**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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



**LAB REPORT**

**on**

**DATA STRUCTURES**

***Submitted by:***

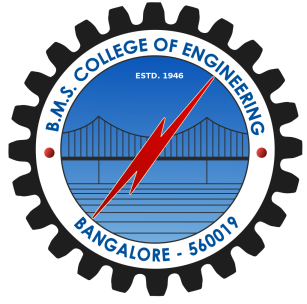
**JIGAR D PATEL (1BM21CS081)**

***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**

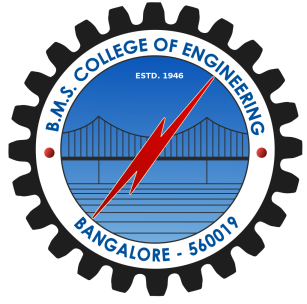
**Oct 2022-Feb 2023**

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

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by **JIGAR D PATEL (1BM21CS081),** 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 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of **Data structures Lab - (22CS3PCDST)** work prescribed for the said degree.

**Dr. Seema Patil**             **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Table Of Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Experiment Title** | | | **Page No.** |
| **1** | **Course Outcomes** | | | **3** |
| **2** | **Experiments** | | | **3-64** |
|  | **2.1** | **Experiment - 1** | | **3-7** |
| **2.1.1** | **Question:**  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. | **3** |
| **2.1.2** | **Code** | **3** |
| **2.1.3** | **Output** | **6** |
| **2.2** | **Experiment - 2** | | **8-11** |
| **2.2.1** | **Question:**  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) | **8** |
| **2.2.2** | **Code** | **8** |
| **2.2.3** | **Output** | **11** |
| **2.3** | **Experiment - 3** | | **12-15** |
| **2.3.1** | **Question:**  WAP to simulate the working of a queue of integers using an array. Provide the following operations:   1. Insert 2. Delete 3. Display   The program should print appropriate messages for queue empty and queue overflow conditions. | **12** |
|  | **2.3.2** | **Code** | **12** |
|  | **2.3.3** | **Output** | **14** |
| **2.4.** | **Experiment - 4** | | **16-20** |
| **2.4.1** | **Question:**  WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:   1. Insert 2. Delete 3. Display   The program should print appropriate messages for queue empty and queue overflow conditions. | **16** |
| **2.4.2** | **Code** | **16** |
| **2.4.3** | **Output** | **19** |
| **2.5.** | **Experiment - 5** | | **21-25** |
| **2.5.1** | **Question:**  WAP to Implement Singly Linked List with following operations:   1. Create a linked list. 2. Insertion of a node at first position, at any position and at end of list. 3. Display the contents of the linked list. | **21** |
| **2.5.2** | **Code** | **21** |
| **2.5.3** | **Output** | **25** |
| **2.6** | **Experiment - 6** | | **26-33** |
| **2.6.1** | **Question:**  WAP to Implement Singly Linked List with following operations:   1. Create a linked list. 2. Deletion of first element, specified element and last element in the list. 3. Display the contents of the linked list. | **26** |
| **2.6.2** | **Code** | **26** |
| **2.6.3** | **Output** | **31** |
| **2.7** | **Experiment - 7** | | **34-42** |
| **2.7.1** | **Question:**  WAP to Implement Single Link List with following operations:   1. Sort the linked list. 2. Reverse the linked list. 3. Concatenation of two linked lists. | **34** |
| **2.7.2** | **Code** | **34** |
| **2.7.3** | **Output** | **41** |
| **2.8** | **Experiment - 8** | | **43-49** |
| **2.8.1** | **Question:**  WAP to implement Stack & Queues using Linked Representation. | **43** |
| **2.8.2** | **Code** | **43** |
| **2.8.3** | **Output** | **47** |
| **2.9** | **Experiment - 9** | | **50-57** |
| **2.9.1** | **Question:**  WAP to Implement doubly link list with primitive operations:   1. Create a doubly linked list. 2. Insert a new node to the left of the node. 3. Delete the node based on a specific value. 4. Display the contents of the list. | **50** |
| **2.9.2** | **Code** | **50** |
| **2.9.3** | **Output** | **56** |
| **2.10** | **Experiment - 10** | | **58-62** |
| **2.10.1** | **Question:**  Write a program:   1. To construct a binary Search tree. 2. To traverse the tree using all the methods i.e., in-order, preorder and post order. 3. To display the elements in the tree. | **58** |
| **2.10.2** | **Code** | **58** |
| **2.10.3** | **Output** | **61** |

1. **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.

1. **Experiments:**

**2.1 Experiment: 1**

**2.1.1 Question:**

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

1. Push
2. Pop
3. Display

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

**2.1.2 Code:**

#include<stdio.h>

int stack [100], choice, n, top,x,i;

void push(void);

void pop(void);

void display(void);

int main ()

{

top=-1;

printf("\nEnter the size of STACK");

scanf("%d",&n);

printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");

do

{

printf("\nEnter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1: push();

break;

case 2: pop();

break;

case 3: display();

break;

case 4: printf("\nExit");

break;

default: printf ("\nEnter a Valid Choice");

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\nSTACK overflow");

}

else

{

printf("\nEnter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack underflow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

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

printf("\n Enter Next Choice");

}

else

{

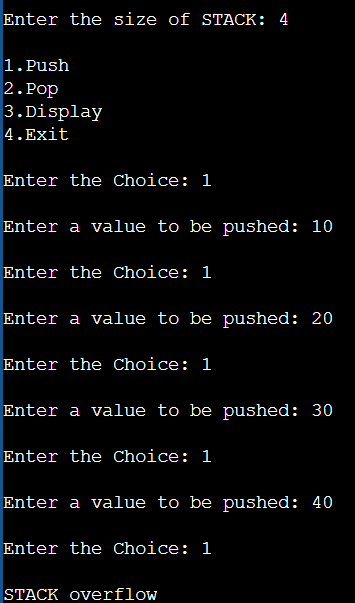
printf("\n The STACK is empty");

}

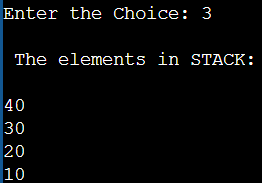
}

**2.1.3 Output:**

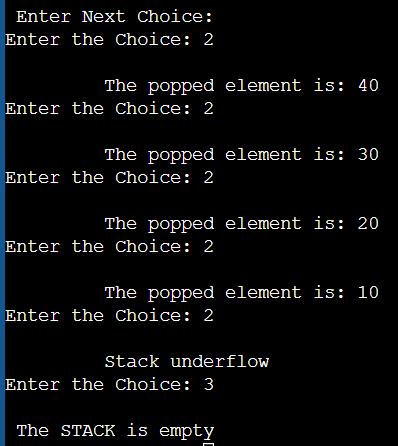
**//Inserting Elements in Stack**



**//Display Elements of Stack**



**//Deleting Elements from Stack**



**2.2 Experiment: 2**

**2.2.1 Question:**

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)

**2.2.2 Code:**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nSTACK Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("STACK Underflow");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

if(symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix\_exp[i];

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

postfix\_exp[j] = '\0';

}

int main()

{

char infix[SIZE], postfix[SIZE];

