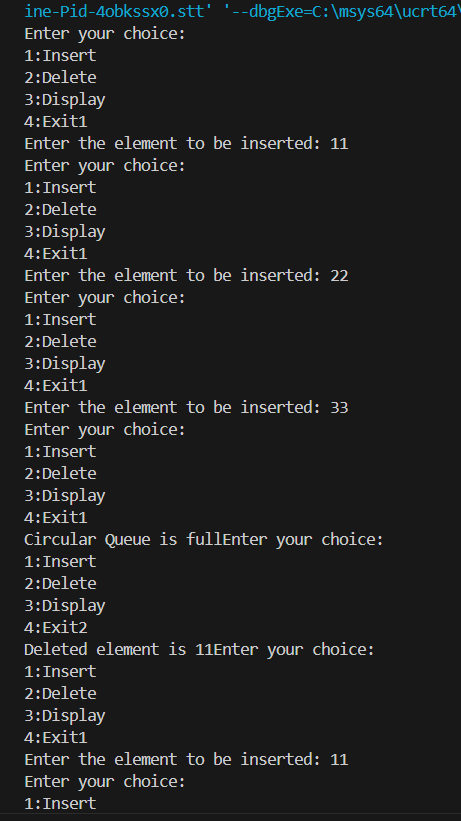
printf("%d\n",queue[i]);

}

}

}

Output:



A screenshot of a computer program

Description automatically generated

**Lab program 4**

**WAP to Implement Singly Linked List with following operations a) Createalinkedlist. 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>

void display();

void insertbeg();

void insertend();

void insertpos();

struct Node {

int data;

struct Node\* link;

};

typedef struct Node node;

node \*new1,\*start = NULL;

int ch;

void main() {

while (1) {

printf("Enter your Choice:\n 1: Insert at the end\n 2: Insert at the beginning\n 3: Insert at a particular position\n 4: Display\n 5: Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1: insertend();

break;

case 2: insertbeg();

break;

case 3: insertpos();

break;

case 4: display();

break;

case 5: exit(0);

break;

default: printf("Invalid choice\n");

break;

}

}

}

void insertend() {

new1 = (node\*)malloc(sizeof(node));

if (new1 == NULL) {

printf("Memory allocation failed\n");

return;

}

printf("Enter value: ");

scanf("%d", &new1->data);

if (start == NULL) {

start = new1;

new1->link = NULL;

return;

}

node\* temp=start;

while(temp!=NULL) {

temp=temp->link;

}

new1->link=NULL;

temp->link=new1;

}

void insertbeg() {

new1 = (node\*)malloc(sizeof(node));

printf("Enter value: ");

scanf("%d", &new1->data);

if (start == NULL) {

start = new1;

new1->link=NULL;

return;

}

new1->link=start;

start=new1;

}

void insertpos() {

int pos;

new1 = (node\*)malloc(sizeof(node));

printf("Enter value: ");

scanf("%d", &new1->data);

printf("Enter position: ");

scanf("%d", &pos);

if(pos==1) {

new1->link=start;

start=new1;

return;

}

int i=1;

node\* temp=start;

while(temp!=NULL) {

temp=temp->link;

i++;

}

if(temp==NULL) {

printf("Enter position greater than the number of elements: ");

return;

}

new1->link=temp->link;

temp->link=new1;

}

void display() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

node\* temp = start;

printf("Elements of the linked list are: ");

while (temp != NULL) {

printf("%d", temp->data);

temp = temp->link;

if (temp != NULL) {

printf(" -> ");

