1. **Write a program for Linear search methods.**

#include <stdio.h>

#include<conio.h>

int main()

{

int array[100], search, c, n;

printf("Enter the number of elements in array\n");

scanf("%d",&n);

printf("Enter %d integer(s)\n", n);

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

scanf("%d", &array[c]);

printf("Enter the number to search\n");

scanf("%d", &search);

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

{

if (array[c] == search) {

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

if (c == n)

printf("%d is not present in array.\n", search);

return 0;

}

1. **Write a program for Binary search methods**.

#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int n, i, arr[50], search, first, last, middle;

printf("Enter total number of elements :");

scanf("%d",&n);

printf("Enter %d number :", n);

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

{

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

}

printf("Enter a number to find :");

scanf("%d", &search);

first = 0;

last = n-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < search)

{

first = middle + 1;

}

else if(arr[middle] == search)

{

printf("%d found at location %d\n", search, middle+1);

break;

}

else

{

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

printf("Not found! %d is not present in the list.",search);

}

getch();

}

**3 Write a program for insertion sort, selection sort and bubble sort.**

**/\* insertion sort \*/**

#include <stdio.h>

#include<conio.h>

int main()

{

int n, array[1000], c, d, t;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

}

for (c = 1 ; c <= n - 1; c++) {

d = c;

while ( d > 0 && array[d] < array[d-1]) {

t = array[d];

array[d] = array[d-1];

array[d-1] = t;

d--;

}

}

printf("Sorted list in ascending order:\n");

for (c = 0; c <= n - 1; c++) {

printf("%d\n", array[c]);

}

return 0;

}

**/\* selection sort\*/**

#include <stdio.h>

#include<conio.h>

int main()

{

int array[100], n, c, d, position, swap;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

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

{

position = c;

for ( d = c + 1 ; d < n ; d++ )

{

if ( array[position] > array[d] )

position = d;

}

if ( position != c )

{

swap = array[c];

array[c] = array[position];

array[position] = swap;

}

}

printf("Sorted list in ascending order:\n");

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

printf("%d\n", array[c]);

return 0;

}

**/\* Bubble sort code \*/**

#include <stdio.h>

#include<conio.h>

int main()

{

int array[100], n, c, d, swap;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

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

{

for (d = 0 ; d < n - c - 1; d++)

{

if (array[d] > array[d+1]) /\* For decreasing order use < \*/

{

swap = array[d];

array[d] = array[d+1];

array[d+1] = swap;

}

}

}

printf("Sorted list in ascending order:\n");

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

printf("%d\n", array[c]);

return 0;

}

**4. Write a program to implement Stack and its operation.**

#include <stdio.h>

#include<conio.h>

#define MAXSIZE 5

struct stack

{

int stk[MAXSIZE];

int top;

};

typedef struct stack STACK;

STACK s;

void push(void);

int pop(void);

void display(void);

void main ()

{

int choice;

int option = 1;

s.top = -1;

printf ("STACK OPERATION\n");

while (option)

{

printf (" 1 --> PUSH \n");

printf (" 2 --> POP \n");

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

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

printf ("Enter your choice\n");

scanf ("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

return;

}

fflush (stdin);

printf ("Do you want to continue(Type 0 or 1)?\n");

scanf ("%d", &option);

}

}

void push ()

{

int num;

if (s.top == (MAXSIZE - 1))

{

printf ("Stack is Full\n");

return;

}

else

{

printf ("Enter the element to be pushed\n");

scanf ("%d", &num);

s.top = s.top + 1;

s.stk[s.top] = num;

}

return;

}

int pop ()

{

int num;

if (s.top == - 1)

{

printf ("Stack is Empty\n");

return (s.top);

}

else

{

num = s.stk[s.top];

printf ("poped element is = %dn", s.stk[s.top]);

s.top = s.top - 1;

}

return(num);

}

void display ()

{

int i;

if (s.top == -1)

{

printf ("Stack is empty\n");

return;

}

else

{

printf ("\n The status of the stack is \n");

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

{

printf ("%d\n", s.stk[i]);

}

}

printf ("\n");

}

**5. Write a program for quick sort.**

#include<stdio.h>

#include<conio.h>

void quicksort(int[],int,int);

int partition(int [],int,int);

void main()

