VISVESVARAYATECHNOLOGICALUNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Deepthi M (1BM23CS088)

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

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 Deepthi M **(1BM23CS088)**, 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.

Dr. Selva Kumar
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

6	Implement Single Link List with following operations: Sort the linked list,Reverse the linked list, Concatenation of two linked lists	31-33
	Single link list to simulate stack and queue operations	34-36
7	Implementation of Doubly LinkedList	37-39
8	Implementation of Binary tree	40-42
9	BFS and DFS	43-46
10	DSTLinear probing	46-48

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: Stack

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\n", st[(*top)--]);
    }
}

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

int main() {
    printf("Deepthi M \n1BM23CS088\n");

    int st[STACK_SIZE], top = -1, c;

    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter 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;
            case 4:
                exit(0);
            default:
                printf("\nInvalid choice!!!\n");
        }
    }

    return 0;
}
```

Output:

```
Deepthi M
1BM23CS088

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1

Enter an item: 2

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

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

Lab program 2: Infix to postfix

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)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>

char s[100];
int top = -1;

void push(char ele) {
   top++;
   s[top] = ele;
}

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

int pr(char op) {
   switch (op) {
      case '#': return 0;
      case '(': return 1;
      case '+': case '-': return 2;
      case '*': case '/': return 3;
      default: return 0;
   }
}

int main() {
   printf("Deepthi M \n1BM23CS088\n");

   char infix[100], ch;
   int i = 0;

   printf("Enter the infix expression: ");
   scanf("%s", infix);

   push('#');

   while (infix[i] != '\0') {
      if (isalpha(infix[i])) {
         printf("%c", infix[i]);
      } else if (infix[i] == '(') {
         push(infix[i]);
      } else if (infix[i] == ')') {
         while (s[top] != '(') {
            ch = pop();
            printf("%c", ch);
         }
         pop();
      } else {
```

```
        while ((s[top] != '#') && (pr(infix[i]) <= pr(s[top]))) {
          ch = pop();
          printf("%c", ch);
        }
        push(infix[i]);
      }
      i++;
    }

    while (top > 0) {
      if (s[top] == '(') {
        printf("INVALID EXPRESSION\n");
        return 1;
      }
      printf("%c", pop());
    }

    return 0;
}
```

Output:

Lab Program 3a: Linear Queue

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 5
```

```
int isFull(int rear) {
   return rear == MAX - 1;
}

int isEmpty(int front) {
   return front == -1;
}

void insert(int queue[], int *front, int *rear, int value) {
   if (isFull(*rear)) {
      printf("Queue Overflow! Cannot insert %d\n", value);
      return;
   }
   if (*rear == -1) {
      *front = 0;
   }
   queue[++(*rear)] = value;
   printf("Inserted %d into the queue\n", value);
}

int delete(int queue[], int *front, int *rear) {
   if (isEmpty(*front)) {
      printf("Queue Underflow! Cannot delete from an empty queue\n");
      return -1;
   }
   int item = queue[*front];
   if (*front == *rear) {
      *front = -1;
      *rear = -1;
   } else {
      (*front)++;
   }
   printf("Deleted %d from the queue\n", item);
```

```
    return item;
}

void display(int queue[], int front, int rear) {
    if (isEmpty(front)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    printf("Deepthi M \n1BM23CS088\n");

    int queue[MAX];
    int front = -1, rear = -1;
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
        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 a value to insert: ");
```

```
            scanf("%d", &value);
            insert(queue, &front, &rear, value);
            break;
        case 2:
            delete(queue, &front, &rear);
            break;
        case 3:
            display(queue, front, rear);
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }
   return 0;
}
```

output:

```
Deepthi M
1BM23CS088

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 2
Inserted 2 into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 4
Inserted 4 into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 2 4
```

Lab Program 3b: Circular Queue

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & DisplayThe program should print appropriate messages for queue empty and queue overflow conditions. The program should be done using pass by reference only.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

int isFull(int front, int rear) {
   return (front == (rear + 1) % MAX);
}

int isEmpty(int front) {
   return front == -1;
}

void enqueue(int queue[], int *front, int *rear, int value) {
   if (isFull(*front, *rear)) {
      printf("Queue is full!\n");
   } else {
      if (*front == -1) {
         *front = 0;
      }
      *rear = (*rear + 1) % MAX;
      queue[*rear] = value;
      printf("Inserted %d\n", value);
   }
}

void dequeue(int queue[], int *front, int *rear) {
   if (isEmpty(*front)) {
      printf("Queue is empty!\n");
   } else {
      printf("Deleted %d\n", queue[*front]);
      if (*front == *rear) {
         *front = -1;
         *rear = -1;
      } else {
         *front = (*front + 1) % MAX;
      }
   }
}

void display(int queue[], int front, int rear) {
   if (isEmpty(front)) {
      printf("Queue is empty!\n");
   } else {
      printf("Queue elements are: ");
      int i = front;
      while (i != rear) {
         printf("%d ", queue[i]);
         i = (i + 1) % MAX;
      }
      printf("%d\n", queue[i]);
   }
}
```

```
int main() {
   printf("Deepthi M \n1BM23CS088\n");

   int queue[MAX];
   int front = -1, rear = -1;
   int choice, value;

   while (1) {
      printf("\nCircular Queue Menu:\n");
      printf("1. Enqueue\n");
      printf("2. Dequeue\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            enqueue(queue, &front, &rear, value);
            break;
         case 2:
            dequeue(queue, &front, &rear);
            break;
         case 3:
            display(queue, front, rear);
            break;
         case 4:
            printf("Exiting...\n");
            exit(0);
         default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

