Data Structures

Lab File

SNEHA GUPTA

2021UCA1859

**Q1. Write a program to find the mean and the median of the numbers stored in an array.**

*Code:*

/\*

Input  : a[] = {1, 3, 4, 2, 6, 5, 8, 7}

Output : Mean = 4.5

         Median = 4.5

Sum of the elements is 1 + 3 + 4 + 2 + 6 +

5 + 8 + 7 = 36

Mean = 36/8 = 4.5

Since number of elements are even, median

is average of 4th and 5th largest elements.

which means (4 + 5)/2 = 4.5

5-2

7-3

9-4

Input  : a[] = {4, 4, 4, 4, 4,4,4,4,4}

Output : Mean = 4

         Median = 4

\*/

#include <iostream>

using namespace std;

int main()

{

    int n,i,a[50],b;

    float mean;

    cout<<"enter the number of elements you want to add in your array";

    cin>>n;

    for(i=0;i<n;i++)

    {

        cout<<"enter your elemnt:";

        cin>>b;

        a[i]=b;

        mean=mean+a[i];

    }

    mean=mean/2;

    if(n%2==0)

    {

        cout<<"even median: "<<(a[(n/2)-1]+a[n/2])/2 <<"\n";

    }

    else(n%2!=0);

    {

        cout<<n%2;

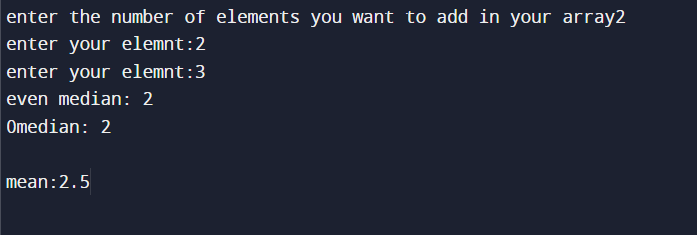
        cout<<"median: "<<a[(n-1)/2]<<"\n";

    }

    cout<<"\n"<<"mean:"<<mean;

}

Output:



**Q2.** Write a program to insert one element in an array and delete an element from an array.

*Code:*

/\*

Write a program to insert one element in an array and delete an element from an array

\*/

#include <iostream>

using namespace std;

int main()

{

    int n,i,a,b[50],c,d,e,f;

    cout<<"enter the number of elements you want to insert in your array (max:50):";

    cin>>n;

    for(i=0;i<n;i++)

    {

        cout<<"enter the element to insert in your array:";

        cin>>a;

        b[i]=a;

    }

    cout<<"\n";

    cout<<"enter 1 if you want to insert an element from your array\n enter2 if you want to delete an element from your array:";

    cin>>c;

    if(c==1)

    {

        cout<<"enter the element you want to insert:";

        cin>>d;

        cout<<"enter the index number where you want to insert your element:";

        cin>>e;

        for(i=n;i>=0;i--)

        {

            b[i]=b[i-1];

            if(i-1==e)

            {

                b[i-1]=d;

                break;

            }

        }

        cout<<"new array:";

        for(i=0;i<=n;i++)

        {

            cout<<b[i];

        }

    }

    if(c==2)

    {

        b[n]=0;

        cout<<"enter the element you want to delete:";

        cin>>f;

        for(i=0;i<n;i++)

        {

            if(b[i]==f)

            {

                b[i]=b[i+1];

                f=b[i+1];

                if(f==b[n])

                {

                    break;

                }

            }

        }

        cout<<"\n"<<"new array:";

        for(i=0;i<n-1;i++)

        {

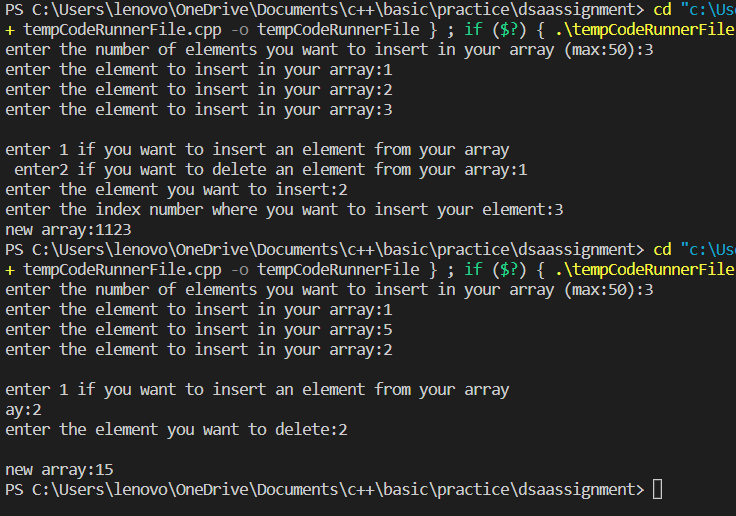
            cout<<b[i];

        }

    }

}

*Output:*



**Q3. Write a program to search for a number in an array.**

*Code:*

/\*

Write a program to search for a number in an array.

\*/

#include <iostream>

using namespace std;

int main()

{

    int n,i,a,b[50],c,d=0;

    cout<<"enter the number of elements you want to insert in your array (max:50):";

    cin>>n;

    for(i=0;i<n;i++)

    {

        cout<<"enter the element to insert in your array:";

        cin>>a;

        b[i]=a;

    }

    cout<<"enter the element you want to search for:";

    cin>>c;

    for(i=0;i<n;i++)

    {

        if(b[i]==c)

        {

            cout<<"yes this element exists in the array at "<<i<<" index number.";

            d=1;

        }

    }

    if(d==0)

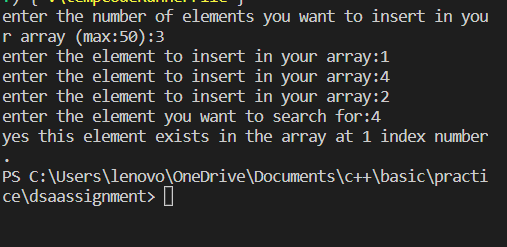
    {

        cout<<"this element does not exists";

    }

}

*Output:*



**Q4. Write a program to sort an array.**

*Code:*

/\*

Write a program to sort an array

\*/

#include <iostream>

using namespace std;

int main()