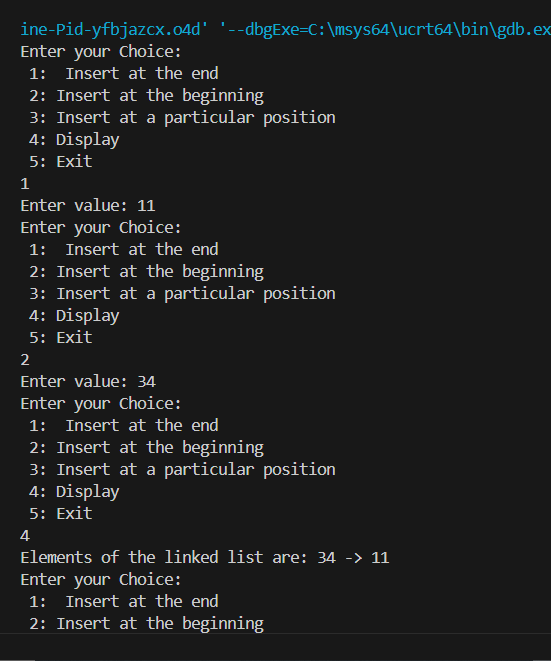
}

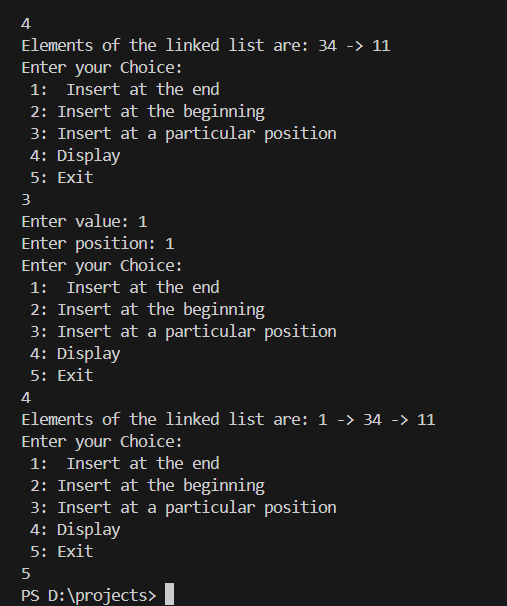
}

printf("\n");

}

Output:



****

**LEETCODE**

**You have a RecentCounter class which counts the number of recent requests within a certain time frame.**

**Implement the RecentCounter class:**

* **RecentCounter() Initializes the counter with zero recent requests.**
* **int ping(int t) Adds a new request at time t, where t represents some time in milliseconds, and returns the number of requests that has happened in the past 3000 milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range [t - 3000, t].**

**It is guaranteed that every call to ping uses a strictly larger value of t than the previous call.**

**Code:**

typedef struct {

int callqueue[10000];

int front;

int rear;

} RecentCounter;

RecentCounter\* recentCounterCreate() {

RecentCounter\* obj = (RecentCounter\*)malloc(sizeof(RecentCounter));

obj->front = 0;

obj->rear = -1;

return obj;

}

int recentCounterPing(RecentCounter\* obj, int t) {

obj->callqueue[++obj->rear]=t;

while (obj->callqueue[obj->front] < t - 3000) {

obj->front++;

}

return obj->rear - obj->front + 1;

}

void recentCounterFree(RecentCounter\* obj) {

free(obj);

}

/\*\*

\* Your RecentCounter struct will be instantiated and called as such:

\* RecentCounter\* obj = recentCounterCreate();

\* int param\_1 = recentCounterPing(obj, t);

\* recentCounterFree(obj);

\*/

**Lab program 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>

void display();

void delend();

void insertend();

void delpos();

void delbeg();

struct Node {

int data;

struct Node\* link;

};

typedef struct Node node;

node\* new1, \*start = NULL;

int ch;

void main() {

while (1) {

printf("Enter your Choice:\n");

printf("1: Insert at the end\n");

printf("2: Delete at the end\n");

printf("3: Delete at the beginning\n");

printf("4: Delete at a position\n");

printf("5: Display\n");

printf("6: Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1:

insertend();

break;

case 2:

delend();

break;

case 3:

delbeg();

break;

case 4:

delpos();

break;

case 5:

display();

break;

case 6:

exit(0);

break;

default:

printf("Invalid choice\n");

break;

}

}

}

void insertend() {

new1 = (node\*)malloc(sizeof(node));

if (new1 == NULL) {

printf("Memory allocation failed\n");

return;

}

printf("Enter value: ");

scanf("%d", &new1->data);

new1->link = NULL;

if (start == NULL) {

start = new1;

return;

}

node\* temp = start;

while (temp->link != NULL) {

temp = temp->link;

}

temp->link = new1;

}

void delbeg() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

node\* temp = start;

start = start->link;

free(temp);

}

void delend() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

if (start->link == NULL) {

free(start);

start = NULL;

return;

}

node\* temp = start;

while (temp->link != NULL && temp->link->link != NULL) {

temp = temp->link;

}

free(temp->link);

temp->link = NULL;

}

void delpos() {

int pos;

printf("Enter the position: ");

scanf("%d", &pos);

if (start == NULL || pos <= 0) {

printf("Invalid position or empty list\n");

return;

}

node\* temp = start;

node\* prev = NULL;

int count = 1;

while (temp != NULL && count < pos) {

prev = temp;

temp = temp->link;

count++;

