Experiment 1:

**Stack implementation using array:**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 4

int top = -1, inp\_array[SIZE];

void push();

void pop();

void show();

int main()

{

int choice;

while (1)

{

printf("\nPerform operations on the stack:");

printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");

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

scanf("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

show();

break;

case 4:

exit(0);

default:

printf("\nInvalid choice!!");

}

}

}

void push()

{

int x;

if (top == SIZE - 1)

{

printf("\nOverflow!!");

}

else

{

printf("\nEnter the element to be added onto the stack: ");

scanf("%d", &x);

top = top + 1;

inp\_array[top] = x;

}

}

void pop()

{

if (top == -1)

{

printf("\nUnderflow!!");

}

else

{

printf("\nPopped element: %d", inp\_array[top]);

top = top - 1;

}

}

void show()

{

if (top == -1)

{

printf("\nUnderflow!!");

}

else

{

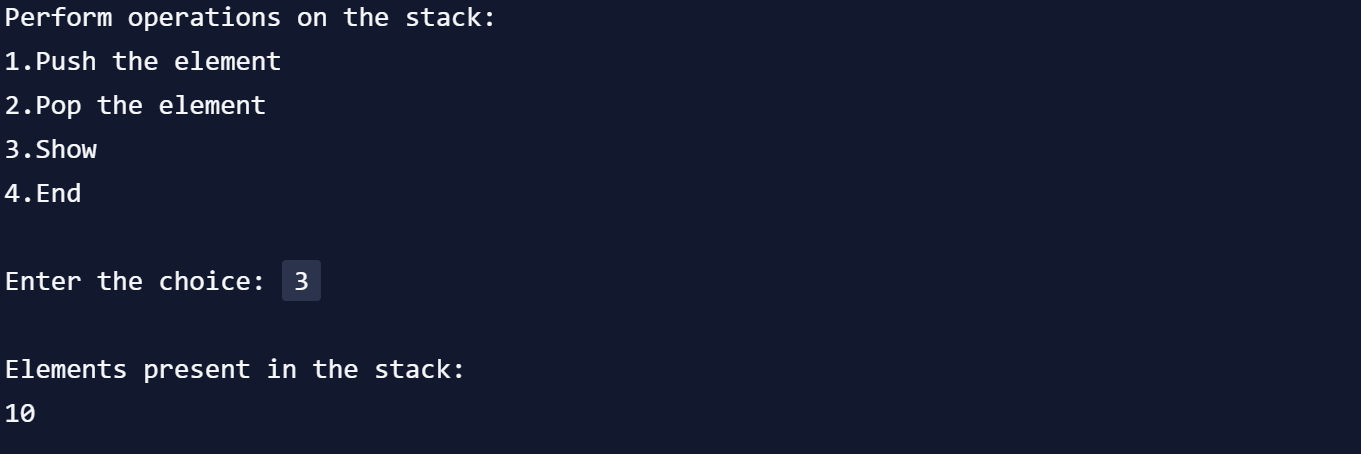
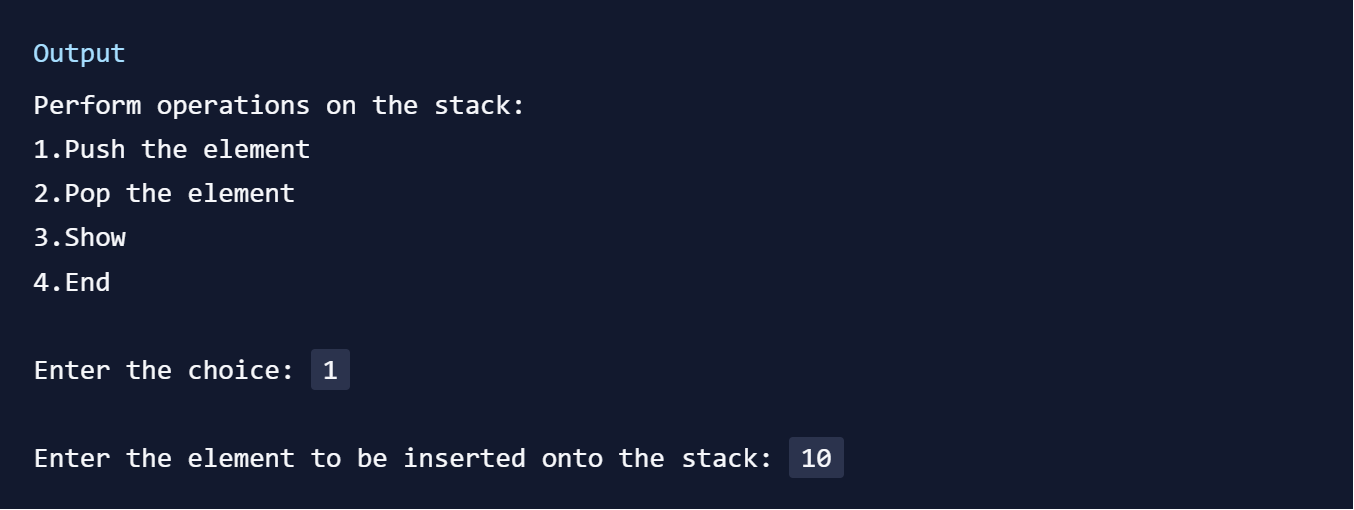
printf("\nElements present in the stack: \n");

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

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

}

}



**Stack implantation using linked list :**

#include <stdio.h>

#include <stdlib.h>

// Structure to create a node with data and the next pointer

struct node {

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

int count = 0;

// Push() operation on a stack

void push(int data) {

if (top == NULL)

{

top =(struct node \*)malloc(1\*sizeof(struct node));

top->ptr = NULL;

top->info = data;

}

else

{

temp =(struct node \*)malloc(1\*sizeof(struct node));

temp->ptr = top;

temp->info = data;

top = temp;

}

count++;

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

}

int pop() {

top1 = top;

if (top1 == NULL)

{

printf("\nStack Underflow\n");

return -1;

}

else

top1 = top1->ptr;

int popped = top->info;

free(top);

top = top1;

count--;

return popped;

}

void display() {

// Display the elements of the stack

top1 = top;

if (top1 == NULL)

{

printf("\nStack Underflow\n");

return;

}

printf("The stack is \n");

while (top1 != NULL)

{

printf("%d--->", top1->info);

top1 = top1->ptr;

}

printf("NULL\n\n");

}

int main() {

int choice, value;

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

while (1) {

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

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:

display();

break;

case 4:

exit(0);

break;

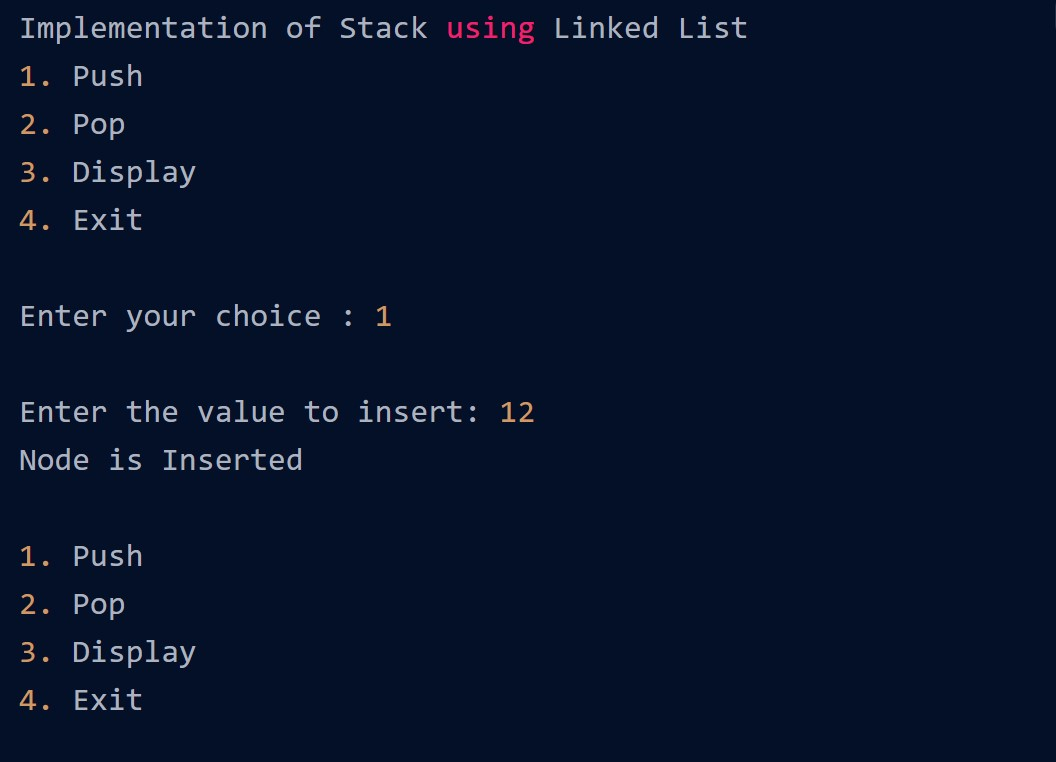
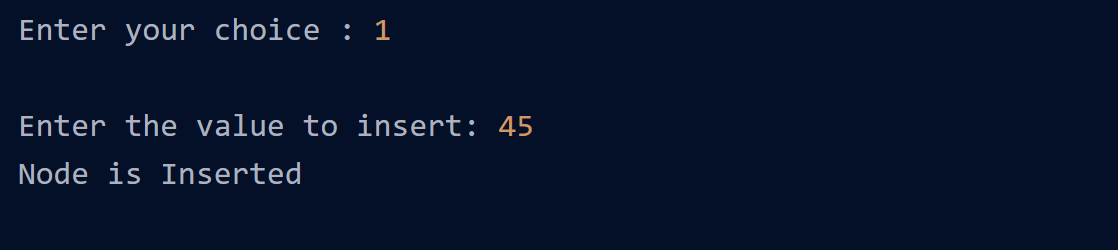
default:

printf("\nWrong Choice\n");