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

gets(infix);

InfixToPostfix(infix,postfix);

printf("Postfix Expression: ");

puts(postfix);

return 0;

}

**2.2.3 Output:**





**2.3 Experiment: 3**

**2.3.1 Question:**

WAP to simulate the working of a queue of integers using an array. Provide the following operations:

1. Insert
2. Delete
3. Display

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

**2.3.2 Code:**

#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

void enqueue(int);

void dequeue();

void display();

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

void main()

{

int value, choice;

printf(“Queue Operations: ”);

printf("1. Insert\n2. Delete\n3. Display\n4. Exit");

while(1)

{

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

enqueue(value);

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nInvalid Input");

}

}

}

void enqueue(int value)

{

if(rear == SIZE-1)

printf("\nQueue is Full!");

else

{

if(front == -1)

front = 0;

rear++;

queue[rear] = value;

}

}

void dequeue()

{

if(front == rear)

printf("\nQueue is Empty!");

else

{

printf("\nDeleted : %d", queue[front]);

front++;

}

}

void display()

{

if(rear == -1)

printf("\nQueue is Empty!!!");

else

{

int i;

printf("\nQueue elements are:\n");

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

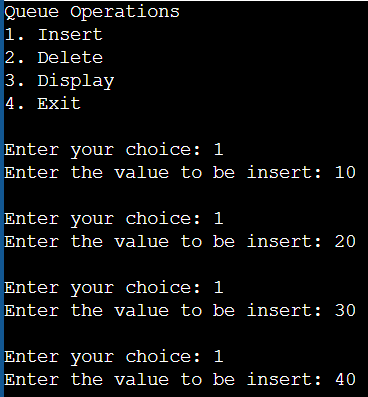
printf("%d\n",queue[i]);

}

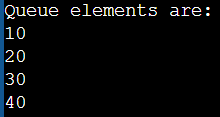
}

**2.3.3 Output:**

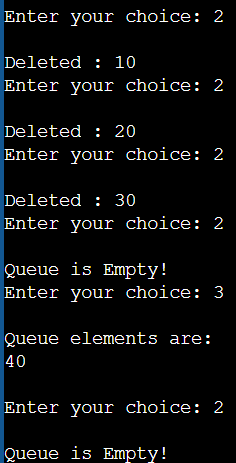
**//Insert in Queue**



**//Display Elements in Queue**



**//Deleting From Queue**



**2.4 Experiment: 4**

**2.4.1 Question:**

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:

1. Insert
2. Delete
3. Display

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

**2.4.2 Code:**

#include<stdio.h>

# define MAX 5

int cqueue\_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item) {

if((front == 0 && rear == MAX-1) || (front == rear+1))

{

printf("\nOVERFLOW\n");

return;

}

if(front == -1)

{

front = 0;

rear = 0;

}

else

{

if(rear == MAX-1)

rear = 0;

else

rear = rear+1;

}

cqueue\_arr[rear] = item ;

}

void delete() {

if(front == -1)

{

printf("\nUNDERFLOW\n");

return ;

}

printf("\nElement deleted from queue is : %d\n",cqueue\_arr[front]);

if(front == rear)

{

front = -1;

rear=-1;

}

else

{

if(front == MAX-1)

front = 0;

else

front = front+1;

}

}

void display()

{

if(front == -1)

{

printf("\nQueue is empty\n");

return;

}

else

{

int i;

printf("\nQueue elements are:\n");

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

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

}

}

int main()

{

int choice,item;

printf("Circular Queue Operations: \n");

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Exit\n");

do

{

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("\nInput the element for insertion in queue : ");

scanf("%d", &item);

insert(item);

break;

case 2: delete();

break;

case 3: display();

break;

case 4: break;

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

}

}

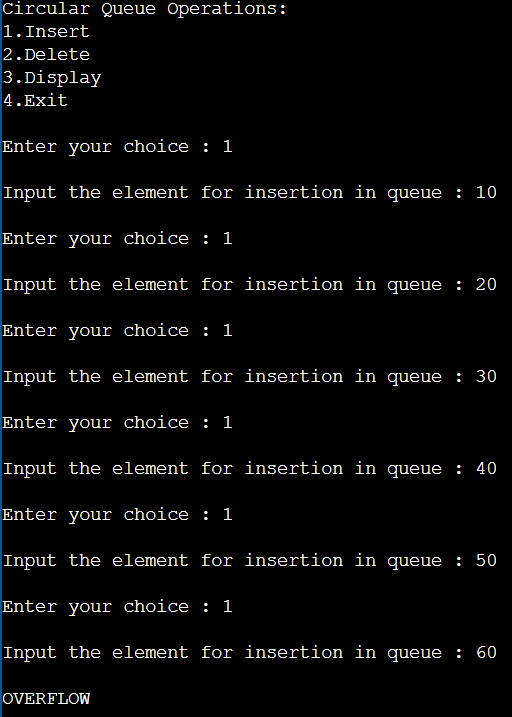
while(choice!=4);

return 0;

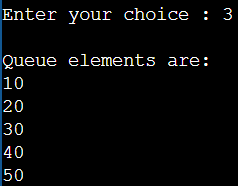
}

**Output:**

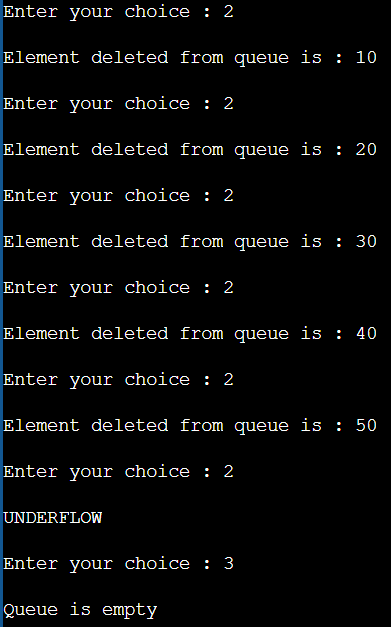
**//Inserting Elements in Circular Queue**



**//Display Elements in Circular Queue**



**//Deleting Elements from Circular Queue**



**2.5 Experiment: 5**

**2.5.1 Question:**

WAP to Implement Singly Linked List with following operations:

1. Create a linked list.
2. Insertion of a node at first position, at any position and at end of list.
3. Display the contents of the linked list.

