

Ans1

A priority queue is an abstract data type in computer programming, supporting the following three operations:

- add an element to the queue with an associated priority
- remove the element from the queue that has the highest priority, and return it
- (optionally) peek at the element with highest priority without removing it

A simple way to implement a priority queue data type is to keep a list of elements, and search through the list for the highest priority element for each "minimum" or "peek" operation. This implementation takes $O(1)$ time to insert an element, and $O(n)$ time for "minimum" or "peek". There are many more efficient implementations available.

If a self-balancing binary search tree is used, all three operations take $O(\log n)$ time; this is a popular solution in environments that already provide balanced trees. The van Emde Boas tree, another associative array data structure, can perform all three operations in $O(\log \log n)$ time, but at a space cost for small queues of about $O(2^{m/2})$, where m is the number of bits in the priority value, which may be prohibitive.

Implementation of Priority Queues:

Priority Queues can be implemented for:

Priority Queues can be implemented for:

1. Bandwidth Management
2. Discrete Event Simulations
3. A* Search Algorithm

Example of Priority Queue:

```

/* Program of Insertion, Deletion and Display in a Priority queue using linked list*/
#include<stdio.h>
#include<malloc.h>

struct node
{
int priority;
int info;
struct node *link;
}*front = NULL;

{
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:

```

```

break;
case 4:
exit(1);
default :
printf("\Wrong choice\n");
}/*End of switch*/
}/*End of while*/
}/*End of main()*/

insert()
{
struct node *tmp,*q;
int added_item,item_priority;
tmp = (struct node *)malloc(sizeof(struct node));
printf("Input the item value to be added in the queue : ");
scanf("%d",&added_item);
printf("Enter its priority : ");
scanf("%d",&item_priority);
tmp->info = added_item;
tmp->priority = item_priority;
/*Queue is empty or item to be added has priority more than first item*/
if( front == NULL || item_priority < front->priority )
{
tmp->priority = item_priority;
/*Queue is empty or item to be added has priority more than first item*/
if( front == NULL || item_priority < front->priority )
{
tmp->link = front;
front = tmp;
}
else
{
q = front;
while( q->link != NULL && q->link->priority <= item_priority )
q=q->link;
tmp->link = q->link;
q->link = tmp;
}/*End of else*/
}/*End of insert()*/

del()
{
struct node *tmp;
if(front == NULL)
printf("Queue Underflow\n");
else
printf("Queue Underflow\n");
else
{
tmp = front;
printf("Deleted item is %d\n",tmp->info);
front = front->link;
free(tmp);
}
}/*End of del()*/

display()
{
struct node *ptr;
ptr = front;
if(front == NULL)
printf("Queue is empty\n");
else
{ printf("Queue is :\n");
printf("Priority Item\n");
while(ptr != NULL)
{
printf("%5d %5d\n",ptr->priority,ptr->info);
ptr = ptr->link;
}
}
}

```

```

display()
{
struct node *ptr;
ptr = front;
if(front == NULL)
printf("Queue is empty\n");
else
{ printf("Queue is :\n");
printf("Priority Item\n");
while(ptr != NULL)
{
printf("%5d %5d\n",ptr->priority,ptr->info);
ptr = ptr->link;
}
}
}/*End of else */
}/*End of display() */

```

Ans2

```

/* Program of polynomial addition using linked list */
#include <stdio.h>
#include <malloc.h>
#include <conio.h>

struct node
{
float coef;
int expo;
struct node *link;
};

struct node *poly_add(struct node *,struct node *);
struct node *enter(struct node *);
struct node *insert(struct node *,float,int);

main( )
{
struct node *p1_start,*p2_start,*p3_start;

p1_start=NULL;

p1_start=NULL;
p2_start=NULL;
p3_start=NULL;

clrscr();
printf("Polynomial 1 :\n");
p1_start=enter(p1_start);

printf("Polynomial 2 :\n");
p2_start=enter(p2_start);

p3_start=poly_add(p1_start,p2_start);

printf("Polynomial 1 is : ");
display(p1_start);
printf("Polynomial 2 is : ");
display(p2_start);
printf("Added polynomial is : ");
display(p3_start);
}/*End of main()*/