{

int a[20],i,n;

clrscr();

printf("Enter the size of array");

scanf("%d",&n);

printf("Enter the elements in the array");

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

{

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

}

quicksort(a,0,n-1);

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

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

getch();

}

void quicksort(int a[],int lb,int ub)

{

int mid;

if(lb<ub)

{

mid=partition(a,lb,ub);

quicksort(a,lb,mid-1);

quicksort(a,mid+1,ub);

}

}

int partition(int a[],int lb,int ub)

{

int i,p,q,t;

p=lb+1;

q=ub;

i=a[lb];

while(q>=p)

{

while(a[p]<i)

p++;

while(a[q]>i)

q--;

if(q>p)

{

t=a[p];

a[p]=a[q];

a[q]=t;

}

}

t=a[lb];

a[lb]=a[q];

a[q]=t;

return q;

}

**6. Write a program for merge sort.**

#include<stdio.h>

#include<conio.h>

void mergesort(int a[],int,int);

void merge(int [],int,int,int);

void main()

{

int a[20],i,n;

clrscr();

printf("Enter the number of elements");

scanf("%d",&n);

printf("Enter the elements");

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

{

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

}

mergesort(a,0,n-1);

printf("Data After Merge Sort");

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

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

getch();

}

void mergesort(int a[],int lb,int ub)

{

int mid;

if(lb<ub)

{

mid=(lb+ub)/2;

mergesort(a,lb,mid);

mergesort(a,mid+1,ub);

merge(a,lb,mid+1,ub);

}

}

void merge(int a[],int lb,int mid,int ub)

{

int k,p1,p2,p3,b[20];

p1=lb;

p3=lb;

p2=mid;

while((p1<mid)&&(p2<=ub))

{

if(a[p1]<=a[p2])

b[p3++]=a[p1++];

else

b[p3++]=a[p2++];

}

while(p1<mid)

{

b[p3++]=a[p1++];

}

while(p2<=ub)

{

b[p3++]=a[p2++];

}

for(k=lb;k<p3;k++)

{

a[k]=b[k];

}

}

**7. Write a program to implement Queue and its operation**.

#include<stdio.h>

#include<conio.h>

#include<process.h>

void insert();

void delet();

void display();

int front,rear;

int q[5];

void main()

{

int choice;

char ch;

front=-1;

rear=-1;

clrscr();

do

{

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

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

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

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

printf("\nEnter your choice");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delet();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("\nBAD CHOICE");

}

printf("\ndo you want to continue y/n");

ch=getche();

}

while(ch=='y'||'Y');

}

void insert()

{

int item;

if(((front==1)&&(rear==5))||(front==rear+1))

{

printf("QUEUE IS FULL");

}

else

{

printf("Enter the element");

scanf("%d",&item);

if(front==-1)

{

front=1;

rear=1;

}

else if(rear==5)

{

rear=0;

}

else

{

rear=rear+1;

}

q[rear]=item;

}

}

void delet()

{

int item;

if(front==-1)

{

printf("QUEUE IS EMPTY");

}

else

{

item=q[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else if(front==5)

{

front=0;

}

else

front=front+1;

printf("%d is deleted",item);

}

}

void display()

{

int i;

if(front==-1)

printf("QUEUE IS EMPTY");

else

{

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

{

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

}

}

}

**8.Write a program to implement Circular Queue and its operation.**

# include<stdio.h>

# define MAX 5

int cqueue\_arr[MAX];

int front = -1;

int rear = -1;

void main()

{

int choice;

while(1)

{

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

insert();

break;

case 2 :

del();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice\n");

}

}

}

int insert()

{

int added\_item;

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

{

printf("Queue Overflow \n");

return;

}

if (front == -1)

{

front = 0;

rear = 0;

}

else

if(rear == MAX-1)

rear = 0;

else

rear = rear+1;

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

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

cqueue\_arr[rear] = added\_item ;

}

del()

{

if (front == -1)

{

printf("Queue Underflow\n");

return ;

}

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

if(front == rear)

{

front = -1;

rear=-1;

}

else

{

if(front == MAX-1)

front = 0;

else

front = front+1;

}

}

display()

{

int front\_pos = front,rear\_pos = rear;

if(front == -1)

{

printf("Queue is empty\n");

return;

