VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Anthra V (1BM23CS044)

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 Anthra V (1BM23CS044), 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. Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course outcomes:

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

Lab program 1:

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

a) Push

b) Pop

c) Display

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

Program:

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 3

int s[10], TOP=-1, i, item, ch;

void push()

{

  if (TOP == MAX - 1)

  {

    printf("Stack overflow\n");

    return;

  }

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

  scanf("%d", &item);

  TOP = TOP + 1;

  s[TOP] = item;
```

```
}

int pop()
{
    if (TOP == -1)
    {
        printf("Stack underflow\n");
        return -1;
    }
    int item = s[TOP];
    TOP = TOP - 1;
    return item;
}

void display()
{
    if (TOP == -1)
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack contents: \n");
```

```c
    for (i = TOP; i >= 0; i--)
    {
        printf("%d \n", s[i]);
    }
}

void main()
{
    while(1)
    {
        printf("1:PUSH \t 2:POP\t 3:DISPLAY\t 4:EXIT \n");
        printf("Enter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                push();
                break;
            case 2:
                item = pop();
                if (item != -1)
                    printf("Popped element: %d\n", item);
```

```
            break;
          case 3:
            display();
            break;
          case 4:
            exit(0);
        }
    }
}
```

Output :

```
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 1
Enter the element to push: 10
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 1
Enter the element to push: 20
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 1
Enter the element to push: 30
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 3
Stack contents:
30
20
10
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 2
Popped element: 30
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 2
Popped element: 20
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 3
Stack contents:
10
1:PUSH   2:POP   3:DISPLAY        4:EXIT
Enter your choice: 4
```

Lab program 2:

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

Program :

```
#include <stdio.h>

#include <string.h>

int i = 0, pos = 0, top = -1, length;

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

void infixtopostfix();

void push(char symbol);

char pop();

int pred(char symb);


int main()

{

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

   scanf("%s", infix);

   infixtopostfix();

   printf("\nInfix expression:\n%s", infix);

   printf("\nPostfix expression:\n%s", postfix);

   return 0;

}
```

```
void infixtopostfix() {
   length = strlen(infix);
   push('#');
   while (i < length) {
      symbol = infix[i];
      switch (symbol) {
         case '(':
            push(symbol);
            break;
         case ')':
            temp = pop();
            while (temp != '(') {
               postfix[pos++] = temp;
               temp = pop();
            }
            break;
         case '+':
         case '-':
         case '*':
         case '/':
         case '^':
            while (pred(stack[top]) >= pred(symbol)) {
```

```
                temp = pop();
                postfix[pos++] = temp;
            }
            push(symbol);
            break;
        default:
            postfix[pos++] = symbol;
        }
        i++;
    }
    while (top > 0) {
        temp = pop();
        postfix[pos++] = temp;
    }
    postfix[pos] = '\0';
}
void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}
char pop() {
    return stack[top--];
}
```

```
int pred(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;
    default:
      p = -1;
      break;
```

```
    }
    return p;
}
```

Output :

```
Enter infix expression:
(a+b)*c^d

Infix expression:
(a+b)*c^d
Postfix expression:
ab+cd^*
```

Lab program 3:

a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display

The program should print appropriate messages for queue empty and queue overflow conditions

Program:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 3


int q[MAX], item, ch, i, front = -1, rear = -1;


void insert() {

  if (rear == MAX - 1) {

    printf("Queue is full\n");

    return;

  }

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

  {

    front = 0;

    rear = 0;

  }

  else

  {
```

```
        rear = rear + 1;
    }
    printf("Enter element to be inserted: ");
    scanf("%d", &item);
    q[rear] = item;
}

void delete() {
    if (front == -1 && rear == -1) {
        printf("Queue is empty\n");
        item = -1;
        return;
    }
    item = q[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else {
        front = front + 1;
    }
}
```

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

void main()
{
   while (1) {
      printf("1.Insert\t2.Delete\t3.Display\t4.Exit\nEnter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            insert();
            break;
         case 2:
```

