**Lab Report**

**NAME : Rohit Satheesh Nair**

**USN:1BM19CS206**

**SUBJECT:DATA STRUCTURES**

**ACADEMIC YEAR-Sept-Dec-2020**

**1)Write a program to simulate the working of stack using an array with the following : a) Push b) Pop c) Display The program should print appropriate messages for stack overflow, stack underflow.**

#include<stdio.h>

#define SIZE 100

struct student

{

int id;

int age;

int marks;

};

int main()

{

struct student s[SIZE];

int i,n;

printf("Please enter the number of students:");

scanf("%d",&n);

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

{

printf("Enter the student id:\n");

scanf("%d",&s[i].id);

printf("Enter the age of the student:\n");

scanf("%d",&s[i].age);

printf("Enter the marks of the student:\n");

scanf("%d",&s[i].marks);

}

printf("The details of student(s) who are qualified:\n");

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

{

if(s[i].age>20 && s[i].marks>=65 && s[i].marks<=100)

{

printf("Student ID:%d\n",s[i].id);

printf("Student age:%d\n",s[i].age);

printf("Student marks:%d\n",s[i].marks);

}

}

return 0;

}

Text

Description automatically generated

**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)**

#include<stdio.h>

#include<ctype.h>

#define SIZE 50

char stack[SIZE];

int top=-1;

void push(char elem)

{

stack[++top]=elem;

}

char pop()

{

return stack[top--];

}

int pr(char symbol)

{

if(symbol=='^')

{

return(3);

}

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

{

return(2);

}

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

{

return(1);

}

else

{

return(0);

}

}

int main()

{

char infix[50],postfix[50],ch,elem;

int i=0,k=0;

printf("Enter the Infix expression: ");

scanf("%s",&infix);

push('#');

while((ch=infix[i++])!='\0')

{

if(ch=='(')push(ch);

else

if(isalnum(ch))postfix[k++]=ch;

else

if(ch==')')

{

while(stack[top]!='(')

postfix[k++]=pop();

elem=pop();

}

else

{

while(pr(stack[top])>=pr(ch))

postfix[k++]=pop();

push(ch);

}

}

while(stack[top]!='#')

postfix[k++]=pop();

postfix[k]='\0';

printf("\nPostfix Expression = %s\n",postfix);

return 0;

}#include<stdio.h>

#include<ctype.h>

#define SIZE 50

char stack[SIZE];

int top=-1;

void push(char elem)

{

stack[++top]=elem;

}

char pop()

{

return stack[top--];

}

int pr(char symbol)

{

if(symbol=='^')

{

return(3);

}

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

{

return(2);

}

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

{

return(1);

}

else

{

return(0);

}

}

int main()

{

char infix[50],postfix[50],ch,elem;

int i=0,k=0;

printf("Enter the Infix expression: ");

scanf("%s",&infix);

push('#');

while((ch=infix[i++])!='\0')

{

if(ch=='(')push(ch);

else

if(isalnum(ch))postfix[k++]=ch;

else

if(ch==')')

{

while(stack[top]!='(')

postfix[k++]=pop();

elem=pop();

}

else

{

while(pr(stack[top])>=pr(ch))

postfix[k++]=pop();

push(ch);

}

}

while(stack[top]!='#')

postfix[k++]=pop();

postfix[k]='\0';

printf("\nPostfix Expression = %s\n",postfix);

return 0;

}

Text

Description automatically generated

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

**a) Insert**

**b) Delete**

**c) Display**

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

#include<stdio.h>

#include<stdlib.h>

#define MAX 50

int queue[MAX];

int front = -1;

int rear = -1;

void insert()

{

int element;

if(rear==MAX-1)

printf("Queue Overflow!\n");

else

{

if(front==-1)

front=0;

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

scanf("%d",&element);

queue[++rear]=element;

printf("Element successfully inserted!!!\n");

}

}

void delete()

{

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

{

printf("Queue underflow!\n");

}

else

{

printf("Deleted Element: %d\n",queue[front++]);

if(front>rear)

{

front=-1;

rear=-1;

}

}

}

void display()

{

int i;

if(front==-1)

{

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

}

else

{

printf("Elements in the Queue are : \n");

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

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

}

}

int main()

{

int choice;

while(1)

{

printf("\n\n");

printf("(1)INSERT\n(2)DELETE\n(3)DISPLAY\n(4)EXIT");