}

}

}

**Queue Implementation using array:**

#include <stdio.h>

#define MAX 50

void insert();

void delete();

void display();

int queue\_array[MAX];

int rear = - 1;

int front = - 1;

main()

{

int choice;

while (1)

{

printf("1.Insert element to queue \n");

printf("2.Delete element from queue \n");

printf("3.Display all elements of queue \n");

printf("4.Quit \n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice \n");

} /\* End of switch \*/

} /\* End of while \*/

} /\* End of main() \*/

void insert()

{

int add\_item;

if (rear == MAX - 1)

printf("Queue Overflow \n");

else

{

if (front == - 1)

/\*If queue is initially empty \*/

front = 0;

printf("Inset the element in queue : ");

scanf("%d", &add\_item);

rear = rear + 1;

queue\_array[rear] = add\_item;

}

} /\* End of insert() \*/

void delete()

{

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

{

printf("Queue Underflow \n");

return ;

}

else

{

printf("Element deleted from queue is : %d\n", queue\_array[front]);

front = front + 1;

}

} /\* End of delete() \*/

void display()

{

int i;

if (front == - 1)

printf("Queue is empty \n");

else

{

printf("Queue is : \n");

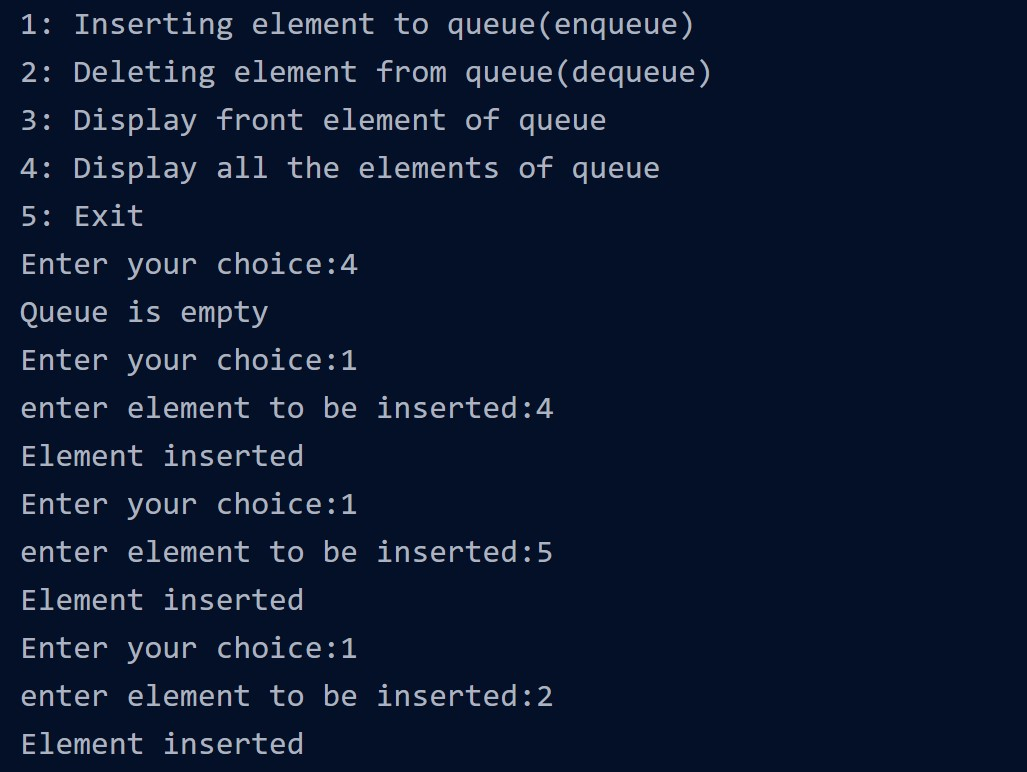
for (i = front; i <= rear; i++)

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

printf("\n");

}

}



#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

/\* Create an empty queue \*/

void create()

{

front = rear = NULL;

}

/\* Returns queue size \*/

void queuesize()

{

printf("\n Queue size : %d", count);

}

/\* Enqueing the queue \*/

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

/\* Displaying the queue elements \*/

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

/\* Dequeing the queue \*/

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

/\* Returns the front element of queue \*/

int frontelement()

{

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

return(front->info);

else

return 0;

}

void empty()

{

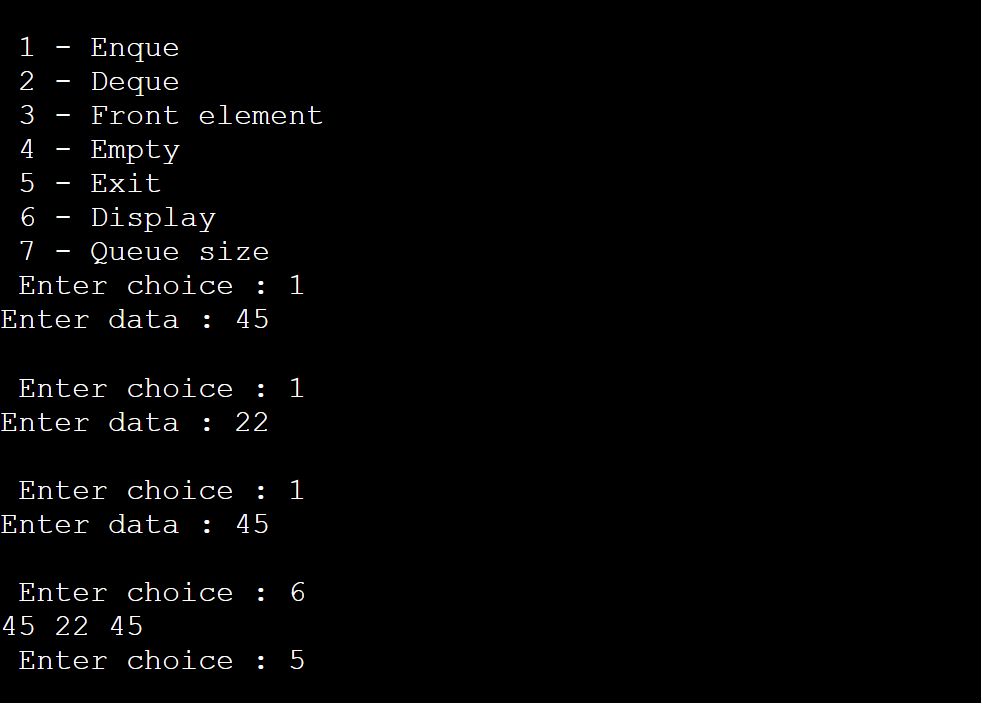
if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}



***Experiment 02***

**Implementation of circular queue using array:**

#include<stdio.h>

using namespace std;

class Queue

{

int rear, front;

int size;

int \*arr;

public:

Queue(int s)

{

front = rear = -1;

size = s;

arr = new int[s];

}

void enQueue(int value);

int deQueue();

void displayQueue();

};

void Queue::enQueue(int value)

{

if ((front == 0 && rear == size-1) ||

((rear+1) % size == front))

{

printf("\nQueue is Full");

return;

}

else if (front == -1)

{

front = rear = 0;

arr[rear] = value;

}

else if (rear == size-1 && front != 0)

{

rear = 0;

arr[rear] = value;

}

else

{

rear++;

arr[rear] = value;

}

}

int Queue::deQueue()

{

if (front == -1)

{

printf("\nQueue is Empty");

return INT\_MIN;

}

int data = arr[front];

arr[front] = -1;

if (front == rear)

{

front = -1;

rear = -1;

}

else if (front == size-1)

front = 0;

else

front++;

return data;

}

void Queue::displayQueue()

{

if (front == -1)

{

printf("\nQueue is Empty");

return;

}

printf("\nElements in Circular Queue are: ");

if (rear >= front)

{

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

printf("%d ",arr[i]);

}

else

{

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

printf("%d ", arr[i]);

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

printf("%d ", arr[i]);

}

}

int main()

{

Queue q(5);

q.enQueue(14);

q.enQueue(22);

q.enQueue(13);

q.enQueue(-6);

q.displayQueue();

printf("\nDeleted value = %d", q.deQueue());

printf("\nDeleted value = %d", q.deQueue());

q.displayQueue();

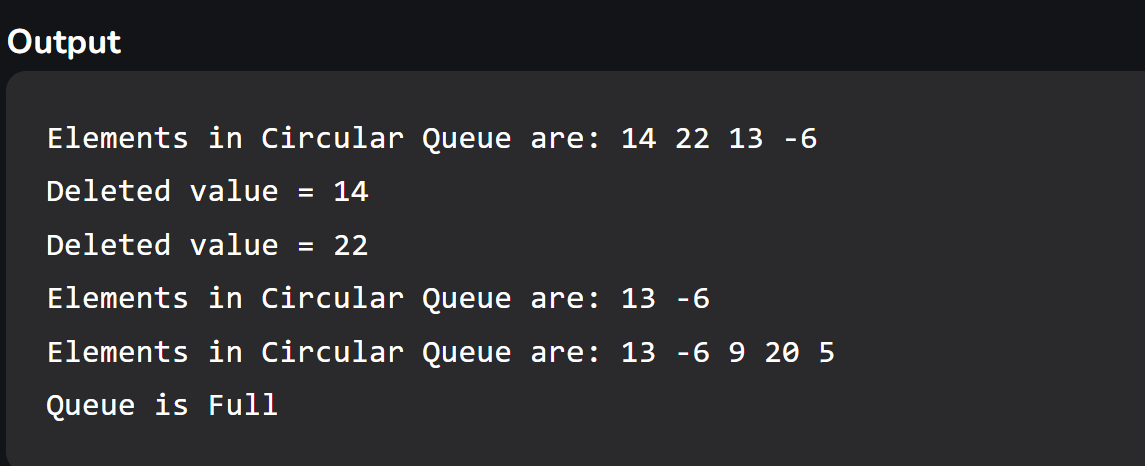
q.enQueue(9);

q.enQueue(20);

q.enQueue(5);

q.displayQueue();

q.enQueue(20);

 return 0;