Output:

```
Deepthi M
1BM23CS088

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 2
Inserted 2

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 5
Inserted 5

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 2 5
```

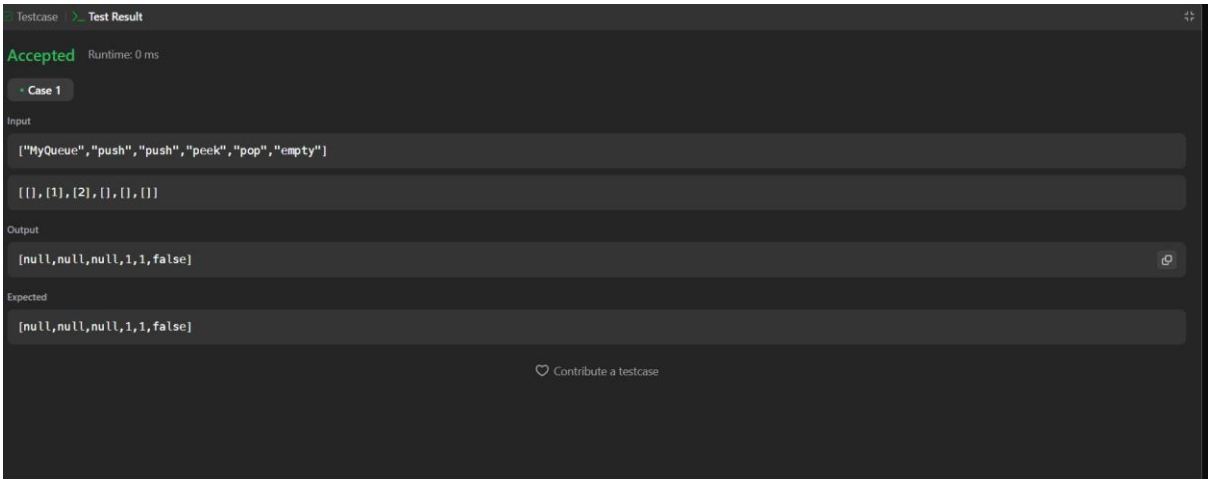
Leet Code: Implement Queue using stacks

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

```
#include<stdlib.h>
#include <stdbool.h>
#define MAX 10


typedef struct {
    int array1[MAX];
    int array2[MAX];
    int top1;
    int top2;

} MyQueue;


MyQueue* myQueueCreate() {
    MyQueue* queue = (MyQueue*)malloc(sizeof(MyQueue));
    queue->top1 = -1;
    queue->top2 = -1;
    return queue;

}

void myQueuePush(MyQueue* obj, int x) {
     if (obj->top1 == MAX - 1)
     {
        printf("Queue is full\n");
        return;
     }
    obj->array1[++(obj->top1)] = x;

}

int myQueuePop(MyQueue* obj) {
    if (obj->top2 == -1)
     {
        if (obj->top1 == -1)
        {
            printf("Queue is empty\n");
            return -1;
        }
        while (obj->top1 != -1)
         {
            obj->array2[++(obj->top2)] = obj->array1[(obj->top1)--];
        }
    }
    return obj->array2[(obj->top2)--];

}

int myQueuePeek(MyQueue* obj) {
     if (obj->top2 == -1)
      {
        if (obj->top1 == -1)
        {
            printf("Queue is empty\n");
            return -1;
        }
```

```
        while (obj->top1 != -1)
        {
            obj->array2[++(obj->top2)] = obj->array1[(obj->top1)--];
        }
    }
    return obj->array2[obj->top2];

}

bool myQueueEmpty(MyQueue* obj) {
    return (obj->top1 == -1 && obj->top2 == -1);

}

void myQueueFree(MyQueue* obj) {
    free(obj);

}
```

Output

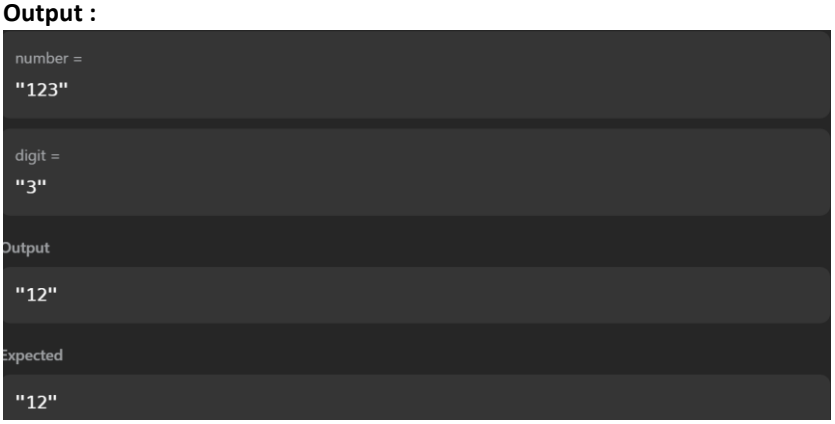


Leet Code: REMOVE DIGIT FROM NUMBER

TO MAXIMISE RESULT

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

char* removeDigit(char* number, char digit) {
    int len = strlen(number);

    for (int i = 0; i < len - 1; i++) {
        if (number[i] == digit && number[i] < number[i + 1]) {
            memmove(&number[i], &number[i + 1], len - i);
            return number;
        }
    }


    for (int i = len - 1; i >= 0; i--) {
        if (number[i] == digit) {

            memmove(&number[i], &number[i + 1], len - i);
            return number;
        }
    }