**2.5.2 Code:**

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node\* link;

};

struct node\* start = NULL;

void createList()

{

if (start == NULL) {

int n;

printf("\nEnter the number of nodes: ");

scanf("%d", &n);

if (n != 0) {

int data;

struct node\* newnode;

struct node\* temp;

newnode = malloc(sizeof(struct node));

start = newnode;

temp = start;

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

start->info = data;

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

newnode = malloc(sizeof(struct node));

temp->link = newnode;

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

newnode->info = data;

temp = temp->link;

}

}

printf("\nList Created\n");

}

else

printf("\nThe list is already created\n");

}

void traverse()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else

{

temp = start;

while (temp != NULL)

{

printf("Element = %d\n", temp->info);

temp = temp->link;

}

}

}

void insertFront()

{

int data;

struct node\* temp;

temp = malloc(sizeof(struct node));

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

temp->info = data;

temp->link = start;

start = temp;

}

void insertEnd()

{

int data;

struct node \*temp, \*head;

temp = malloc(sizeof(struct node));

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

temp->link = 0;

temp->info = data;

head = start;

while (head->link != NULL) {

head = head->link;

}

head->link = temp;

}

void insertPosition()

{

struct node \*temp, \*newnode;

int pos, data, i = 1;

newnode = malloc(sizeof(struct node));

printf("\nEnter position and data :");

scanf("%d %d", &pos, &data);

temp = start;

newnode->info = data;

newnode->link = 0;

while (i < pos - 1) {

temp = temp->link;

i++;

}

newnode->link = temp->link;

temp->link = newnode;

}

int main()

{

printf("Singly Linked List Insertion and Display Operations");

printf("\n1. Insert At First Position");

printf("\n2. Insert At Last Position");

printf("\n3. Insert At Any Position");

printf("\n4. Display Elements");

printf("\n5. Exit");

while(1){

int choice;

printf("\n\nEnter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1: insertFront();

break;

case 2: insertEnd();

break;

case 3: insertPosition();

break;

case 4: traverse();

break;

case 5: exit(1);

break;

default: printf("Wrong choice!!!!!!\n");

}

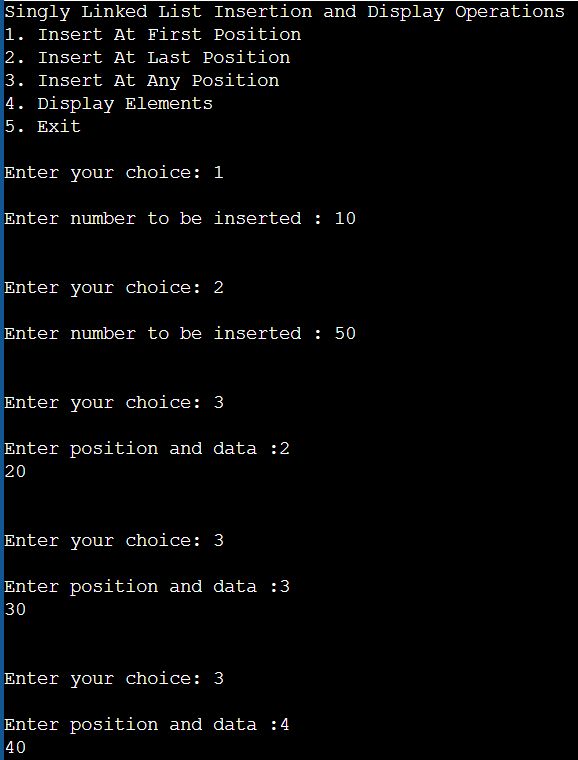
}

return 0;

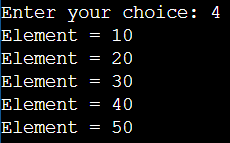
}

**2.5.3 Output:**

**//Inserting Elements in A Singly Linked List**



**//Display Elements in A Singly Linked List**



**2.6 Experiment: 6**

**2.6.1 Question:**

WAP to Implement Singly Linked List with following operations:

1. Create a linked list.
2. Deletion of first element, specified element and last element in the list.
3. Display the contents of the linked list.

**2.6.2 Code:**

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node\* link;

};

struct node\* start = NULL;

void createList()

{

if (start == NULL) {

int n;

printf("\nEnter the number of nodes: ");

scanf("%d", &n);

if (n != 0) {

int data;

struct node\* newnode;

struct node\* temp;

newnode = malloc(sizeof(struct node));

start = newnode;

temp = start;

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

start->info = data;

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

newnode = malloc(sizeof(struct node));

temp->link = newnode;

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

newnode->info = data;

temp = temp->link;

}

}

printf("\nList Created\n");

}

else

printf("\nThe list is already created\n");

}

void insertFront()

{

int data;

struct node\* temp;

temp = malloc(sizeof(struct node));

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

temp->info = data;

temp->link = start;

start = temp;

}

void insertEnd()

{

int data;

struct node \*temp, \*head;

temp = malloc(sizeof(struct node));

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

temp->link = 0;

temp->info = data;

head = start;

while (head->link != NULL) {

head = head->link;

}

head->link = temp;

}

void insertPosition()

{

struct node \*temp, \*newnode;

int pos, data, i = 1;

newnode = malloc(sizeof(struct node));

printf("\nEnter position and data :");

scanf("%d %d", &pos, &data);

temp = start;

newnode->info = data;

newnode->link = 0;

while (i < pos - 1) {

temp = temp->link;

i++;

}

newnode->link = temp->link;

temp->link = newnode;

}

void deleteFirst()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else

{

temp = start;

start = start->link;

free(temp);

}

}

void deleteLast()

{

struct node \*temp, \*prevnode;

if (start==NULL)

printf("\nList is Empty\n");

else

{

temp=start;

while(temp->link!=0)

{

prevnode=temp;

temp=temp->link;

}

free(temp);

prevnode->link = 0;

}

}

void deletePosition()

{

struct node \*temp, \*position;

int i=1, pos;

if(start==NULL)

printf("\nList is Empty\n");

else

{

printf("Enter Position: ");

scanf("%d", &pos);

position=malloc(sizeof(struct node));

temp=start;

while(i<pos-1)

{

temp=temp->link;

i++;

}

position = temp->link;

temp->link = position->link;

free(position);

}

}

void display()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else

{

temp = start;

while (temp != NULL)

{

printf("Element = %d\n", temp->info);

temp = temp->link;

}

}

}

int main()

{

printf("Singly Linked List Deletion and Display Operations");

printf("\n1. Insert At First Position");

printf("\n2. Insert At Last Position");

printf("\n3. Insert At Any Position");

printf("\n4. Delete At First Position");

printf("\n5. Delete At Last Position");

printf("\n6. Delete At Any Position");

printf("\n7. Display Elements");

printf("\n8. Exit\n");

while(1){

int choice;

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1: insertFront();

break;

case 2: insertEnd();

break;

case 3: insertPosition();

break;

case 4: deleteFirst();

display();

break;

case 5: deleteLast();

display();

break;

case 6: deletePosition();

display();

break;

case 7: display();

break;

case 8: exit(1);

break;

default: printf("Wrong choice!!!!!!\n");