}

printf("Queue elements :\n");

if( front\_pos <= rear\_pos )

while(front\_pos <= rear\_pos)

{

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

front\_pos++;

}

else

{

while(front\_pos <= MAX-1)

{

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

front\_pos++;

}

front\_pos = 0;

while(front\_pos <= rear\_pos)

{

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

front\_pos++;

}

}

printf("\n");

}

**9. Write a program to implement singly linked list for the following operations: Create, Display, searching, traversing and deletion.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

}\*head;

void append(int num)

{

struct node \*temp,\*right;

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

temp->data=num;

right=(struct node \*)head;

while(right->next != NULL)

right=right->next;

right->next =temp;

right=temp;

right->next=NULL;

}

void add( int num )

{

struct node \*temp;

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

temp->data=num;

if (head== NULL)

{

head=temp;

head->next=NULL;

}

else

{

temp->next=head;

head=temp;

}

}

void addafter(int num, int loc)

{

int i;

struct node \*temp,\*left,\*right;

right=head;

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

{

left=right;

right=right->next;

}

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

temp->data=num;

left->next=temp;

left=temp;

left->next=right;

return;

}

void insert(int num)

{

int c=0;

struct node \*temp;

temp=head;

if(temp==NULL)

{

add(num);

}

else

{

while(temp!=NULL)

{

if(temp->data<num)

c++;

temp=temp->next;

}

if(c==0)

add(num);

else if(c<count())

addafter(num,++c);

else

append(num);

}

}

int delete(int num)

{

struct node \*temp, \*prev;

temp=head;

while(temp!=NULL)

{

if(temp->data==num)

{

if(temp==head)

{

head=temp->next;

free(temp);

return 1;

}

else

{

prev->next=temp->next;

free(temp);

return 1;

}

}

else

{

prev=temp;

temp= temp->next;

}

}

return 0;

}

void display(struct node \*r)

{

r=head;

if(r==NULL)

{

return;

}

while(r!=NULL)

{

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

r=r->next;

}

printf("\n");

}

int count()

{

struct node \*n;

int c=0;

n=head;

while(n!=NULL)

{

n=n->next;

c++;

}

return c;

}

int main()

{

int i,num;

struct node \*n;

head=NULL;

while(1)

{

printf("\nList Operations\n");

printf("===============\n");

printf("1.Insert\n");

printf("2.Display\n");

printf("3.Size\n");

printf("4.Delete\n");

printf("5.Exit\n");

printf("Enter your choice : ");

if(scanf("%d",&i)<=0){

printf("Enter only an Integer\n");

exit(0);

} else {

switch(i)

{

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

scanf("%d",&num);

insert(num);

break;

case 2: if(head==NULL)

{

printf("List is Empty\n");

}

else

{

printf("Element(s) in the list are : ");

}

display(n);

break;

case 3: printf("Size of the list is %d\n",count());

break;

case 4: if(head==NULL)

printf("List is Empty\n");

else{

printf("Enter the number to delete : ");

scanf("%d",&num);

if(delete(num))

printf("%d deleted successfully\n",num);

else

printf("%d not found in the list\n",num);

}

break;

case 5: return 0;

default: printf("Invalid option\n");

}

}

}

return 0;

}

**10. Write a program to implement doubly linked list for the following operations: Create, Display, inserting, counting, searching, traversing and deletion.**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node

{

struct node \*previous;

int data;

struct node \*next;

}\*head, \*last;

void insert\_begning(int value)

{

struct node \*var,\*temp;

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

var->data=value;

if(head==NULL)

{

head=var;

head->previous=NULL;

head->next=NULL;

last=head;

}

else

{

temp=var;

temp->previous=NULL;

temp->next=head;

head->previous=temp;

head=temp;

}

}

void insert\_end(int value)

{

struct node \*var,\*temp;

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

var->data=value;

if(head==NULL)

{

head=var;

head->previous=NULL;

head->next=NULL;

last=head;

}

else

{

last=head;

while(last!=NULL)

{

temp=last;

last=last->next;

}

last=var;

temp->next=last;

last->previous=temp;

last->next=NULL;

}

}

int insert\_after(int value, int loc)

{

struct node \*temp,\*var,\*temp1;

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

var->data=value;

if(head==NULL)