{

    int n,i,a,b[50],j,c;

    cout<<"enter the number of elements you want to insert in your array (max:50):";

    cin>>n;

    for(i=0;i<n;i++)

    {

        cout<<"enter the element to insert in your array:";

        cin>>a;

        b[i]=a;

    }

    cout<<"\nold array:";

    for(i=0;i<n;i++)

    {

        cout<<b[i];

    }

    for(i=0;i<n;i++)

    {

        for(j=0;j<n;j++)

        {

            if(b[j]>b[j+1])

            {

                c=b[j];

                b[j]=b[j+1];

                b[j+1]=c;

            }

        }

    }

    cout<<"\nnew array:";

    for(i=0;i<n;i++)

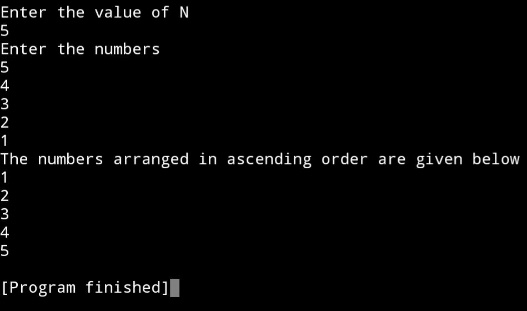
    {

        cout<<b[i];

    }

}

*Output:*



**Q5. Write a program to merge two sorted arrays.**

*Code:*

/\*

Write a program to merge two sorted arrays.

\*/

#include <iostream>

using namespace std;

int main()

{

    int n,i,b[50],a,c,e[50],j,f[50],d;

    cout<<"enter the number of elements you want to insert in your 1st array (max:50):";

    cin>>n;

    for(i=0;i<n;i++)

    {

        cout<<"enter the element to insert in your array:";

        cin>>a;

        b[i]=a;

    }

    cout<<"enter the number of elements you want to insert in your 2nd array (max:50):";

    cin>>c;

    for(i=0;i<c;i++)

    {

        cout<<"enter the element to insert in your array:";

        cin>>d;

        e[i]=d;

    }

    //couting 1st array before sorting

    cout<<"\n1st array:";

    for(i=0;i<n;i++)

    {

        cout<<b[i];

    }

    //coting 2nd array before sorting

    cout<<"\n2nd array:";

    for(i=0;i<c;i++)

    {

        cout<<e[i];

    }

    //merging

    i=0;

    while(i<n+c)

    {

        for(j=0;j<n;j++)

        {

            f[i]=b[j];

            i++;

        }

        for(j=0;j<c;j++)

        {

            f[i]=e[j];

            i++;

        }

    }

    //couting merged array

    cout<<"\nmerged array:";

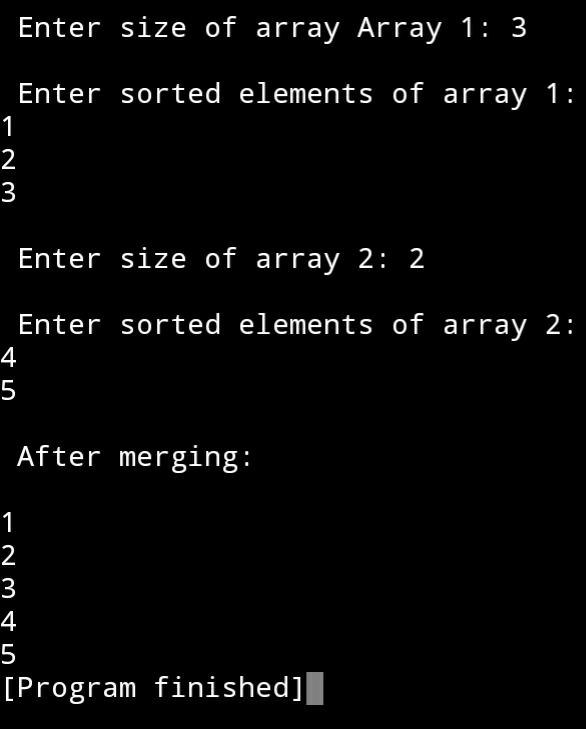
    for(i=0;i<n+c;i++)

    {

        cout<<f[i];

    }

}

*Output:*

**Q6. Write a program to store the marks obtained by 10 students in 5 courses in a two-dimensional array.**

*Code:*

/\*

Write a program to store the marks obtained by 10 students in 5

courses in a two-dimensional array.

\*/

#include <iostream>

using namespace std;

int main()

{

    int a=10,b=5,i,arr[10][5],j;

    for(i=0;i<10;i++)

    {

        for(j=0;j<5;j++)

        {

            cout<<"enter the marks of student "<<i+1<<":";

            cin>>arr[i][j];

        }

    }

    for(int i=0;i<10;i++)

    {

        cout<<"student"<<i+1<<": ";

        for(j=0;j<5;j++)

        {

            cout<<arr[i][j]<<" ";

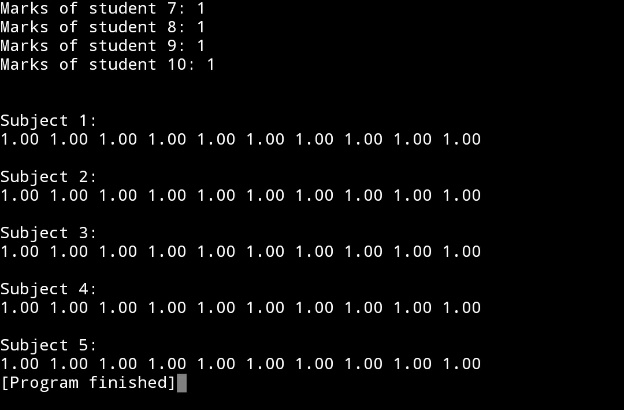
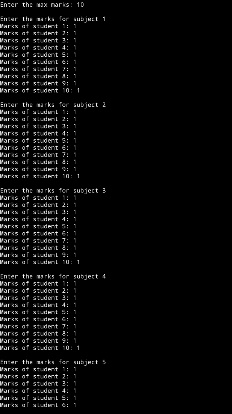
        }

        cout<<endl;

    }

}

*Output:*



**Q7. Write a program to implement a linked list.**

*Code:*

/\*

Write a program to implement a

linked list

\*/

#include <iostream>

using namespace std;

struct node

{

    int data;

    struct node \*link;