}

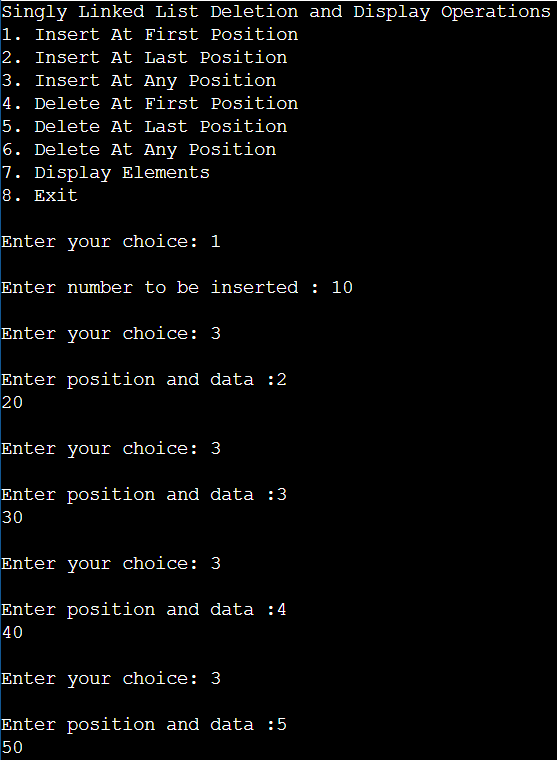
}

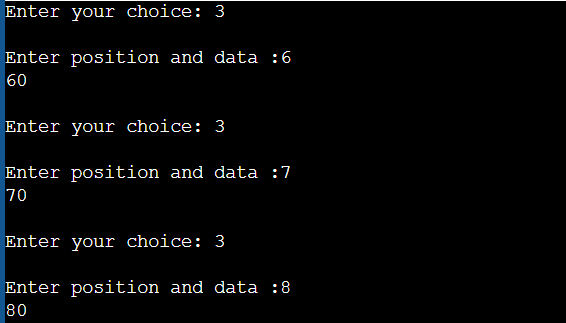
return 0;

}

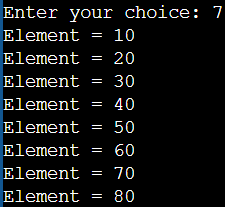
**2.6.3 Output:**

**//Inserting Elements in Singly Linked List**

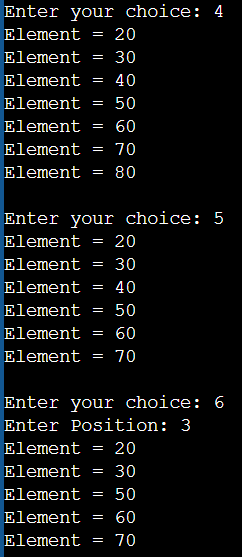




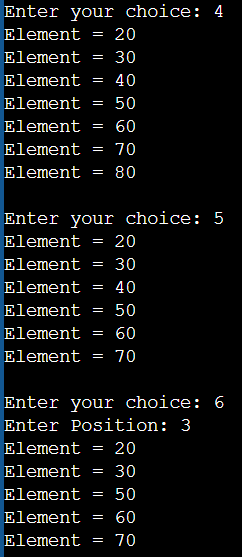
**//Display Elements**



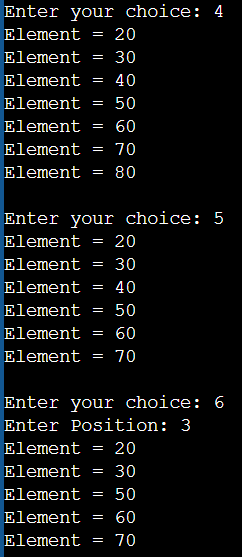
**//Delete At First Position**



**//Delete At Last Position**



**//Delete At Any Position**



**2.7 Experiment: 7**

**2.7.1 Question:**

WAP to Implement Single Link List with following operations:

1. Sort the linked list.
2. Reverse the linked list.
3. Concatenation of two linked lists.

**2.7.2 Code:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*link;

};

struct node\* start = NULL;

struct node \*create(struct node \*start);

struct node \*insert\_s(struct node \*start,int data);

struct node \*insert(struct node \*start,int data);

void display(struct node \*start );

void merge(struct node \*p1,struct node \*p2);

void merge(struct node \*p1,struct node \*p2)

{

struct node \*start3;

start3=NULL;

while(p1!=NULL && p2!=NULL)

{

if(p1->info < p2->info)

{

start3=insert(start3,p1->info);

p1=p1->link;

}

else if(p2->info < p1->info)

{

start3=insert(start3,p2->info);

p2=p2->link;

}

else if(p1->info==p2->info)

{

start3=insert(start3,p1->info);

p1=p1->link;

p2=p2->link;

}

}

while(p1!=NULL)

{

start3=insert(start3,p1->info);

p1=p1->link;

}

while(p2!=NULL)

{

start3=insert(start3,p2->info);

p2=p2->link;

}

printf("Merged list is : ");

display(start3);

}

struct node \*create(struct node \*start )

{

int i,n,data;

printf("Enter the number of nodes : ");

scanf("%d",&n);

start=NULL;

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

{

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

scanf("%d",&data);

start=insert\_s(start, data);

}

return start;

}

struct node \*insert\_s(struct node \*start,int data)

{

struct node \*p,\*tmp;

tmp=(struct node \*)malloc(sizeof(struct node));

tmp->info=data;

if(start==NULL || data<start->info)

{

tmp->link=start;

start=tmp;

return start;

}

else

{

p=start;

while(p->link!=NULL && p->link->info < data)

p=p->link;

tmp->link=p->link;

p->link=tmp;

}

return start;

}

struct node \*insert(struct node \*start,int data)

{

struct node \*p,\*tmp;

tmp=(struct node \*)malloc(sizeof(struct node));

tmp->info=data;

if(start==NULL)

{

tmp->link=start;

start=tmp;

return start;

}

else

{

p=start;

while(p->link!=NULL)

p=p->link;

tmp->link=p->link;

p->link=tmp;

}

return start;

}

void display(struct node \*start)

{

struct node \*p;

if(start==NULL)

{

printf("List is empty\n");

return;

}

p=start;

while(p!=NULL)

{

printf("%d ",p->info);

p=p->link;

}

printf("\n");

}

void sort()

{

struct node\* current = start;

struct node\* index = NULL;

int temp;

if (start == NULL) {

return;

}

else {

while (current != NULL) {

index = current->link;

while (index != NULL) {

if (current->info > index->info) {

temp = current->info;

current->info = index->info;

index->info = temp;

}

index = index->link;

}

current = current->link;

}

}

}

void reverseLL()

{

struct node \*t1, \*t2, \*temp;

t1 = t2 = NULL;

if (start == NULL)

printf("List is empty\n");

else {

while (start != NULL) {

t2 = start->link;

start->link = t1;

t1 = start;

start = t2;

}

start = t1;

temp = start;

printf("Reversed linked list is : ");

while (temp != NULL) {

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

temp = temp->link;

}

}

}

void insertAtFront()

{

int data;

struct node\* temp;

temp = malloc(sizeof(struct node));

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

temp->info = data;

temp->link = start;

start = temp;

}

void insertAtEnd()

{

int data;

struct node \*temp, \*head;

temp = malloc(sizeof(struct node));

printf("\nEnter number to be inserted : ");

scanf("%d", &data);

temp->link = 0;

temp->info = data;

head = start;

while (head->link != NULL) {

head = head->link;

}

head->link = temp;

}

void insertAtPosition()

{

struct node \*temp, \*newnode;

int pos, data, i = 1;

newnode = malloc(sizeof(struct node));

printf("\nEnter position and data :");

scanf("%d %d", &pos, &data);

temp = start;

newnode->info = data;

newnode->link = 0;

while (i < pos - 1) {

temp = temp->link;

i++;

}

newnode->link = temp->link;

temp->link = newnode;