    return number;
}
```

Output :



number =

"1231"

digit =

"1"

Output

"231"

Expected

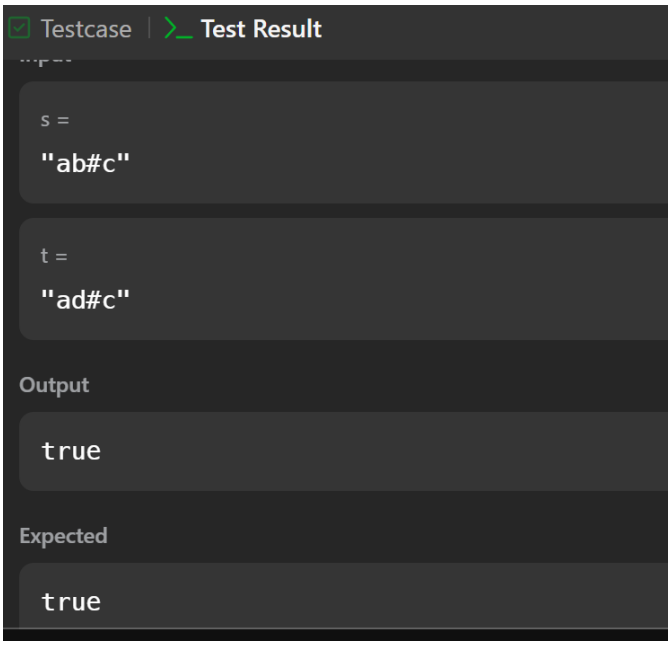
"231"

Leet Code : Backspace String compare

```
typedef struct {
    char items[200];
    int top;
} Stack;


void init(Stack* stack) {
    stack->top = -1;
}


void push(Stack* stack, char c) {
    stack->items[++(stack->top)] = c;
}


void pop(Stack* stack) {
    if (stack->top >= 0) {
        --(stack->top);
    }
}


bool backspaceCompare(char* s, char* t) {
    Stack stack1, stack2;
    init(&stack1);
```

```
    init(&stack2);


    for (int i = 0; i < strlen(s); ++i) {
        if (s[i] == '#') {
            pop(&stack1);
        } else {
            push(&stack1, s[i]);
        }
    }


    for (int i = 0; i < strlen(t); ++i) {
        if (t[i] == '#') {
            pop(&stack2);
        } else {
            push(&stack2, t[i]);
        }
    }


    if (stack1.top != stack2.top) {
        return false;
    }


    for (int i = 0; i <= stack1.top; ++i) {
        if (stack1.items[i] != stack2.items[i]) {
            return false;
        }
    }

    return true;
}
```

Output :



Lab Program 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.

c) Display the contents of the linked list.

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

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

typedef struct node* NODE;

NODE getnode() {
   NODE new_node = (NODE)malloc(sizeof(struct node));
   if (new_node == NULL) {
      printf("Memory allocation failed.\n");
      exit(1);
   }
   return new_node;
}

NODE insert_pos(NODE first, int item, int pos) {
   NODE new = getnode();
   new->value = item;
   new->next = NULL;
   if (first == NULL && pos == 1) {
      return new;
   }
   if (pos < 1) {
      printf("INVALID POSITION\n");
      free(new);
      return first;
   }
   int count = 1;
   NODE current = first, prev = NULL;
   while (count < pos && current != NULL) {
      prev = current;
      current = current->next;
      count++;
   }
   if (count != pos) {
      printf("INVALID POSITION\n");
      free(new);
      return first;
   }
   new->next = current;
   if (prev != NULL) {
      prev->next = new;
   } else {
      return new;
   }
   return first;
```

```
}

NODE insert_beg(int item, NODE first) {
   NODE new = getnode();
   new->value = item;
   new->next = first;
   return new;
}

NODE insert_end(int item, NODE first) {
   NODE new_end = getnode();
   new_end->value = item;
   new_end->next = NULL;
   if (first == NULL) {
      return new_end;
   }
   NODE current = first;
   while (current->next != NULL) {
      current = current->next;
   }
   current->next = new_end;
   return first;
}

NODE delete_first(NODE first) {
   if (first == NULL) {
      printf("LINKED LIST IS EMPTY\n");
      return NULL;
   }
   NODE temp = first;
   first = first->next;
   free(temp);
   return first;
}

NODE delete_end(NODE first) {
   if (first == NULL) {
      printf("LINKED LIST IS EMPTY\n");
      return NULL;
   }
   if (first->next == NULL) {
      free(first);
      return NULL;
   }
   NODE prev = NULL, last = first;
   while (last->next != NULL) {
      prev = last;
      last = last->next;
   }
   prev->next = NULL;
   free(last);
   return first;
}

NODE delete_value(NODE first, int value) {
   if (first == NULL) {
      printf("LINKED LIST IS EMPTY\n");
```

```
        return NULL;
    }
    NODE current = first, prev = NULL;
    while (current != NULL && current->value != value) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        printf("VALUE NOT FOUND\n");
        return first;
    }
    if (prev == NULL) {
        first = current->next;
    } else {
        prev->next = current->next;
    }
    free(current);
    return first;
}

void display(NODE first) {
    if (first == NULL) {
        printf("LINKED LIST IS EMPTY\n");
        return;
    }
    NODE temp = first;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d ", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    printf("Deepthi M \n1BM23CS088\n");

    NODE first = NULL;
    int choice, item, pos;

    do {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete First\n");
        printf("5. Delete Last\n");
        printf("6. Delete by Value\n");
        printf("7. Display\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &item);
```

```
            first = insert_beg(item, first);
            break;
        case 2:
            printf("Enter value to insert at end: ");
            scanf("%d", &item);
            first = insert_end(item, first);
            break;
        case 3:
            printf("Enter value and position to insert: ");
            scanf("%d %d", &item, &pos);
            first = insert_pos(first, item, pos);
            break;
        case 4:
            first = delete_first(first);
            break;
        case 5:
            first = delete_end(first);
            break;
        case 6:
            printf("Enter value to delete: ");
            scanf("%d", &item);
            first = delete_value(first, item);
            break;
        case 7:
            display(first);
            break;
        case 8:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice.\n");
      }
   } while (choice != 8);

   return 0;
}
```