};

void print\_data(struct node \*head)

{

    if(head==NULL)

    {

        cout<<"linked list is empty";

    }

    struct node \*ptr=NULL;

    ptr=head;

    while(ptr!=NULL)

    {

        cout<<ptr->data<<" ";

        ptr=ptr->link;

    }

}

int main()

{

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

    head->data=45;

    head->link=NULL;

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

    current->data=98;

    current->link =NULL;

    head->link=current;

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

    current->data=3;

    current->link=NULL;

    head->link->link=current;

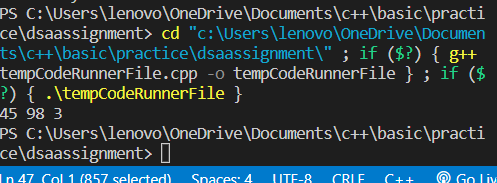
    print\_data(head);

    return 0;

}

}

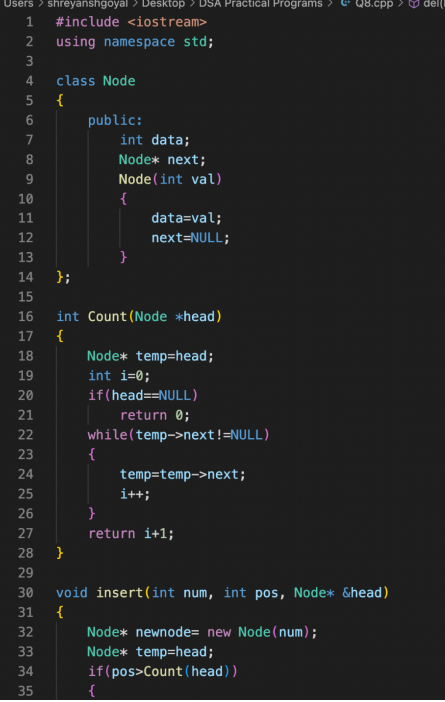
*Output:*

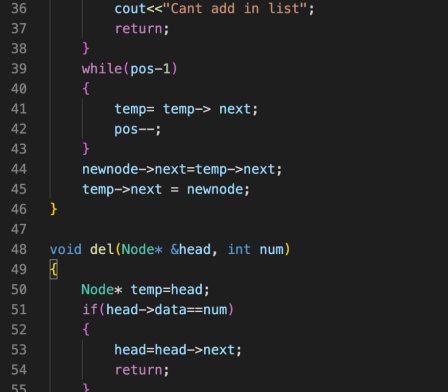


**Q8. Write a program to insert a node in a linked list and delete a node from a linked**

**list.**

*Code:*







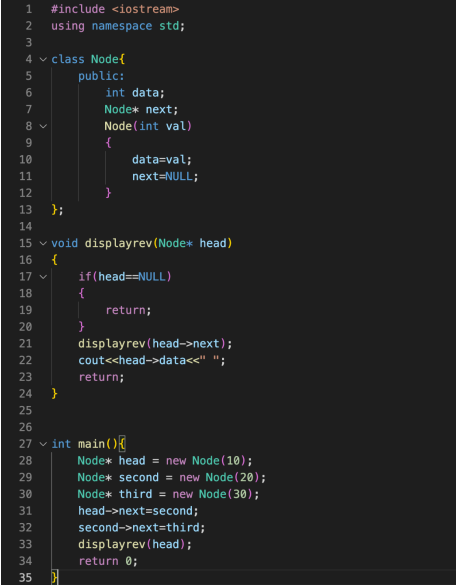
*Output:*



**Q9. Write a program to print the elements of a linked list in reverse order without**

**disturbing the linked list.**

*Code:*



*Output:*



**Q10. Write a program to reverse a linked list.**

*Code:*



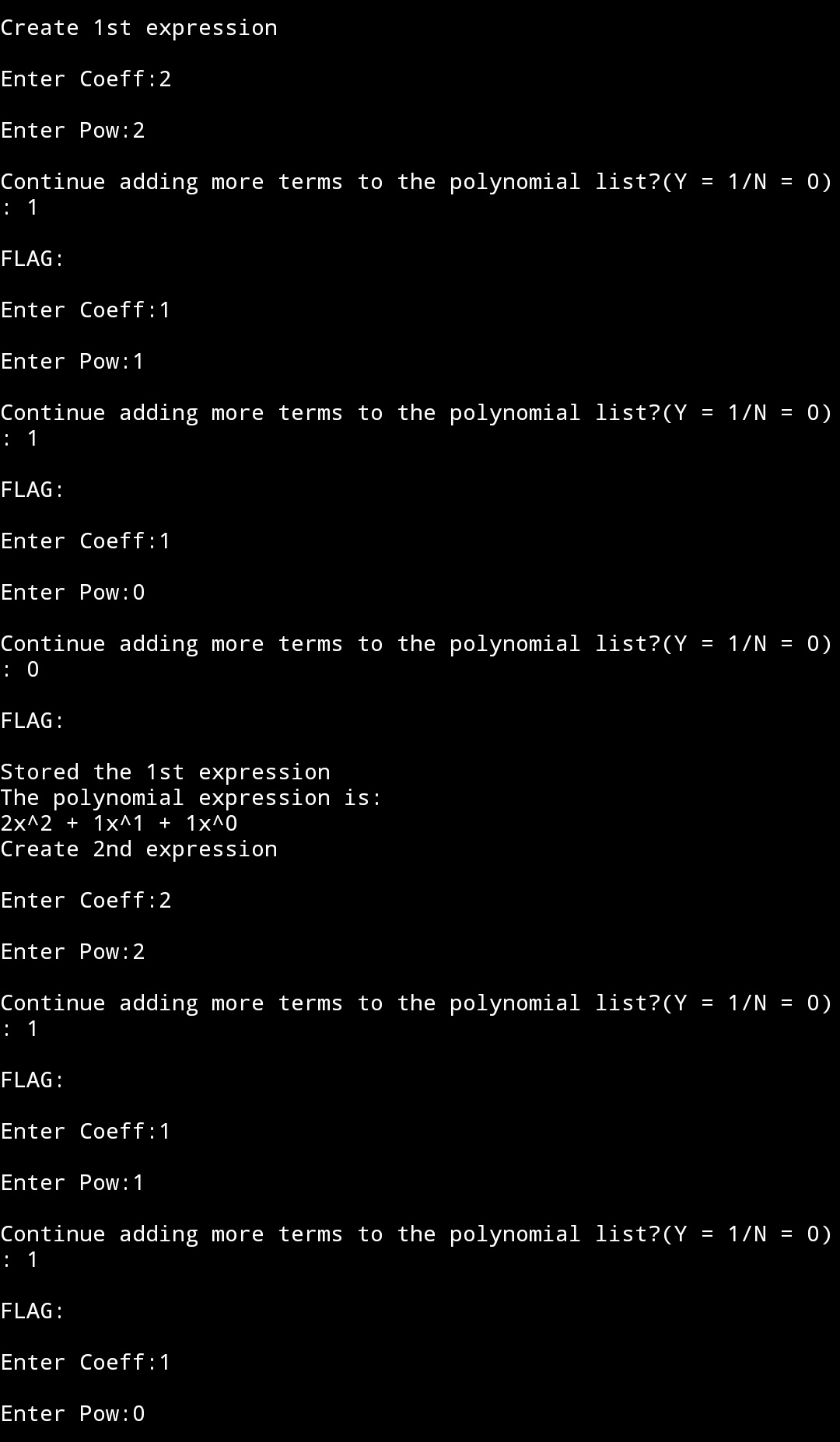
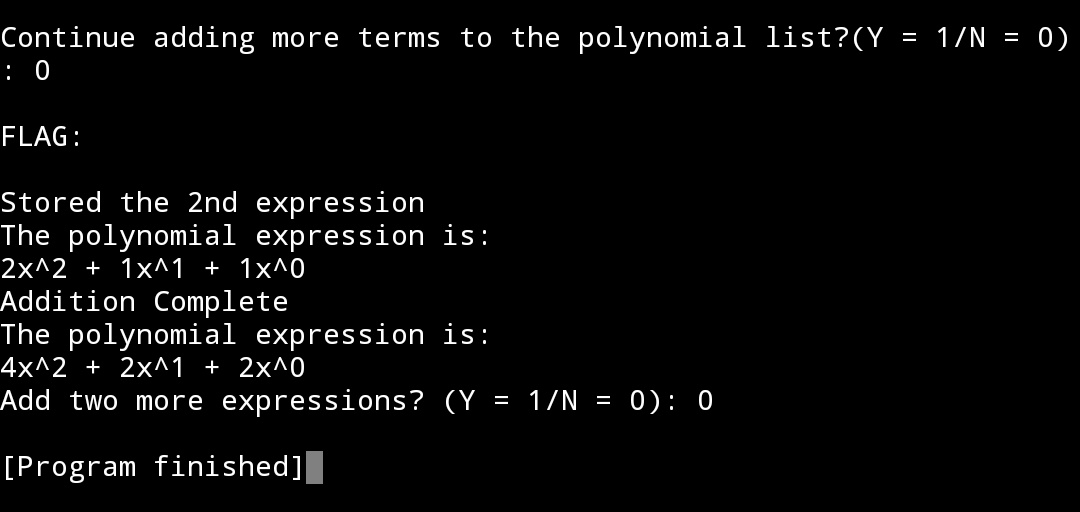
*Output:*

**

**Q11. Write a program to add two polynomials using linked lists.**

*Code:*

*Output:*



**Q12. Write a program to implement a doubly-linked list.**

*Code:*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

struct node {

int data;

int key;

struct node \*next;

struct node \*prev;

};

struct node \*head = NULL;

struct node \*last = NULL;

struct node \*current = NULL;

bool isEmpty() {

return head == NULL;

}

int length() {

int length = 0;

struct node \*current;

for(current = head; current != NULL; current = current->next){

length++;

}

return length;

}

void displayForward() {

struct node \*ptr = head;

printf("\n[ ");

while(ptr != NULL) {

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

ptr = ptr->next;

}

printf(" ]");

}

void displayBackward() {

struct node \*ptr = last;

printf("\n[ ");

while(ptr != NULL) {

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

ptr = ptr ->prev;

}

}

void insertFirst(int key, int data) {