}

void traverse()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else {

temp = start;

while (temp != NULL) {

printf("Data = %d\n", temp->info);

temp = temp->link;

}

}

}

int main()

{

int choice;

struct node \*start1=NULL,\*start2=NULL;

printf("Singly Link List Sort, Reverse & Concatenate Operations");

printf("\n1. Display List\n");

printf("2. Insert At First Position\n");

printf("3. Insert At Last Position\n");

printf("4. Insert At Any Position\n");

printf("5. Sort The List\n");

printf("6. Reverse the list\n");

printf("7. Concatenate 2 Linked Lists\n");

printf("8. To exit\n");

while (1) {

printf("\nEnter Choice: \n");

scanf("%d", &choice);

switch (choice) {

case 1: traverse();

break;

case 2: insertAtFront();

break;

case 3: insertAtEnd();

break;

case 4: insertAtPosition();

break;

case 5: sort();

traverse();

break;

case 6: reverseLL();

break;

case 7: start1=create(start1);

start2=create(start2);

printf("List1 : ");

display(start1);

printf("List2 : ");

display(start2);

merge(start1, start2);

break;

case 8: exit(1);

break;

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

}

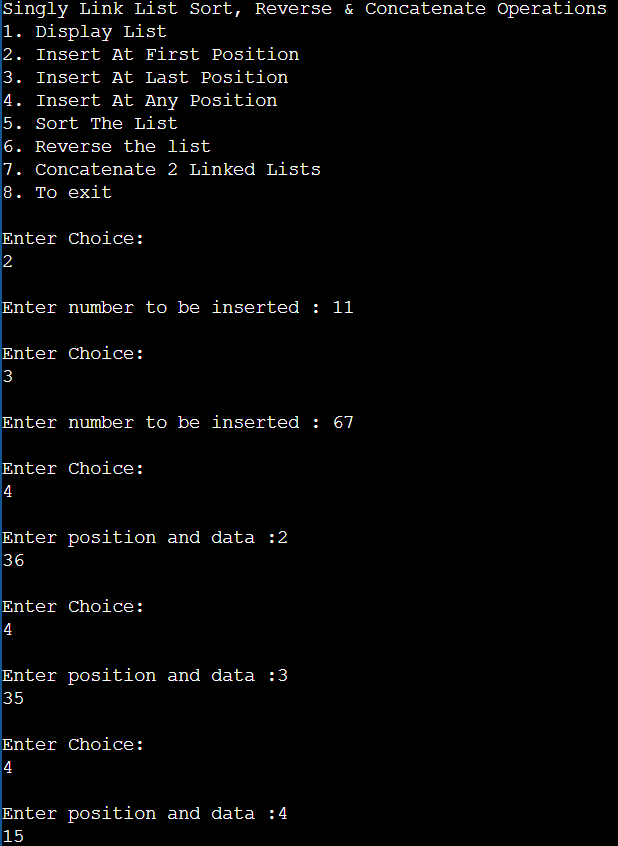
}

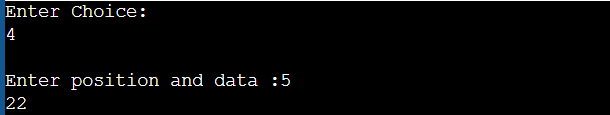
return 0;

}

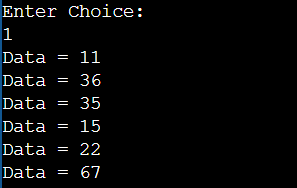
**2.7.3 Output:**

**//Insert Elements in Singly Linked List**

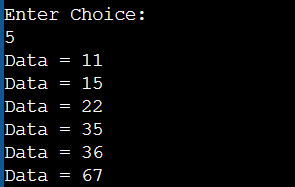




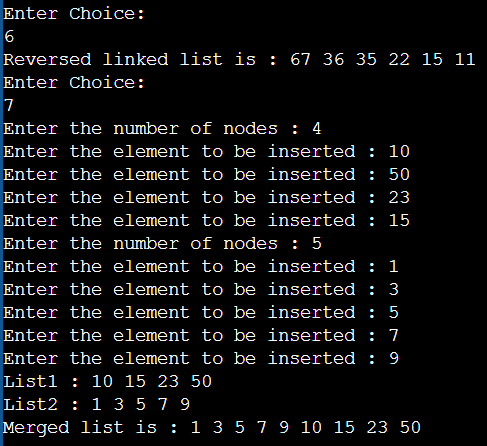
**//Display Elements**



**//Sort the List**



**//Concatenate 2 Linked List**



**2.8 Experiment: 8**

**2.8.1 Question:**

WAP to implement Stack & Queues using Linked Representation.

**2.8.2 Code:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int choice, value;

printf("\nImplementation of Stack and Queues using Linked List\n");

printf("\nStack Operations\n");

printf("1. Push\n2. Pop\n3. Display\n");

printf("\nQueue Operations\n");

printf("4.Enqueue\n5.Dequeue\n6.Display\n7.Exit\n");

while (1)

{

printf("\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1: printf("\nEnter the value to insert: ");

scanf("%d", &value);

push(value);

break;

case 2: printf("Popped element is :%d\n", pop());

break;

case 3: displayStack();

break;

case 4: printf("\nEnter the value to insert: ");

scanf("%d", & value);

enqueue(value);

break;

case 5: printf("Popped element is :%d\n", dequeue());

break;

case 6: displayQueue();

break;

case 7: exit(0);

break;

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

}

}

}

//Stack Implementation

struct Node

{

int data;

struct Node \*next;

};

struct Node\* top = NULL;

void push(int value)

{

struct Node \*newNode;

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

newNode->data = value;

if (top == NULL)

{

newNode->next = NULL;

}

else

{

newNode->next = top;

}

top = newNode;

printf("Node is Inserted\n\n");

}

int pop()

{

if (top == NULL)

{

printf("\nStack Underflow\n");

}

else

{

struct Node \*temp = top;

int temp\_data = top->data;

top = top->next;

free(temp);

return temp\_data;

}

}

void displayStack()

{

if (top == NULL)

{

printf("\nStack Underflow\n");

}

else

{

printf("The stack is \n");

struct Node \*temp = top;

while (temp->next != NULL)

{

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

temp = temp->next;

}

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

}

}

//Queue Implementation

struct node

{

int data;

struct node \* next;

};

struct node \* front = NULL;

struct node \* rear = NULL;

void enqueue(int value)

{

struct node \* ptr;

ptr = (struct node \*) malloc(sizeof(struct node));

ptr -> data = value;

ptr -> next = NULL;

if ((front == NULL) && (rear == NULL))

{

front = rear = ptr;

}

else

{

rear -> next = ptr;

rear = ptr;

}

printf("Node is Inserted\n\n");

}

int dequeue()

{

if (front == NULL)

{

printf("\nUnderflow\n");

}

else

{

struct node \* temp = front;

int temp\_data = front -> data;

front = front -> next;

free(temp);

return temp\_data;

}

}

void displayQueue()

{

struct node \* temp;

if ((front == NULL) && (rear == NULL))

{

printf("\nQueue is Empty\n");

}

else

{

printf("The queue is \n");

temp = front;

while (temp)

{

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

temp = temp -> next;