}

if (temp == NULL) {

printf("Position out of range\n");

return;

}

if (prev == NULL) {

start = temp->link;

} else {

prev->link = temp->link;

}

free(temp);

}

void display() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

node\* temp = start;

printf("Elements of the linked list are: ");

while (temp != NULL) {

printf("%d", temp->data);

temp = temp->link;

if (temp != NULL) {

printf(" -> ");

}

}

printf("\n");

}

Output:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Leetcode**

**Given a circular integer array nums of length n, return *the maximum possible sum of a non-empty subarray of*nums.**

**A circular array means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].**

**A subarray may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.**

**Code:**

int kadane(int\* nums, int numsSize, int findMax) {

    int curr = nums[0];

    int result = nums[0];

    for (int i = 1; i < numsSize; i++) {

        if (findMax) {

            curr = nums[i] > curr + nums[i] ? nums[i] : curr + nums[i];

            result = result > curr ? result : curr;

        } else {

            curr = nums[i] < curr + nums[i] ? nums[i] : curr + nums[i];

            result = result < curr ? result : curr;

        }

    }

    return result;

}

int maxSubarraySumCircular(int\* nums, int numsSize) {

    int maxKadane = kadane(nums, numsSize, 1); // Maximum subarray sum without wrapping

    int minKadane = kadane(nums, numsSize, 0); // Minimum subarray sum

    int total = 0;

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

        total += nums[i];

    }

    // Special case: if all elements are negative, return maxKadane (non-wrapping)

    if (maxKadane < 0) {

        return maxKadane;

    }

    return maxKadane > (total - minKadane) ? maxKadane : (total - minKadane);

}

**LAB program 6**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*link;

};

typedef struct Node node;

node \*ptr, \*start2 = NULL, \*start = NULL, \*new1;

void concatenate()

{

printf("Concatenating Lists\n");

if(start == NULL && start2 == NULL)

{

printf("Both Linked Lists are empty\n");

return;

}

if(start == NULL)

{

start = start2;

display(start);

return;

}

if(start2 == NULL)

{

display(start);

return;

}

ptr = start;

while(ptr->link != NULL)

{

ptr = ptr->link;

}

ptr->link = start2;

start2 = NULL;

display(start);

}

void reverse()

{

printf("Reversing the list\n");

if(start == NULL)

{

printf("Linked list is empty, cannot reverse\n");

return;

}

node \*a = start, \*b = NULL, \*c = NULL;

while(a != NULL)

{

c = b;

b = a;

a = a->link;

b->link = c;

}

start = b;

display(start);

}

node \*curr;

void createA()

{

int c = 1;

while(c == 1)

{

new1 = (node \*) malloc(sizeof(node));

if (new1 == NULL) {

printf("Memory allocation failed\n");

return;

}

printf("Enter data: ");

scanf("%d", &new1->data);

if(start == NULL)

{

start = new1;

curr = new1;

}

else

{

curr->link = new1;

curr = new1;

}

printf("Create another element?(1: Yes) ");

scanf("%d",&c);

}

curr->link = NULL;

}

void sort()

{

printf("Sorting the list\n");

if(start == NULL)

{

printf("Linked list is empty, cannot sort\n");

return;

}

int len = 0, itr = 0, swapped;

for(ptr = start; ptr != NULL; ptr = ptr->link)

{

len++;

}

while (itr < len)

{

node \*traverse = start;

node \*prev = start;

swapped = 0;

while (traverse->link != NULL)

{

ptr = traverse->link;

if (traverse->data > ptr->data)

{

swapped = 1;

if (traverse == start)

{

traverse->link = ptr->link;

ptr->link = traverse;

prev = ptr;

start = prev;

}

else

{

traverse->link = ptr->link;

ptr->link = traverse;

prev->link = ptr;

prev = ptr;

}

}

prev = traverse;

traverse = traverse->link;

}

if (!swapped)

break;

++itr;

}

display(start);

}

void display(node \*startTemp)

{

if(startTemp == NULL)

{

printf("Linked list is empty\n");

return;

}

printf("List Contents:\n");

for(ptr = startTemp; ptr != NULL; ptr = ptr->link)

printf("%d\n", ptr->data);

}

void main()

{

printf("Create Linked List:\n");

createA();

while(1){

int ch;

printf("Select:\n1. Sort\n2. Reverse\n3. Concatenate\n4. Exit\n");

scanf("%d", &ch);

switch(ch){

case 1: sort(); break;

case 2: reverse(); break;

case 3:

printf("\n\nCreate List 2:\n");

createA();

concatenate();

break;

case 4: exit(0); break;