printf("\nEnter you choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

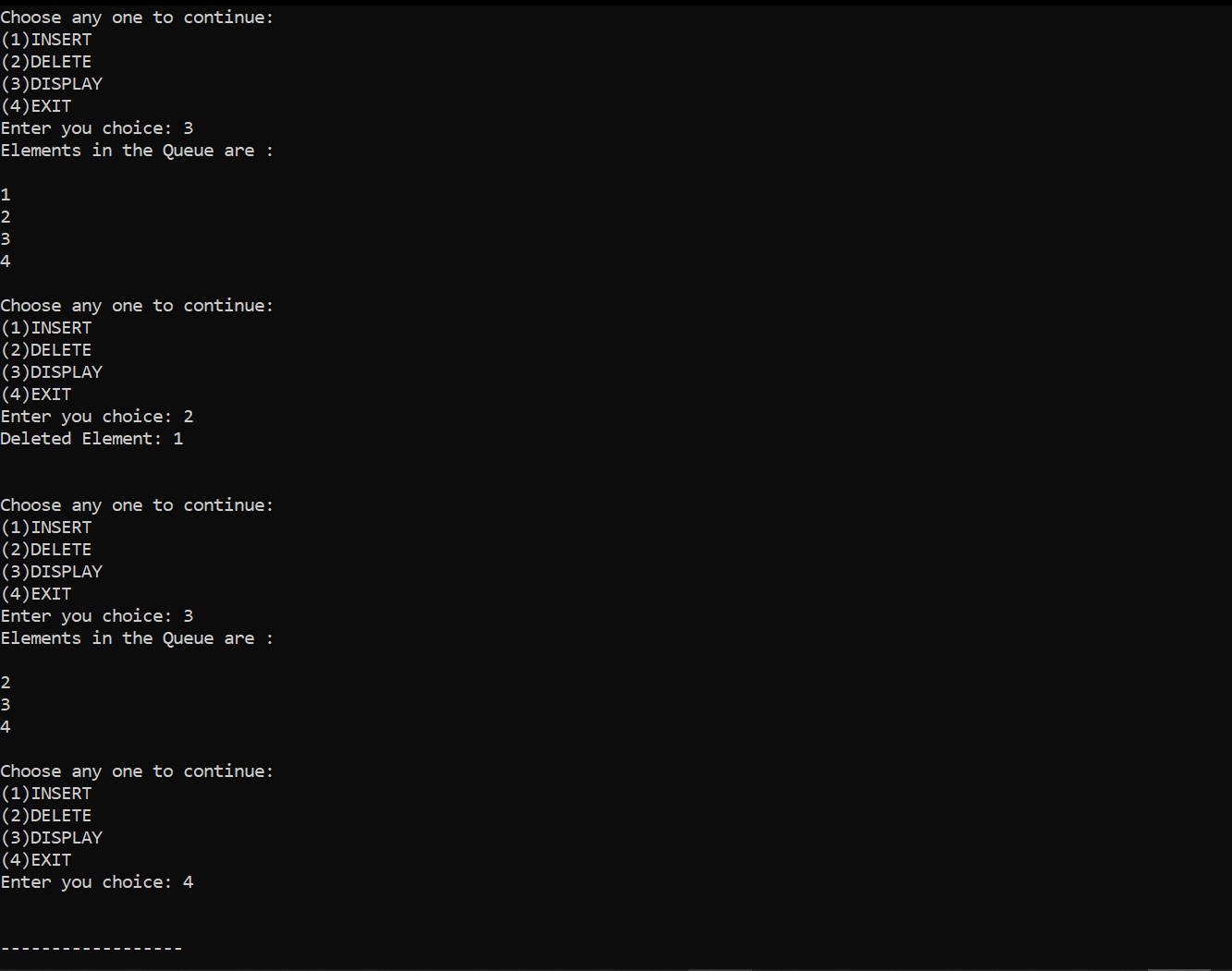
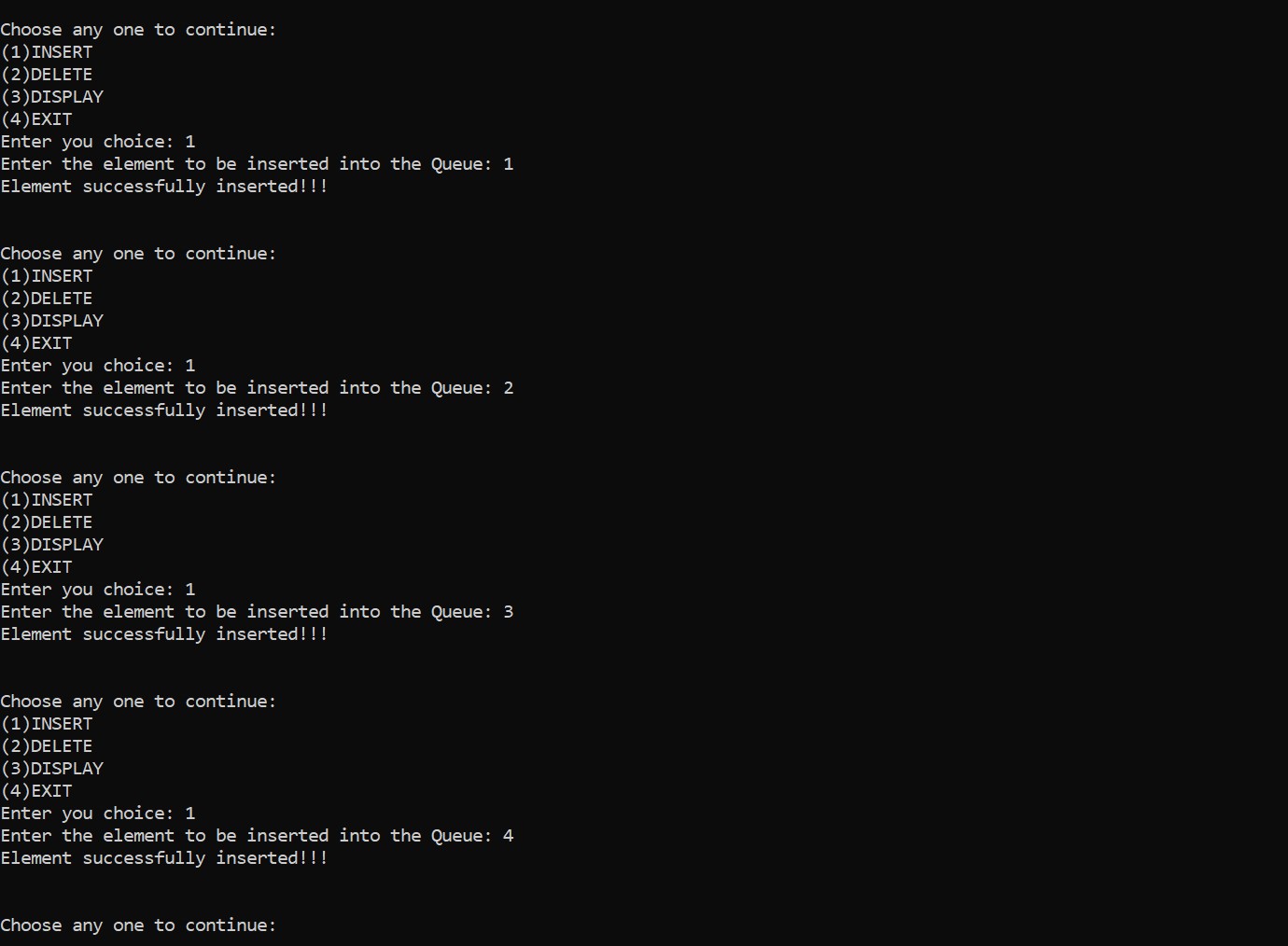
printf("Invalid choice!\n");