}

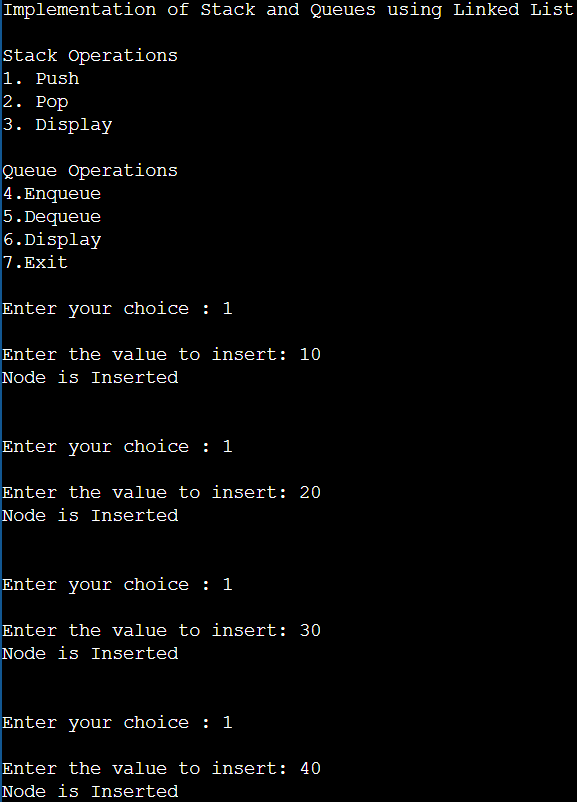
printf("NULL\n\n");

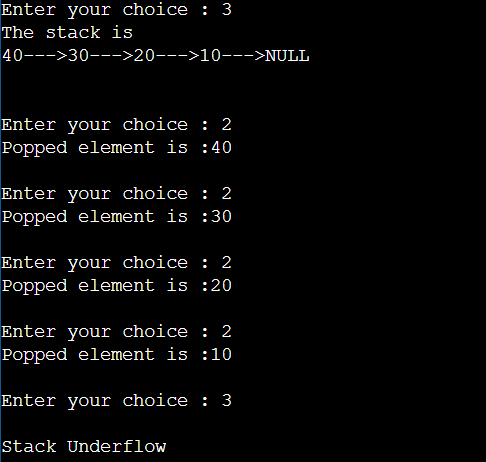
}

}

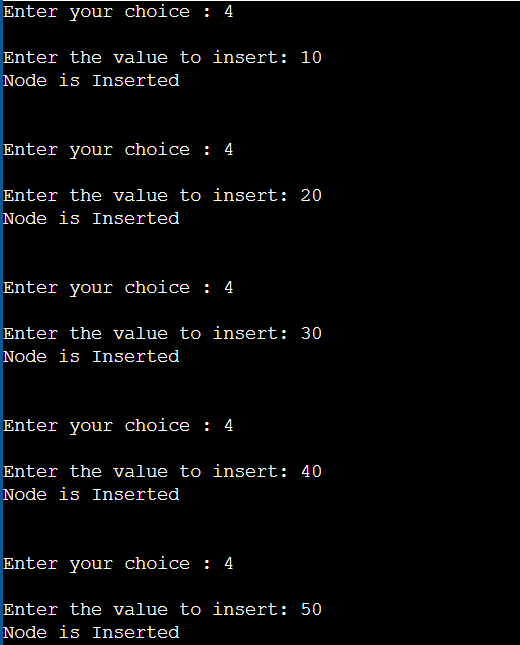
**Output:**

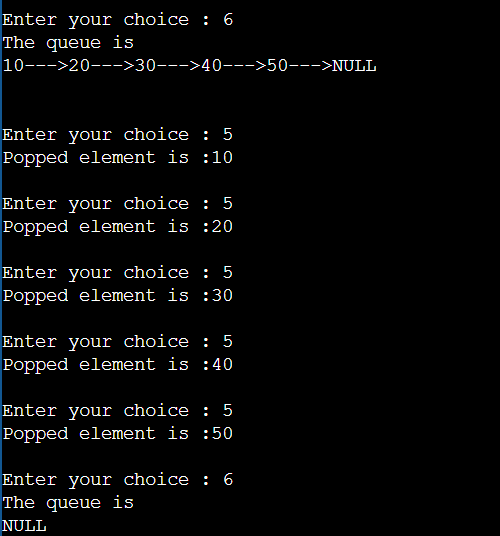
**//Stack Operations**





**//Queue Operations**





**2.9 Experiment: 9**

**2.9.1 Question:**

WAP to Implement doubly link list with primitive operations:

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

**2.9.2 Code:**

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*prev, \*next;

};

struct node\* start = NULL;

void insertFront()

{

int data;

struct node\* temp;

temp=(struct node\*)malloc(sizeof(struct node));

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

scanf("%d",&data);

temp->info = data;

temp->prev = NULL;

temp->next = start;

start = temp;

}

void insertEnd()

{

int data;

struct node \*temp, \*trav;

temp=(struct node\*)malloc(sizeof(struct node));

temp->prev = NULL;

temp->next = NULL;

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

scanf("%d", &data);

temp->info=data;

temp->next=NULL;

trav=start;

if (start == NULL)

{

start = temp;

}

else

{

while (trav->next != NULL)

trav = trav->next;

temp->prev = trav;

trav->next = temp;

}

}

void insertPosition()

{

int data, pos, i=1;

struct node \*temp, \*newnode;

newnode = malloc(sizeof(struct node));

newnode->prev = NULL;

newnode->next = NULL;

printf("Enter position: ");

scanf("%d", &pos);

if (start == NULL) {

start = newnode;

newnode->prev = NULL;

newnode->next = NULL;

}

else if (pos == 1)

{

insertFront();

}

else

{

printf("\nEnter number to be inserted: ");

scanf("%d", &data);

newnode->info = data;

temp = start;

while (i < pos - 1)

{

temp = temp->next;

i++;

}

newnode->next = temp->next;

newnode->prev = temp;

temp->next = newnode;

temp->next->prev = newnode;

}

}

void deleteFirst()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else

{

temp = start;

start = start->next;

if (start != NULL)

start->prev = NULL;

free(temp);

}

}

void deleteEnd()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

temp = start;

while (temp->next != NULL)

temp = temp->next;

if (start->next == NULL)

start = NULL;

else {

temp->prev->next = NULL;

free(temp);

}

}

void deletePosition()

{

int pos, i = 1;

struct node \*temp, \*position;

temp = start;

if (start == NULL)

printf("\nList is empty\n");

else

{

printf("\nEnter position : ");

scanf("%d", &pos);

if (pos == 1)

{

deleteFirst();

if (start != NULL)

{

start->prev = NULL;

}

free(position);

return;

}

while (i < pos - 1)

{

temp = temp->next;

i++;

}

position = temp->next;

if (position->next != NULL)

position->next->prev = temp;

temp->next = position->next;

free(position);

}

}

void traverse()

{

if (start == NULL)

{

printf("\nList is empty\n");

return;

}

struct node\* temp;

temp = start;

while (temp != NULL)

{

printf("Data = %d\n", temp->info);

temp = temp->next;