```

```

struct node *enter(struct node *start)
{
int i,n,ex;
float co;
printf("How many terms u want to enter : ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("Enter coeficient for term %d : ",i);
scanf("%f",&co);
printf("Enter exponent for term %d : ",i);
scanf("%d",&ex);
start=insert(start,co,ex);
}
return start;
}/*End of enter()*/

```

```

struct node *insert(struct node *start,float co,int ex)
{
struct node *ptr,*tmp;
tmp= malloc(sizeof(struct node));

```

```

struct node *ptr,*tmp;
tmp= malloc(sizeof(struct node));
tmp->coef=co;
tmp->expo=ex;

/*list empty or exp greater than first one */
if(start==NULL || ex>start->expo)
{
tmp->link=start;
start=tmp;
}
else
{
ptr=start;
while(ptr->link!=NULL && ptr->link->expo>ex)
ptr=ptr->link;
tmp->link=ptr->link;
ptr->link=tmp;
if(ptr->link==NULL) /*item to be added in the end */
tmp->link=NULL;
}
return start;
}/*End of insert()*/

```

```

struct node *poly_add(struct node *p1,struct node *p2)
{
struct node *p3_start,*p3,*tmp;
p3_start=NULL;
if(p1==NULL && p2==NULL)
return p3_start;

while(p1!=NULL && p2!=NULL )
{
tmp=malloc(sizeof(struct node));
if(p3_start==NULL)
{
p3_start=tmp;
p3=p3_start;
}
else
{
p3->link=tmp;
p3=p3->link;
}
if(p1->expo > p2->expo)
{
tmp->coef=p1->coef;

```

```
if(p1->expo > p2->expo)
{
tmp->coef=p1->coef;
tmp->expo=p1->expo;
p1=p1->link;
}
else
if(p2->expo > p1->expo)
{
tmp->coef=p2->coef;
tmp->expo=p2->expo;
p2=p2->link;
}
else
if(p1->expo == p2->expo)
{
tmp->coef=p1->coef + p2->coef;
tmp->expo=p1->expo;
p1=p1->link;
p2=p2->link;
}
}/*End of while*/
while(p1!=NULL)
{
```

```
while(p1!=NULL)
{
tmp=malloc(sizeof(struct node));
tmp->coef=p1->coef;
tmp->expo=p1->expo;
if (p3_start==NULL) /*poly 2 is empty*/
{
p3_start=tmp;
p3=p3_start;
}
else
{
p3->link=tmp;
p3=p3->link;
}
p1=p1->link;
}/*End of while */
while(p2!=NULL)
{
tmp=malloc(sizeof(struct node));
tmp->coef=p2->coef;
tmp->expo=p2->expo;
if (p3_start==NULL) /*poly 1 is empty*/
{
```

```

if (p3_start==NULL) /*poly 2 is empty*/
{
p3_start=tmp;
p3=p3_start;
}
else
{
p3->link=tmp;
p3=p3->link;
}
p1=p1->link;
}/*End of while */
while(p2!=NULL)
{
tmp=malloc(sizeof(struct node));
tmp->coef=p2->coef;
tmp->expo=p2->expo;
if (p3_start==NULL) /*poly 1 is empty*/
{
p3_start=tmp;
p3=p3_start;
}
else

```

```

{
p3->link=tmp;
p3=p3->link;
}
p2=p2->link;
}/*End of while*/
p3->link=NULL;
return p3_start;
}/*End of poly_add() */

```

```

display(struct node *ptr)
{
if(ptr==NULL)
{
printf("Empty\n");
return;
}
while(ptr!=NULL)
{
printf("%.1fx^%d) + ", ptr->coef,ptr->expo);
ptr=ptr->link;
}
printf("\b\b\n"); /* \b\b to erase the last + sign */

```

```

printf("Empty\n");
return;
}
while(ptr!=NULL)
{
printf("%.1fx^%d) + ", ptr->coef,ptr->expo);
ptr=ptr->link;
}
printf("\b\b\n"); /* \b\b to erase the last + sign */
}/*End of display()*/