Output:

```
Deepthi M
1BM23CS088

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 1
Enter value to insert at beginning: 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 3
Enter value and position to insert: 2
5
INVALID POSITION
```

```
6. Delete by Value
7. Display
8. Exit
Enter your choice: 2
Enter value to insert at end: 7

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 5

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 7
Linked List: 2
```

Lab Program 4b:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

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

c) Display the contents of the linked list.

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

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

typedef struct node* NODE;

NODE getnode() {
   NODE new_node =
(NODE)malloc(sizeof(struct node));
   if (new_node == NULL) {
      printf("Memory allocation failed.\n");
      exit(1);
   }
   return new_node;
}

NODE insert_pos(NODE first, int item, int pos) {
   NODE new = getnode();
   new->value = item;
   new->next = NULL;
   if (first == NULL && pos == 1) {
      return new;
   }
   if (pos < 1) {
      printf("INVALID POSITION\n");
      free(new);
      return first;
   }
   int count = 1;
   NODE current = first, prev = NULL;
   while (count < pos && current != NULL) {
      prev = current;
      current = current->next;
      count++;
   }
   if (count != pos) {
      printf("INVALID POSITION\n");
      free(new);
      return first;
   }
   new->next = current;
   if (prev != NULL) {
      prev->next = new;
   } else {
      return new;
   }
```

```
    return first;
}

NODE insert_beg(int item, NODE first) {
    NODE new = getnode();
    new->value = item;
    new->next = first;
    return new;
}

NODE insert_end(int item, NODE first) {
    NODE new_end = getnode();
    new_end->value = item;
    new_end->next = NULL;
    if (first == NULL) {
        return new_end;
    }
    NODE current = first;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = new_end;
    return first;
}

NODE delete_first(NODE first) {
    if (first == NULL) {
        printf("LINKED LIST IS EMPTY\n");
        return NULL;
    }
    NODE temp = first;
    first = first->next;
    free(temp);
    return first;
}

NODE delete_end(NODE first) {
    if (first == NULL) {
        printf("LINKED LIST IS EMPTY\n");
        return NULL;
    }
    if (first->next == NULL) {
        free(first);
        return NULL;
    }
    NODE prev = NULL, last = first;
    while (last->next != NULL) {
        prev = last;
        last = last->next;
    }
    prev->next = NULL;
    free(last);
    return first;
}

NODE delete_value(NODE first, int value) {
```

```
    if (first == NULL) {
        printf("LINKED LIST IS EMPTY\n");
        return NULL;
    }
    NODE current = first, prev = NULL;
    while (current != NULL && current->value !=
value) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        printf("VALUE NOT FOUND\n");
        return first;
    }
    if (prev == NULL) {
        first = current->next;
    } else {
        prev->next = current->next;
    }
    free(current);
    return first;
}

void display(NODE first) {
    if (first == NULL) {
        printf("LINKED LIST IS EMPTY\n");
        return;
    }
    NODE temp = first;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d ", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    printf("Deepthi M \n1BM23CS088\n");

    NODE first = NULL;
    int choice, item, pos;

    do {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete First\n");
        printf("5. Delete Last\n");
        printf("6. Delete by Value\n");
        printf("7. Display\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter value to insert at
beginning: ");
                scanf("%d", &item);
                first = insert_beg(item, first);
                break;
            case 2:
                printf("Enter value to insert at end: ");
                scanf("%d", &item);
                first = insert_end(item, first);
                break;
            case 3:
                printf("Enter value and position to
insert: ");
                scanf("%d %d", &item, &pos);
                first = insert_pos(first, item, pos);
                break;
            case 4:
                first = delete_first(first);
                break;
            case 5:
                first = delete_end(first);
                break;
            case 6:
                printf("Enter value to delete: ");
                scanf("%d", &item);
                first = delete_value(first, item);
                break;
            case 7:
                display(first);
                break;
            case 8:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 8);

    return 0;
}
```

Output:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 5
LINKED LIST IS EMPTY

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 7
linked list is empty
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 1
Enter value to insert at beginning: 1

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 2
Enter value to insert at end: 2
```

```
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 2
Enter value to insert at end: 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 3
Enter value and position to insert: 3
4
INVALID POSITION

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 7
1
2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete by Value
7. Display
8. Exit
Enter your choice: 8

Process returned 0 (0x0)   execution time : 94.875 s
Press any key to continue.
```

Lab Program 5a:

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;
};

void append(struct Node** head, int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
   if (*head == NULL) {
      *head = newNode;
   } else {
      struct Node* temp = *head;
      while (temp->next)
         temp = temp->next;
      temp->next = newNode;
   }
}

void reverse(struct Node** head) {
   struct Node *prev = NULL, *current = *head, *next = NULL;
   while (current) {
      next = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   *head = prev;
}

void sort(struct Node* head) {
   struct Node *i, *j;
   int temp;
   for (i = head; i != NULL; i = i->next) {
      for (j = i->next; j != NULL; j = j->next) {
         if (i->data > j->data) {
            temp = i->data;
            i->data = j->data;
            j->data = temp;
         }
      }
   }
}

void concatenate(struct Node** head1, struct Node* head2) {
   if (*head1 == NULL) {
      *head1 = head2;
```