{

head=var;

head->previous=NULL;

head->next=NULL;

}

else

{

temp=head;

while(temp!=NULL && temp->data!=loc)

{

temp=temp->next;

}

if(temp==NULL)

{

printf("\n%d is not present in list ",loc);

}

else

{

temp1=temp->next;

temp->next=var;

var->previous=temp;

var->next=temp1;

temp1->previous=var;

}

}

last=head;

while(last->next!=NULL)

{

last=last->next;

}

}

int delete\_from\_end()

{

struct node \*temp;

temp=last;

if(temp->previous==NULL)

{

free(temp);

head=NULL;

last=NULL;

return0;

}

printf("\nData deleted from list is %d \n",last->data);

last=temp->previous;

last->next=NULL;

free(temp);

return0;

}

int delete\_from\_middle(int value)

{

struct node \*temp,\*var,\*t, \*temp1;

temp=head;

while(temp!=NULL)

{

if(temp->data == value)

{

if(temp->previous==NULL)

{

free(temp);

head=NULL;

last=NULL;

return0;

}

else

{

var->next=temp1;

temp1->previous=var;

free(temp);

return0;

}

}

else

{

var=temp;

temp=temp->next;

temp1=temp->next;

}

}

printf("data deleted from list is %d",value);

}

void display()

{

struct node \*temp;

temp=head;

if(temp==NULL)

{

printf("List is Empty");

}

while(temp!=NULL)

{

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

temp=temp->next;

}

}

int main()

{

int value, i, loc;

head=NULL;

printf("Select the choice of operation on link list");

printf("\n1.) insert at begning\n2.) insert at at\n3.) insert at middle");

printf("\n4.) delete from end\n5.) reverse the link list\n6.) display list\n7.)exit");

while(1)

{

printf("\n\nenter the choice of operation you want to do ");

scanf("%d",&i);

switch(i)

{

case1:

{

printf("enter the value you want to insert in node ");

scanf("%d",&value);

insert\_begning(value);

display();

break;

}

case2:

{

printf("enter the value you want to insert in node at last ");

scanf("%d",&value);

insert\_end(value);

display();

break;

}

case3:

{

printf("after which data you want to insert data ");

scanf("%d",&loc);

printf("enter the data you want to insert in list ");

scanf("%d",&value);

insert\_after(value,loc);

display();

break;

}

case4:

{

delete\_from\_end();

display();

break;

}

case5:

{

printf("enter the value you want to delete");

scanf("%d",value);

delete\_from\_middle(value);

display();

break;

}

case6 :

{

display();

break;

}

case7 :

{

exit(0);

break;

}

}

}

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

display();

getch();

}

**11.Write a program to implement circular linked list for the following operations: Create, Display,inserting, counting, searching, traversing and deletion.**

#include<stdio.h>

#include<stdlib.h>

struct Node;

typedef struct Node \* PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

struct Node

{

    int e;

    Position next;

};

void Insert(int x, List l, Position p)

{

    Position TmpCell;

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

    if(TmpCell == NULL)

        printf("Memory out of space\n");

    else

    {

        TmpCell->e = x;

        TmpCell->next = p->next;

        p->next = TmpCell;

    }

}

int isLast(Position p, List l)

{

    return (p->next == l);

}

Position FindPrevious(int x, List l)

{

    Position p = l;

    while(p->next != l && p->next->e != x)

        p = p->next;

    return p;

}

Position Find(int x, List l)

{

    Position p = l->next;

    while(p != l && p->e != x)

        p = p->next;

    return p;

}

void Delete(int x, List l)

{

    Position p, TmpCell;

    p = FindPrevious(x, l);

    if(!isLast(p, l))

    {

        TmpCell = p->next;

        p->next = TmpCell->next;

        free(TmpCell);

    }

    else

        printf("Element does not exist!!!\n");

}

void Display(List l)

{

    printf("The list element are :: ");

    Position p = l->next;

    while(p != l)

    {

        printf("%d -> ", p->e);

        p = p->next;

    }

}

void main()

{

    int x, pos, ch, i;

    List l, l1;

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

    l->next = l;

    List p = l;

    printf("CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");

    do

    {

        printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");

        scanf("%d", &ch);

        switch(ch)

        {

            case 1:

                p = l;