```

Ans3

```
/* Program of double linked list – Create, Reverse and Display */
#include <stdio.h>
#include <malloc.h>
```

```
struct node
{
struct node *prev;
int info;
struct node *next;
}*start;
```

```
main()
{
int choice,n,m,po,i;
start=NULL;
while(1)
{
printf("1.Create List\n");
printf("2.Add at begining\n");
printf("3.Add after\n");
printf("4.Delete\n");
printf("5.Display\n");
```

```
printf("3.Add after\n");
printf("4.Delete\n");
printf("5.Display\n");
printf("6.Count\n");
printf("7.Reverse\n");
printf("8.exit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("How many nodes you want : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the element : ");
scanf("%d",&m);
create_list(m);
}
break;
case 2:
printf("Enter the element : ");
scanf("%d",&m);
addatbeg(m);
break;
case 3:
printf("Enter the element : ");
scanf("%d",&m);
printf("Enter the position after which this element is inserted : ");
scanf("%d",&po);
addafter(m,po);
break;
case 4:
printf("Enter the element for deletion : ");
scanf("%d",&m);
del(m);
break;
case 5:
display();
break;
case 6:
count();
break;
```

```

        break;
    case 6:
        count();
        break;
    case 7:
        rev();
        break;
    case 8:
        exit();
    default:
        printf("Wrong choice\n");
    }/*End of switch*/
}/*End of while*/
}/*End of main()*/

create_list(int num)
{
    struct node *q,*tmp;
    tmp= malloc(sizeof(struct node));
    tmp->info=num;
    tmp->next=NULL;
    if(start==NULL)
    {

```

```

        {
        tmp->prev=NULL;
        start->prev=tmp;
        start=tmp;
        }
    else
    {
        q=start;
        while(q->next!=NULL)
        q=q->next;
        q->next=tmp;
        tmp->prev=q;
    }
}/*End of create_list()*/

```

```

addatbeg(int num)
{
    struct node *tmp;
    tmp=malloc(sizeof(struct node));
    tmp->prev=NULL;
    tmp->info=num;
    tmp->next=start;

```

```

    tmp->next=start;
    start->prev=tmp;
    start=tmp;
}/*End of addatbeg()*/

```

```

addafter(int num,int c)
{
    struct node *tmp,*q;
    int i;
    q=start;
    for(i=0;i<c-1;i++)
    {
        q=q->next;
        if(q==NULL)
        {
            printf("There are less than %d elements\n",c);
            return;
        }
    }
    tmp=malloc(sizeof(struct node) );
    tmp->info=num;
    q->next->prev=tmp;
    tmp->next=q->next;

```

```

int i;
q=start;
for(i=0;i<c-1;i++)
{
q=q->next;
if(q==NULL)
{
printf("There are less than %d elements\n",c);
return;
}
}
tmp=malloc(sizeof(struct node) );
tmp->info=num;
q->next->prev=tmp;
tmp->next=q->next;
tmp->prev=q;
q->next=tmp;
}/*End of addafter() */

del(int num)
{
struct node *tmp,*q;
if(start->info==num)

```

```

{
struct node *tmp,*q;
if(start->info==num)
{
tmp=start;
start=start->next; /*first element deleted*/
start->prev = NULL;
free(tmp);
return;
}
q=start;
while(q->next->next!=NULL)
{
if(q->next->info==num) /*Element deleted in between*/
{
tmp=q->next;
q->next=tmp->next;
tmp->next->prev=q;
free(tmp);
return;
}
q=q->next;
}
}/*End of del()*/

```

```

}
q=q->next;
}
if(q->next->info==num) /*last element deleted*/
{ tmp=q->next;
free(tmp);
q->next=NULL;
return;
}
printf("Element %d not found\n",num);
}/*End of del()*/

```

```

display()
{
struct node *q;
if(start==NULL)
{
printf("List is empty\n");
return;
}
q=start;
printf("List is :\n");
while(q!=NULL)

```

```

while(q!=NULL)
{
printf("%d ", q->info);
q=q->next;
}
printf("\n");
}/*End of display() */

count()
{ struct node *q=start;
int cnt=0;
while(q!=NULL)
{
q=q->next;
cnt++;
}
printf("Number of elements are %d\n",cnt);
}/*End of count()*/

rev()
{
struct node *p1,*p2;
p1=start;