}

***Experiment 3***

//Write a program to implement **double-ended queue** (dequeue) using array:

# include<stdio.h>

# define MAX 5

int deque\_arr[MAX];

int left = -1;

int right = -1;

void insert\_right()

{

int added\_item;

if((left == 0 && right == MAX-1) || (left == right+1))

{ printf("Queue Overflow\n");

return;}

if (left == -1)

{ left = 0;

right = 0;}

else

if(right == MAX-1)

right = 0;

else

right = right+1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[right] = added\_item ;

}

void insert\_left()

{ int added\_item;

if((left == 0 && right == MAX-1) || (left == right+1))

{ printf("Queue Overflow \n");

return; }

if (left == -1)

{ left = 0;

right = 0; }

else

if(left== 0)

left=MAX-1;

else

left=left-1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[left] = added\_item ; }

void delete\_left()

{ if (left == -1)

{ printf("Queue Underflow\n");

return ; }

printf("Element deleted from queue is : %d\n",deque\_arr[left]);

if(left == right)

{ left = -1;

right=-1; }

else

if(left == MAX-1)

left = 0;

else

left = left+1;

}

void delete\_right()

{if (left == -1)

{printf("Queue Underflow\n");

return ; }

printf("Element deleted from queue is : %d\n",deque\_arr[right]);

if(left == right)

{ left = -1;

right=-1; }

else

if(right == 0)

right=MAX-1;

else

right=right-1; }

void display\_queue()

{ int front\_pos = left,rear\_pos = right;

if(left == -1)

{ printf("Queue is empty\n");

return; }

printf("Queue elements :\n");

if( front\_pos <= rear\_pos )

{ while(front\_pos <= rear\_pos)

{ printf("%d ",deque\_arr[front\_pos]);

front\_pos++; } }

else

{ while(front\_pos <= MAX-1)

{ printf("%d ",deque\_arr[front\_pos]);

front\_pos++; }

front\_pos = 0;

while(front\_pos <= rear\_pos)

{ printf("%d ",deque\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

void input\_que()

{ int choice;

do

{ printf("1.Insert at right\n");

printf("2.Delete from left\n");

printf("3.Delete from right\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{ case 1:

insert\_right();

break;

case 2:

delete\_left();

break;

case 3:

delete\_right();

break;

case 4:

display\_queue();

break;

case 5:

break;

default:

printf("Wrong choice\n");

}

}while(choice!=5);

}

void output\_que()

{ int choice;

do

{ printf("1.Insert at right\n");

printf("2.Insert at left\n");

printf("3.Delete from left\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert\_right();

break;

case 2:

insert\_left();

break;

case 3:

delete\_left();

break;

case 4:

display\_queue();

break;

case 5:

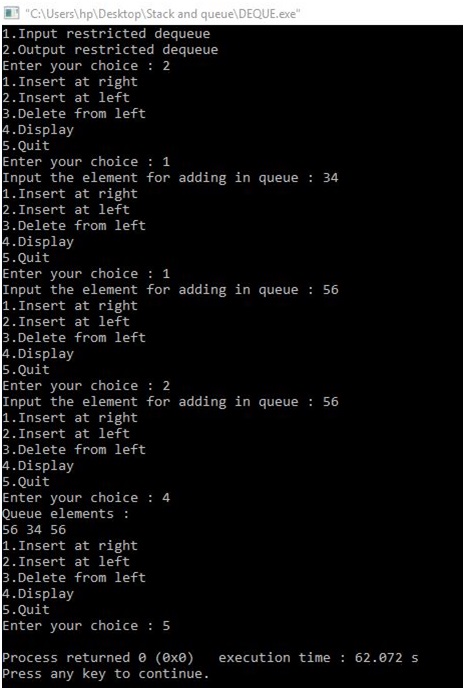
break;

default:

printf("Wrong choice\n");

}

}while(choice!=5);

}

main()

{ int choice;

printf("1.Input restricted dequeue\n");

printf("2.Output restricted dequeue\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

input\_que();

break;

case 2:

output\_que();

break;

default:

printf("Wrong choice\n");

}

}

**Implementation of Priority Queue using Array:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

void insert\_by\_priority(int);

void delete\_by\_priority(int);

void create();

void check(int);

void display\_pqueue();

int pri\_que[MAX];

int front, rear;

void main()

{

int n, ch;

printf("\n1 - Insert an element into queue");

printf("\n2 - Delete an element from queue");

printf("\n3 - Display queue elements");

printf("\n4 - Exit");

create();

while (1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

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

scanf("%d",&n);

insert\_by\_priority(n);

break;

case 2:

printf("\nEnter value to delete : ");

scanf("%d",&n);

delete\_by\_priority(n);

break;

case 3:

display\_pqueue();

break;

case 4:

exit(0);

break;

default:

printf("\nChoice is incorrect, Enter a correct choice");

}

}

}

/\* Function to create an empty priority queue \*/

void create()

{

front = rear = -1;

}

/\* Function to insert value into priority queue \*/

void insert\_by\_priority(int data)

{

if (rear >= MAX - 1)

{

printf("\nQueue overflow no more elements can be inserted");

return;

}

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

{

front++;

rear++;

pri\_que[rear] = data;

return;

}

else

check(data);

rear++;

}

/\* Function to check priority and place element \*/

void check(int data)

{

int i,j;

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

{

if (data >= pri\_que[i])

{

for (j = rear + 1; j > i; j--)

{

pri\_que[j] = pri\_que[j - 1];

}

pri\_que[i] = data;

return;

}

}

pri\_que[i] = data;

}

/\* Function to delete an element from queue \*/

void delete\_by\_priority(int data)

{

int i;

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

{

printf("\nQueue is empty no elements to delete");

return;

}

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

{

if (data == pri\_que[i])

{

for (; i < rear; i++)

{

pri\_que[i] = pri\_que[i + 1];

}

pri\_que[i] = -99;

rear--;

if (rear == -1)

front = -1;

return;

}

}

printf("\n%d not found in queue to delete", data);

}

/\* Function to display queue elements \*/

void display\_pqueue()

{

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

{

printf("\nQueue is empty");

return;

}

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

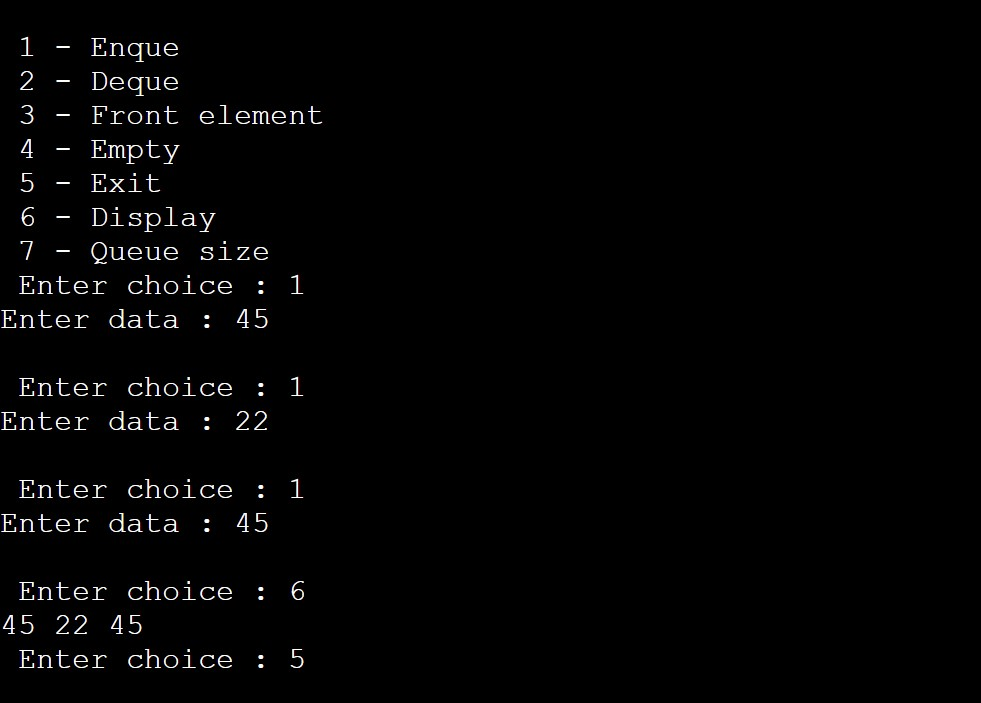
{

printf(" %d ", pri\_que[front]);

}

front = 0;

}



***Experiment 4***

**C program for Infix to Postfix Expression using stack :**

#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 under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

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

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')

{

return(3);

}

else 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);

}

if(top>0)

{

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

getchar();

exit(1);

}

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

}

int main()

{

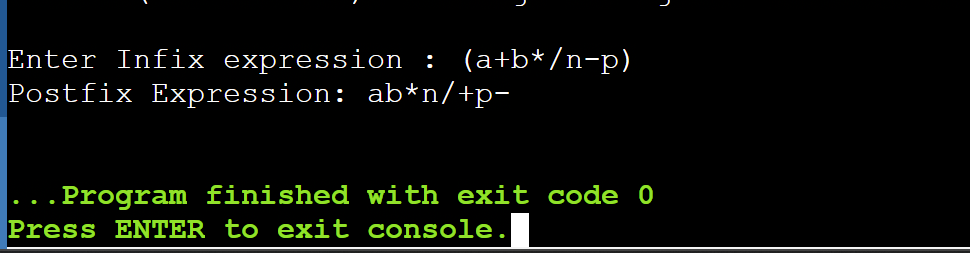
char infix[SIZE], postfix[SIZE];