}

}

return 0;

}



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

**a)  Insert**

**b)  Delete**

**c)  Display**

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

#include<stdio.h>

#include<stdlib.h>

#define size 5

int Q[size];

int front=-1,rear=-1;

int isfull()

{

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

{

return 1;

}

return 0;

}

int isempty()

{

if(front==-1)

{

return 1;

}

return 0;

}

void enqueue()

{

int element;

if(isfull())

{

printf("\nQueue Overflow!!!\n");

}

else

{

if(front==-1)

{

front=0;

}

printf("\nEnter the element to be inserted into the Queue: ");

scanf("%d",&element);

rear = (rear+1)%size;

Q[rear]=element;

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

}

}

void dequeue()

{

int element;

if(isempty())

{

printf("\nQueue Underflow!\n");

}

else

{

element=Q[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

front=(front+1)%size;

}

printf("\nDeleted Element is %d\n",element);

}

}

void display()

{

int i;

if(isempty())

{

printf("\nQueue is Empty!!Enter some Elements!!\n");

}

else

{

printf("\nFront--> %d",front);

printf("\nQueue Elements: \n");

for(i=front;i!=rear;i=(i+1)%size)

{

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

}

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

printf("\nRear--> %d\n",rear);

}

}

int main()

{

int choice;

while(1)

{

printf("\n-----MENU-----\n");

printf("(1)INSERT\n");

printf("(2)DELETE\n");

printf("(3)DISPLAY\n");

printf("(4)EXIT\n");

printf("Enter you choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

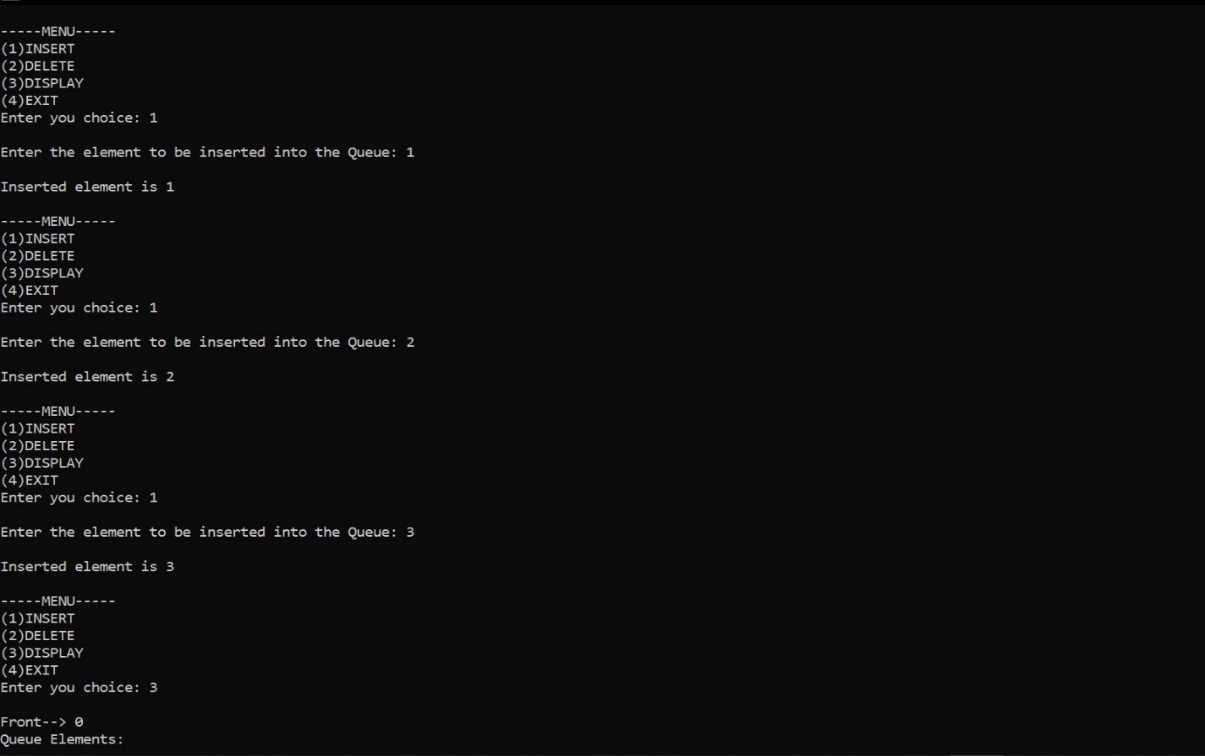
printf("\nInvalid choice!!Choose another one\n");