```

```

}

rev()
{
struct node *p1,*p2;
p1=start;
p2=p1->next;
p1->next=NULL;
p1->prev=p2;
while(p2!=NULL)
{
p2->prev=p2->next;
p2->next=p1;
p1=p2;
p2=p2->prev; /*next of p2 changed to prev */
}
start=p1;
}/*End of rev()*/

```

Ans4

```

/* Program for traversing a graph through and DFS */
#include<stdio.h>
#define MAX 20

typedef enum boolean{false,true} bool;
int adj[MAX][MAX];
bool visited[MAX];
int n; /* Denotes number of nodes in the graph */
main()
{
int i,v,choice;

create_graph();
while(1)
{
printf("\n");
printf("1. Adjacency matrix\n");
printf("2. Depth First Search using stack\n");
printf("3. Depth First Search through recursion\n");
printf("5. Adjacent vertices\n");
printf("6. Components\n");
printf("7. Exit\n");
printf("Enter your choice : ");

```

```

printf("5. Adjacent vertices\n");
printf("6. Components\n");
printf("7. Exit\n");
printf("Enter your choice : ");
scanf("%d",&choice);

switch(choice)
{
case 1:
printf("Adjacency Matrix\n");
display();
break;
case 2:
printf("Enter starting node for Depth First Search : ");
scanf("%d",&v);
for(i=1;i<=n;i++)
visited[i]=false;
dfs(v);
break;
case 3:
printf("Enter starting node for Depth First Search : ");
scanf("%d",&v);
for(i=1;i<=n;i++)
visited[i]=false;
scanf("%d",&v);
for(i=1;i<=n;i++)
visited[i]=false;
dfs_rec(v);
break;
case 4:printf("\n Not Used !");
case 5:
printf("Enter node to find adjacent vertices : ");
scanf("%d", &v);
printf("Adjacent Vertices are : ");
adj_nodes(v);
break;
case 6:
components();
break;
case 7:
exit(1);

default:
printf("\Wrong choice\n");
break;
}/*End of switch*/
}/*End of while*/
}/*End of main()*/

create_graph()
{
int i,max_edges,origin,destin;

printf("Enter number of nodes : ");
scanf("%d",&n);
max_edges=n*(n-1);

for(i=1;i<=max_edges;i++)
{
printf("Enter edge %d( 0 0 to quit ) : ",i);
scanf("%d %d",&origin,&destin);

if((origin==0) && (destin==0))
break;
}
}

```

```

    return( 0);
}

if((origin==0) && (destin==0))
break;

if( origin > n || destin > n || origin<=0 || destin<=0)
{
printf("Invalid edge!\n");
i--;
}
else
{
adj[origin][destin]=1;
}
}/*End of for*/
}/*End of create_graph()*/

```

```

display()
{
int i,j;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)

```

```

for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%4d",adj[i][j]);
printf("\n");
}
}/*End of display()*/

```

```

dfs_rec(int v)
{
int i;
visited[v]=true;
printf("%d ",v);
for(i=1;i<=n;i++)
if(adj[v][i]==1 && visited[i]==false)
dfs_rec(i);
}/*End of dfs_rec()*/

```

```

dfs(int v)
{
int i,stack[MAX],top=-1,pop_v,j,t;
int ch;

```

```

top++;
stack[top]=v;
while (top>=0)
{
pop_v=stack[top];
top--; /*pop from stack*/
if( visited[pop_v]==false)
{
printf("%d ",pop_v);
visited[pop_v]=true;
}
else
continue;

```

```

for(i=n;i>=1;i--)
{
if( adj[pop_v][i]==1 && visited[i]==false)
{
top++; /* push all unvisited neighbours of pop_v */
stack[top]=i;
}/*End of if*/
}/*End of for*/
}/*End of while*/

```

```

adj_nodes(int v)
{
int i;
for(i=1;i<=n;i++)
if(adj[v][i]==1)
printf("%d ",i);
printf("\n");
}/*End of adj_nodes()*/

components()
{
int i;
for(i=1;i<=n;i++)
visited[i]=false;
for(i=1;i<=n;i++)
{
if(visited[i]==false)
dfs_rec(i);
}
printf("\n");
}/*End of components()*/