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

link->key = key;

link->data = data;

if(isEmpty()) {

last = link;

} else {

head->prev = link;

}

link->next = head;

head = link;

}

void insertLast(int key, int data) {

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

link->key = key;

link->data = data;

if(isEmpty()) {

last = link;

} else {

last->next = link;

link->prev = last;

}

last = link;

}

struct node\* deleteFirst() {

struct node \*tempLink = head;

if(head->next == NULL){

last = NULL;

} else {

head->next->prev = NULL;

}

head = head->next;

return tempLink;

}

struct node\* deleteLast() {

struct node \*tempLink = last;

if(head->next == NULL) {

head = NULL;

} else {

last->prev->next = NULL;

}

last = last->prev;

return tempLink;

}

struct node\* delete(int key) {

struct node\* current = head;

struct node\* previous = NULL;

if(head == NULL) {

return NULL;

}

while(current->key != key) {

if(current->next == NULL) {

return NULL;

} else {

previous = current;

current = current->next;

}

}

if(current == head) {

head = head->next;

} else {

current->prev->next = current->next;

}

if(current == last) {

last = current->prev;

} else {

current->next->prev = current->prev;

}

return current;

}

bool insertAfter(int key, int newKey, int data) {

struct node \*current = head;

if(head == NULL) {

return false;

}

while(current->key != key) {

if(current->next == NULL) {

return false;

} else {

current = current->next;

}

}

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

newLink->key = newKey;

newLink->data = data;

if(current == last) {

newLink->next = NULL;

last = newLink;

} else {

newLink->next = current->next;

current->next->prev = newLink;

}

newLink->prev = current;

current->next = newLink;

return true;

}

void main() {

insertFirst(1,10);

insertFirst(2,20);

insertFirst(3,30);

insertFirst(4,1);

insertFirst(5,40);

insertFirst(6,56);

printf("\nList (First to Last): ");

displayForward();

printf("\n");

printf("\nList (Last to first): ");

displayBackward();

printf("\nList , after deleting first record: ");

deleteFirst();

displayForward();

printf("\nList , after deleting last record: ");

deleteLast();

displayForward();

printf("\nList , insert after key(4) : ");

insertAfter(4,7, 13);

