Practical No-1

#include<iostream>

#include<string.h>

using namespace std;

struct node

{

int value;

node\* next;

}\*HashTable[10];

class hashing

{

public:

hashing()

{

for(int i=0 ; i<10 ; i++)

{

HashTable[i]=NULL;

}

}

int HashFunction(int value)

{

return (value%10);

}

node\* create\_node(int x)

{

node\* temp=new node;

temp->next=NULL;

temp->value=x;

return temp;

}

void display()

{

for(int i=0 ; i< 10; i++)

{

node \* temp=new node;

temp=HashTable[i];

cout<<"a["<<i<<"] : ";

while(temp !=NULL)

{

cout<<" ->"<<temp->value;

temp=temp->next;

}

cout<<"\n";

}

}

int searchElement(int value)

{

bool flag = false;

int hash\_val = HashFunction(value);

node\* entry = HashTable[hash\_val];

cout<<"\nElement found at : ";

while (entry != NULL)

{

if (entry->value==value)

{

cout<<hash\_val<<" : "<<entry->value<<endl;

flag = true;

}

entry = entry->next;

}

if (!flag)

return -1;

}

void deleteElement(int value)

{

int hash\_val = HashFunction(value);

node\* entry = HashTable[hash\_val];

if (entry == NULL )

{

cout<<"No Element found ";

return;

}

if(entry->value==value)

{

HashTable[hash\_val]=entry->next;

return;

}

while ((entry->next)->value != value)

{

entry = entry->next;

}

entry->next=(entry->next)->next;

}

void insertElement(int value)

{

int hash\_val = HashFunction(value);

// node\* prev = NULL;

//node\* entry = HashTable[hash\_val];

node\* temp=new node;

node\* head=new node;

head = create\_node(value);

temp=HashTable[hash\_val];

if (temp == NULL)

{

HashTable[hash\_val] =head;

}

else

{

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next =head;

}

}

};

int main()

{

int ch;

int data,search,del;

hashing h;

do

{

cout<<"\nTelephone : \n1.Insert \n2.Display \n3.Search \n4.Delete \n5.Exit";

cin>>ch;

switch(ch)

{

case 1:cout<<"\nEnter phone no. to be inserted : ";

cin>>data;

h.insertElement(data);

break;

case 2:h.display();

break;

case 3:cout<<"\nEnter the no to be searched : ";

cin>>search;

if (h.searchElement(search) == -1)

{

cout<<"No element found at key ";

continue;

}

break;

case 4:cout<<"\nEnter the phno. to be deleted : ";

cin>>del;

h.deleteElement(del);

cout<<"Phno. Deleted"<<endl;

break;

}

}while(ch!=5);

return 0;

}

Practical No-1 Output

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit1

Enter phone no. to be inserted : 56777

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit1

Enter phone no. to be inserted : 89997

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit455

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit2

a[0] :

a[1] :

a[2] :

a[3] :

a[4] :

a[5] :

a[6] :

a[7] : ->56777 ->89997

a[8] :

a[9] :

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit56777

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit3

Enter the no to be searched : 89997

Element found at : 7 : 89997

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit4

Enter the phno. to be deleted : 56777

Phno. Deleted

Telephone :

1.Insert

2.Display

3.Search

4.Delete

5.Exit5

--------------------------------

Process exited after 47.31 seconds with return value 0

Press any key to continue . . .

Practical No-2

#include<iostream>

#include<string.h>

using namespace std;

class HashFunction

{

typedef struct hash

{

long key;

char name[10];

}hash;

hash h[10];

public:

HashFunction();

void insert();

void display();

int find(long);

void Delete(long);

};

HashFunction::HashFunction()

{

int i;

for(i=0;i<10;i++)

{

h[i].key=-1;

strcpy(h[i].name,"NULL");

}

}

void HashFunction::Delete(long k)

{

int index=find(k);

if(index==-1)

{

cout<<"\n\tKey Not Found";

}

else

{

h[index].key=-1;

strcpy(h[index].name,"NULL");

cout<<"\n\tKey is Deleted";

}

}

int HashFunction::find(long k)

{

int i;

for(i=0;i<10;i++)

{

if(h[i].key==k)

{

cout<<"\n\t"<<h[i].key<<" is Found at "<<i<<" Location With Name "<<h[i].name;

return i;

}

}

if(i==10)

{

return -1;

}

}

void HashFunction::display()

{

int i;

cout<<"\n\t\tKey\t\tName";

for(i=0;i<10;i++)