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix);

printf("Postfix Expression: ");

puts(postfix);

return 0;

}

***Experiment 5***

**C program to evaluate a given Postfix Expression Using Stacks:**

#include<stdio.h>

#include<ctype.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

case '^':

{

n3 = n2 ^ n1;

break;

}

}

push(n3);

}

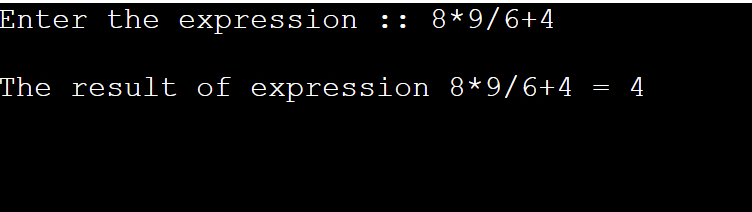
e++;

}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

return 0;

}



***Experiment 6***

**C program to implement a Stack using two Queue:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 20

int queue1[MAX\_SIZE], queue2[MAX\_SIZE];

int front1 = -1, rear1 = -1;

int front2 = -1, rear2 = -1;

int count = 0;

void enqueue1(int x);

int dequeue1();

void enqueue2(int x);

int dequeue2();

void push(int x);

int pop();

void display();

int main() {

int choice, num;

while (choice != 4) {

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

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &num);

push(num);

break;

case 2:

printf("Item deleted: %d\n", pop());

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nInvalid Choice!\n");

}

}

return 0;

}

void enqueue1(int x) {

if (rear1 == MAX\_SIZE - 1) {

printf("Overflow\n");

} else {

if (front1 == -1) {

front1 = 0;

}

rear1++;

queue1[rear1] = x;

}

}

int dequeue1() {

int temp;

if (front1 == -1 || front1 > rear1) {

printf("Underflow\n");

} else {

temp = queue1[front1];

front1++;

}

return temp;

}

void enqueue2(int x) {

if (rear2 == MAX\_SIZE - 1) {

printf("Overflow\n");

} else {

if (front2 == -1) {

front2 = 0;

}

rear2++;

queue2[rear2] = x;

}

}

int dequeue2() {

int temp;

if (front2 == -1 || front2 > rear2) {

printf("Underflow\n");

} else {

temp = queue2[front2];

front2++;

}

return temp;

}

void push(int x) {

int i;

enqueue1(x);

for (i = 0; i < count; i++) {

enqueue1(dequeue2());

}

count++;

for (i = 0; i < count; i++) {

enqueue2(dequeue1());

}

}

int pop() {

count--;

return dequeue2();

}

void display() {

int i;

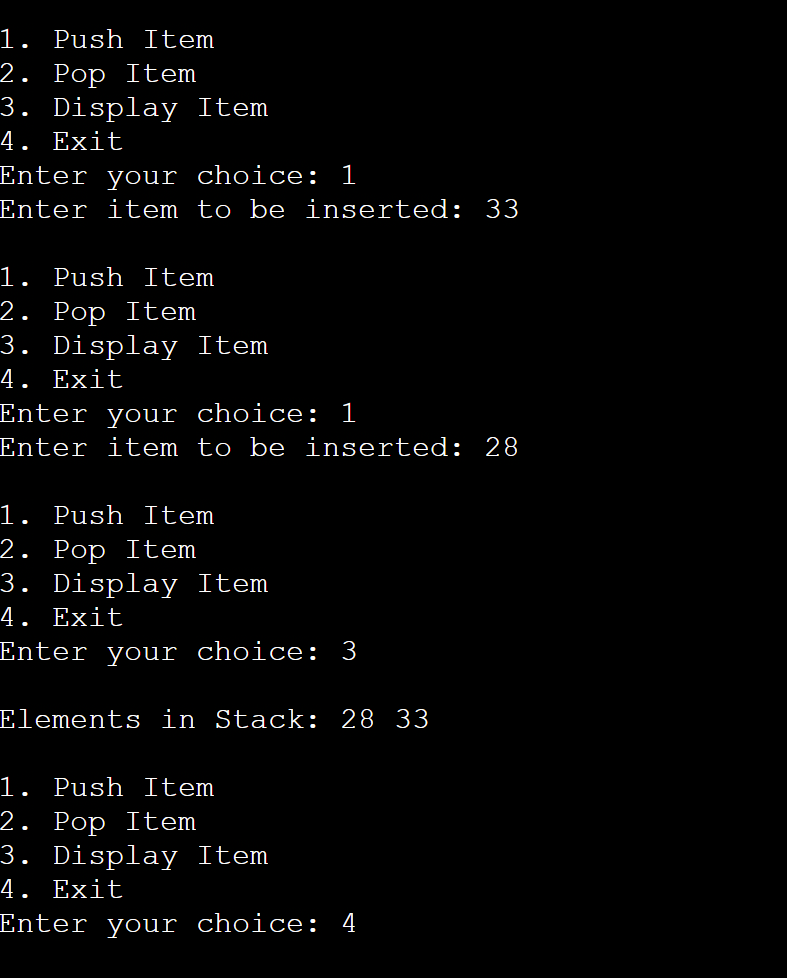
printf("\nElements in Stack: ");

for (i = front2; i <= rear2; i++) {

printf("%d ", queue2[i]);

}

printf("\n");

}

}

***Experiment 7***

**C program to implement a Queue using two stacks:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 20

int stack1[MAX\_SIZE], stack2[MAX\_SIZE];

int top1 = -1, top2 = -1;

void push(int stack[], int value, int \*top) {

if (\*top == MAX\_SIZE - 1) {

printf("Overflow\n");

return;

}

(\*top)++;

stack[\*top] = value;

}

int pop(int stack[], int \*top) {

if (\*top == -1) {

printf("Underflow\n");

return -1;

}

int value = stack[\*top];

(\*top)--;

return value;

}

void enqueue(int value) {

push(stack1, value, &top1);

}

int dequeue() {

if (top1 == -1 && top2 == -1) {

printf("Queue is empty\n");

return -1;

}

if (top2 == -1) {

while (top1 != -1) {

int value = pop(stack1, &top1);

push(stack2, value, &top2);

}

}

int dequeuedValue = pop(stack2, &top2);

return dequeuedValue;

}

void displayQueue() {

if (top1 == -1 && top2 == -1) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

for (int i = top2; i >= 0; i--) {

printf("%d ", stack2[i]);

}

for (int i = 0; i <= top1; i++) {

printf("%d ", stack1[i]);

}

printf("\n");

}

int main() {

int choice, num;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Display Queue\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &num);

enqueue(num);

break;

case 2:

printf("Item deleted: %d\n", dequeue());

break;

case 3:

displayQueue();

break;

case 4:

exit(0);

default:

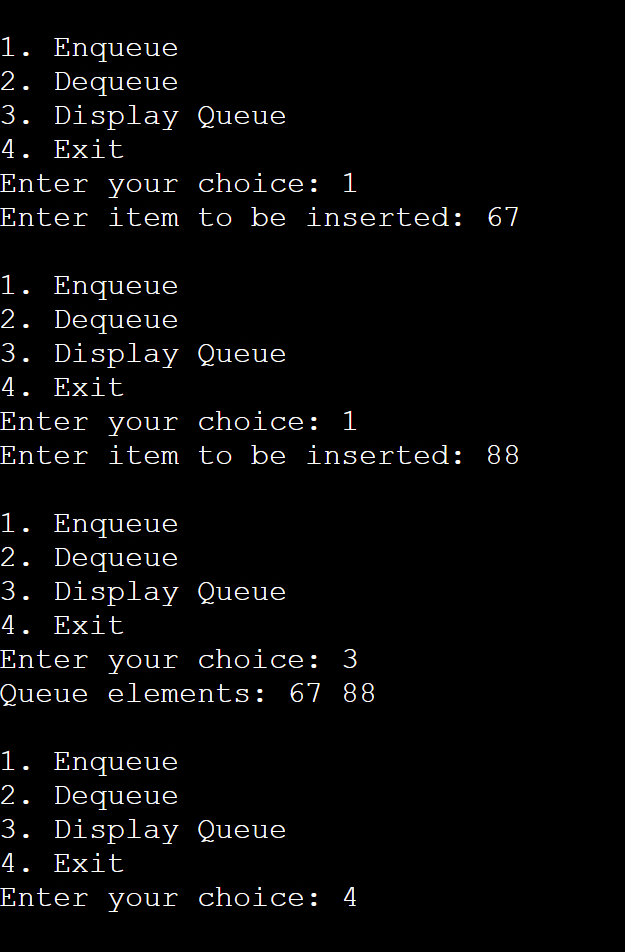
printf("\nInvalid Choice!\n");

}

}

return 0;

}



***Experiment 8***

**C programs to implement the following data structures:**

1. **Single Linked list (b) Double Linked list**

#include<stdio.h>

#include<stdlib.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

struct node\* head = NULL;

struct node

{

int data;

struct node\* next;

};

int main()

{

int choice;

while(1)

{

printf("\n\*\*\*\*\*\n");

printf("0. Create\n");

printf("1. display\n");

printf("2. Insert Node at beginning\n");

printf("3. Insert Node in specific position\n");

printf("4. Insert Node at end of LinkedList\n");

printf("5. Delete Node at beginning\n");

printf("6. Delete Node at end\n");

printf("7. Delete Node at position\n");

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

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

scanf("%d",&choice);

switch(choice)

{

case 0: create();

break;

case 1: display();

break;

case 2: insert\_begin();

break;

case 3: insert\_pos();

break;

case 4: insert\_end();

break;

case 5: delete\_begin();

break;

case 6: delete\_end();

break;

case 7: delete\_pos();

break;

case 8: exit(0);