```

Ans5

```

/* Program of single linked list with n integers
displayed in 4 seperate Lists */

```

```

#include <stdio.h>
#include <malloc.h>
#include <conio.h>

```

```

struct node
{
int info;
struct node *link;
}*start,*list1,*list2,*list3,*list4;

```

```

void main()
{
int choice,n,m,position,i;
int c1=0,c2=1,c3=2,c4=3;
list1=NULL;
list2=NULL;
list3=NULL;

```

```

int choice,n,m,position,i;
int c1=0,c2=1,c3=2,c4=3;
list1=NULL;
list2=NULL;
list3=NULL;
list4=NULL;

```

```

while(1)
{ clrscr();
printf("\n\n");
printf("\n\n0.Create List");
printf("\n\n1.Display 1st List");
printf("\n\n2.Display 2nd List");
printf("\n\n3.Display 3rd List");
printf("\n\n4.Display 4th List");
printf("\n\n5.Quit");
printf("\n\n\nEnter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 0:
printf("How many nodes you want : ");
scanf("%d",&n);

```

```

{
case 0:
printf("How many nodes you want : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
if(i==c1)
{
printf("Enter the element : ");
scanf("%d",&m);
create_list1(m);
c1=c1+4;
}

if(i==c2)
{
printf("Enter the element : ");
scanf("%d",&m);
create_list2(m);
c2=c2+4;
}

if(i==c3)

```

```

{
scanf("%d",&m);
create_list3(m);
c3=c3+4;
}

if(i==c4)
{
printf("Enter the element : ");
scanf("%d",&m);
create_list4(m);
c4=c4+4;
}
} // End of For Loop
break;

case 1:display1();
break;

```

```

case 2:
display2();
break;

case 3:display3();
break;

case 4:display4();
break;

case 5:exit();
break;

default:
printf("Wrong choice\n");
}/*End of switch */
}/*End of while */
}/*End of main()*/

```

```

create_list1(int data)
{

```

```

create_list1(int data)
{
    struct node *q,*tmp;
    tmp= malloc(sizeof(struct node));
    tmp->info=data;
    tmp->link=NULL;

    if(list1==NULL) /*If list is empty */
    list1=tmp;
    else
    { /*Element inserted at the end */
    q=list1;
    while(q->link!=NULL)
    q=q->link;
    q->link=tmp;
    }
} /*End of create_list1()*/

create_list2(int data)
{
    struct node *q,*tmp;
    tmp= malloc(sizeof(struct node));

```

```

{
    struct node *q,*tmp;
    tmp= malloc(sizeof(struct node));
    tmp->info=data;
    tmp->link=NULL;

    if(list2==NULL) /*If list is empty */
    list2=tmp;
    else
    { /*Element inserted at the end */
    q=list2;
    while(q->link!=NULL)
    q=q->link;
    q->link=tmp;
    }
} /*End of create_list2()*/

create_list3(int data)
{
    struct node *q,*tmp;
    tmp= malloc(sizeof(struct node));
    tmp->info=data;
    tmp->link=NULL;

```

```

    if(list3==NULL) /*If list is empty */
    list3=tmp;
    else
    { /*Element inserted at the end */
    q=list3;
    while(q->link!=NULL)
    q=q->link;
    q->link=tmp;
    }
} /*End of create_list3()*/

```

```

create_list4(int data)
{
    struct node *q,*tmp;
    tmp= malloc(sizeof(struct node));
    tmp->info=data;
    tmp->link=NULL;