```
            delete();
            if (item != -1)
                printf("Deleted item is %d\n", item);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid Choice !!\n");
        }
    }
}
```

Output:

```
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 1
Enter element to be inserted: 10
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 1
Enter element to be inserted: 20
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 1
Enter element to be inserted: 30
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 1
Queue is full
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 3
Queue contents:
10 20 30
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 2
Deleted item is 10
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 2
Deleted item is 20
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 2
Deleted item is 30
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 2
Queue is empty
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 3
Queue is empty
1.Insert    2.Delete    3.Display   4.Exit
Enter your choice: 4
```

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

Program:

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 3


int q[SIZE], ch, i, value, front = -1, rear = -1;


void insert() {

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

    printf("Queue is full\n");

    return;

  }

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

    front = 0;

    rear = 0;

  } else {

    rear = (rear + 1) % SIZE;

  }

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

  scanf("%d", &value);

  q[rear] = value;
```

```
}

int delete() {
    if (front == -1 && rear == -1) {
        printf("Queue is empty\n");
        return -1;
    }
    value = q[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % SIZE;
    }
    return value;
}

void display() {
    if (front == -1 && rear == -1) {
        printf("Circular queue is empty\n");
        return;
    }
    printf("Circular queue contents: ");
```

```
    if (front <= rear) {
        for (i = front; i <= rear; i++) {
            printf("%d\n", q[i]);
        }
    } else {
        for (i = front; i < SIZE; i++) {
            printf("%d\n", q[i]);
        }
        for (i = 0; i <= rear; i++) {
            printf("%d\n", q[i]);
        }
    }
}

void main() {
    while (1) {
        printf("1.Insert \t 2.Delete \t 3.Display \t 4.Exit \n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                insert();
                break;
```

```
        case 2:
          value = delete();
          if (value != -1)
            printf("Deleted element is %d\n", value);
          break;
        case 3:
          display();
          break;
        case 4:
          exit(0);
        default:
          printf("Invalid choice !!\n");
      }
  }
}
```

Output:

```
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 1
Enter element to be inserted: 10
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 1
Enter element to be inserted: 20
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 1
Enter element to be inserted: 30
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 1
Queue is full
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 3
Circular queue contents: 10
20
30
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 2
Deleted element is 10
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 2
Deleted element is 20
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 2
Deleted element is 30
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 2
Queue is empty
1.Insert          2.Delete        3.Display       4.Exit
Enter your choice: 4
[Inferior 1 (process 690) exited normally]
```

Lab program 4:

WAP to Implement Singly Linked List with following operations

a) Create a linkedlist.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

Program:

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *link;

};


typedef struct Node node;

node *pos, *new1, *curr, *start = NULL;


void create();

void display();

void insert();

void beg();

void end();
```

```
void loc();

int main() {
   int ch;
   while (1) {
      printf("1.Create 2.Insert 3.Display 4.Exit\n Enter choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            create();
            break;
         case 2:
            insert();
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
         default:
            printf("Invalid choice\n");
      }
   }
```

```
    return 0;
}

void create() {
    char ch;
    do {
        new1 = (node*)malloc(sizeof(node));
        printf("Enter value: ");
        scanf("%d", &new1->data);
        if (start == NULL) {
            start = new1;
            curr = new1;
        } else {
            curr->link = new1;
            curr = new1;
        }
        printf("Do you want to add another element? (Y/N): ");
        scanf(" %c", &ch);
    } while (ch == 'Y' || ch == 'y');
    curr->link = NULL;
}

void insert() {
```

```
    int x;
    printf("Where to insert the element? 1.Beginning 2.End 3.Location\n Enter your choice: ");
    scanf("%d", &x);
    switch(x) {
        case 1:
            beg();
            break;
        case 2:
            end();
            break;
        case 3:
            loc();
            break;
        default:
            printf("Invalid choice\n");
    }
}

void beg() {
    new1 = (node*) malloc(sizeof(node));
    printf("Enter element to be inserted: ");
    scanf("%d", &new1->data);
```