```
    } else {
        struct Node* temp = *head1;
        while (temp->next)
            temp = temp->next;
        temp->next = head2;
    }
}

void printList(struct Node* head) {
    while (head) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

int main() {
    printf("Deepthi M \n1BM23CS088\n");

    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice, item;

    do {
        printf("\nENTER THE OPERATION YOU WANT TO PERFORM:\n");
        printf("1. Insert for first node\n");
        printf("2. Insert for second node\n");
        printf("3. Concatenate\n");
        printf("4. Reverse\n");
        printf("5. Sort\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value for first node: ");
                scanf("%d", &item);
                append(&list1, item);
                break;
            case 2:
                printf("Enter value for second node: ");
                scanf("%d", &item);
                append(&list2, item);
                break;
            case 3:
                concatenate(&list1, list2);
                printf("Concatenated List: ");
                printList(list1);
                break;
            case 4:
                reverse(&list1);
                printf("Reversed List 1: ");
                printList(list1);
                reverse(&list2);
                printf("Reversed List 2: ");
                printList(list2);
```

```
            break;
        case 5:
            sort(list1);
            printf("Sorted List 1: ");
            printList(list1);
            sort(list2);
            printf("Sorted List 2: ");
            printList(list2);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. \n");
      }
   } while (choice != 6);

   return 0;
}
```

Output :

```
4. Reverse
5. Sort
6. Exit
Enter your choice: 1
Enter value for first node: 2

ENTER THE OPERATION YOU WANT TO PERFORM:
1. Insert for first node
2. Insert for second node
3. Concatenate
4. Reverse
5. Sort
6. Exit
Enter your choice: 1
Enter value for first node: 4

ENTER THE OPERATION YOU WANT TO PERFORM:
1. Insert for first node
2. Insert for second node
3. Concatenate
4. Reverse
5. Sort
6. Exit
Enter your choice: 4
Reversed List 1: 4 2
Reversed List 2:
```

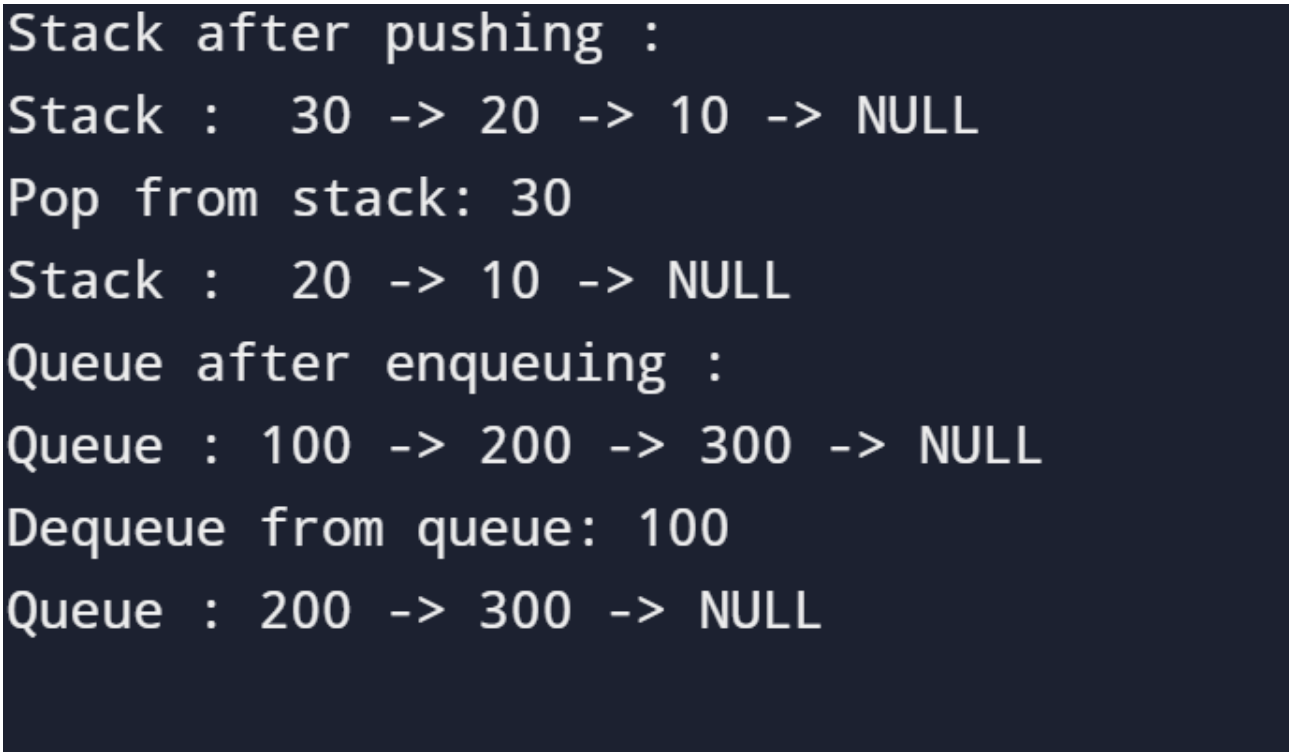
Lab Program 5b: Queue and stack Implementaion

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

// Name: Deepthi M
// USN: 1BM23CS088

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

struct Stack {
   struct Node* top;
};

struct Queue {
   struct Node* front;
   struct Node* rear;
};

void initStack(struct Stack* stack) {
   stack->top = NULL;
}

void initQueue(struct Queue* queue) {
   queue->front = queue->rear = NULL;
}

void push(struct Stack* stack, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = stack->top;
   stack->top = newNode;
}

int pop(struct Stack* stack) {
   if (stack->top == NULL)
      return -1;
   struct Node* temp = stack->top;
   int data = temp->data;
   stack->top = stack->top->next;
   free(temp);
   return data;
}

int peek(struct Stack* stack) {
   if (stack->top == NULL)
      return -1;
   return stack->top->data;
}

void enqueue(struct Queue* queue, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
```

```
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }
    queue->rear->next = newNode;
    queue->rear = newNode;
}

int dequeue(struct Queue* queue) {
    if (queue->front == NULL)
        return -1;
    struct Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;
    if (queue->front == NULL)
        queue->rear = NULL;
    free(temp);
    return data;
}

int isEmptyStack(struct Stack* stack) {
    return stack->top == NULL;
}

int isEmptyQueue(struct Queue* queue) {
    return queue->front == NULL;
}

void displayStack(struct Stack* stack) {
    struct Node* temp = stack->top;
    printf("Stack :  ");
    while (temp) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void displayQueue(struct Queue* queue) {
    struct Node* temp = queue->front;
    printf("Queue : ");
    while (temp) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Stack stack;
    struct Queue queue;
    initStack(&stack);
    initQueue(&queue);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
```

```
    printf("Stack after pushing :\n");
    displayStack(&stack);

    printf("Pop from stack: %d\n", pop(&stack));
    displayStack(&stack);

    enqueue(&queue, 100);
    enqueue(&queue, 200);
    enqueue(&queue, 300);
    printf("Queue after enqueuing :\n");
    displayQueue(&queue);

    printf("Dequeue from queue: %d\n", dequeue(&queue));
    displayQueue(&queue);

    return 0;
}
```