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

                scanf("%d",&x);

                printf("Enter the position of the element :: ");

                scanf("%d",&pos);

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

                {

                    p = p->next;

                }

                Insert(x,l,p);

                break;

            case 2:

                p = l;

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

                scanf("%d",&x);

                Delete(x,p);

                break;

            case 3:

                p = l;

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

                scanf("%d",&x);

                p = Find(x,p);

                if(p == l)

                    printf("Element does not exist!!!\n");

                else

                    printf("Element exist!!!\n");

                break;

            case 4:

                Display(l);

                break;

        }

    }while(ch<5);

    return 0;

}

**12. Write a program to implement insertion, deletion and traversing in B tree.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 4

#define MIN 2

struct btreeNode {

int val[MAX + 1], count;

struct btreeNode \*link[MAX + 1];

};

struct btreeNode \*root;

struct btreeNode \* createNode(int val, struct btreeNode \*child) {

struct btreeNode \*newNode;

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

newNode->val[1] = val;

newNode->count = 1;

newNode->link[0] = root;

newNode->link[1] = child;

return newNode;

}

void addValToNode(int val, int pos, struct btreeNode \*node,

struct btreeNode \*child) {

int j = node->count;

while (j > pos) {

node->val[j + 1] = node->val[j];

node->link[j + 1] = node->link[j];

j--;

}

node->val[j + 1] = val;

node->link[j + 1] = child;

node->count++;

}

void splitNode (int val, int \*pval, int pos, struct btreeNode \*node,

struct btreeNode \*child, struct btreeNode \*\*newNode) {

int median, j;

if (pos > MIN)

median = MIN + 1;

else

median = MIN;

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

j = median + 1;

while (j <= MAX) {

(\*newNode)->val[j - median] = node->val[j];

(\*newNode)->link[j - median] = node->link[j];

j++;

}

node->count = median;

(\*newNode)->count = MAX - median;

if (pos <= MIN) {

addValToNode(val, pos, node, child);

} else {

addValToNode(val, pos - median, \*newNode, child);

}

\*pval = node->val[node->count];

(\*newNode)->link[0] = node->link[node->count];

node->count--;

}

int setValueInNode(int val, int \*pval,

struct btreeNode \*node, struct btreeNode \*\*child) {

int pos;

if (!node) {

\*pval = val;

\*child = NULL;

return 1;

}

if (val < node->val[1]) {

pos = 0;

} else {

for (pos = node->count;

(val < node->val[pos] && pos > 1); pos--);

if (val == node->val[pos]) {

printf("Duplicates not allowed\n");

return 0;

}

}

if (setValueInNode(val, pval, node->link[pos], child)) {

if (node->count < MAX) {

addValToNode(\*pval, pos, node, \*child);

} else {

splitNode(\*pval, pval, pos, node, \*child, child);

return 1;

}

}

return 0;

}

void insertion(int val) {

int flag, i;

struct btreeNode \*child;

flag = setValueInNode(val, &i, root, &child);

if (flag)

root = createNode(i, child);

}

void copySuccessor(struct btreeNode \*myNode, int pos) {

struct btreeNode \*dummy;

dummy = myNode->link[pos];

for (;dummy->link[0] != NULL;)

dummy = dummy->link[0];

myNode->val[pos] = dummy->val[1];

}

void removeVal(struct btreeNode \*myNode, int pos) {

int i = pos + 1;

while (i <= myNode->count) {

myNode->val[i - 1] = myNode->val[i];

myNode->link[i - 1] = myNode->link[i];

i++;

}

myNode->count--;

}

void doRightShift(struct btreeNode \*myNode, int pos) {

struct btreeNode \*x = myNode->link[pos];

int j = x->count;

while (j > 0) {

x->val[j + 1] = x->val[j];

x->link[j + 1] = x->link[j];

}

x->val[1] = myNode->val[pos];

x->link[1] = x->link[0];

x->count++;

x = myNode->link[pos - 1];

myNode->val[pos] = x->val[x->count];

myNode->link[pos] = x->link[x->count];

x->count--;

return;