```
    if (start == NULL) {
        start = new1;
        new1->link = NULL;
    } else {
        new1->link = start;
        start = new1;
    }
}

void end() {
    node *temp;
    new1 = (node*) malloc(sizeof(node));
    printf("Enter element to be inserted: ");
    scanf("%d", &new1->data);
    if (start == NULL) {
        start = new1;
        new1->link = NULL;
    } else {
        temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        new1->link = NULL;
```

```
        temp->link = new1;
    }
}

void loc() {
    node *temp;
    int position, i = 1;
    new1 = (node*) malloc(sizeof(node));
    printf("Enter element to be inserted: ");
    scanf("%d", &new1->data);
    if (start == NULL) {
        start = new1;
        new1->link = NULL;
    } else {
        printf("Enter the position where to insert: ");
        scanf("%d", &position);
        temp = start;
        while (temp != NULL && i < position - 1) {
            temp = temp->link;
            i++;
        }
        if (temp == NULL) {
            printf("Position is greater than the number of elements.\n");
```

```
        } else {
            new1->link = temp->link;
            temp->link = new1;
        }
    }
}

void display() {
    node *temp;
    if (start == NULL) {
        printf("Linked list is empty\n");
        return;
    }
    printf("Elements in the list: ");
    temp = start;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}
```

```
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 1
Enter value: 10
Do you want to add another element? (Y/N): y
Enter value: 20
Do you want to add another element? (Y/N): y
Enter value: 30
Do you want to add another element? (Y/N): n
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 3
Elements in the list: 10 20 30
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 2
Where to insert the element? 1.Beginning 2.End 3.Location
 Enter your choice: 1
Enter element to be inserted: 5
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 2
Where to insert the element? 1.Beginning 2.End 3.Location
 Enter your choice: 2
Enter element to be inserted: 40
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 3
Elements in the list: 5 10 20 30 40
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 2
Where to insert the element? 1.Beginning 2.End 3.Location
 Enter your choice: 3
Enter element to be inserted: 15
Enter the position where to insert: 3
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 3
Elements in the list: 5 10 15 20 30 40
1.Create 2.Insert 3.Display 4.Exit
 Enter choice: 4
```

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

Progarm :

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *link;

};


typedef struct Node node;

node *start = NULL;

node *new1, *curr, *ptr;



void create();

void display();

void DeleteStart();
```

```
void DeletePosition();
void DeleteEnd();


void main() {
  int ch;
  while (1) {
    printf("\n1. Create \n2. Display \n3. Delete from Beginning \n4. Delete at
Position \n5. Delete at End \n6. Exit");
    printf("\nEnter Your Choice: ");
    scanf("%d", &ch);

    switch (ch) {
      case 1: create();
        break;
      case 2: display();
        break;
      case 3: DeleteStart();
        break;
      case 4: DeletePosition();
        break;
      case 5: DeleteEnd();
        break;
```

```
        case 6: exit(0);
      }
   }
}

void create() {
   char ch;

   do {
      new1 = (node*)malloc(sizeof(node));
     printf("\nEnter Value: ");
     scanf("%d",&new1->data);
     if (start==NULL)
     {
        start=new1;
        curr=new1;
     }
     else {
        curr->link = new1;
        curr=new1;
     }

      printf("Do You Want to Add an Element (Y/N)? ");
```

```
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

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

void DeleteStart() {
    if (start == NULL) {
```

```
        printf("\nLinked List is Empty.\n");
        return;
    }

    node *temp = start;
    start = start->link;
    free(temp);
    printf("\nFirst Element Deleted.\n");
}
void DeletePosition() {
    int i=1,pos;
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }
    printf("\nEnter Position: ");
    scanf("%d", &pos);
    node *temp = start;
    node *prev = NULL;
    if (pos == 1) {
        start = temp->link;
        free(temp);
        printf("\nElement at Position %d Deleted.\n", pos);
```

```
        return;
    }
    while (temp != NULL && i < pos) {
        prev = temp;
        temp = temp->link;
        i++;
    }
    if (temp == NULL) {
        printf("\nPosition Not Found.\n");
        return;
    }
    prev->link = temp->link;
    free(temp);
    printf("\nElement at Position %d Deleted\n", pos);
}
void DeleteEnd() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }
    node *temp = start;
    node *prev = NULL;
    if (start->link == NULL) {
```

```
        start = NULL;
        free(temp);
        printf("\nLast Element Deleted.\n");
        return;
    }
    while (temp->link != NULL) {
        prev = temp;
        temp = temp->link;
    }
    prev->link = NULL;
    free(temp);
    printf("\nLast element Deleted.\n");
}
```