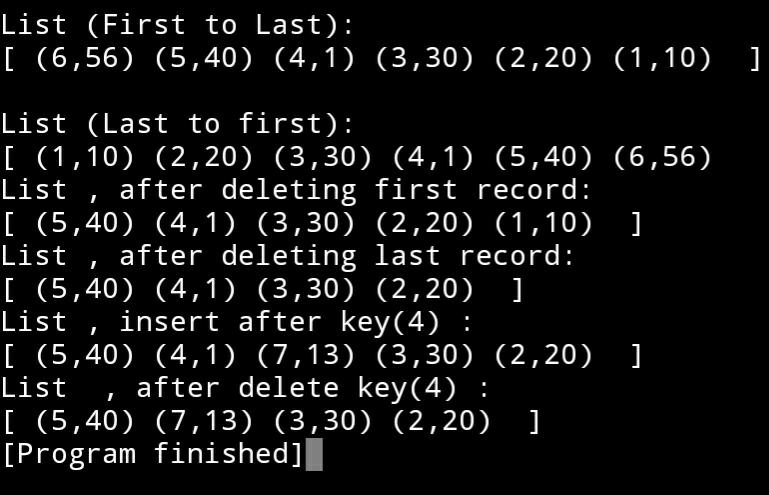
displayForward();

printf("\nList , after delete key(4) : ");

delete(4);

displayForward();

}

*Output:*

**Q13. Write a program to implement a stack using an array.**

*Code:*

#include <stdio.h>

#define SIZE 3

int arr[SIZE], top = -1;

void peek() {

top == -1 ? printf("Stack is empty!\n") : printf("%d\n", arr[top]);

}

void push(int val) {

if (top == SIZE - 1) {

printf("Overflow!\n");

} else {

arr[++top] = val;

printf("Successfully pushed %d!\n", val);

}

}

int pop() {

if (top == -1) {

printf("Underflow!\n");

} else {

int temp = arr[top--];

return temp;

}

}

int main() {

peek();

push(3);

push(4);

push(5);

push(6);

peek();

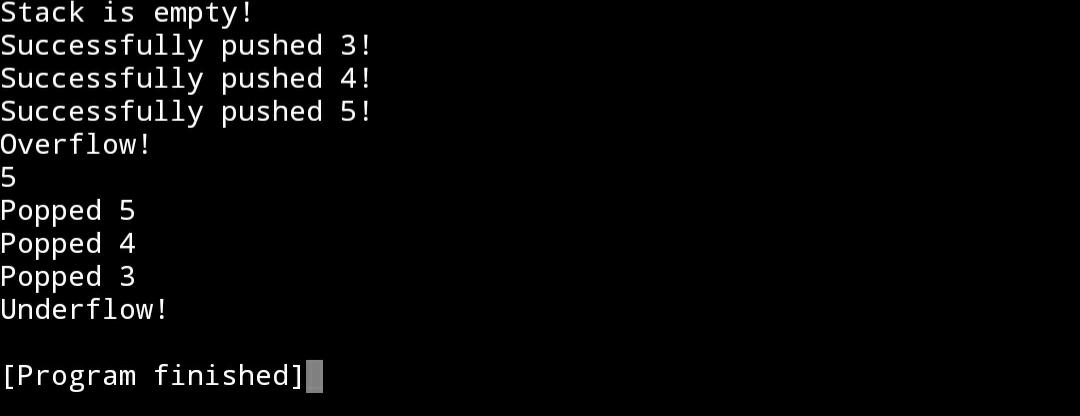
printf("Popped %d", pop());

printf("\nPopped %d", pop());

printf("\nPopped %d\n", pop());

pop();

}

*Output:*

**Q14. Write a program to implement a stack using a linked list.**

*Code:*

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

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

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack\_count();

void create();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Push");

printf("\n 2 - Pop");

printf("\n 3 - Top");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Dipslay");

printf("\n 7 - Stack Count");

printf("\n 8 - Destroy stack");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

push(no);

break;

case 2:

pop();

break;

case 3:

if (top == NULL)

printf("No elements in stack");

else

{

e = topelement();

printf("\n Top element : %d", e);

}

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

stack\_count();

break;

case 8:

destroy();

break;

default :

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

break;

}

}

}

void create()

{

top = NULL;

}

void stack\_count()

{

printf("\n No. of elements in stack : %d", count);

}

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++;

}

void display()

{

top1 = top;

if (top1 == NULL)

{

printf("Stack is empty");

return;

}

while (top1 != NULL)

{

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

top1 = top1->ptr;

}

}

void pop()

{

top1 = top;

if (top1 == NULL)

{

printf("\n Error : Trying to pop from empty stack");

return;

}

else

top1 = top1->ptr;

printf("\n Popped value : %d", top->info);

free(top);

top = top1;

count--;

}

int topelement()

{

return(top->info);

}

void empty()

{

if (top == NULL)

printf("\n Stack is empty");

else

printf("\n Stack is not empty with %d elements", count);

}

void destroy()

{

top1 = top;

while (top1 != NULL)

{

top1 = top->ptr;

free(top);

top = top1;

top1 = top1->ptr;

}

free(top1);