}

void doLeftShift(struct btreeNode \*myNode, int pos) {

int j = 1;

struct btreeNode \*x = myNode->link[pos - 1];

x->count++;

x->val[x->count] = myNode->val[pos];

x->link[x->count] = myNode->link[pos]->link[0];

x = myNode->link[pos];

myNode->val[pos] = x->val[1];

x->link[0] = x->link[1];

x->count--;

while (j <= x->count) {

x->val[j] = x->val[j + 1];

x->link[j] = x->link[j + 1];

j++;

}

return;

}

void mergeNodes(struct btreeNode \*myNode, int pos) {

int j = 1;

struct btreeNode \*x1 = myNode->link[pos], \*x2 = myNode->link[pos - 1];

x2->count++;

x2->val[x2->count] = myNode->val[pos];

x2->link[x2->count] = myNode->link[0];

while (j <= x1->count) {

x2->count++;

x2->val[x2->count] = x1->val[j];

x2->link[x2->count] = x1->link[j];

j++;

}

j = pos;

while (j < myNode->count) {

myNode->val[j] = myNode->val[j + 1];

myNode->link[j] = myNode->link[j + 1];

j++;

}

myNode->count--;

free(x1);

}

void adjustNode(struct btreeNode \*myNode, int pos) {

if (!pos) {

if (myNode->link[1]->count > MIN) {

doLeftShift(myNode, 1);

} else {

mergeNodes(myNode, 1);

}

} else {

if (myNode->count != pos) {

if(myNode->link[pos - 1]->count > MIN) {

doRightShift(myNode, pos);

} else {

if (myNode->link[pos + 1]->count > MIN) {

doLeftShift(myNode, pos + 1);

} else {

mergeNodes(myNode, pos);

}

}

} else {

if (myNode->link[pos - 1]->count > MIN)

doRightShift(myNode, pos);

else

mergeNodes(myNode, pos);

}

}

}

int delValFromNode(int val, struct btreeNode \*myNode) {

int pos, flag = 0;

if (myNode) {

if (val < myNode->val[1]) {

pos = 0;

flag = 0;

} else {

for (pos = myNode->count;

(val < myNode->val[pos] && pos > 1); pos--);

if (val == myNode->val[pos]) {

flag = 1;

} else {

flag = 0;

}

}

if (flag) {

if (myNode->link[pos - 1]) {

copySuccessor(myNode, pos);

flag = delValFromNode(myNode->val[pos], myNode->link[pos]);

if (flag == 0) {

printf("Given data is not present in B-Tree\n");

}

} else {

removeVal(myNode, pos);

}

} else {

flag = delValFromNode(val, myNode->link[pos]);

}

if (myNode->link[pos]) {

if (myNode->link[pos]->count < MIN)

adjustNode(myNode, pos);

}

}

return flag;

}

void deletion(int val, struct btreeNode \*myNode) {

struct btreeNode \*tmp;

if (!delValFromNode(val, myNode)) {

printf("Given value is not present in B-Tree\n");

return;

} else {

if (myNode->count == 0) {

tmp = myNode;

myNode = myNode->link[0];

free(tmp);

}

}

root = myNode;

return;

}

void searching(int val, int \*pos, struct btreeNode \*myNode) {

if (!myNode) {

return;

}

if (val < myNode->val[1]) {

\*pos = 0;

} else {

for (\*pos = myNode->count;

(val < myNode->val[\*pos] && \*pos > 1); (\*pos)--);

if (val == myNode->val[\*pos]) {

printf("Given data %d is present in B-Tree", val);

return;

}

}

searching(val, pos, myNode->link[\*pos]);

return;

}

void traversal(struct btreeNode \*myNode) {

int i;

if (myNode) {

for (i = 0; i < myNode->count; i++) {

traversal(myNode->link[i]);

printf("%d ", myNode->val[i + 1]);

}

traversal(myNode->link[i]);

}

}

int main() {

int val, ch;

while (1) {

printf("1. Insertion\t2. Deletion\n");

printf("3. Searching\t4. Traversal\n");

printf("5. Exit\nEnter your choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter your input:");

scanf("%d", &val);

insertion(val);

break;

case 2:

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

scanf("%d", &val);

deletion(val, root);

break;

case 3:

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

scanf("%d", &val);

searching(val, &ch, root);

break;

case 4:

traversal(root);

break;

case 5:

exit(0);

default:

printf("U have entered wrong option!!\n");

break;

}

printf("\n");

}

}