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

break;

}

}

}

void create()

{

struct node\* temp;

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

printf("Enter node data: ");

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

temp->next = NULL;

if(head==NULL)

{

head = temp;

}

else

{

struct node\* ptr = head;

while(ptr->next!=NULL)

{

ptr = ptr->next;

}

ptr->next = temp;

}

}

void display()

{

if(head==NULL)

{

printf("Linked List is Empty\n");

return;

}

printf("LinkedList: ");

struct node\* ptr = head;

while(ptr!=NULL)

{

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

ptr = ptr->next;

}

printf("\n");

}

void insert\_begin()

{

struct node\* temp;

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

printf("Enter node data: ");

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

temp->next = NULL;

if(head==NULL)

{

head = temp;

return;

}

else

{

temp->next = head;

head = temp;

}

}

void insert\_pos()

{

struct node\* temp;

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

printf("Enter node data: ");

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

temp->next = NULL;

if(head==NULL)

{

head = temp;

return;

}

else

{

struct node\* prev\_ptr;

struct node\* ptr = head;

int pos;

printf("Enter position: ");

scanf("%d",&pos);

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

{

prev\_ptr = ptr;

ptr = ptr->next;

}

temp->next = ptr;

prev\_ptr->next = temp;

}

}

void insert\_end()

{

struct node\* temp;

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

printf("Enter node data: ");

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

temp->next = NULL;

if(head==NULL)

{

head = temp;

return;

}

else{

struct node\* ptr = head;

while(ptr->next!=NULL)

{

ptr = ptr->next;

}

ptr->next = temp;

}

}

void delete\_begin()

{

if(head==NULL)

{

printf("Linked List is empty | Nothing to delete \n");

return;

}

else

{

struct node\* ptr = head;

head = head->next;

free(ptr);

printf("Node Deleted \n");

}

}

void delete\_end() {

if (head == NULL) {

printf("Linked List is empty | Nothing to delete \n");

return;

} else if (head->next == NULL) {

free(head);

head = NULL;

} else {

struct node\* ptr = head;

while (ptr->next != NULL) {

ptr = ptr->next;

}

struct node\* prev\_ptr = head;

while (prev\_ptr->next != ptr) {

prev\_ptr = prev\_ptr->next;

}

prev\_ptr->next = NULL;

free(ptr);

}

}

void delete\_pos()

{

int pos;

printf("Enter node position to delete: ");

scanf("%d",&pos);

struct node\* ptr=head;

if(head==NULL)

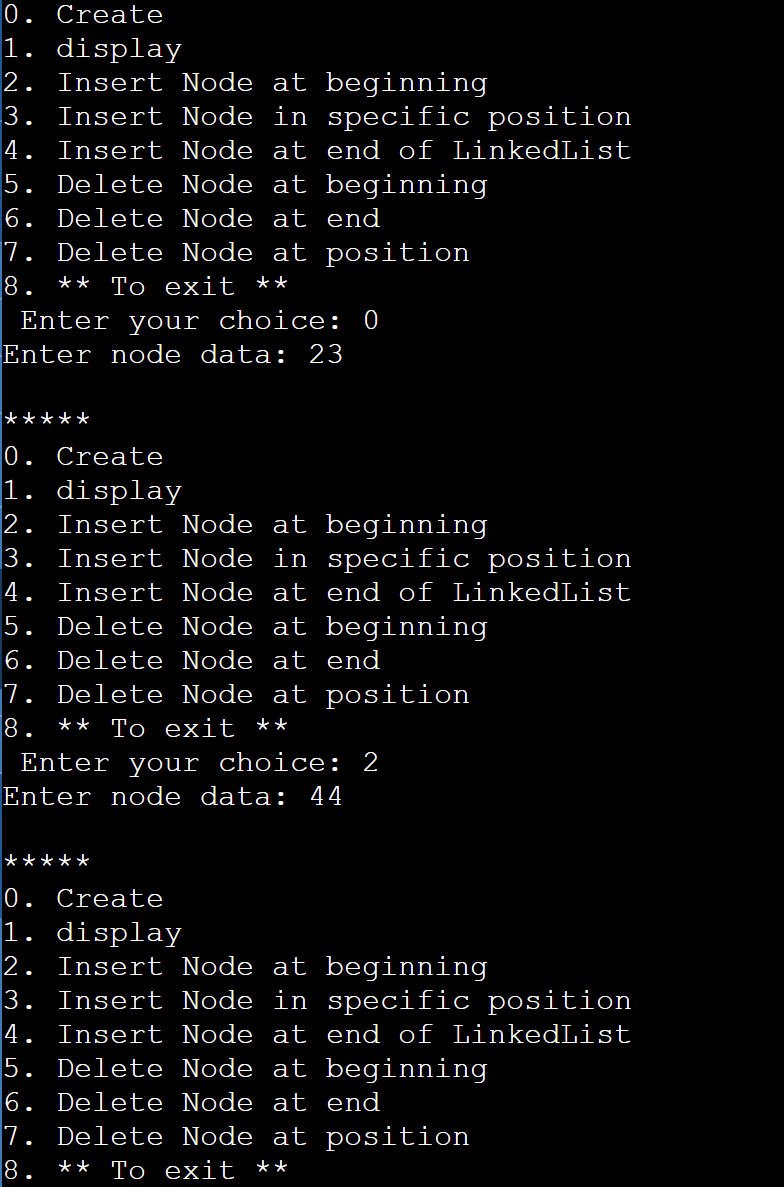
{

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

return;

}

else if(pos == 0)

 {

ptr = head;

head=ptr->next;

free(ptr);

}

else

{

struct node\* prev\_ptr;

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

{

prev\_ptr = ptr;

ptr = ptr->next;

}

prev\_ptr->next = ptr->next;

free(ptr);

}

}

1. **Double Linked list**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* next;

struct node\* prev;

};

struct node\* head = NULL;

void create() {

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

printf("Enter node data: ");

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

temp->next = NULL;

temp->prev = NULL;

if (head == NULL) {

head = temp;

} else {

struct node\* ptr = head;

while (ptr->next != NULL) {

ptr = ptr->next;

}

ptr->next = temp;

temp->prev = ptr;

}

}

void display() {

if (head == NULL) {

printf("Doubly Linked List is Empty\n");

return;

}

printf("Doubly Linked List: ");

struct node\* ptr = head;

while (ptr != NULL) {

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

ptr = ptr->next;

}

printf("\n");

}

void insert\_begin() {

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

printf("Enter node data: ");

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

temp->next = NULL;

temp->prev = NULL;

if (head == NULL) {

head = temp;

} else {

temp->next = head;

head->prev = temp;

head = temp;

}

}

void insert\_end() {

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

printf("Enter node data: ");

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

temp->next = NULL;

temp->prev = NULL;

if (head == NULL) {

head = temp;

return;

}

struct node\* ptr = head;

while (ptr->next != NULL) {

ptr = ptr->next;

}

ptr->next = temp;

temp->prev = ptr;

}

void insert\_pos() {

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

printf("Enter node data: ");

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

temp->next = NULL;

temp->prev = NULL;

if (head == NULL) {

head = temp;

return;

}

int pos;

printf("Enter position: ");

scanf("%d", &pos);

struct node\* ptr = head;

struct node\* prev\_ptr = NULL;

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

prev\_ptr = ptr;

ptr = ptr->next;

}

temp->next = ptr;

temp->prev = prev\_ptr;

if (prev\_ptr != NULL) {

prev\_ptr->next = temp;

} else {

head = temp;

}

if (ptr != NULL) {

ptr->prev = temp;

}

}

void delete\_begin() {

if (head == NULL) {

printf("Doubly Linked List is empty | Nothing to delete\n");

return;

}

struct node\* ptr = head;

head = head->next;

if (head != NULL) {

head->prev = NULL;

}

free(ptr);

}

void delete\_end() {

if (head == NULL) {

printf("Doubly Linked List is empty | Nothing to delete\n");

return;

}

struct node\* ptr = head;

struct node\* prev\_ptr = NULL;

while (ptr->next != NULL) {

prev\_ptr = ptr;

ptr = ptr->next;

}

if (prev\_ptr != NULL) {

prev\_ptr->next = NULL;

} else {

head = NULL;

}

free(ptr);

}

void delete\_pos() {

if (head == NULL) {

printf("Doubly Linked List is empty | Nothing to delete\n");

return;

}

int pos;

printf("Enter node position to delete: ");

scanf("%d", &pos);

struct node\* ptr = head;

struct node\* prev\_ptr = NULL;

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

prev\_ptr = ptr;

ptr = ptr->next;

}

if (prev\_ptr != NULL) {

prev\_ptr->next = ptr->next;

} else {

head = ptr->next;

}

if (ptr->next != NULL) {

ptr->next->prev = prev\_ptr;

}

free(ptr);

}

int main() {

int choice;

while (1) {

printf("\n\*\*\*\*\*\n");

printf("0. Create\n");

printf("1. Display\n");

printf("2. Insert Node at beginning\n");

printf("3. Insert Node in specific position\n");

printf("4. Insert Node at end\n");

printf("5. Delete Node at beginning\n");

printf("6. Delete Node at end\n");

printf("7. Delete Node at position\n");

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

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 0: create();

break;

case 1: display();

break;

case 2: insert\_begin();

break;

case 3: insert\_pos();

break;

case 4: insert\_end();

break;

case 5: delete\_begin();

break;

case 6: delete\_end();

break;

case 7: delete\_pos();

break;

case 8: exit(0);

default:printf("\nWrong Choice");