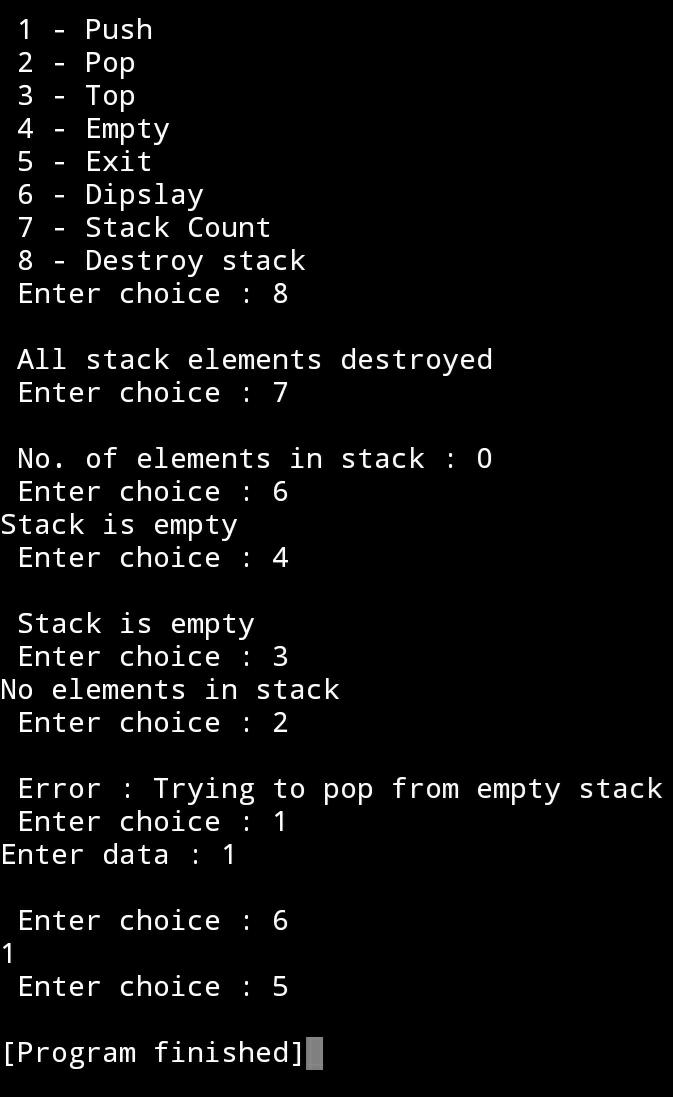
top = NULL;

printf("\n All stack elements destroyed");

count = 0;

}

*Output:*



**Q15. Write a program to implement a queue using an array.**

*Code:*

#include <stdio.h>

#define MAX 10

int queue[MAX];

int f = -1, r = -1, size = -1;

void enqueue(int val) {

if(size < MAX) {

if (size < 0) {

queue[0] = val;

f++; r++;

size = 1;

} else if (r == MAX-1) {

queue[0] = val;

r = 0;

size++;

} else {

queue[++r] = val;

size++;

}

} else {

printf("Queue is full\n");

}

}

int dequeue() {

if (size < 0) {

printf("Queue is empty\n");

} else {

size--;

f++;

}

}

void display()

{

int i;

if( r >= f ) {

for (i = f; i <= r; i++) {

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

}

} else {

for (i = f; i < MAX; i++) {

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

}

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

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

}

}

}

int main()

{

enqueue(24);

enqueue(9);

enqueue(22);

enqueue(93);

display();

dequeue();

printf("\nAfter dequeue\n");

display();

enqueue(8);

enqueue(63);

enqueue(57);

enqueue(900);

dequeue();

enqueue(84);

enqueue(73);

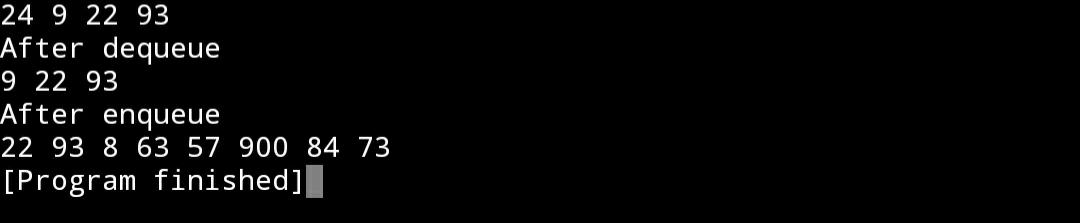
printf("\nAfter enqueue\n");

display();

return 0;

}

*Output:*

**

**Q16. Write a program to implement a queue using a linked list.**

*Code:*

#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;

}

}

}

void create()

{

front = rear = NULL;

}

void queuesize()

{

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

}

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++;

}

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);

}

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--;

}

int frontelement()

{

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

return(front->info);

else

return 0;

}

void empty()

{

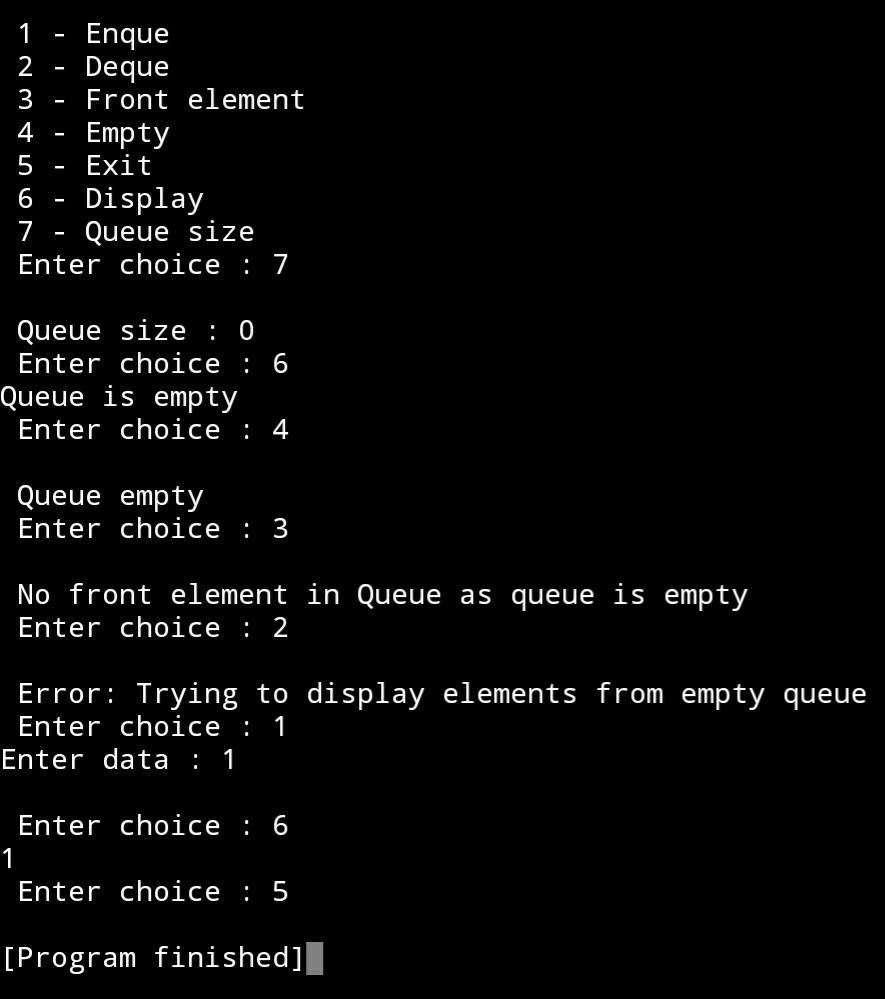
if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}