{

cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t\t"<<h[i].name;

}

}

void HashFunction::insert()

{

char ans,n[10],ntemp[10];

long k,temp;

int v,hi,cnt=0,flag=0,i;

do

{

if(cnt>=10)

{

cout<<"\n\tHash Table is FULL";

break;

}

cout<<"\n\tEnter a Telephone No: ";

cin>>k;

cout<<"\n\tEnter a Client Name: ";

cin>>n;

hi=k%10;// hash function

if(h[hi].key==-1)

{

h[hi].key=k;

strcpy(h[hi].name,n);

}

else

{

if(h[hi].key%10!=hi)

{

temp=h[hi].key;

strcpy(ntemp,h[hi].name);

h[hi].key=k;

strcpy(h[hi].name,n);

for(i=hi+1;i<10;i++)

{

if(h[i].key==-1)

{

h[i].key=temp;

strcpy(h[i].name,ntemp);

flag=1;

break;

}

}

for(i=0;i<hi && flag==0;i++)

{

if(h[i].key==-1)

{

h[i].key=temp;

strcpy(h[i].name,ntemp);

break;

}

}

}

else

{

for(i=hi+1;i<10;i++)

{

if(h[i].key==-1)

{

h[i].key=k;

strcpy(h[i].name,n);

flag=1;

break;

}

}

for(i=0;i<hi && flag==0;i++)

{

if(h[i].key==-1)

{

h[i].key=k;

strcpy(h[i].name,n);

break;

}

}

}

}

flag=0;

cnt++;

cout<<"\n\t..... Do You Want to Insert More Key: y/n";

cin>>ans;

}while(ans=='y'||ans=='Y');

}

int main()

{

long k;

int ch,index;

char ans;

HashFunction obj;

do

{

cout<<"\n\t\*\* Telephone (ADT) \*\*";

cout<<"\n\t1. Insert\n\t2. Display\n\t3. Find\n\t4. Delete\n\t5. Exit";

cout<<"\n\t..... Enter Your Choice: ";

cin>>ch;

switch(ch)

{

case 1: obj.insert();

break;

case 2: obj.display();

break;

case 3: cout<<"\n\tEnter a Key Which You Want to Search: ";

cin>>k;

index=obj.find(k);

if(index==-1)

{

cout<<"\n\tKey Not Found";

}

break;

case 4: cout<<"\n\tEnter a Key Which You Want to Delete: ";

cin>>k;

obj.Delete(k);

break;

case 5:

break;

}

cout<<"\n\t..... Do You Want to Continue in Main Menu:y/n ";

cin>>ans;

}while(ans=='y'||ans=='Y');

}

Practical No-2 Output

\*\* Telephone (ADT) \*\*

1. Insert

2. Display

3. Find

4. Delete

5. Exit

..... Enter Your Choice: 1

Enter a Telephone No: 4556

Enter a Client Name: divya

..... Do You Want to Insert More Key: y/ny

Enter a Telephone No: 9887

Enter a Client Name: rutuja

..... Do You Want to Insert More Key: y/n

..... Do You Want to Continue in Main Menu:y/n

--------------------------------

Process exited after 45.73 seconds with return value 0

Press any key to continue . . .

Practical No-3

# include <iostream>

# include <cstdlib>

# include <string.h>

using namespace std;

/\*

\* Node Declaration

\*/

struct node

{

char label[10];

int ch\_count;

struct node \*child[10];

}\*root;

/\*

\* Class Declaration

\*/

class GT

{

public:

void create\_tree();

void display(node \* r1);

GT()

{

root = NULL;

}

};

void GT::create\_tree()

{

int tbooks,tchapters,i,j,k;

root = new node;

cout<<"Enter name of book";

cin>>root->label;

cout<<"Enter no. of chapters in book";

cin>>tchapters;

root->ch\_count = tchapters;

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

{

root->child[i] = new node;

cout<<"Enter Chapter name\n";

cin>>root->child[i]->label;

cout<<"Enter no. of sections in Chapter: "<<root->child[i]->label;

cin>>root->child[i]->ch\_count;

for(j=0;j<root->child[i]->ch\_count;j++)

{

root->child[i]->child[j] = new node;

cout<<"Enter Section "<<j+1<<"name\n";

cin>>root->child[i]->child[j]->label;

//cout<<"Enter no. of subsections in "<<r1->child[i]->child[j]->label;

//cin>>r1->child[i]->ch\_count;

}

}

}

void GT::display(node \* r1)

{

int i,j,k,tchapters;

if(r1 != NULL)