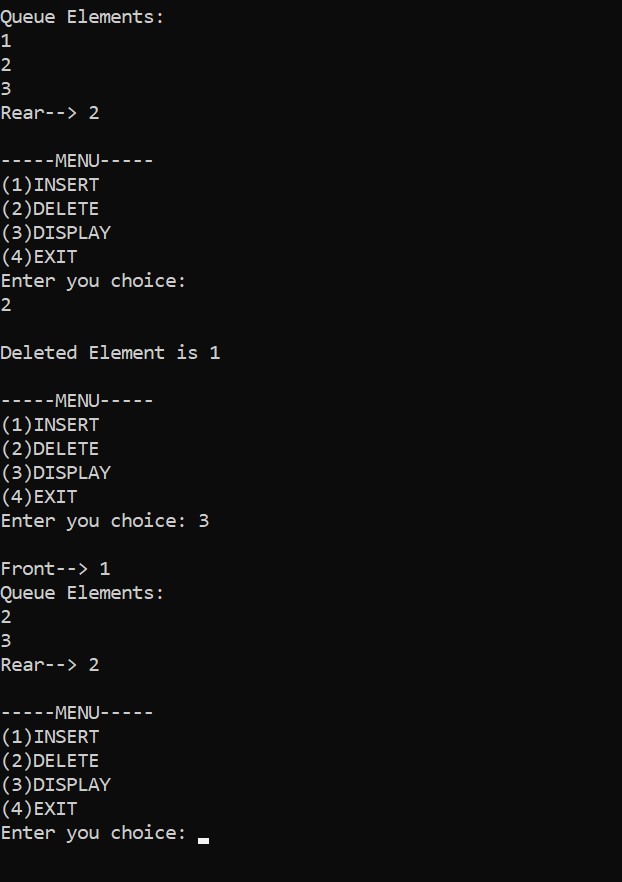
}

}

return 0;

}





**5)WAP to Implement Singly Linked List with following operations**

**a)  Create a linked list.**

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

**c)  Display the contents of the linked list.**

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main()

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

printf("\n 1. Create a list");

printf("\n 2. Display the list");

printf("\n 3. Insert node at the beginning");

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

printf("\n 5. Insert node at any specified position");

printf("\n 9. Exit");

int choice;

while(1){

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

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

display();

break;

case 3:

insert\_begin();

break;

case 4:

insert\_end();

break;

case 5:

insert\_pos();

break;

case 9:

exit(0);

break;

default:

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

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

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

printf("\nEnter the value for the node: ");

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

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

if(start==NULL)

{

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

return;

}

else

{

ptr=start;

printf("\n The List elements are:\n");

while(ptr!=NULL)

{

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

ptr=ptr->next ;

}

}

}

void insert\_begin()

{

struct node \*temp;

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

printf("\nEnter the value for the node: " );

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

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->next=start;

start=temp;

}

}

void insert\_end()

{

struct node \*temp,\*ptr;

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

printf("\nEnter the value for the node: " );

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

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next !=NULL)

{

ptr=ptr->next ;

}

ptr->next =temp;

}

}

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

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

printf("\nEnter the position for the new node to be inserted: ");

scanf("%d",&pos);

printf("\nEnter the value of the node: ");

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

temp->next=NULL;

if(pos==0)

{

temp->next=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++)

{

ptr=ptr->next;

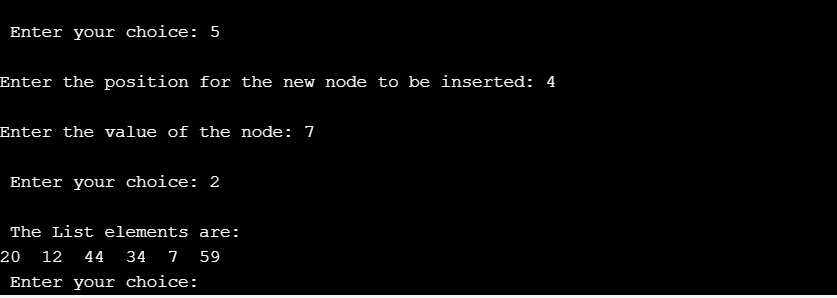
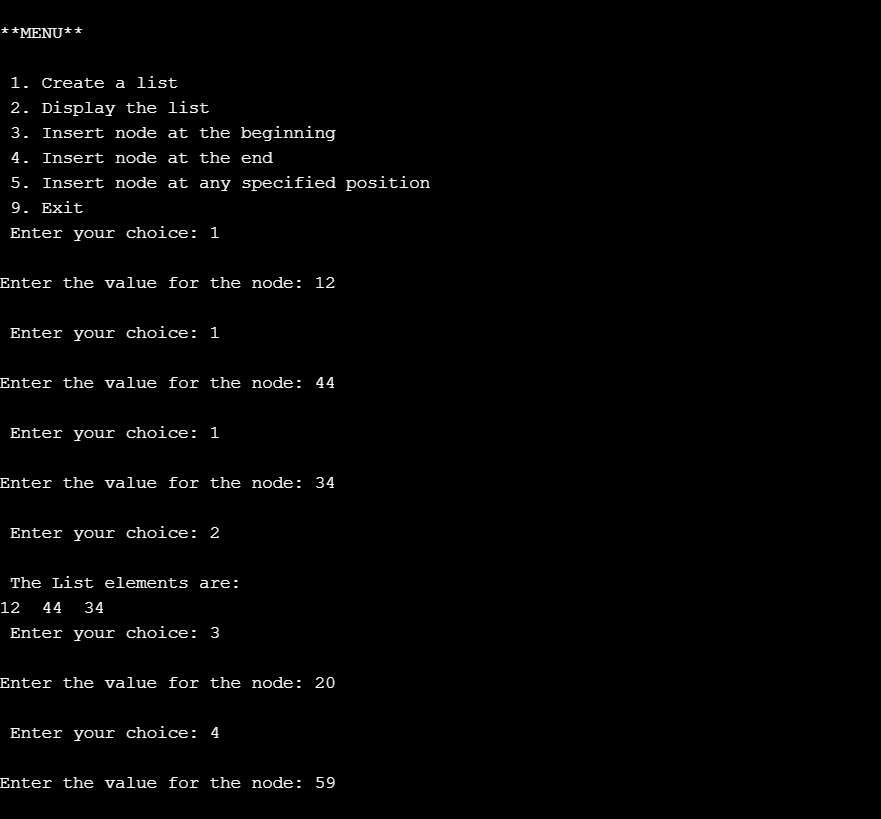
}

temp->next =ptr->next ;

ptr->next=temp;