}

}

int main()

{ printf("Doubly Linked List Insert & Delete Operations");

printf("\n1. Display List\n");

printf("2. Insert At Front\n");

printf("3. Insert At End\n");

printf("4. Insert At Position\n");

printf("5. Delete First Element\n");

printf("6. Delete Last Element\n");

printf("7. Delete From Any Position\n");

printf("8. Exit\n");

while (1)

{

int choice;

printf("\nEnter Choice: \n");

scanf("%d", &choice);

switch (choice)

{

case 1: traverse();

break;

case 2: insertFront();

break;

case 3: insertEnd();

break;

case 4: insertPosition();

break;

case 5: deleteFirst();

traverse();

break;

case 6: deleteEnd();

traverse();

break;

case 7: deletePosition();

traverse();

break;

case 8: exit(1);

break;

default: printf("Incorrect Choice. Try Again \n");

}

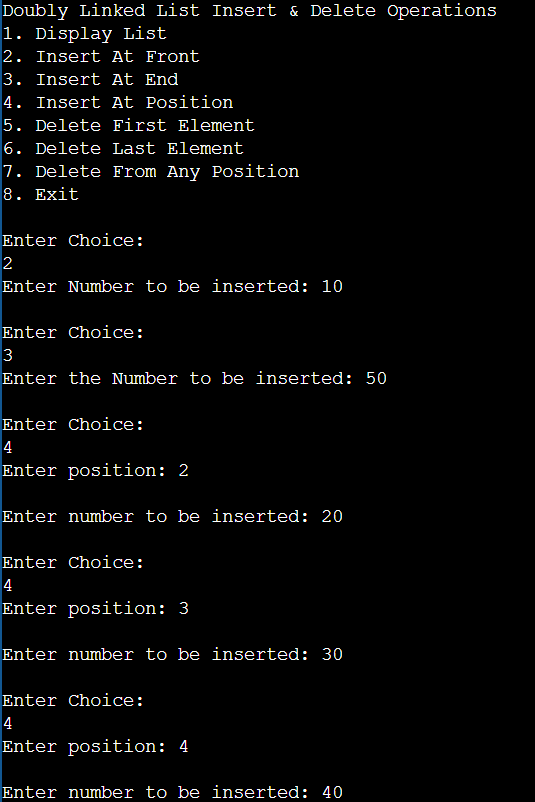
}

return 0;

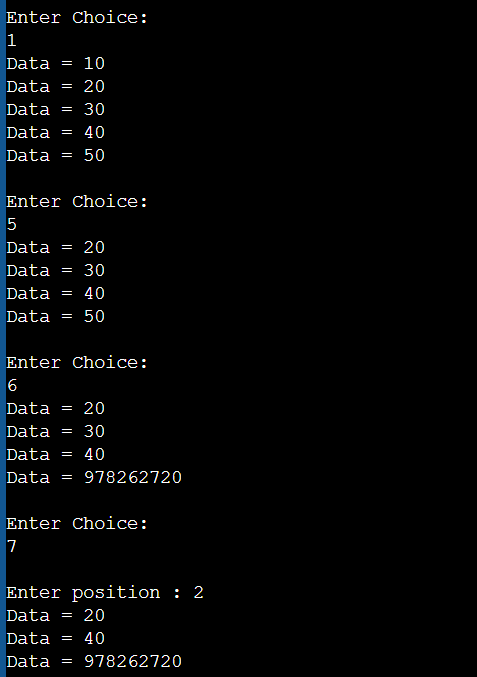
}

**2.9.3 Output:**

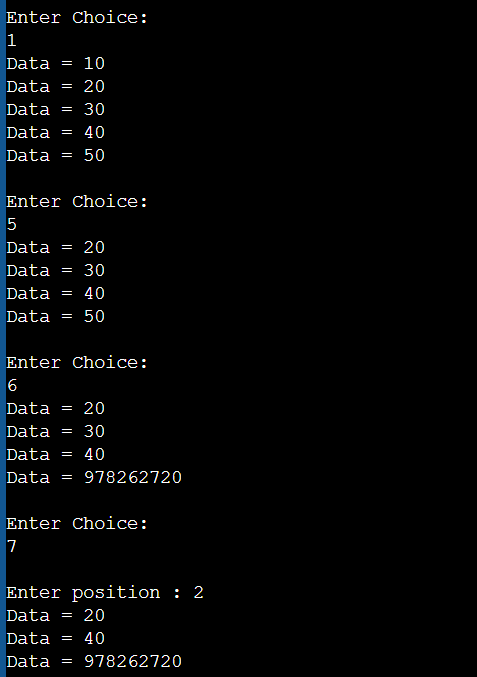
**//Inserting Elements in Doubly Linked List**



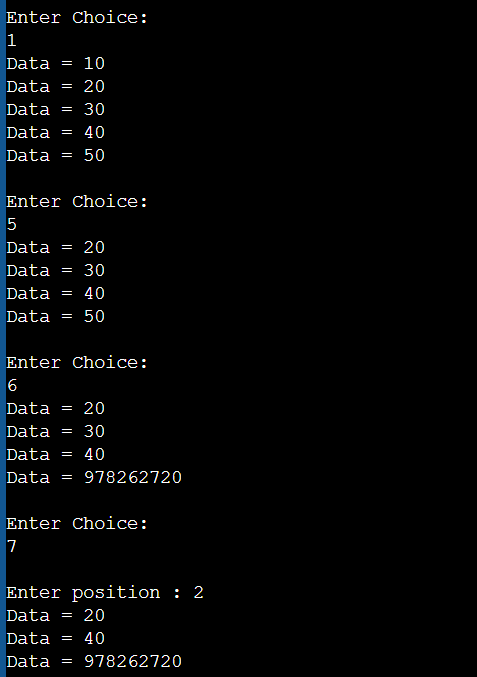
**//Display Elements in Doubly Linked List**



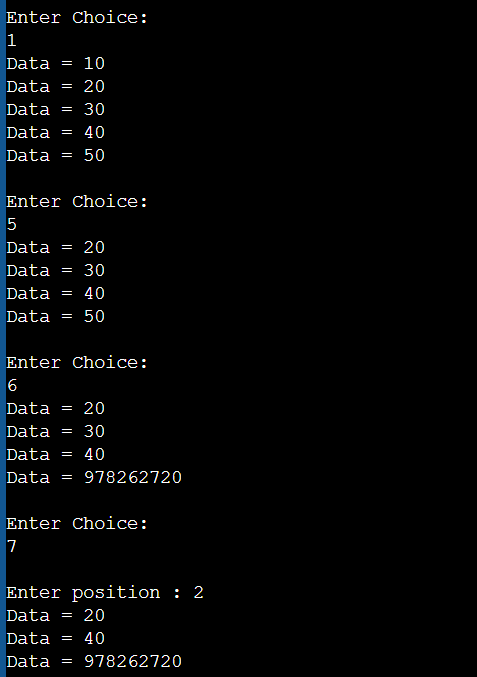
**//Delete First Element**



**//Delete Last Element**



**//Delete Element from Any Position**



**2.10 Experiment: 10**

**2.10.1 Question:**

Write a program:

1. To construct a binary Search tree.
2. To traverse the tree using all the methods i.e., in-order, preorder and post order.
3. To display the elements in the tree.