{

cout<<"\n-----Book Hierarchy---";

cout<<"\n Book title : "<<r1->label;

tchapters = r1->ch\_count;

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

{

cout<<"\n Chapter "<<i+1;

cout<<" "<<r1->child[i]->label;

cout<<"\n Sections";

for(j=0;j<r1->child[i]->ch\_count;j++)

{

//cin>>r1->child[i]->child[j]->label;

cout<<"\n "<<r1->child[i]->child[j]->label;

}

}

}

}

/\*

\* Main Contains Menu

\*/

int main()

{

int choice;

GT gt;

while (1)

{

cout<<"-----------------"<<endl;

cout<<"Book Tree Creation"<<endl;

cout<<"-----------------"<<endl;

cout<<"1.Create"<<endl;

cout<<"2.Display"<<endl;

cout<<"3.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

gt.create\_tree();

case 2:

gt.display(root);

break;

case 3:

exit(1);

default:

cout<<"Wrong choice"<<endl;

}

}

}

Practical No-3 Output

Book Tree Creation

-----------------

1.Create

2.Display

3.Quit

Enter your choice : 1

Enter name of bookr

Enter no. of chapters in book3

Enter Chapter name

datastructure

Enter no. of sections in Chapter: datastructure2

Enter Section 1name

a

Enter Section 2name

b

Enter Chapter name

o

Enter no. of sections in Chapter: o1

Enter Section 1name

p

Enter Chapter name

mp

Enter no. of sections in Chapter: mp1

Enter Section 1name

t

-----Book Hierarchy---

Book title : r

Chapter 1 datastructur?

Sections

a

b

Chapter 2 o

Sections

p

Chapter 3 mp

Sections

t-----------------

Book Tree Creation

-----------------

1.Create

2.Display

3.Quit

Enter your choice : 2

-----Book Hierarchy---

Book title : r

Chapter 1 datastructur?

Sections

a

b

Chapter 2 o

Sections

p

Chapter 3 mp

Sections

t-----------------

Book Tree Creation

-----------------

1.Create

2.Display

3.Quit

Enter your choice : 3

--------------------------------

Process exited after 87.26 seconds with return value 1

Press any key to continue . . .

Practical No-4

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class node

{

public:

class node \*left;

class node \*right;

int data;

};

class tree: public node

{

public:

int stk[50], top;

node \*root;

tree()

{

root = NULL;

top = 0;

}

void insert(int ch)

{

node \*temp, \*temp1;

if (root == NULL)

{

root = new node;

root->data = ch;

root->left = NULL;

root->right = NULL;

return;

}

temp1 = new node;

temp1->data = ch;

temp1->right = temp1->left = NULL;

temp = search(root, ch);

if (temp->data > ch)

temp->left = temp1;

else

temp->right = temp1;

}

node \*search(node \*temp, int ch)

{

if (root == NULL)

{

cout << "no node present";

return NULL;

}

if (temp->left == NULL && temp->right == NULL)

return temp;

if (temp->data > ch)

{

if (temp->left == NULL)

return temp;

search(temp->left, ch);

}

else

{

if (temp->right == NULL)

return temp;

search(temp->right, ch);

}

}

void display(node \*temp)

{

if (temp == NULL)

return;

display(temp->left);

cout << temp->data << " ";

display(temp->right);

}

void postorder(node \*root)

{

node \*p;

p = root;

top = 0;

while (1)

{

while (p != NULL)

{

stk[top] = p->data;

top++;

if (p->right != NULL)

stk[top++] = -p->right->data;

p = p->left;

}

while (stk[top - 1] > 0 || top == 0)

{

if (top == 0)

return;

cout << stk[top - 1] << " ";

p = pop(root);

}

if (stk[top - 1] < 0)

{

stk[top - 1] = -stk[top - 1];

p = pop(root);

}

}

}

node \* pop(node \*p)

{

int ch;

ch = stk[top - 1];

if (p->data == ch)

{

top--;

return p;

}

if (p->data > ch)

pop(p->left);

else

pop(p->right);

}

};

int main(int argc, char \*\*argv)

{

tree t1;

int ch, n, i;

while (1)

{

cout

<< "\n1.INSERT\n2.DISPLAY\n3.POSTORDER TRAVERSE\n4.EXIT\nEnter your choice:";

cin >> ch;

switch (ch)

{

case 1:

cout << "Enter no of elements to insert: ";

cin >> n;

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

{

cin >> ch;

t1.insert(ch);