*Output:*

**Q17. Write a program to implement a circular queue using an array.**

*Code:*

#include <stdio.h>

#define SIZE 5

int cirqueue[SIZE], f = -1, r = -1;

int isFull() {

if ((f == r + 1) || (f == 0 && r == SIZE - 1)) { return 1; }

return 0;

}

int isEmpty() {

if (f == -1) { return 1; }

return 0;

}

void enqueue(int val) {

if (isFull()) { printf("\nQueue is full!\n"); }

else {

if (f == -1) { f = 0; }

r = (r + 1) % SIZE;

cirqueue[r] = val;

printf("\nInserted: %d", val);

}

}

int dequeue() {

int val;

if (isEmpty()) {

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

} else {

val = cirqueue[f];

if (f == r) { f = r = -1;}

else {

f = (f + 1) % SIZE;

} printf("\nDeleted element: %d \n", val);

return val;

}

}

void display() {

int i;

if (isEmpty())

printf("\nEmpty Queue\n");

else {

printf("\nFront: %d ", f);

printf("\nCircular Queue: ");

for (i = f; i != r; i = (i + 1) % SIZE) {

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

}

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

printf("\nrear: %d \n", r);

}

}

int main() {

dequeue();

enqueue(1);

enqueue(2);

enqueue(3);

enqueue(4);

enqueue(5);

enqueue(6);

display();

dequeue();

display();

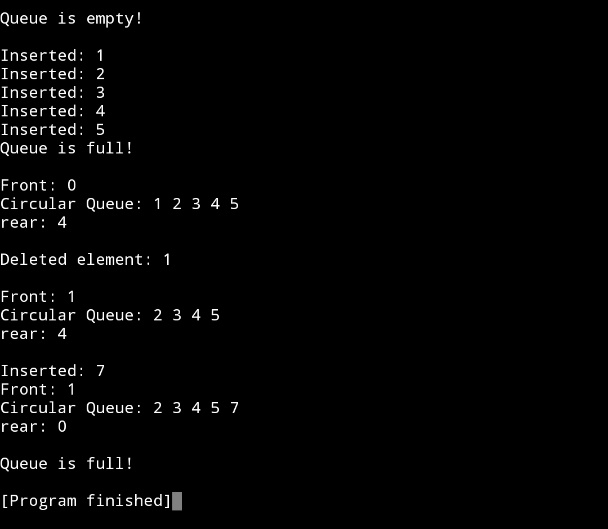
enqueue(7);

display();

enqueue(8);

return 0;

}

*Output:*

**Q18. Write a program to implement a priority queue using a linked list.**

*Code:*

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int val;

int priority;

struct node\* next;

} Node;

Node\* initNode(int v, int p) {

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

temp -> val = v;

temp -> priority = p;

temp -> next = NULL;

return temp;

}

int peek(Node\*\* head) {

return (\*head) -> val;

}

void pop(Node\*\* head) {

Node\* temp = \*head;

(\*head) = (\*head) -> next;

free(temp);

}

void push(Node\*\* head, int v, int p) {

Node\* first = (\*head);

Node\* temp = initNode(v, p);

if ((\*head) -> priority > p) {

temp -> next = \*head;

(\*head) = temp;

} else {

while (first -> next != NULL && first -> next -> priority < p) {

first = first -> next;

}

temp -> next = first -> next;

first -> next = temp;

} printf("Successfuly pushed %d!\n", v);

}

int isEmpty(Node\*\* head) {

return (\*head) == NULL;

}

int main() {

Node\* pq = initNode(7, 1);

printf("Created linked list with value 7.\n");

push(&pq, 1, 2);

push(&pq, 3, 3);

push(&pq, 2, 0);

while (!isEmpty(&pq)) {

printf("Popped %d\n", peek(&pq));

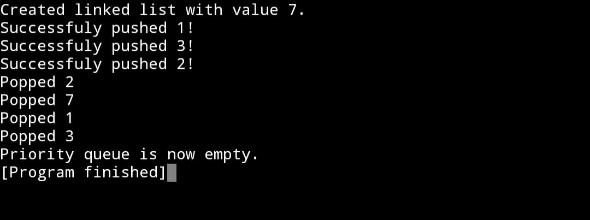
pop(&pq);

}

printf("Priority queue is now empty.");

return 0;

}

*Output:*

**Q19. Write a program to implement a double-ended queue using a linked list.**

*Code:*

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node \*prev, \*next;

} Node;

Node \*head = NULL, \*end = NULL;

Node\* initNode(int data) {

Node \*new = (Node \*) malloc(sizeof (Node));

new -> data = data;

new -> next = new -> prev = NULL;

return new;

}

void makeEnds() {

head = initNode(0);

end = initNode(0);

head -> next = end;

end -> prev = head;

}

void enqueueFront(int data) {

Node \*new, \*temp;

new = initNode(data);

temp = head -> next;

head -> next = new;

new -> prev = head;

new -> next = temp;

temp -> prev = new;

}

void enqueueRear(int data) {

Node \*new, \*temp;

new = initNode(data);

temp = end -> prev;

end -> prev = new;

new -> next = end;

new -> prev = temp;

temp -> next = new;

}

void dequeueFront() {

Node \*temp;

if (head -> next == end) {

printf("Queue is empty\n");

} else {

temp = head -> next;

head -> next = temp -> next;

temp -> next -> prev = head;

free(temp);

} return;

}

void dequeueRear() {

Node \*temp;

if (end -> prev == head) {

printf("Queue is empty\n");

} else {

temp = end -> prev;

end -> prev = temp -> prev;

temp -> prev -> next = end;

free(temp);

} return;

}

void display() {

Node \*temp;

if (head -> next == end) {

printf("Queue is empty\n");

return;

}

temp = head -> next;

while (temp != end) {

printf("%-3d", temp -> data);

temp = temp -> next;

}

printf("\n");