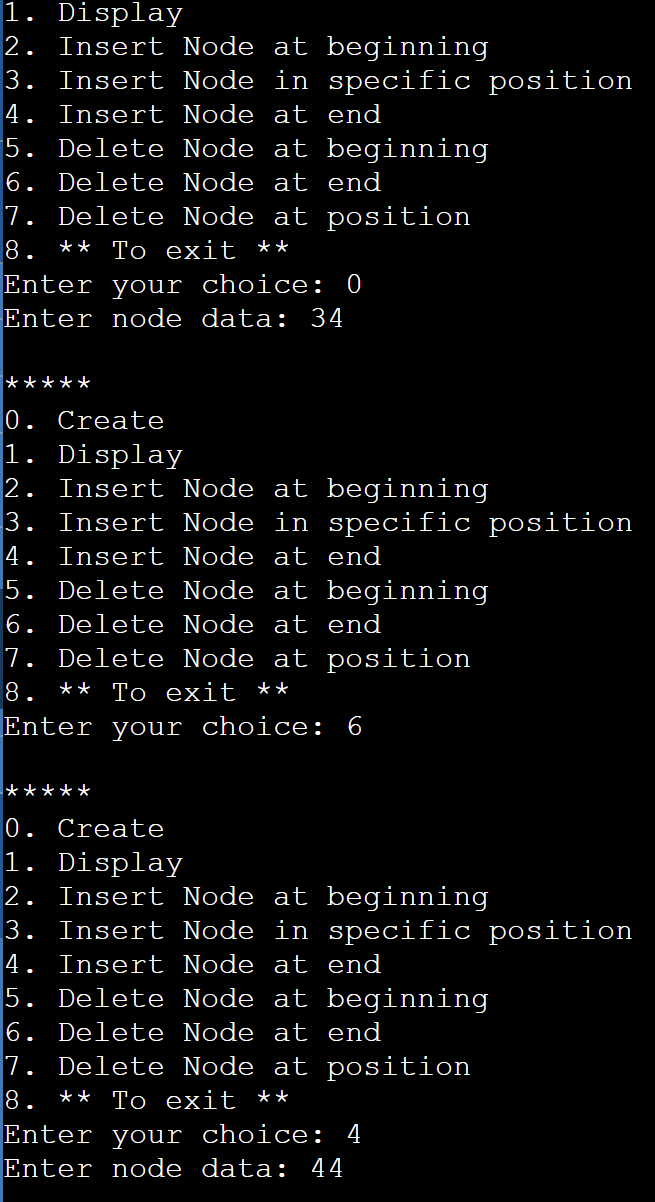
break;

}

}

return 0;

}



***Experiment 9***

**C program to implement the Circular Singly linked list:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

int main() {

int choice;

while (1) {

printf("\n\*\*\*\*\*\n");

printf("0. Create\n");

printf("1. Display\n");

printf("2. Insert Node at beginning\n");

printf("3. Insert Node in specific position\n");

printf("4. Insert Node at end of LinkedList\n");

printf("5. Delete Node at beginning\n");

printf("6. Delete Node at end\n");

printf("7. Delete Node at position\n");

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

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 0: create();

break;

case 1: display();

break;

case 2: insert\_begin();

break;

case 3: insert\_pos();

break;

case 4: insert\_end();

break;

case 5: delete\_begin();

break;

case 6: delete\_end();

break;

case 7: delete\_pos();

break;

case 8: exit(0);

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

break;

}

}

}

void create() {

struct Node\* temp;

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

printf("Enter node data: ");

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

temp->next = temp;

if (head == NULL) {

head = temp;

} else {

struct Node\* ptr = head;

while (ptr->next != head) {

ptr = ptr->next;

}

ptr->next = temp;

temp->next = head;

}

}

void display() {

if (head == NULL) {

printf("Circular Linked List is Empty\n");

return;

}

printf("Circular LinkedList: ");

struct Node\* ptr = head;

do {

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

ptr = ptr->next;

} while (ptr != head);

printf("\n");

}

void insert\_begin() {

struct Node\* temp;

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

printf("Enter node data: ");

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

temp->next = temp;

if (head == NULL) {

head = temp;

temp->next = head;

} else {

struct Node\* ptr = head;

while (ptr->next != head) {

ptr = ptr->next;

}

temp->next = head;

head = temp;

ptr->next = head;

}

}

void insert\_pos() {

struct Node\* temp;

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

printf("Enter node data: ");

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

temp->next = temp;

int pos;

printf("Enter position: ");

scanf("%d", &pos);

if (pos == 0) {

temp->next = head;

head = temp;

} else {

struct Node\* ptr = head;

for (int i = 0; i < pos - 1; i++) {

ptr = ptr->next;

}

temp->next = ptr->next;

ptr->next = temp;

}

}

void insert\_end() {

struct Node\* temp;

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

printf("Enter node data: ");

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

temp->next = temp;

if (head == NULL) {

head = temp;

temp->next = head;

} else {

struct Node\* ptr = head;

while (ptr->next != head) {

ptr = ptr->next;

}

ptr->next = temp;

temp->next = head;

}

}

void delete\_begin() {

if (head == NULL) {

printf("Circular Linked List is empty | Nothing to delete \n");

return;

} else if (head->next == head) {

struct Node\* ptr = head;

head = NULL;

free(ptr);

} else {

struct Node\* ptr = head;

while (ptr->next != head) {

ptr = ptr->next;

}

ptr->next = head->next;

struct Node\* temp = head;

head = head->next;

free(temp);

}

}

void delete\_end() {

if (head == NULL) {

printf("Circular Linked List is empty | Nothing to delete \n");

return;

} else if (head->next == head) {

struct Node\* ptr = head;

head = NULL;

free(ptr);

} else {

struct Node\* ptr = head;

struct Node\* prev\_ptr = NULL;

while (ptr->next != head) {

prev\_ptr = ptr;

ptr = ptr->next;

}

prev\_ptr->next = head;

free(ptr);

}

}

void delete\_pos() {

int pos;

printf("Enter node position to delete: ");

scanf("%d", &pos);

if (head == NULL) {

printf("Circular Linked List is empty | Nothing to delete \n");

return;

} else if (pos == 0) {

struct Node\* ptr = head;

while (ptr->next != head) {

ptr = ptr->next;

}

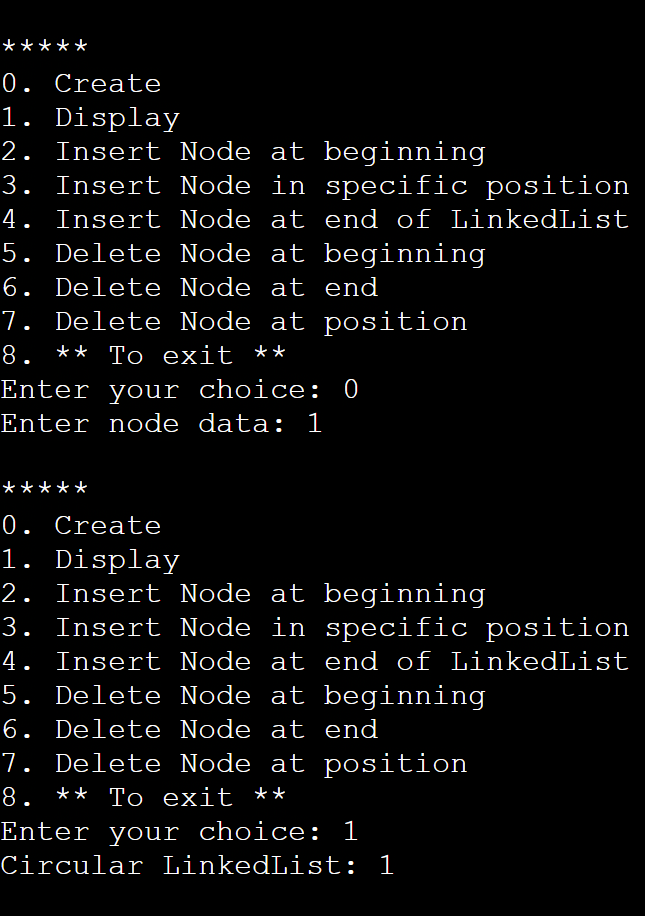
ptr->next = head->next;

struct Node\* temp = head;

head = head->next;

free(temp);

} else {

 struct Node\* ptr = head;

struct Node\* prev\_ptr;

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

prev\_ptr = ptr;

ptr = ptr->next;

}

prev\_ptr->next = ptr->next;

free(ptr);

}

}

***Experiment 10***

**C program to implement Binary Tree:**

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

struct node \*create();

void inorder(struct node \*root);

void preorder(struct node \*root);

void postorder(struct node \*root);

int main() {

struct node \*root = NULL;

int choice;

while(1) {

printf("1. Create\n2. Inorder\n3. Preorder\n4. Postorder\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1: root = create();

break;

case 2: inorder(root);

break;

case 3: preorder(root);

break;

case 4: postorder(root);

break;

case 5: exit(0);

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

}

}

return 0;

}

struct node \*create() {

struct node \*temp;

int data;

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

if(temp == NULL) {

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

exit(1);

}

printf("Enter data (Press 0 to exit): ");

scanf("%d", &data);

if(data == 0) {

free(temp);

return NULL;

}

temp->data = data;

printf("Enter the left child of %d: ", data);

temp->left = create();

printf("Enter the right child of %d: ", data);

temp->right = create();

return temp;