}

break;

case 2:

t1.display(t1.root);

break;

case 3:

t1.postorder(t1.root);

break;

case 4:

exit(1);

}

}

}

Practical No-4 Output

1.INSERT

2.DISPLAY

3.POSTORDER TRAVERSE

4.EXIT

Enter your choice:1

Enter no of elements to insert: 4

1

2

3

4

1.INSERT

2.DISPLAY

3.POSTORDER TRAVERSE

4.EXIT

Enter your choice:3

4 3 2 1

1.INSERT

2.DISPLAY

3.POSTORDER TRAVERSE

4.EXIT

Enter your choice:4

--------------------------------

Process exited after 21.82 seconds with return value 1

Press any key to continue . . .

Practical No-5

#include <iostream>

#include<string>

using namespace std;

class dictionary;

class node

{

string word,meaning;

node \*left,\*right;

public:

friend class dictionary;

node()

{

left=NULL;

right=NULL;

}

node(string word, string meaning)

{

this->word=word;

this->meaning=meaning;

left=NULL;

right=NULL;

}

};

class dictionary

{

node \*root;

public:

dictionary()

{

root=NULL;

}

void create();

void inorder\_rec(node \*rnode);

void postorder\_rec(node \*rnode);

void inorder()

{

inorder\_rec(root);

}

void postorder();

bool insert(string word,string meaning);

int search(string key);

};

int dictionary::search(string key)

{

node \*tmp=root;

int count;

if(tmp==NULL)

{

return -1;

}

if(root->word==key)

return 1;

while(tmp!=NULL)

{

if((tmp->word)>key)

{

tmp=tmp->left;

count++;

}

else if((tmp->word)<key)

{

tmp=tmp->right;

count++;

}

else if(tmp->word==key)

{

return ++count;

}

}

return -1;

}

void dictionary::postorder()

{

postorder\_rec(root);

}

void dictionary::postorder\_rec(node \*rnode)

{

if(rnode)

{

postorder\_rec(rnode->right);

cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;

postorder\_rec(rnode->left);

}

}

void dictionary::create()

{

int n;

string wordI,meaningI;

cout<<"\nHow many Word to insert?:\n";

cin>>n;

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

{

cout<<"\nENter Word: ";

cin>>wordI;

cout<<"\nEnter Meaning: ";

cin>>meaningI;

insert(wordI,meaningI);

}

}

void dictionary::inorder\_rec(node \*rnode)

{

if(rnode)

{

inorder\_rec(rnode->left);

cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;

inorder\_rec(rnode->right);

}

}

bool dictionary::insert(string word, string meaning)

{

node \*p=new node(word, meaning);

if(root==NULL)

{

root=p;

return true;

}

node \*cur=root;

node \*par=root;

while(cur!=NULL) //traversal

{

if(word>cur->word)

{par=cur;

cur=cur->right;

}

else if(word<cur->word)

{

par=cur;

cur=cur->left;

}

else

{

cout<<"\nWord is already in the dictionary.";

return false;

}

}

if(word>par->word) //insertion of node

{

par->right=p;

return true;

}

else

{

par->left=p;

return true;

}

}

int main() {

string word;

dictionary months;

months.create();

cout<<"Ascending order\n";

months.inorder();

cout<<"\nDescending order:\n";

months.postorder();

cout<<"\nEnter word to search: ";

cin>>word;

int comparisons=months.search(word);

if(comparisons==-1)

{

cout<<"\nNot found word";

}

else

{

cout<<"\n "<<word<<" found in "<<comparisons<<" comparisons";

}

return 0;

}

Practical No-5 Output

How many Word to insert?:

3

ENter Word: january

Enter Meaning: feb

ENter Word: february

Enter Meaning: mar

ENter Word: march

Enter Meaning: march

Ascending order

february : mar

january : feb

march : march

Descending order:

march : march

january : feb

february : mar

Enter word to search: jan

Not found word

--------------------------------

Process exited after 72.25 seconds with return value 0

Press any key to continue . . .