}

}

}

Output:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Lab Program 6**

**WAP to Implement Single Link List to simulate Stack & Queue Operations.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

void display();

void push();

void pop();

void insert();

void delete();

struct Node {

int data;

struct Node\* link;

};

typedef struct Node node;

node \*new1, \*start = NULL;

int ch;

void main() {

char c;

printf("Stack or Queue? (s/q): ");

scanf(" %c", &c);

if (c == 's' || c == 'S') {

while (1) {

printf("Enter your Choice:\n 1: Push\n 2: Pop\n 3: Display\n 4: Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1: push();

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

break;

default: printf("Invalid choice\n");

break;

}

}

} else if (c == 'q' || c == 'Q') {

while (1) {

printf("Enter your Choice:\n 1: Insert\n 2: Delete\n 3: Display\n 4: Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1: insert();

break;

case 2: delete();

break;

case 3: display();

break;

case 4: exit(0);

break;

default: printf("Invalid choice\n");

break;

}

}

} else {

printf("Invalid input. Please restart the program.\n");

}

}

void insert() {

new1 = (node\*)malloc(sizeof(node));

if (new1 == NULL) {

printf("Memory allocation failed\n");

return;

}

printf("Enter value: ");

scanf("%d", &new1->data);

new1->link = NULL;

if (start == NULL) {

start = new1;

} else {

node\* temp = start;

while (temp->link != NULL) {

temp = temp->link;

}

temp->link = new1;

}

}

void push() {

new1 = (node\*)malloc(sizeof(node));

if (new1 == NULL) {

printf("Memory allocation failed\n");

return;

}

printf("Enter value: ");

scanf("%d", &new1->data);

new1->link = start;

start = new1;

}

void display() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

node\* temp = start;

printf("Elements of the linked list are: ");

while (temp != NULL) {

printf("%d", temp->data);

temp = temp->link;

if (temp != NULL) {

printf(" -> ");

}

}

printf("\n");

}

void pop() {

if (start == NULL) {

printf("Stack is empty\n");

return;

}

node\* temp = start;

printf("Popped value: %d\n", temp->data);

start = start->link;

free(temp);

}

void delete() {

if (start == NULL) {

printf("Queue is empty\n");

return;

}

node\* temp = start;

printf("Deleted value: %d\n", temp->data);

start = start->link;

free(temp);

}

Output:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Lab program 7**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

void create();

void display();

void insertbeg();

void delpos();

struct Node {

int data;

struct Node \*prev, \*next;

};

typedef struct Node node;

node \*new1, \*start = NULL, \*temp, \*prev, \*curr;

int ch;

void main() {

while (1) {

printf("Enter your Choice:\n 1: Create\n 2: Insert at the beginning\n 3: Delete a given element\n 4: Display\n 5: Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1: create();

break;

case 2: insertbeg();

break;

case 3: delpos();

break;

case 4: display();

break;

case 5: exit(0);

break;

default: printf("Invalid choice\n");

break;

}

}

}

void create() {

char ch;

new1 = (node\*)malloc(sizeof(node));

printf("Enter data: ");

scanf("%d", &new1->data);

new1->prev = NULL;

new1->next = NULL;

start = new1;

curr = new1;

while (1) {

printf("Do you want to add another node? (Y/n): ");

getchar();

scanf("%c", &ch);

if(ch == 'y' || ch == 'Y') {

new1 = (node\*)malloc(sizeof(node));

printf("Enter data: ");

scanf("%d", &new1->data);

new1->prev = curr;

curr->next = new1;

curr = new1;

} else {

curr->next = NULL;

return;

}

}

}

void insertbeg() {

new1 = (node\*)malloc(sizeof(node));

printf("Enter value: ");

scanf("%d", &new1->data);

if (start == NULL) {

new1->prev = NULL;

new1->next = NULL;

start = new1;

return;

}

new1->prev = NULL;

new1->next = start;

start->prev = new1;

start = new1;

}

void display() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

temp = start;

printf("Elements of the linked list are: ");

while (temp != NULL) {

printf("%d", temp->data);

temp = temp->next;

if (temp != NULL) {

printf(" -> ");

}

}

printf("\n");

}

void delpos() {

if (start == NULL) {

printf("Linked list is empty\n");

return;

}

int ele;

printf("Enter the element to delete: ");

scanf("%d", &ele);

if (start->data == ele) {

temp = start;

start = start->next;

if (start != NULL) {

start->prev = NULL;