}

void preorder(struct node \*root) {

if(root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

}

void inorder(struct node \*root) {

if(root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

void postorder(struct node \*root) {

if(root != NULL) {

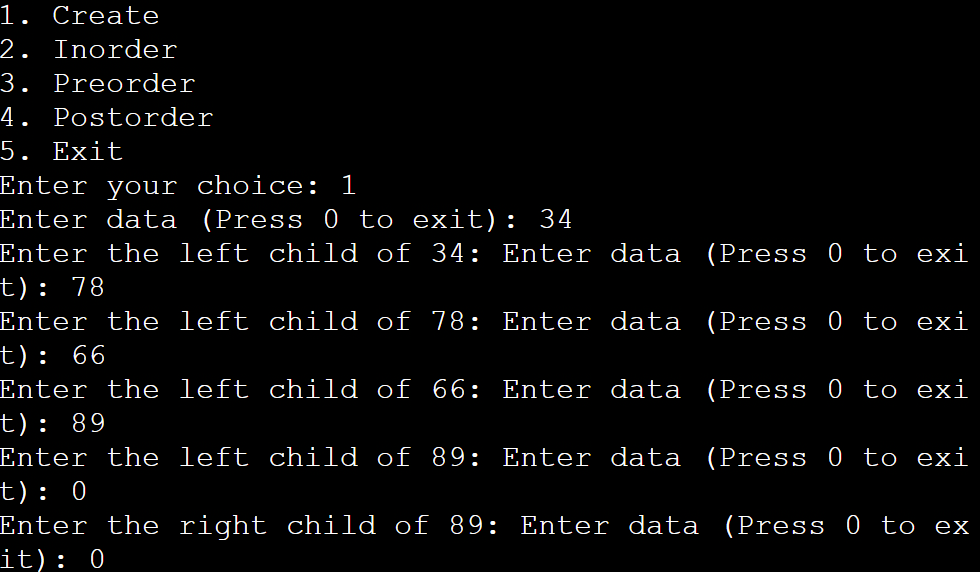
postorder(root->left);

postorder(root->right);

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

}

}



***Experiment 11***

**C program to implement a binary search tree:**

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*left, \*right;

};

struct node \*createTree(struct node \*root, int data);

void search(struct node \*root);

void findMax(struct node \*root);

struct node \*deleteNode(struct node \*root, int data);

struct node \*findMin(struct node \*root);

void preOrder(struct node \*root);

void inOrder(struct node \*root);

void postOrder(struct node \*root);

struct node \*root = NULL;

int main() {

int data, choice, i, n;

while(1) {

printf("\n1. Insertion in Binary Search Tree");

printf("\n2. Search Element in Binary Search Tree");

printf("\n3. Delete Element in Binary Search Tree");

printf("\n4. Inorder\n5. Preorder\n6. Postorder");

printf("\n7. Find Min\n8. Find Max\n9. Exit");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: printf("\nEnter how many nodes you want to insert: ");

scanf("%d", &n);

printf("\nEnter values: ");

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

scanf("%d", &data);

root = createTree(root, data);

}

break;

case 2: search(root);

break;

case 3: printf("\nEnter the element to delete: ");

scanf("%d", &data);

root = deleteNode(root, data);

break;

case 4: printf("\nInorder Traversal: ");

inOrder(root);

break;

case 5: printf("\nPreorder Traversal: ");

preOrder(root);

break;

case 6: printf("\nPostorder Traversal: ");

postOrder(root);

break;

case 7: {

struct node \*temp = findMin(root);

if (temp != NULL)

printf("\n%d is the minimum value in BST", temp->data);

else

printf("\nBST is empty");

break;

}

case 8: findMax(root);

break;

case 9: exit(0);

default: printf("WRONG CHOICE");

break;

}

}

return 0;

}

struct node \*createTree(struct node \*root, int data) {

if (root == NULL) {

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

if (temp == NULL) {

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

exit(1);

}

temp->data = data;

temp->left = NULL;

temp->right = NULL;

return temp;

}

if (data < (root->data)) {

root->left = createTree(root->left, data);

} else if (data > root->data) {

root->right = createTree(root->right, data);

}

return root;

}

void preOrder(struct node \*root) {

if (root != NULL) {

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

preOrder(root->left);

preOrder(root->right);

}

}

void inOrder(struct node \*root) {

if (root != NULL) {

inOrder(root->left);

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

inOrder(root->right);

}

}

void postOrder(struct node \*root) {

if (root != NULL) {

postOrder(root->left);

postOrder(root->right);

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

}

}

struct node \*deleteNode(struct node \*root, int data) {

if (root == NULL) {

printf("\nElement not found");

} else if (data < root->data) {

root->left = deleteNode(root->left, data);

} else if (data > root->data) {

root->right = deleteNode(root->right, data);

} else {

if (root->right && root->left) {

struct node \*temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

} else {

struct node \*temp = root;

if (root->left == NULL) {

root = root->right;

} else if (root->right == NULL) {

root = root->left;

}

free(temp);

}

}

return root;

}

struct node \*findMin(struct node \*root) {

if (root == NULL) {

return NULL;

}

if (root->left != NULL) {

return findMin(root->left);

} else {

return root;

}

}

void findMax(struct node \*root) {

if (root == NULL) {

printf("\nBST is empty");

return;

}

while (root->right != NULL) {

root = root->right;

}

printf("\n%d is the maximum value in BST", root->data);

}

void search(struct node \*root) {

int data;

if (root == NULL) {

printf("\nBST is empty.");

return;

}

printf("\nEnter element to be searched: ");

scanf("%d", &data);

while (root != NULL) {

if (root->data == data) {

printf("\nKey element is present in BST");

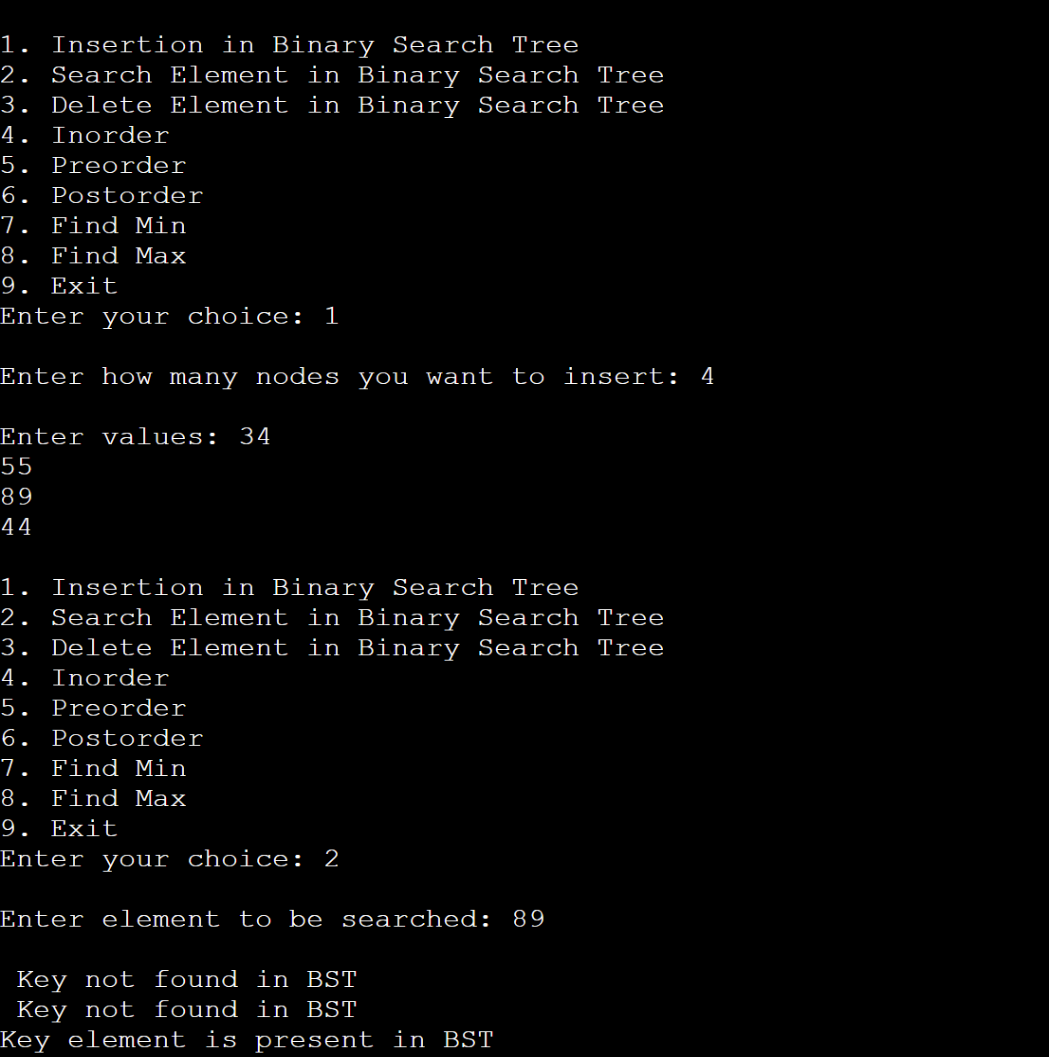
return;

}

if (data < root->data) {

root = root->left;

} else {

 root = root->right;

}

printf("\n Key not found in BST");

}

}

*Experiment 12*

**C program to implement Hash table with the linear probing method:**

#include<stdio.h>

#include<stdlib.h>

#define size 10

int arr[size];

void init();

void insert(int value);

void del(int value);

void printTable();

int main() {

int ch = 0, temp;

init();

while (ch != 4) {

printf("\n1. Insert\n2. Delete\n3. Print hash table\n4. Exit\nEnter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

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

scanf("%d", &temp);

insert(temp);

break;

case 2:

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

scanf("%d", &temp);

del(temp);

break;

case 3:

printTable();

break;

case 4:

exit(0);

break;

default:

printf("Wrong input!\n");

break;

}

}

return 0;

}

void init() {

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

arr[i] = -1;

}

void insert(int value) {

int key = value % size;

if (arr[key] == -1) {

arr[key] = value;

printf("%d inserted with key %d!\n", value, key);

} else {

printf("Collision: key %d has element %d already!\n", key, arr[key]);

while (arr[key] != -1) {

key++;

if (key == size) {

printf("Unable to insert %d\n", value);

return;

}

}

arr[key] = value;

printf("%d inserted with key %d!\n", value, key);

}

}

void del(int value) {

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

if (arr[i] == value) {

arr[i] = -1;

printf("%d is deleted\n", value);

return;

}

}

printf("%d not found in the hash table\n", value);