}

}



**6)WAP to Implement Singly Linked List with following operations**

**a)  Create a linked list.**

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

**c)  Display the contents of the linked list.**

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void delete\_begin();

void delete\_end();

void delete\_pos();

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main()

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

printf("\n 1. Create a list");

printf("\n 2. Display the list");

printf("\n 3. Delete node from the beginning");

printf("\n 4. Delete node from the end");

printf("\n 5. Delete node from any specified position");

printf("\n 9. Exit");

int choice;

while(1){

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

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

display();

break;

case 3:

delete\_begin();

break;

case 4:

delete\_end();

break;

case 5:

delete\_pos();

break;

case 9:

exit(0);

break;

default:

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

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

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

printf("\nEnter the value for the node: ");

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

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

if(start==NULL)

{

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

return;

}

else

{

ptr=start;

printf("\n The List elements are:\n");

while(ptr!=NULL)

{

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

ptr=ptr->next ;

}

}

}

void delete\_begin()

{

struct node \*ptr;

if(start==NULL)

{

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

return;

}

else

{

ptr=start;

start=start->next ;

printf("\nThe deleted element is : %d ",ptr->info);

free(ptr);

}

}

void delete\_end()

{

struct node \*temp,\*ptr;

if(start==NULL)

{

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

exit(0);

}

else if(start->next ==NULL)

{

ptr=start;

start=NULL;

printf("\nThe deleted element is: %d ",ptr->info);

free(ptr);

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=NULL;

printf("\nThe deleted element is: %d ",ptr->info);

free(ptr);

}

}

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\nThe List is Empty!\n");

exit(0);

}

else

{

printf("\nEnter the position of the node to be deleted: \n");

scanf("%d",&pos);

if(pos==0)

{

ptr=start;

start=start->next ;

printf("\nThe deleted element is: %d ",ptr->info );

free(ptr);

}

else

{

ptr=start;

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

{

temp=ptr;

ptr=ptr->next ;

if(ptr==NULL)

{

printf("\nPosition not Found!\n");

return;

}

}

temp->next =ptr->next ;

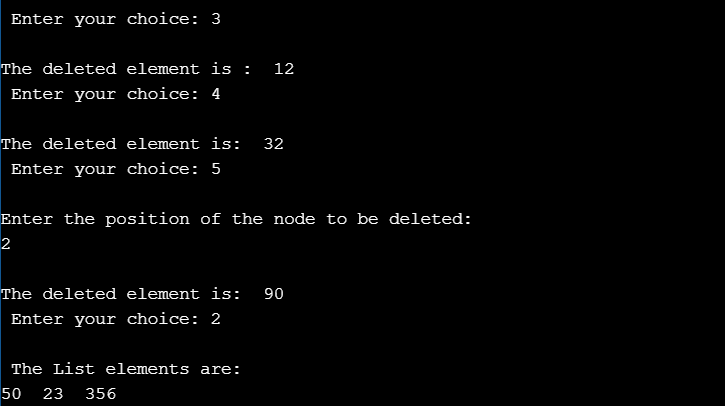
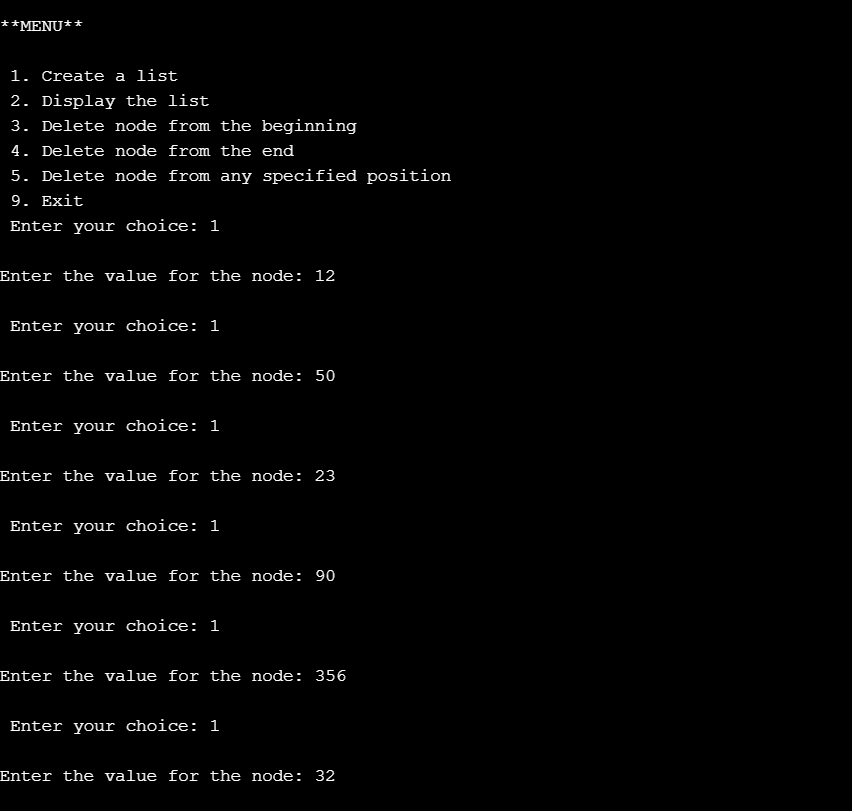
printf("\nThe deleted element is: %d ",ptr->info );

free(ptr);