```

```

    if(list4==NULL) /*If list is empty */
    list4=tmp;
    else

```

```
    { /*Element inserted at the end */
      q=list4;
      while(q->link!=NULL)
        q=q->link;
      q->link=tmp;
    }
  } /*End of create_list4()*/
```

```
display1()
{
  struct node *q;
  if(list1 == NULL)
  {
    printf("List is empty\n");
    return;
  }
  q=list1;
  printf("List is :\n");
  while(q!=NULL)
  {
    printf("%d ", q->info);
    q=q->link;
  }
}
```

```

}
printf("\n");
getch();
} /*End of display1() */
```

```
display2()
{
  struct node *q;
  if(list2 == NULL)
  {
    printf("List is empty\n");
    return;
  }
  q=list2;
  printf("List is :\n");
  while(q!=NULL)
  {
    printf("%d ", q->info);
    q=q->link;
  }
  printf("\n");
  getch();
} /*End of display2() */
```

```

printf("\n");
getch();
} /*End of display2() */
```

```
display3()
{
  struct node *q;
  if(list3 == NULL)
  {
    printf("List is empty\n");
    return;
  }
  q=list3;
  printf("List is :\n");
  while(q!=NULL)
  {
    printf("%d ", q->info);
    q=q->link;
  }
  printf("\n");
  getch();
} /*End of display3() */
```

```

printf("n"),
getch();
}/*End of display3() */

display4()
{
struct node *q;
if(list4 == NULL)
{
printf("List is empty\n");
return;
}
q=list4;
printf("List is :\n");
while(q!=NULL)
{
printf("%d ", q->info);
q=q->link;
}
printf("\n");
getch();
}/*End of display4() */

```

Ans6

```

/*Program of Create deletion and Display as per Question in B tree*/
#include<stdlib.h>
#include<stdio.h>
#define M 5

struct node{
int n; /* n < M No. of keys in node will always less than order of B tree */
int keys[M-1]; /*array of keys*/
struct node *p[M]; /* (n+1 pointers will be in use) */
}*root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys };

void insert(int key);
void display(struct node *root,int);
void DelNode(int x);
void search(int x);

void search(int x);
enum KeyStatus ins(struct node *r, int x, int* y, struct node** u);
int searchPos(int x,int *key_arr, int n);
enum KeyStatus del(struct node *r, int x);

int main()
{
int key;
int choice;
printf("Creation of B tree for node %d\n",M);
while(1)
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Search\n");
printf("4.Display\n");
printf("5.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);

switch(choice)
{
case 1:

```

```

t
case 1:
printf("Enter the key : ");
scanf("%d",&key);
insert(key);
break;
case 2:
printf("Enter the key : ");
scanf("%d",&key);
DelNode(key);
break;
case 3:
printf("Enter the key : ");
scanf("%d",&key);
search(key);
break;
case 4:
printf("Btree is :\n");
display(root,0);
break;
case 5:
exit(1);
default:
printf("\Wrong choice\n");

```

```

case 0:
exit(1);
default:
printf("\Wrong choice\n");
break;
}/*End of switch*/
}/*End of while*/
return 0;
}/*End of main()*/

```

```

void insert(int key)
{
struct node *newnode;
int upKey;
enum KeyStatus value;
value = ins(root, key, &upKey, &newnode);
if (value == Duplicate)
printf("Key already available\n");
if (value == InsertIt)
{
struct node *uproot = root;
root=malloc(sizeof(struct node));
root->n = 1;
root->key[0] = upKey;

```

```

root->keys[0] = upKey,
root->p[0] = uproot,
root->p[1] = newnode;
}/*End of if */
}/*End of insert()*/

enum KeyStatus ins(struct node *ptr, int key, int *upKey, struct node **newnode)
{
struct node *newPtr, *lastPtr;
int pos, i, n, splitPos;
int newKey, lastKey;
enum KeyStatus value;
if (ptr == NULL)
{
*newnode = NULL;
*upKey = key;
return InsertIt;
}
n = ptr->n;
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
return Duplicate;
value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)

```

```

if (value != InsertIt)
return value;
/*If keys in node is less than M-1 where M is order of B tree*/
if (n < M - 1)
{
pos = searchPos(newKey, ptr->keys, n);
/*Shifting the key and pointer right for inserting the new key*/
for (i=n; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
/*Key is inserted at exact location*/
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
++ptr->n; /*incrementing the number of keys in node*/
return Success;
}/*End of if */
/*If keys in nodes are maximum and position of node to be inserted is last*/
if (pos == M - 1)
{
lastKey = newKey;
lastPtr = newPtr;
,