}

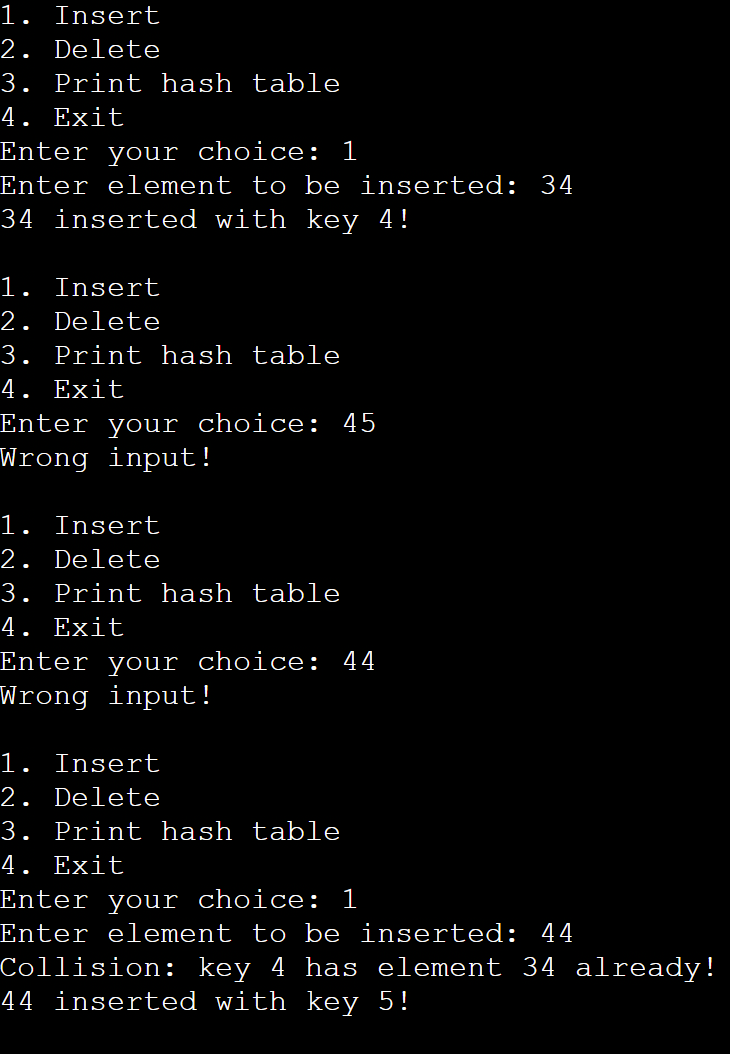
void printTable() {

printf("\nVALUE\t---\tKEY\n===================\n");

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

if (arr[i] != -1) {

printf("%d\t---\t%d\n", arr[i], i);

 }

}

}

***Experiment 13***

**C program for implementation of graph traversals by applying :**

1. **BFS b) DFS**

#include <stdio.h>

int n, i, j, visited[10], queue[10], front = -1, rear = -1;

int adj[10][10];

void bfs(int v)

{

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

if (adj[v][i] && !visited[i])

queue[++rear] = i;

if (front <= rear)

{

visited[queue[front]] = 1;

bfs(queue[front++]);

}

}

int main()

{

int v;

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

scanf("%d", &n);

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

{

queue[i] = 0;

visited[i] = 0;

}

printf("Enter graph data in matrix form: \n");

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

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

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

printf("Enter the starting vertex: ");

scanf("%d", &v);

bfs(v);

printf("The node which are reachable are: \n");

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

if (visited[i])

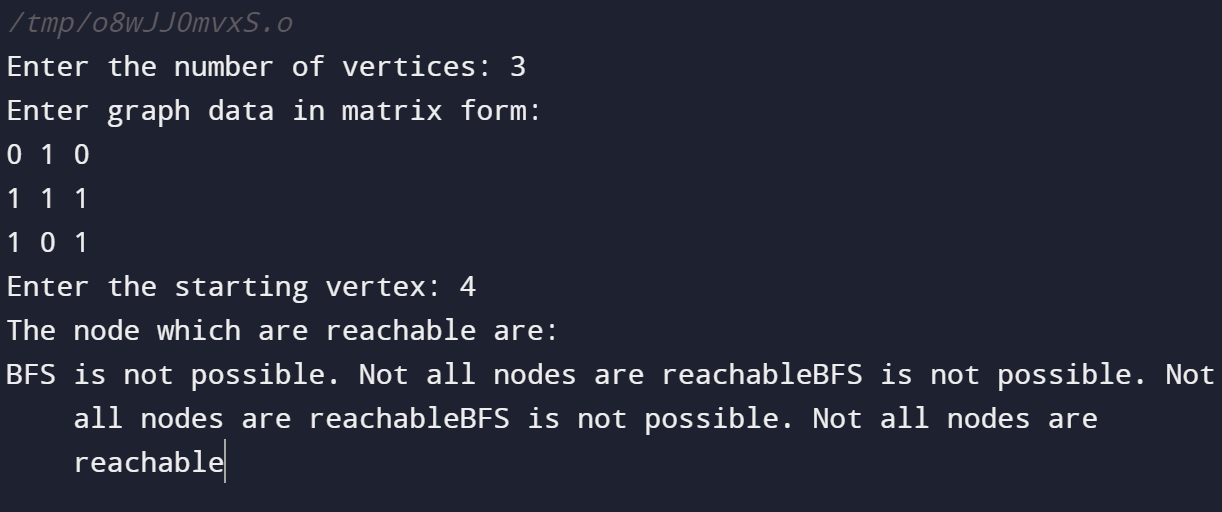
printf("%d\t", i);

else

printf("BFS is not possible. Not all nodes are reachable");

return 0;

}



#include<stdio.h>

#include<stdlib.h>

int visited[7] = {0,0,0,0,0,0,0};

int A [7][7] = {

{0,1,1,1,0,0,0},

{1,0,1,0,0,0,0},

{1,1,0,1,1,0,0},

{1,0,1,0,1,0,0},

{0,0,1,1,0,1,1},

{0,0,0,0,1,0,0},

{0,0,0,0,1,0,0}

};

void DFS(int i){

printf("%d ", i);

visited[i] = 1;

for (int j = 0; j < 7; j++)

{

if(A[i][j]==1 && !visited[j]){

DFS(j);

}

}

}

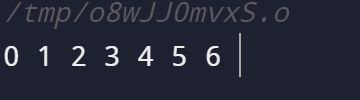
int main(){

// DFS Implementation

DFS(0);

return 0;

}



***Experiment 14***

**C program to implement insertion sort and selection sort:**

#include <stdio.h>

void insertionSort(int array[], int n)

{

int i, element, j;

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

{

element = array[i];

j = i - 1;

while (j >= 0 && array[j] > element) {

array[j + 1] = array[j];

j = j - 1;

}

array[j + 1] = element;

}

}

void printArray(int array[], int n)

{

int i;

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

printf("%d ", array[i]);

printf("\n");

}

// Main Function

int main()

{

int array[] = { 54, 43, 39, 8, 67 };

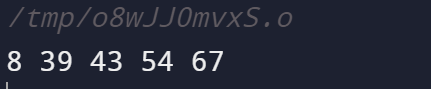
int n = sizeof(array) / sizeof(array[0]);

insertionSort(array, n);

printArray(array, n);

return 0;

}



#include <stdio.h>

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void selectionSort(int array[], int n)

{

int i, j, min\_element;

for (i = 0; i < n-1; i++)

{

min\_element = i;

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

if (array[j] < array[min\_element])

min\_element = j;

swap(&array[min\_element], &array[i]);

}

}

void printArray(int array[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", array[i]);

printf("null");

}

// Main Function

int main()

{

int array[] = {8, 12, 90, 42, 88};

int size = sizeof(array)/sizeof(array[0]);

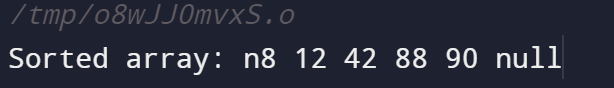
selectionSort(array, size);

printf("Sorted array: n");

printArray(array, size);

return 0;

}



***Experiment 15***

**C program to implement Quick sort and merge sort :**

#include<stdio.h>

// Function to swap two elements

void swapElements(int\* x, int\* y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

// Partition function

int partition (int arr[], int lowIndex, int highIndex)

{

int pivotElement = arr[highIndex];

int i = (lowIndex - 1);

for (int j = lowIndex; j <= highIndex- 1; j++)

{

if (arr[j] <= pivotElement)

{

i++;

swapElements(&arr[i], &arr[j]);

}

}

swapElements(&arr[i + 1], &arr[highIndex]);

return (i + 1);

}

// QuickSort Function

void quickSort(int arr[], int lowIndex, int highIndex)

{

if (lowIndex < highIndex)

{

int pivot = partition(arr, lowIndex, highIndex);

// Separately sort elements before & after partition

quickSort(arr, lowIndex, pivot - 1);

quickSort(arr, pivot + 1, highIndex);

}

}

// Function to print array

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

}

// Main Function

int main()

{

int arr[] = {8, 54, 40, 28, 88, 3};

int n = sizeof(arr)/sizeof(arr[0]);

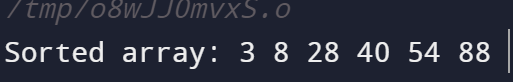
quickSort(arr, 0, n-1);

printf("Sorted array: ");

printArray(arr, n);

return 0;

}



b)

#include<stdlib.h>

#include<stdio.h>

// Merge Function

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

// Merge Sort Function in C

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

// Functions to Print Elements of Array

void printArray(int A[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", A[i]);

printf("n");

}

// Main Method

int main()

{

int arr[] = {4, 53, 68, 86, 21, 43, 9, 26};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\n Sorted array is \n");

printArray(arr, arr\_size);

return 0;

}