}

}

}



**7)WAP Implement Single Link List with following operations**

**a)  Sort the linked list.**

**b)  Reverse the linked list.**

**c)  Concatenation of two linked lists**

#include <stdlib.h>

#include<stdio.h>

#include <string.h>

struct node

{

int sem;

struct node \*next;

};

struct node \*head= NULL;

struct node \*head2= NULL;

int c=0;

void Insert()

{

struct node \*newnode;

struct node \*temp;

int s;

printf("Enter integer : ");

scanf("%d",&s);

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

newnode->sem =s;

if (head==NULL)

{

newnode->next=NULL;

head=newnode;

c++;

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

c++;

\

}

}

void Insert2()

{

struct node \*newnode;

struct node \*temp;

int s,y;

printf("enter elements to create list 2\n");

do

{

printf("Enter integer : \n");

scanf("%d",&s);

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

newnode->sem =s;

if (head2==NULL)

{

newnode->next=NULL;

head2=newnode;

c++;

}

else

{

temp=head2;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

c++;

printf("Node created\n");

}

printf("do you want to continue adding:1(yes) or 0(no)\n");

scanf("%d",&y);

}while(y!=0);

}

void bubbleSort()

{

int swapped, i;

struct node \*ptr1;

struct node \*lptr = NULL;

if (head == NULL)

return;

do

{

swapped = 0;

ptr1 = head;

while (ptr1->next != lptr)

{

if (ptr1->sem > ptr1->next->sem)

{

int temp = ptr1->sem;

ptr1->sem = ptr1->next->sem;

ptr1->next->sem = temp;

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

}

while (swapped);

}

void reverse()

{

struct node\* prev = NULL;

struct node\* current = head;

struct node\* next ;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head= prev;

}

void concat()

{

struct node \*ptr;

if(head==NULL)

{

head=head2;

}

if(head2==NULL)

{

head2=head;

}

ptr=head;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=head2;

}

void display1()

{

struct node \*ptr;

ptr=head;

int i=1;

if(ptr==NULL)

{

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

}

else

{

while(ptr!= NULL)

{

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

i++;

ptr=ptr->next;

}

}

}

void display2()

{

struct node \*ptr;

ptr=head2;

int i=1;

if(ptr==NULL)

{

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

}

else

{

while(ptr!= NULL)

{

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

printf("\n");

i++;

ptr=ptr->next;

}

}

}

int main()

{ printf("\n1. Insert node \n2. sort node\n3. reverse node\n4.concat 2 lists \n5.exit\n");

int choice,pos;

do

{

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

Insert();

break;

case 2:

bubbleSort();

display1();

break;

case 3:

reverse();

display1();

break;

case 4:

Insert2();

concat();

display1();

break;

case 5:

break;

default:

printf("Wrong choice!\n");

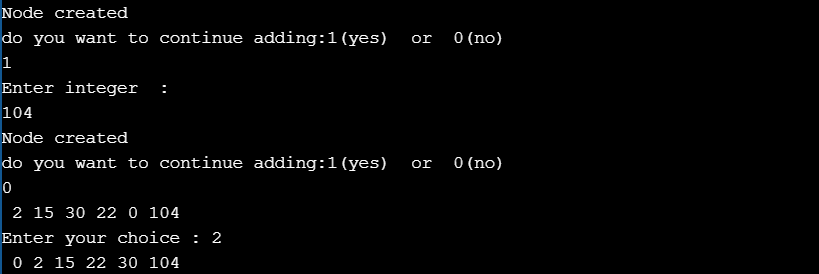
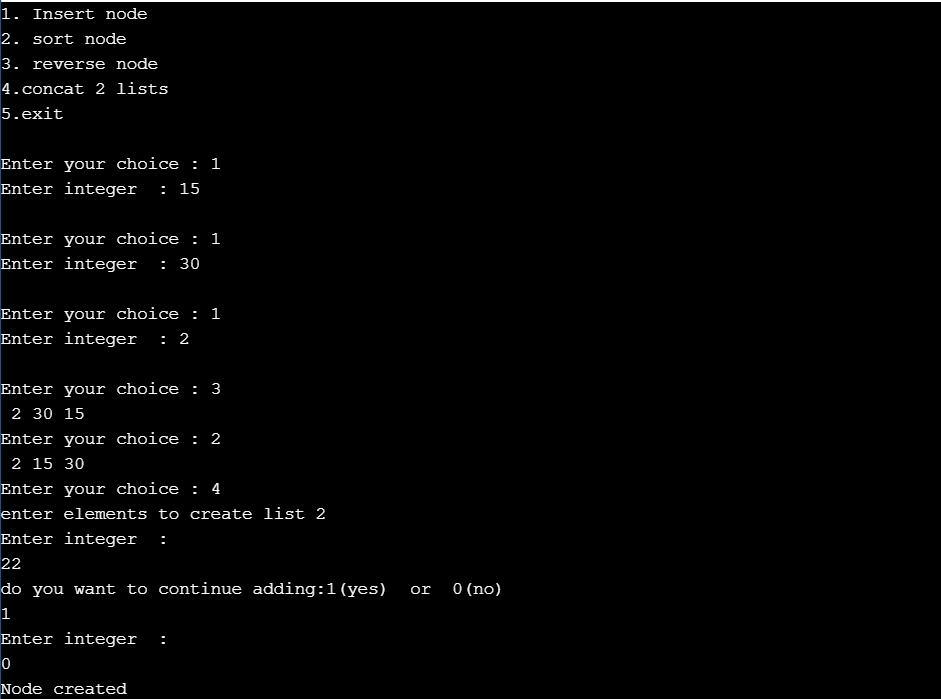
break;