**2.10.2 Code:**

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*tree;

void create\_tree(struct node \*);

struct node \*insertElement(struct node \*, int);

void preorderTraversal(struct node \*);

void inorderTraversal(struct node \*);

void postorderTraversal(struct node \*);

int main()

{

int option, val;

struct node \*ptr;

create\_tree(tree);

printf("Binary Search Tree Operations:\n");

printf("1. Insert Element");

printf("\n2. Preorder Traversal");

printf("\n3. Inorder Traversal");

printf("\n4. Postorder Traversal");

printf("\n5. Exit");

do

{

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

scanf("%d", &option);

switch(option)

{

case 1:

printf("\n Enter the value of the new node : ");

scanf("%d", &val);

tree = insertElement(tree, val);

break;

case 2:

printf("\n The elements of the tree are : \n");

preorderTraversal(tree);

break;

case 3:

printf("\n The elements of the tree are : \n");

inorderTraversal(tree);

break;

case 4:

printf("\n The elements of the tree are : \n");

postorderTraversal(tree);

break;

}

}

while(option!=14);

getch();

return 0;

}

void create\_tree(struct node \*tree)

{

tree = NULL;

}

struct node \*insertElement(struct node \*tree, int val)

{

struct node \*ptr, \*nodeptr, \*parentptr;

ptr = (struct node\*)malloc(sizeof(struct node));

ptr->data = val;

ptr->left = NULL;

ptr->right = NULL;

if(tree==NULL)

{

tree=ptr;

tree->left=NULL;

tree->right=NULL;

}

else

{

parentptr=NULL;

nodeptr=tree;

while(nodeptr!=NULL)

{

parentptr=nodeptr;

if(val<nodeptr->data)

nodeptr=nodeptr->left;

else

nodeptr = nodeptr->right;

}

if(val<parentptr->data)

parentptr->left = ptr;

else

parentptr->right = ptr;

}

return tree;

}

void preorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

printf("%d\t", tree->data);

preorderTraversal(tree->left);

preorderTraversal(tree->right);

}

}

void inorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

inorderTraversal(tree->left);

printf("%d\t", tree->data);

inorderTraversal(tree->right);

}

}

void postorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

postorderTraversal(tree->left);

postorderTraversal(tree->right);

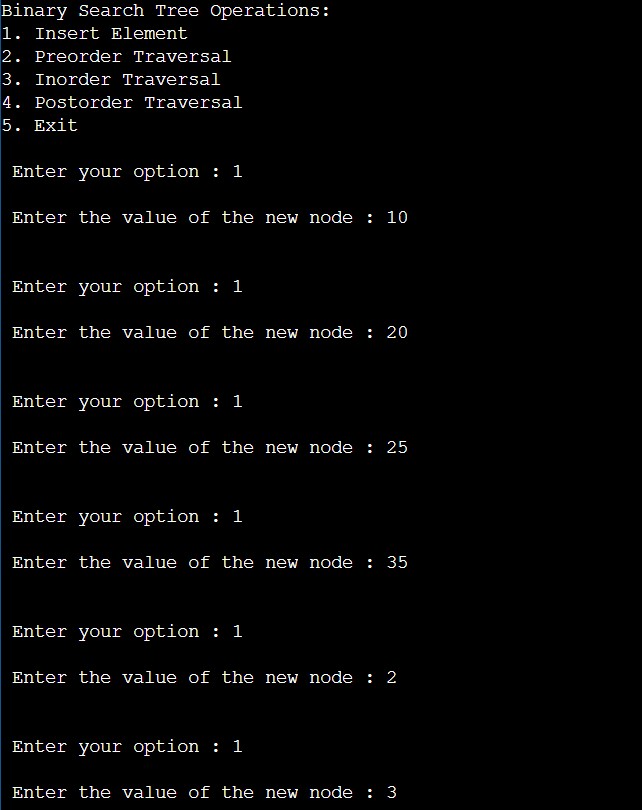
printf("%d\t", tree->data);

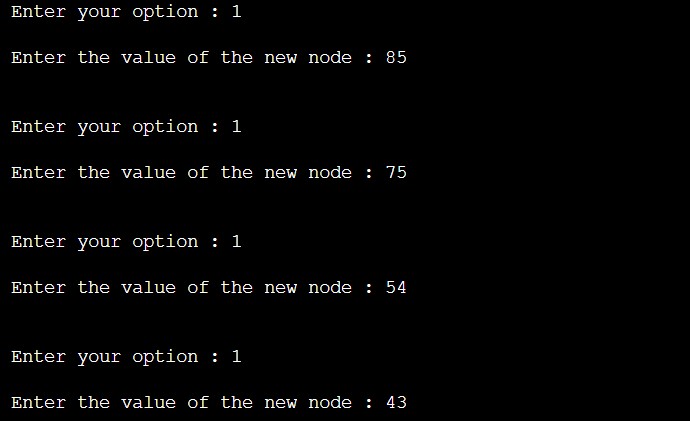
}

}

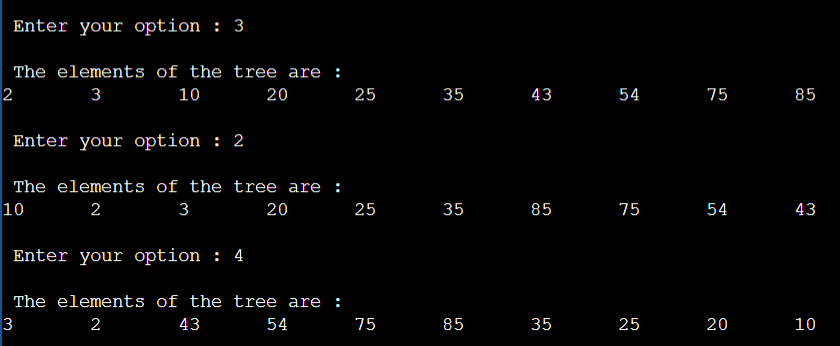
**2.10.3 Output:**

**//Insertion of Number in tree**

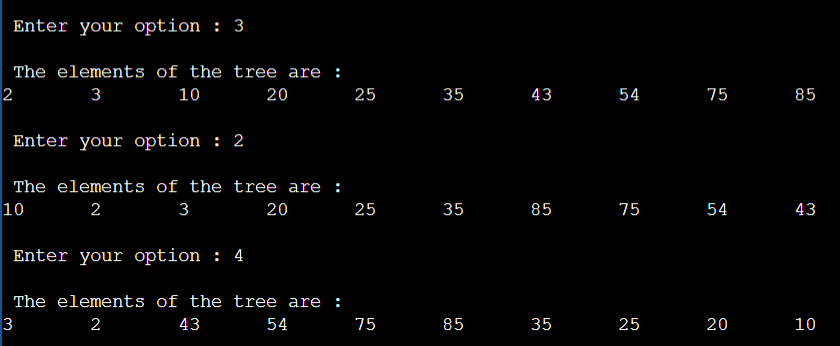




**//In - Order traversal**



**//Pre - Order traversal**



**//Post - Order traversal**