Output :

```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 1

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
10 20 30

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 3

First Element Deleted.

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 5

Last element Deleted.
```

Lab program 6:

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

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

#include <stdlib.h>


struct Node {

   int data;

   struct Node *link;

};

typedef struct Node node;


node *start = NULL;

int ch;

char c;


void createList(node **head);

void sort(node *head);

void reverse(node **head);

void display(node *head);

void concatenate();
```

```
void createList(node **head) {
   node *new1, *curr = NULL;
   do {
      new1 = (node*)malloc(sizeof(node));
      if (new1 == NULL) {
         printf("Memory allocation failed!\n");
         exit(1);
      }
      printf("Enter Value: ");
      scanf("%d", &new1->data);
      new1->link = NULL;

      if (*head == NULL) {
         *head = new1;
         curr = new1;
      } else {
         curr->link = new1;
         curr = new1;
      }
      printf("Do you want to add another element (Y/N): ");
      scanf(" %c", &c);
   } while (c == 'y' || c == 'Y');
}
```

```
void sort(node *head) {
    if (head == NULL) {
        printf("The Linked List is Empty.\n");
        return;
    }

    node *i, *j;
    int tempData;
    for (i = head; i != NULL; i = i->link) {
        for (j = i->link; j != NULL; j = j->link) {
            if (i->data > j->data) {
                tempData = i->data;
                i->data = j->data;
                j->data = tempData;
            }
        }
    }
    printf("Linked List is Sorted.\n");
}

void reverse(node **head) {
    node *a = *head, *b = NULL;
```

```
    while (a != NULL) {
        node *temp = a->link;
        a->link = b;
        b = a;
        a = temp;
    }
    *head = b;
    printf("Linked List is Reversed.\n");
}

void display(node *head) {
    if (head == NULL) {
        printf("Linked list is Empty\n");
        return;
    }

    node *temp = head;
    printf("Elements in Linked List:\n");
    while (temp != NULL) {
        printf("%d\t", temp->data);
        temp = temp->link;
    }
    printf("\n");
```

```
}

void concatenate() {
    node *start2 = NULL;

    printf("Creating the second linked list:\n");
    createList(&start2);

    if (start == NULL) {
        start = start2;
    } else {
        node *temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = start2;
    }
    printf("Lists concatenated successfully.\n");
}

int main() {
    while (1) {
```

```
    printf("\n1. Create 1st Linked List\n2. Sort Linked List\n3. Reverse Linked List\n4. Concatenate Linked Lists\n5. Display Linked List\n6. Exit\n");
    printf("Enter Your Choice: ");
    scanf("%d", &ch);
    switch (ch) {
      case 1:
        createList(&start);
        break;
      case 2:
        sort(start);
        break;
      case 3:
        reverse(&start);
        break;
      case 4:
        concatenate();
        break;
      case 5:
        display(start);
        break;
      case 6:
        exit(0);
        break;
```

```
        default:

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

          break;

      }

    }

}
```

```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 1
Enter Value: 20
Do you want to add another element (Y/N): y
Enter Value: 10
Do you want to add another element (Y/N): y
Enter Value: 15
Do you want to add another element (Y/N): n

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
20      10      15

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 2
Linked List is Sorted.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      15      20

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 3
Linked List is Reversed.
```

```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
20      15      10

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 4
Creating the second linked list:
Enter Value: 25
Do you want to add another element (Y/N): y
Enter Value: 30
Do you want to add another element (Y/N): y
Enter Value: 35
Do you want to add another element (Y/N): n
Lists concatenated successfully.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
20      15      10      25      30      35