}

}while(choice!=5);

return 0;

}



**8)WAP to implement Stack & Queues using Linked Representation**

**Stack using linked list**

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

int data;

struct node \*link;

}node;

node \*top=NULL;

void push()

{

node \*temp;

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

printf("Enter node element\n");

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

temp->link=NULL;

if(top==NULL)

{

top=temp;

}

else

{

temp->link=top;

top=temp;

}

}

void pop()

{

node \*temp;

if(top==NULL)

{

printf("Stack is empty\n");

}

else

{

temp=top;

top=temp->link;

temp->link=NULL;

free(temp);

}

}

void display()

{

node \*temp=top;

if(temp==NULL)

{

printf("Stack is empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int main()

{

printf("Enter the operation\n1.Push\n");

printf("2.Pop\n3.Display\n4.Exit\n");

int op,len;

while(1)

{ printf("choose your option : ");

scanf("%d",&op);

switch (op)

{

case 1:push();

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

break;

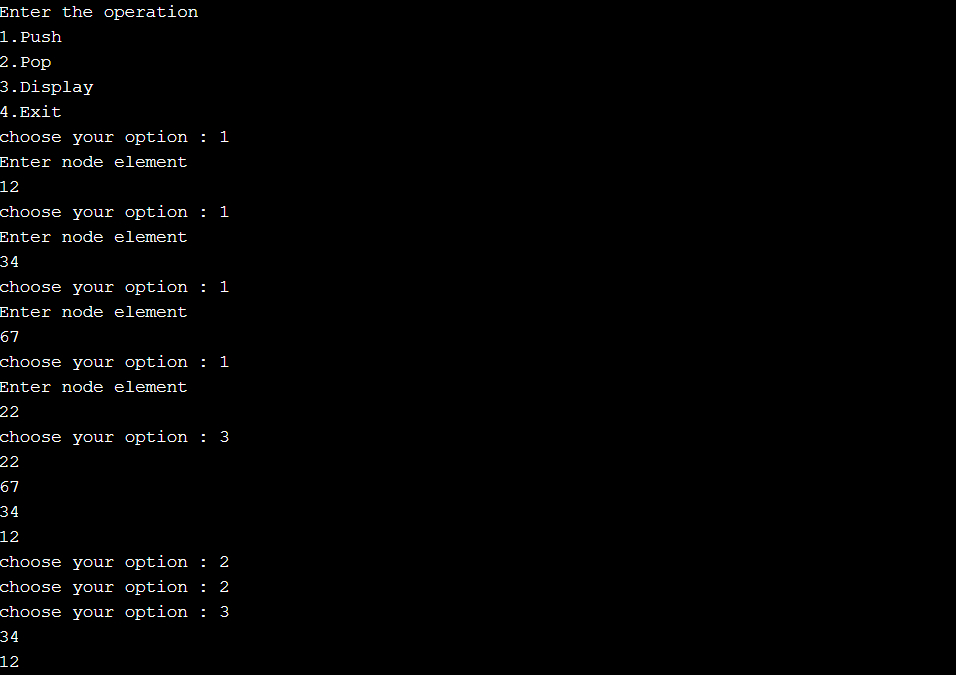
default: printf("No such operation\n");

}

}

return 0;

}

****

**Queue using linked list**

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

int data;

struct node \*link;

}node;

node \*root=NULL;

void enqueue()

{ //insert at end (rear)

node \*temp;

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

printf("Enter the node element\n");

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

temp->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

node \*p=root;

while(p->link!=NULL)

{

p=p->link;

}

p->link=temp;

}

}

void dequeue()

{

node \*temp;

if(root==NULL) //delete from front

{

printf("Queue is empty\n");

}

else

{

temp=root;

root=temp->link;

temp->link=NULL;

free(temp);

}

}

void display()

{

node \*temp=root;

if(temp==NULL)

{

printf("Queue is empty\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int main()

{

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

int op,len;

while(1)

{ printf("Enter the operation: ");

scanf("%d",&op);

switch (op)

{

case 1:enqueue();

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

break;

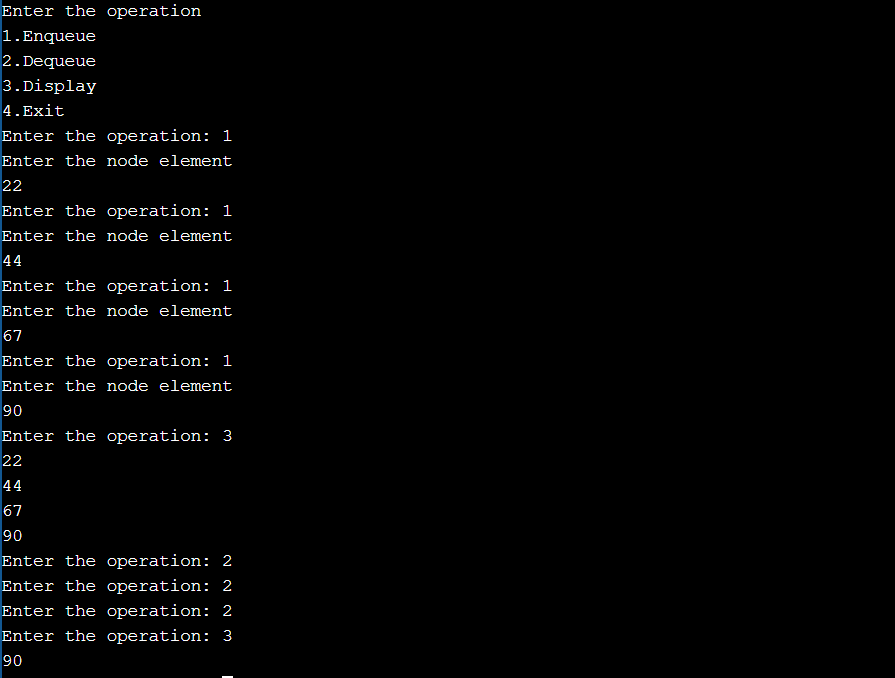
default: printf("No such operation\n");