Output:



Lab Program 6 : DOUBLE LINKED LIST

WAP to Implement doubly link list with primitive operations

a) **Create a doubly linked list.**

b) **Insert a new node to the left of the node.**

c) **Delete the node based on a specific value**

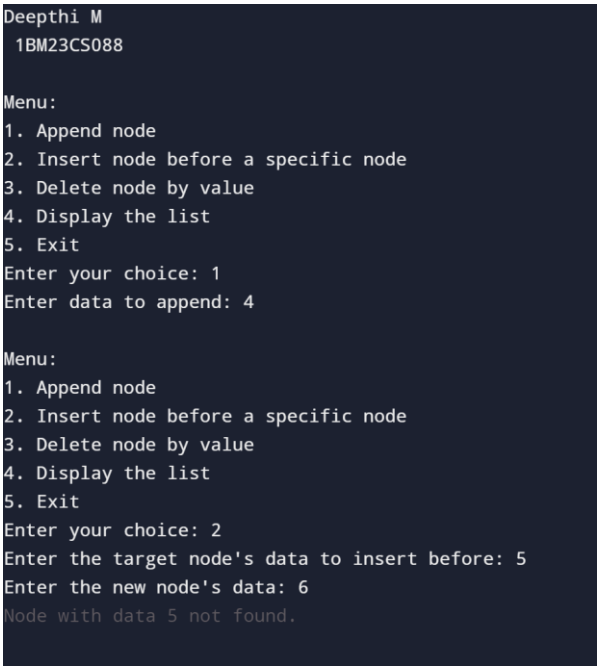
d) **Display the contents of the list**

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

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}

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

void insertBefore(struct Node** head, int targetData, int newData) {
   struct Node* temp = *head;
   while (temp != NULL && temp->data != targetData) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Node with data %d not found.\n", targetData);
      return;
   }
   struct Node* newNode = createNode(newData);
   if (temp == *head) {
      newNode->next = *head;
      (*head)->prev = newNode;
      *head = newNode;
      return;
```

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

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

void display(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <=> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, targetData, newData;
     printf(“Deepthi M \n 1BM23CS088”);
    while (1) {
        printf("\nMenu:\n");
        printf("1. Append node\n");
        printf("2. Insert node before a specific node\n");
        printf("3. Delete node by value\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to append: ");
                scanf("%d", &data);
                append(&head, data);
                break;
            case 2:
                printf("Enter the target node's data to insert before: ");
                scanf("%d", &targetData);
                printf("Enter the new node's data: ");
                scanf("%d", &newData);
                insertBefore(&head, targetData, newData);
                break;
            case 3:
                printf("Enter the data of the node to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 4:
                display(head);
                break;
            case 5:
                printf("Exiting the program...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

Output:



Lab program-8: BINARY TREE

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

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

struct Node {
   int data;
   struct Node *left, *right;
};

struct Node* newNode(int data) {
   struct Node* node = (struct
Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->left = node->right = NULL;
   return node;
}

struct Node* insert(struct Node* root, int data)
{
   if (root == NULL) {
      return newNode(data);
   }
   if (data < root->data) {
      root->left = insert(root->left, data);
   } else {
      root->right = insert(root->right, data);
   }
   return root;
}

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

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

```
}

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

void displayMenu() {
    printf("\nBinary Search Tree Operations\n");
    printf("1. Insert a node\n");
    printf("2. In-order Traversal\n");
    printf("3. Pre-order Traversal\n");
    printf("4. Post-order Traversal\n");
    printf("5. Exit\n");
}

int main() {
    struct Node* root = NULL;
    int choice, data;
   printf(“Deepthi M \n 1BM23CS088”);

    while (1) {
        displayMenu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Post-order Traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                printf("Exiting...\n");
                exit(0);
                break;
            default:
```

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

    return 0;
```