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 6
```

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

Program :

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
   int data;
   struct Node *link;
};
typedef struct Node node;

node *top=NULL;
void push();
void pop();
void displayStack();

void push(){
   node *new1=(node*)malloc(sizeof(node));
   if(new1==NULL){
      printf("\nStack Overflow.\n");
      return;
   }
```

```
    printf("\nEnter Value to Push: ");
    scanf("%d", &new1->data);
    new1->link=top;
    top=new1;
}

void pop(){
    if(top==NULL){
        printf("\nStack Underflow.\n");
        return;
    }

    node *temp=top;
    printf("\nPopped Element: %d\n", temp->data);
    top=top->link;
    free(temp);
}

void displayStack(){
    if(top==NULL){
        printf("\nThe Stack is Empty.\n");
        return;
```

```
    }

    printf("\nElements in the Stack: ");
    node *temp=top;
    while(temp!=NULL){
        printf("%d ", temp->data);
        temp=temp->link;
    }
    printf("\n");
}

node *front=NULL, *rear=NULL;

void insert();
void del();
void displayQueue();

void insert(){
    node *new1=(node*)malloc(sizeof(node));
    if(new1==NULL){
        printf("\nQueue Full.\n");
        return;
    }
```

```
    printf("\nEnter Value to Insert: ");
    scanf("%d", &new1->data);
    new1->link=NULL;

    if(rear==NULL){
        front=rear=new1;
        return;
    }
    rear->link=new1;
    rear=new1;
}

void del(){
    if(front==NULL){
        printf("\nQueue Empty.\n");
        return;
    }

    node *temp=front;
    printf("\nDeleted Element: %d\n", temp->data);
    front=front->link;
```

```
    if(front==NULL){
        rear=NULL;
    }
    free(temp);
}

void displayQueue(){
    if(front==NULL){
        printf("\nThe Queue is Empty.\n");
        return;
    }

    printf("\nElements in the Queue: ");
    node *temp=front;
    while(temp!=NULL){
        printf("%d ", temp->data);
        temp=temp->link;
    }
    printf("\n");
}

void main(){
    int ch;
```

```
    while(1){
        printf("\n1. Push (Stack) \n2. Pop (Stack) \n3. Display (Stack)");
        printf("\n4. Insert (Queue) \n5. Delete (Queue) \n6. Display (Queue) \n7.
Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                insert();
                break;
            case 5:
                del();
```

```
            break;
          case 6:
            displayQueue();
            break;
          case 7:
            exit(0);
          default:
            printf("\nEnter Your Choice: \n");
        }
    }
}
```

Output :

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 20

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 30

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 3

Elements in the Stack: 30 20 10
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 2

Popped Element: 30

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 2

Popped Element: 20

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 2

Popped Element: 10
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 5

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 15

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 25

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 5 15 25
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 5