}

int main() {

makeEnds();

enqueueFront(23);

enqueueRear(29);

enqueueRear(30);

enqueueFront(40);

display();

dequeueFront();

dequeueRear();

display();

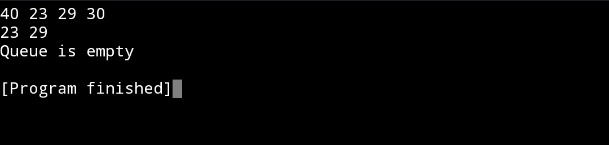
dequeueFront();

dequeueRear();

display();

return 0;

}

*Output:*

**Q20. Write a program to construct a binary tree and display its preorder, inorder and postorder traversals.**

*Code:*

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node\* left;

struct node\* right;

} Node;

Node\* newNode(int data) {

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

node -> data = data;

node -> left = NULL;

node -> right = NULL;

return node;

}

void Postorder(Node\* node) {

if (node == NULL) { return; }

Postorder(node -> left);

Postorder(node -> right);

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

}

void Inorder(Node\* node)

{

if (node == NULL) { return; }

Inorder(node -> left);

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

Inorder(node -> right);

}

void Preorder(Node\* node)

{

if (node == NULL) { return; }

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

Preorder(node -> left);

Preorder(node -> right);

}

int main()

{

Node\* base = newNode(1);

base -> left = newNode(2);

base -> right = newNode(3);

base -> left -> left = newNode(4);

base -> left -> right = newNode(5);

base -> right -> left = newNode(6);

base -> right -> right = newNode(7);

printf("\nPreorder traversal of binary tree:\n");

Preorder(base);

printf("\nInorder traversal of binary tree:\n");

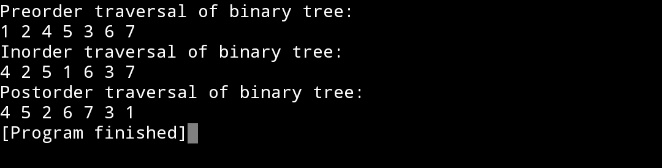
Inorder(base);

printf("\nPostorder traversal of binary tree:\n");

Postorder(base);

return 0;

}

*Output:*

**Q21. Write a program to construct a binary search tree.**

*Code:*

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

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

void delete1();

void insert();

void delete();

void inorder(struct btnode \*t);

void create();

void search(struct btnode \*t);

void preorder(struct btnode \*t);

void postorder(struct btnode \*t);

void search1(struct btnode \*t,int data);

int smallest(struct btnode \*t);

int largest(struct btnode \*t);

int flag = 1;

void main()

{

int ch;

printf("\nOPERATIONS ---");

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

printf("2 - Delete an element from the tree\n");

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

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

printf("5 - Postorder Traversal\n");

printf("6 - Exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

inorder(root);

break;

case 4:

preorder(root);

break;

case 5:

postorder(root);

break;

case 6:

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 delete()

{

int data;

if (root == NULL)

{

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

return;

}

printf("Enter the data to be deleted : ");

scanf("%d", &data);

t1 = root;

t2 = root;

search1(root, data);

}

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);

}

void search1(struct btnode \*t, int data)

{

if ((data>t->value))

{

t1 = t;

search1(t->r, data);

}

else if ((data < t->value))

{

t1 = t;

search1(t->l, data);

}

else if ((data==t->value))

{

delete1(t);

}

}

void delete1(struct btnode \*t)

{

int k;

if ((t->l == NULL) && (t->r == NULL))

{

if (t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

return;

}

else if ((t->r == NULL))

{

if (t1 == t)

{

root = t->l;

t1 = root;

}

else if (t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

return;

}

else if (t->l == NULL)

{

if (t1 == t)

{

root = t->r;

t1 = root;

}

else if (t1->r == t)

t1->r = t->r;

else

t1->l = t->r;

t == NULL;

free(t);

return;

}

else if ((t->l != NULL) && (t->r != NULL))

{

t2 = root;

if (t->r != NULL)

{

k = smallest(t->r);

flag = 1;

}

else

{

k =largest(t->l);

flag = 2;

}

search1(root, k);

t->value = k;

}

}

int smallest(struct btnode \*t)

{

t2 = t;

if (t->l != NULL)

{

t2 = t;

return(smallest(t->l));

}

else

return (t->value);

}

int largest(struct btnode \*t)

{

if (t->r != NULL)

{

t2 = t;

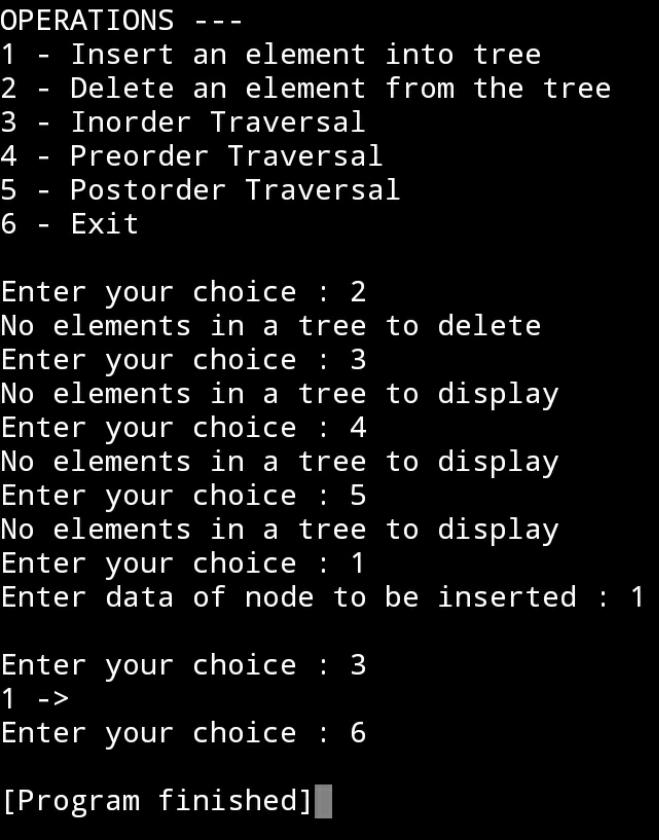
return(largest(t->r));

}

else

return(t->value);

}

*Output:*