Output:

```
Binary Search Tree Operations
1. Insert a node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 4

Binary Search Tree Operations
1. Insert a node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 7

Binary Search Tree Operations
1. Insert a node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 2 4 7
```

LeetCode: struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode *headB) {

if(headA==NULL||headB==NULL) return NULL; struct ListNode *t1=headA,*t2=headB; while(t1!=t2)

```
    {
        t1=t1->next; t2=t2-
        >next; if(t1==t2) return
```

```
        t1; if(t1==NULL)
        t1=headB; if(t2==NULL)
        t2=headA;
    }
    return t1;
}
```

Output:

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

```
#include<stdio.h>

void bfs(int); int a[10][10],
vis[10], n; void main() {
    int i, j, src; 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;
    }
```

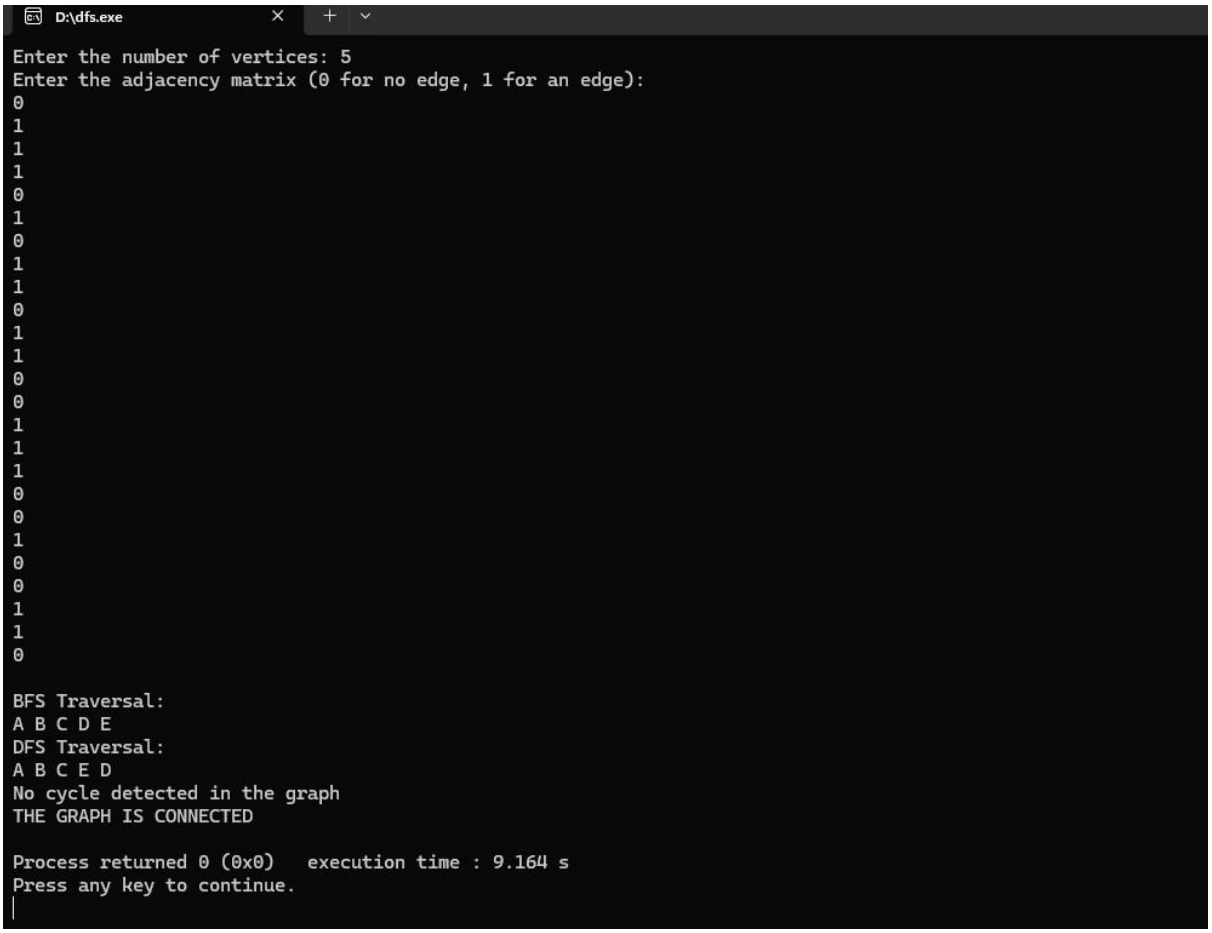
```
    printf("enter the src vertex\n"); scanf("%d", &src);
    printf("nodes reachable from src vertex\n"); bfs(src);
}

void bfs(int v) { int q[10], f = 1, r = 1, u, i; q[r]
    = v; vis[v] = 1; while(f <= r) { u = q[f];
    printf("%d ", u); for(i = 1; i <= n; i++) {
    if(a[v][i] == 1 && vis[i] == 0) { vis[i] = 1;
            r = r + 1;
            q[r] = i;
        }
      }
      f = f + 1;
    }
}
```

```
D:\dfs.exe
Enter the number of vertices: 5
Enter the adjacency matrix (0 for no edge, 1 for an edge):
0
1
1
1
0
1
0
1
1
0
1
1
0
0
1
1
1
0
0
1
0
0
1
1
0

BFS Traversal:
A B C D E
DFS Traversal:
A B C E D
No cycle detected in the graph
THE GRAPH IS CONNECTED

Process returned 0 (0x0)   execution time : 9.164 s
Press any key to continue.
```