Practical No-6

# include <iostream>

# include <cstdlib>

using namespace std;

class node

{

public:

int info;

struct node \*left;

struct node \*right;

}\*root;

class BST

{

public:

node \*root;

void insert(node \*,node \*);

void display(node \*, int);

int min(node \*);

int height(node \*);

void mirror(node \*);

void preorder(node \*);

void inorder(node \*);

void postorder(node \*);

void search(node \*,int);

BST()

{

root = NULL;

}

};

int main()

{

int choice, num;

BST bst;

node \*temp;

while (1)

{

cout<<"-----------------"<<endl;

cout<<"Operations on BST"<<endl;

cout<<"-----------------"<<endl;

cout<<"1.Insert Element "<<endl;

cout<<"2.Display"<<endl;

cout<<"3.Min value find"<<endl;

cout<<"4.Height"<<endl;

cout<<"5.Mirror of node"<<endl;

cout<<"6.Preorder"<<endl;

cout<<"7.Inorder"<<endl;

cout<<"8.Postorder"<<endl;

cout<<"9.No. of nodes in longest path"<<endl;

cout<<"10.Search an element"<<endl;

cout<<"11.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

temp = new node();

cout<<"Enter the number to be inserted : ";

cin>>temp->info;

bst.insert(bst.root, temp);

break;

case 2:

cout<<"Display BST:"<<endl;

bst.display(bst.root,1);

cout<<endl;

break;

case 3:

cout<<"Min value of tree"<<endl;

cout<<temp->info;

bst.min(bst.root);

cout<<endl;

break;

case 4:

int h;

h=bst.height(bst.root);

cout<<"Height of tree="<<h;

cout<<endl;

break;

case 5:

cout<<"Mirror";

bst.mirror(bst.root);

bst.display(bst.root,1);

break;

case 6:

cout<<" \n Display preorder Binary tree = ";

bst.preorder(bst.root);

cout<<endl;

break;

case 7:

cout<<" \n Display inorder Binary tree = ";

bst.inorder(bst.root);

cout<<endl;

break;

case 8:

cout<<" \n Display postorder Binary tree = ";

bst.postorder(bst.root);

cout<<endl;

break;

case 9:

int nodes;

nodes=bst.height(bst.root);

cout<<"No. of nodes in longest path from root is "<<nodes;

cout<<endl;

break;

case 10:

int searchdata;

cout<<"Enter the element to ne searched:";

cin>>searchdata;

bst.search(bst.root, searchdata);

cout<<endl;

break;

case 11:

exit(1);

default:

cout<<"Wrong choice"<<endl;

}

}

}

void BST::insert(node \*tree, node \*newnode)

{

if (root == NULL)

{

root = new node;

root->info = newnode->info;

root->left = NULL;

root->right = NULL;

cout<<"Root Node is Added"<<endl;

return;

}

if (tree->info == newnode->info)

{

cout<<"Element already in the tree"<<endl;

return;

}

if (tree->info > newnode->info)

{

if (tree->left != NULL)

{

insert(tree->left, newnode);

}

else

{

tree->left = newnode;

(tree->left)->left = NULL;

(tree->left)->right = NULL;

cout<<"Node Added To Left"<<endl;

return;

}

}

else

{

if (tree->right != NULL)

{

insert(tree->right, newnode);

}

else

{

tree->right = newnode;

(tree->right)->left = NULL;

(tree->right)->right = NULL;

cout<<"Node Added To Right"<<endl;

return;

}

}

}

void BST::display(node \*ptr, int level)

{

int i;

if (ptr != NULL)

{

display(ptr->right, level+1);

cout<<endl;

if (ptr == root)

cout<<"Root->: ";

else

{

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

cout<<" ";

}

cout<<ptr->info;

display(ptr->left, level+1);

}

}

int BST::min(node \*root)

{

node \*temp;

if(root==NULL)

{

cout<<"Tree is empty";

}

else

{

temp=root;

while(temp->left!=NULL)

{

temp=temp->left;

}

return(temp->info);

}

}

int BST::height(node \*root)

{

int htleft,htright;

if(root==NULL)

{

//cout<<"Tree is empty"<<endl;

return(0);

}

else if(root->left==NULL && root->right==NULL)

{

return(1);

}

htleft=height(root->left);

htright=height(root->right);

if(htright>=htleft)

{

return(htright+1);

}

else

{

return(htleft+1);

}

}

void BST::mirror(node \*root)

{

node \*temp;

if(root!=NULL)

{

temp=root->left;

root->left=root->right;

root->right=temp;

mirror(root->left);

mirror(root->right);

}

}

void BST::preorder(node \*ptr)

{

if(ptr!=NULL)

{

cout<<ptr->info<<"\t";

preorder(ptr->left);

preorder(ptr->right);

cout<<endl;

}

}

void BST::inorder(node \*ptr)

{

if(ptr!=NULL)

{

inorder(ptr->left);

cout<<ptr->info<<"\t";

inorder(ptr->right);

cout<<endl;