```

```

lastKey = newKey;
lastPtr = newPtr;
}
else /*If keys in node are maximum and position of node to be inserted is not last*/
{
lastKey = ptr->keys[M-2];
lastPtr = ptr->p[M-1];
for (i=M-2; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
}
splitPos = (M - 1)/2;
(*upKey) = ptr->keys[splitPos];

(*newnode)=malloc(sizeof(struct node));/*Right node after split*/
ptr->n = splitPos; /*No. of keys for left splitted node*/
(*newnode)->n = M-1-splitPos; /*No. of keys for right splitted node*/
for (i=0; i < (*newnode)->n; i++)
{

```

```

(*newnode)->n = M-1-splitPos; /*No. of keys for right splitted node*/
for (i=0; i < (*newnode)->n; i++)
{
(*newnode)->p[i] = ptr->p[i + splitPos + 1];
if(i < (*newnode)->n - 1)
(*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
else
(*newnode)->keys[i] = lastKey;
}
(*newnode)->p[(*)newnode)->n] = lastPtr;
return InsertIt;
}/*End of ins()*/

```

```

void display(struct node *ptr, int blanks)
{
if (ptr)
{
int i;
for(i=1; i<=blanks; i++)
printf(" ");
for (i=0; i < ptr->n; i++)
printf("%d ", ptr->keys[i]);
printf("\n");
}
}

```

```

for (i=0; i <= ptr->n; i++)
display(ptr->p[i], blanks+10);
}/*End of if*/
}/*End of display()*/

```

```

void search(int key)
{
int pos, i, n;
struct node *ptr = root;
printf("Search path:\n");
while (ptr)
{
n = ptr->n;
for (i=0; i < ptr->n; i++)
printf(" %d", ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
{
printf("Key %d found in position %d of last dispalyed node\n",key,i);
return;
}
}
ptr = ptr->p[pos];
}

```

```

}
printf("Key %d is not available\n",key);
}/*End of search()*/

int searchPos(int key, int *key_arr, int n)
{
int pos=0;
while (pos < n && key > key_arr[pos])
pos++;
return pos;
}/*End of searchPos()*/

void DelNode(int key)
{
struct node *uproot;
enum KeyStatus value;
value = del(root,key);
switch (value)
{
case SearchFailure:
printf("Key %d is not available\n",key);
break;

```

```

case LessKeys:
uproot = root;
root = root->p[0];
free(uproot);
break;
}/*End of switch*/
}/*End of delnode()*/

```

```

enum KeyStatus del(struct node *ptr, int key)
{
int pos, i, pivot, n ,min;
int *key_arr;
enum KeyStatus value;
struct node **p,*lptr,*rptr;

if (ptr == NULL)
return SearchFailure;
/*Assigns values of node*/
n=ptr->n;
key_arr = ptr->keys;
p = ptr->p;
min = (M - 1)/2;/*Minimum number of keys*/

```

```

p = ptr->p;
min = (M - 1)/2;/*Minimum number of keys*/

pos = searchPos(key, key_arr, n);
if (p[0] == NULL)
{
if (pos == n || key < key_arr[pos])
return SearchFailure;
/*Shift keys and pointers left*/
for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
p[i] = p[i+1];
}
return --ptr->n >= (ptr==root ? 1 : min) ? Success : LessKeys;
}/*End of if */

if (pos < n && key == key_arr[pos])
{
struct node *qp = p[pos], *qp1;
int nkey;
while(1)
{

```

```

int nkey;
while(1)
{
nkey = qp->n;
qp1 = qp->p[nkey];
if (qp1 == NULL)
break;
qp = qp1;
}/*End of while*/
key_arr[pos] = qp->keys[nkey-1];
qp->keys[nkey - 1] = key;
}/*End of if */
value = del(p[pos], key);
if (value != LessKeys)
return value;

if (pos > 0 && p[pos-1]->n > min)
{
pivot = pos - 1; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pos];
/*Assigns values for right node*/
rptr->p[rptr->n + 1] = rptr->p[rptr->n];

rptr = p[pivot+ 1];
/*merge right node with left node*/
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
for (i=0; i < rptr->n; i++)
{
lptr->keys[lptr->n + 1 + i] = rptr->keys[i];
lptr->p[lptr->n + 2 + i] = rptr->p[i+1];
}
lptr->n = lptr->n + rptr->n + 1;
free(rptr); /*Remove right node*/
for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
p[i] = p[i+1];
}
return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;
}/*End of del()*/

```