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

```
#include<stdio.h>


void bfs(int); int a[10][10],
vis[10], n;


void main() {
   int i, j, src; 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;
   }
   printf("enter the src vertex\n"); scanf("%d", &src);
   printf("nodes reachable from src vertex\n"); bfs(src);
}


void bfs(int v) { int q[10], f = 1, r = 1, u, i; q[r]
   = v; vis[v] = 1; while(f <= r) { u = q[f];
   printf("%d ", u); for(i = 1; i <= n; i++) {
   if(a[v][i] == 1 && vis[i] == 0) { vis[i] = 1;
            r = r + 1;
            q[r] = i;
         }
      }
      f = f + 1;
   }
}
```

Output:



Lab Program 10: Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K-> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

#include <stdio.h> #include

<stdlib.h>

int *ht, key[20], n, m; int

index; int count = 0;

void insert(int key) {

index = key % m;

while (ht[index] != -1) {
index = (index + 1) % m;

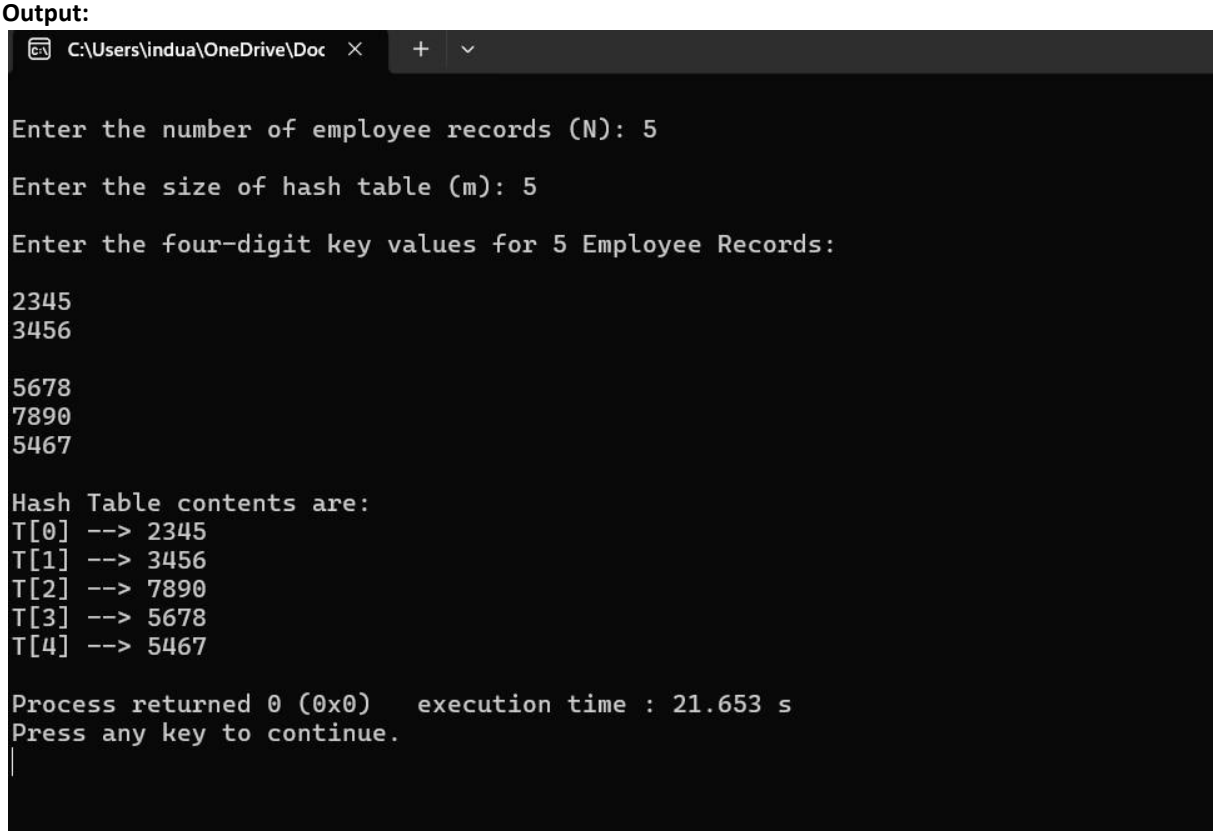
}

ht[index] = key; count++;

}

```
void display() {
   int i;
   if (count == 0) { printf("\nHash Table is
      empty\n"); return;
   }
   printf("\nHash Table contents are:\n"); for (i = 0; i <
   m; i++) {
      printf("T[%d] --> %d\n", i, ht[i]);
   }
}

int main() {
   int i;
   printf("\nEnter the number of employee records (N): "); scanf("%d", &n);

   printf("\nEnter the size of hash table (m): "); scanf("%d", &m);

   ht = (int *)malloc(m * sizeof(int)); for (i = 0; i
   < m; i++) {
      ht[i] = -1;
   }

   printf("\nEnter the four-digit key values for %d Employee Records:\n", n); for (i = 0; i < n; i++) { scanf("%d", &key[i]);
   }

   for (i = 0; i < n; i++) { if (count == m) { printf("\nHash table is full. Cannot insert the record
      with key %d\n", key[i]); break;
      } insert(key[i]);
   }

   display(); free(ht);
   return 0;
}
```

Output:


C:\Users\indua\OneDrive\Doc
Enter the number of employee records (N): 5
Enter the size of hash table (m): 5
Enter the four-digit key values for 5 Employee Records:
2345
3456
5678
7890
5467
Hash Table contents are:
T[0] --> 2345
T[1] --> 3456
T[2] --> 7890
T[3] --> 5678
T[4] --> 5467
Process returned 0 (0x0) execution time : 21.653 s
Press any key to continue.