}

free(temp);

if (start == NULL) {

printf("The list is now empty.\n");

}

return;

}

temp = start;

while (temp != NULL && temp->data != ele) {

temp = temp->next;

}

if (temp == NULL) {

printf("Element not found in the list.\n");

return;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

if (temp->prev != NULL) {

temp->prev->next = temp->next;

}

free(temp);

printf("Node deleted successfully.\n");

}

**Output:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Lab Program 8**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

struct Node \*left;

int data;

struct Node \*right;

};

typedef struct Node node;

node \*root = NULL, \*new1, \*curr, \*ptr;

int ch, c = 0;

char ch1;

void create();

void traverse\_preorder(node \*temp);

void traverse\_inorder(node \*temp);

void traverse\_postorder(node \*temp);

void display(node \*temp, int level);

int main() {

while (1) {

printf("1. Construct a binary Search tree.\n2. To traverse the tree using all the methods i.e., in-order, preorder, and postorder\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

create();

break;

case 2:

printf("Preorder:\n");

traverse\_preorder(root);

printf("\nInorder:\n");

traverse\_inorder(root);

printf("\nPostorder:\n");

traverse\_postorder(root);

printf("\n");

break;

case 3:

printf("Level: 0\n");

display(root, 0);

break;

case 4:

exit(0);

}

}

return 0;

}

void create() {

new1 = (node \*)malloc(sizeof(node));

printf("Enter the element (integer): ");

scanf("%d", &new1->data);

new1->left = NULL;

new1->right = NULL;

root = new1;

while (1) {

printf("Do you want to add more elements (Y/N): ");

scanf(" %c", &ch1);

if (ch1 == 'Y' || ch1 == 'y') {

new1 = (node \*)malloc(sizeof(node));

printf("Enter the element (integer): ");

scanf("%d", &new1->data);

new1->left = NULL;

new1->right = NULL;

curr = root;

while (curr != NULL) {

ptr = curr;

if (new1->data >= curr->data) {

curr = curr->right;

} else {

curr = curr->left;

}

}

if (new1->data < ptr->data) {

ptr->left = new1;

} else {

ptr->right = new1;

}

} else {

return;

}

}

}

void traverse\_preorder(node \*temp) {

if (temp != NULL) {

printf("%d ", temp->data);

traverse\_preorder(temp->left);

traverse\_preorder(temp->right);

}

}

void traverse\_inorder(node \*temp) {

if (temp != NULL) {

traverse\_inorder(temp->left);

printf("%d ", temp->data);

traverse\_inorder(temp->right);

}

}

void traverse\_postorder(node \*temp) {

if (temp != NULL) {

traverse\_postorder(temp->left);

traverse\_postorder(temp->right);

printf("%d ", temp->data);

}

}

void display(node \*temp, int level) {

if (temp == NULL) {

return;

}

printf("Data: %d, Level: %d\n", temp->data, level);

display(temp->left, level + 1);

display(temp->right, level + 1);

}

Output:

A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

**Lab program 9**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 20

int adjMatrix[MAX][MAX], visited[MAX], queue[MAX], front = 0, rear = -1;

void enqueue(int v) {

queue[++rear] = v;

}

int dequeue() {

return queue[front++];

}

void BFS(int start, int vertices) {

enqueue(start);

visited[start] = 1;

printf("BFS traversal starting from vertex %d: ", start);

while (front <= rear) {

int v = dequeue();

printf("%d ", v);

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

if (adjMatrix[v][i] && !visited[i]) {

enqueue(i);

visited[i] = 1;

}

}

}

printf("\n");

}

int main() {

int vertices, edges, u, v;

printf("Enter number of vertices: ");

scanf("%d", &vertices);

printf("Enter number of edges: ");

scanf("%d", &edges);

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

printf("Enter edge (u v): ");

scanf("%d %d", &u, &v);

adjMatrix[u][v] = adjMatrix[v][u] = 1;

}

printf("Enter the starting vertex for BFS: ");

scanf("%d", &u);

BFS(u, vertices);

return 0;

}

**Output:**

**A screenshot of a computer program

Description automatically generated**

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

**Code:**

void dfs(int);

int n,i,j,a[10][10],vis[10],parent[10],cycle = 0;

void main()

{

printf("enter the number of vertices\n");

scanf("%d",&n);

printf("enter the adjacency matrix\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&a[i][j]);

}

vis[i]=0;

parent[i] = -1;

}

printf("dfs traversal\n");

for(i=1;i<=n;i++)