}

}

return 0;

}

****

**9)WAP 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<conio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head;

void insertbeginning(int item)

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

if(head==NULL)

{ ptr->next = NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr; }

else

{ ptr->data=item;

ptr->prev=NULL;

ptr->next = head;

head->prev=ptr;

head=ptr; }

}

void delete\_specified( )

{ struct node \*ptr, \*temp;

int val;

printf("Enter the value");

scanf("%d",&val);

temp = head;

while(temp -> data != val)

temp = temp -> next;

if(temp -> next == NULL)

{ printf("\nCan't delete\n"); }

else if(temp -> next -> next == NULL)

{ temp ->next = NULL;

printf("\nNode Deleted\n");

}

else

{ ptr = temp -> next;

temp -> next = ptr -> next;

ptr -> next -> prev = temp;

free(ptr);

printf("\nNode Deleted\n");

}

}

void display()

{

struct node \*ptr;

ptr=head;

if(ptr==NULL)

{

printf("empty ");

}

else

{

while(ptr!=NULL)

{

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

ptr=ptr -> next;

}

printf("\n");

}

}

void main()

{

int op=0;int a;

printf("\n 1.Insert to the left\n2.Delete\n3.Display\n ");

while(op!=4)

{

printf("\nenter your choice : ");

scanf("%d",&op);

switch(op)

{

case 1:printf("enter value to be inserted ");

scanf("%d",&a);

insertbeginning(a);

break;

case 2:delete\_specified();

break;

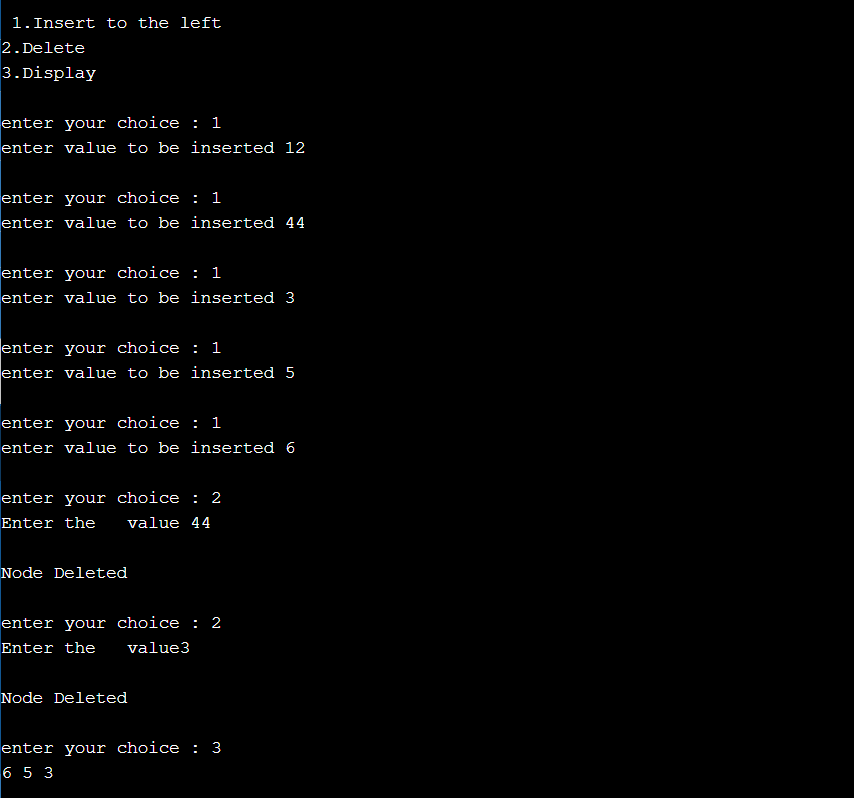
case 3: display();

break;

}

}

}



**10)Write a program**

**a) To construct a binary Search tree.**

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

**c) To display the elements in the tree.**

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

void insert();

void inorder(struct btnode \*t);

void create();

void search(struct btnode \*t);

void preorder(struct btnode \*t);

void postorder(struct btnode \*t);

int flag = 1;

void main()

{

int ch;

printf("\nOPERATIONS ---");

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

printf("2- Inorder Traversal\n");

printf("3 - Preorder Traversal\n");

printf("4- Postorder Traversal\n");

printf("5- Exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

inorder(root);

break;

case 3:

preorder(root);

break;

case 4:

postorder(root);

break;

case 5:

exit(0);

default :

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

break;

}

}

}

void insert()

{

create();

if (root == NULL)

root = temp;

else

search(root);

}

void create()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

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

temp->value = data;

temp->l = temp->r = NULL;

}

void search(struct btnode \*t)

{

if ((temp->value > t->value) && (t->r != NULL))

search(t->r);

else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp;

else if ((temp->value < t->value) && (t->l != NULL))

search(t->l);

else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp;

}

void inorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

if (t->l != NULL)

inorder(t->l);

printf("%d -> ", t->value);

if (t->r != NULL)

inorder(t->r);

}

void preorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

printf("%d -> ", t->value);

if (t->l != NULL)

preorder(t->l);

if (t->r != NULL)

preorder(t->r);

}

void postorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display ");

return;

}

if (t->l != NULL)

postorder(t->l);

if (t->r != NULL)

postorder(t->r);

printf("%d -> ", t->value);

}