Deleted Element: 5

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 15 25

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 7
```

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

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   struct Node *prev;

   int data;

   struct Node *next;

};


typedef struct Node node;


node *start = NULL, *curr = NULL;


void create_dll() {

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

   printf("Enter element: ");
```

```
scanf("%d", &new1->data);
new1->prev = NULL;
new1->next = NULL;
start = new1;
curr = new1;

char ch;
while (1) {
  printf("Do you want to add another element (Y/N): ");
  scanf(" %c", &ch);
  if (ch == 'Y' || ch == 'y') {
    new1 = (node *)malloc(sizeof(node));
    printf("Enter element: ");
    scanf("%d", &new1->data);
    new1->prev = curr;
    new1->next = NULL;
    curr->next = new1;
    curr = new1;
  } else {
    curr->next = NULL;
    return;
  }
}
```

```
}

void insert_left() {
    node *new1 = (node *)malloc(sizeof(node));
    printf("Enter element: ");
    scanf("%d", &new1->data);
    printf("Enter position: ");
    int pos;
    scanf("%d", &pos);

    if (pos == 1) {
        new1->next = start;
        if (start != NULL) {
            start->prev = new1;
        }
        new1->prev = NULL;
        start = new1;
        return;
    }

    int i = 1;
    node *temp = start;
    while (i < pos - 1 && temp != NULL) {
```

```
        temp = temp->next;
        i++;
    }

    if (temp == NULL) {
        printf("Entered position is greater than the number of elements.\n");
        free(new1);
        return;
    }

    new1->next = temp->next;
    new1->prev = temp;

    if (temp->next != NULL) {
        temp->next->prev = new1;
    }
    temp->next = new1;
}

void delete_loc() {
    int ele;
    if (start == NULL) {
        printf("Doubly Linked list is empty\n");
```

```
    return;
}
printf("Enter element: ");
scanf("%d", &ele);

node *temp = start;
if (start->data == ele) {
    start = start->next;
    if (start != NULL) {
        start->prev = NULL;
    }
    free(temp);
    return;
}

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

if (temp == NULL) {
    printf("Element not found\n");
    return;
}
```

```c
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    free(temp);
}

void display_dll() {
    node *temp = start;
    if (temp == NULL) {
        printf("Doubly Linked list is empty\n");
        return;
    }

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

```
}

int main() {
   while (1) {
      printf("1. Create DLL\t2. Insert at left\t3. Delete given element\t4. Display
DLL\t5. Exit\nEnter your choice: ");
      int ch;
      scanf("%d", &ch);

      switch (ch) {
         case 1: create_dll();
               break;
         case 2: insert_left();
               break;
         case 3: delete_loc();
               break;
         case 4: display_dll();
               break;
         case 5: exit(0);
         default: printf("Invalid choice\n");
      }
   }
   return 0;
```

```
}
```

Output:

```
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 1
Enter element: 10
Do you want to add another element (Y/N): y
Enter element: 20
Do you want to add another element (Y/N): y
Enter element: 25
Do you want to add another element (Y/N): y
Enter element: 30
Do you want to add another element (Y/N): n
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 4
Doubly Linked List: 10 20 25 30
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 2
Enter element: 15
Enter position: 2
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 4
Doubly Linked List: 10 15 20 25 30
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 3
Enter element: 30
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 4
Doubly Linked List: 10 15 20 25
1. Create DLL   2. Insert at left       3. Delete given element 4. Display DLL  5. Exit
Enter your choice: 5


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

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.

Program :

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

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

node* createNode(int data) {
   node* new1 = (node*)malloc(sizeof(node));
   new1->data = data;
   new1->left = new1->right = NULL;
   return new1;
}
```

```
node* insertNode(node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }
   if (data < root->data) {
      root->left = insertNode(root->left, data);
   } else {
      root->right = insertNode(root->right, data);
   }
   return root;
}

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

void preorderTraversal(node* root) {
   if (root != NULL) {
      printf("%d ", root->data);
```

```
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

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

void displayTree(node* root, int space) {
    if (root == NULL) {
        return;
    }

    space += 10;

    displayTree(root->right, space);

    printf("\n");
```

```
    for (int i = 10; i < space; i++) {
        printf(" ");
    }
    printf("%d\n", root->data);

    displayTree(root->left, space);
}

int main() {
    node* root = NULL;
    int choice, value;

    printf("Binary Search Tree Operations:\n");
    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-
order Traversal\n5. Display Tree\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
    root = insertNode(root, value);
    break;
case 2:
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");
    break;
case 3:
    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");
    break;
case 4:
    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
    break;
case 5:
    printf("Tree Representation:\n");
    displayTree(root, 0);
    printf("\n");
    break;
case 6:
```

```
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display Tree
6. Exit
Enter your choice: 2
In-order Traversal: 1 2 3 4 5

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display Tree
6. Exit
Enter your choice: 3
Pre-order Traversal: 5 3 2 1 4

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display Tree
6. Exit
Enter your choice: 4
Post-order Traversal: 1 2 4 3 5

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Tree Representation:

5

                    4

          3

                    2

                              1
```

Lab program 9:

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

Program:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100


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


void enqueue(int item) {

   if (rear == MAX - 1) {

      printf("Queue Overflow\n");

      return;

   }

   if (front == -1) front = 0;

   queue[++rear] = item;

}


int dequeue() {

   if (front == -1 || front > rear) {

      printf("Queue Underflow\n");

      return -1;

   }
```

```
    return queue[front++];
}

void bfs(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    int i;
    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 1; i <= n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}
```

```
void main() {
    int n, i, j, start;
    int graph[MAX][MAX], visited[MAX] = {0};

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &graph[i][j]);

    printf("Enter the starting vertex: ");
    scanf("%d", &start);

    bfs(graph, visited, start, n);
}
```

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
Enter the starting vertex: 3
BFS Traversal: 3 1 4 5 2
```

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

Program:

```
#include <stdio.h>
#define MAX 10

int a[MAX][MAX], vis[MAX], n;

void dfs(int v);

int main() {
  int i, j;

  printf("Enter number of vertices: ");
  scanf("%d", &n);

  printf("Enter adjacency matrix:\n");
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
```

```
for (i = 1; i <= n; i++) {
   vis[i] = 0;
}

printf("DFS traversal: ");
dfs(1);



int isConnected = 1;
for (i = 1; i <= n; i++) {
   if (vis[i] == 0) {
      isConnected = 0;
      break;
   }
}

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

```
    printf("\n");
    return 0;
}


void dfs(int v) {
    printf("%d ", v);
    vis[v] = 1;


    for (int i = 1; i <= n; i++) {
        if (a[v][i] == 1 && vis[i] == 0) {
            dfs(i);
        }
    }
}
```

Output :

```
Enter number of vertices: 5
Enter adjacency matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
DFS traversal: 1 3 4 2 5
The graph is connected.
```

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.

Program :

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_EMPLOYEES 100

#define TABLE_SIZE 10


typedef struct {

   int key;

   char name[50];

} Employee;


Employee hashTable[TABLE_SIZE];

int occupied[TABLE_SIZE] = {0};
```

```
int hashFunction(int key) {
  return key % TABLE_SIZE;
}

void insertEmployee(int key, const char *name) {
  int index = hashFunction(key);
  while (occupied[index]) {
    index = (index + 1) % TABLE_SIZE;
  }
  hashTable[index].key = key;
  strcpy(hashTable[index].name, name);
  occupied[index] = 1;
}

Employee* searchEmployee(int key) {
  int index = hashFunction(key);
  while (occupied[index]) {
    if (hashTable[index].key == key) {
      return &hashTable[index];
    }
    index = (index + 1) % TABLE_SIZE;
  }
  return NULL;
```

```
}


void displayHashTable() {
   for (int i = 0; i < TABLE_SIZE; i++) {
      if (occupied[i]) {
         printf("Index %d: Key = %d, Name = %s\n", i, hashTable[i].key,
hashTable[i].name);
      } else {
         printf("Index %d: Empty\n", i);
      }
   }
}


int main() {
   insertEmployee(1234, "Alice");
   insertEmployee(2345, "Bob");
   insertEmployee(3456, "Charlie");
   insertEmployee(4567, "David");
   insertEmployee(5678, "Eve");
   insertEmployee(6789, "Frank");

   displayHashTable();
```

```
    int searchKey = 2345;

    Employee* emp = searchEmployee(searchKey);

    if (emp) {

        printf("Found: Key = %d, Name = %s\n", emp->key, emp->name);

    } else {

        printf("Employee with key %d not found.\n", searchKey);

    }


    return 0;

}
```

Output :

```
Index 0: Empty
Index 1: Empty
Index 2: Empty
Index 3: Empty
Index 4: Key = 1234, Name = Alice
Index 5: Key = 2345, Name = Bob
Index 6: Key = 3456, Name = Charlie
Index 7: Key = 4567, Name = David
Index 8: Key = 5678, Name = Eve
Index 9: Key = 6789, Name = Frank
Found: Key = 2345, Name = Bob
```