{

if(vis[i]==0)

dfs(i);

}

if(cycle == 1)

printf("\nGraph contains a cycle.\n");

else

printf("\nNo cycle detected in the graph.\n");

for(i=1;i<=n;i++)

{

if(vis[i] == 0)

{

printf("\nGraph is not connected.\n");

return;

}

}

printf("\nGraph is connected.\n");

getch();

}

void dfs(int v)

{

vis[v]=1;

printf("%d ",v);

for(j=1;j<=n;j++)

{

if(a[v][j]==1 && vis[j]==0)

{

parent[j] = v;

dfs(j);

}

else if(a[v][j]==1 && parent[v] != j)

{

cycle = 1;

}

}

}

Output:

A screenshot of a computer program

Description automatically generated

**Lab program 10**

**Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 10

typedef struct {

int key;

char name[50];

} Employee;

Employee hashTable[MAX];

int m = MAX;

void initHashTable() {

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

hashTable[i].key = -1;

}

}

int hashFunction(int key) {

return key % m;

}

int linearProbing(int key) {

int index = hashFunction(key);

int start = index;

while (hashTable[index].key != -1 && hashTable[index].key != key) {

index = (index + 1) % m;

if (index == start) {

printf("Hash Table is full\n");

return -1;

}

}

return index;

}

void insertRecord(int key, char name[]) {

int index = linearProbing(key);

if (index != -1) {

hashTable[index].key = key;

strcpy(hashTable[index].name, name);

printf("Employee record inserted at index %d\n", index);

}

}

void displayHashTable() {

printf("\nHash Table:\n");

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

if (hashTable[i].key != -1) {

printf("Index %d: Key = %d, Name = %s\n", i, hashTable[i].key, hashTable[i].name);

} else {

printf("Index %d: Empty\n", i);

}

}

}

int main() {

int choice, key;

char name[50];

initHashTable();

while (1) {

printf("\nMenu:\n");

printf("1. Insert Employee Record\n");

printf("2. Display Hash Table\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter employee key (4-digit): ");

scanf("%d", &key);

printf("Enter employee name: ");

getchar();

fgets(name, sizeof(name), stdin);

name[strcspn(name, "\n")] = '\0';

insertRecord(key, name);

break;

case 2:

displayHashTable();

break;

case 3:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

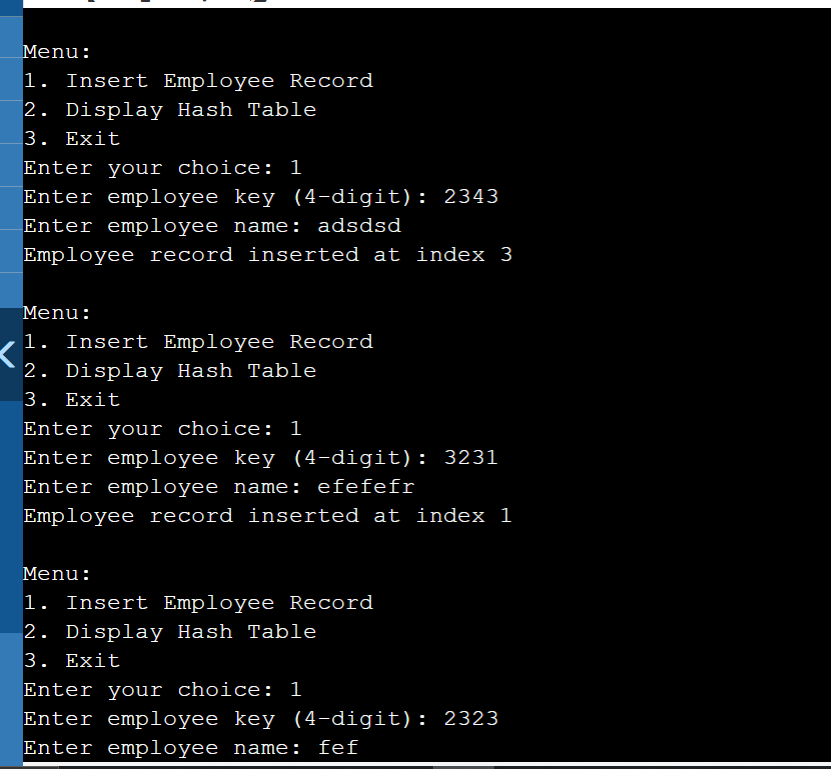
}

}

return 0;

}

Output:



A screenshot of a computer

Description automatically generated