}

}

void BST::postorder(node \*ptr)

{

if(ptr!=NULL)

{

postorder(ptr->left);

postorder(ptr->right);

cout<<ptr->info<<"\t";

cout<<endl;

}

}

void BST::search(node \*ptr, int searchdata)

{

if (ptr->info==searchdata)

{

cout<<"Element Found..."<<endl;

}

else if (ptr->info<searchdata && ptr->right!=NULL)

{

search(ptr->right, searchdata);

}

else if (ptr->info>searchdata && ptr->left!=NULL)

{

search(ptr->left, searchdata);

}

else

{

cout<<"Element not found..."<<endl;

}

}

Practical No-6 Output

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 1

Enter the number to be inserted : 3

Root Node is Added

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 2

Display BST:

Root->: 3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 3

Min value of tree

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 4

Height of tree=1

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 5

Mirror

Root->: 3-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 6

Display preorder Binary tree = 3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 7

Display inorder Binary tree = 3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 8

Display postorder Binary tree = 3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 9

No. of nodes in longest path from root is 1

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Enter your choice : 10

Enter the element to ne searched:3

Element Found...

-----------------

Operations on BST

-----------------

1.Insert Element

2.Display

3.Min value find

4.Height

5.Mirror of node

6.Preorder

7.Inorder

8.Postorder

9.No. of nodes in longest path

10.Search an element

11.Quit

Practical No-7

#include<iostream>

#include<stdlib.h>

#include<string.h>

using namespace std;

struct node

{ string vertex;

int time;

node \*next;

};

class adjmatlist

{ int m[10][10],n,i,j; char ch; string v[20]; node \*head[20]; node \*temp=NULL;

public:

adjmatlist()

{ for(i=0;i<20;i++)

{ head[i]=NULL; }

}

void getgraph();

void adjlist();

void displaym();

void displaya();

};

void adjmatlist::getgraph()

{

cout<<"\n enter no. of cities(max. 20)";

cin>>n;

cout<<"\n enter name of cities";

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

cin>>v[i];

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

{

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

{ cout<<"\n if path is present between city "<<v[i]<<" and "<<v[j]<<" then press enter y otherwise n";

cin>>ch;

if(ch=='y')

{

cout<<"\n enter time required to reach city "<<v[j]<<" from "<<v[i]<<" in minutes";

cin>>m[i][j];

}

else if(ch=='n')

{ m[i][j]=0; }

else

{ cout<<"\n unknown entry"; }

}

}

adjlist();

}

void adjmatlist::adjlist()

{ cout<<"\n \*\*\*\*";

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

{ node \*p=new(struct node);

p->next=NULL;

p->vertex=v[i];

head[i]=p; cout<<"\n"<<head[i]->vertex;

}

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

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

{

if(m[i][j]!=0)

{

node \*p=new(struct node);

p->vertex=v[j];

p->time=m[i][j];

p->next=NULL;

if(head[i]->next==NULL)

{ head[i]->next=p; }

else

{ temp=head[i];

while(temp->next!=NULL)

{ temp=temp->next; }

temp->next=p;

}

}

}

}

}

void adjmatlist::displaym()

{ cout<<"\n";

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

{ cout<<"\t"<<v[j]; }

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

{ cout<<"\n "<<v[i];

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

{ cout<<"\t"<<m[i][j];

}

cout<<"\n";

}

}

void adjmatlist::displaya()

{

cout<<"\n adjacency list is";

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

{

if(head[i]==NULL)

{ cout<<"\n adjacency list not present"; break; }

else

{

cout<<"\n"<<head[i]->vertex;

temp=head[i]->next;

while(temp!=NULL)

{ cout<<"-> "<<temp->vertex;

temp=temp->next; }

}

}

cout<<"\n path and time required to reach cities is";

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

{

if(head[i]==NULL)

{ cout<<"\n adjacency list not present"; break; }

else

{

temp=head[i]->next;

while(temp!=NULL)

{ cout<<"\n"<<head[i]->vertex;

cout<<"-> "<<temp->vertex<<"\n [time required: "<<temp->time<<" min ]";

temp=temp->next; }

}

}

}

int main()

{ int m;

adjmatlist a;

while(1)

{

cout<<"\n\n enter the choice";

cout<<"\n 1.enter graph";

cout<<"\n 2.display adjacency matrix for cities";

cout<<"\n 3.display adjacency list for cities";

cout<<"\n 4.exit";

cin>>m;

switch(m)

{ case 1: a.getgraph();

break;

case 2: a.displaym();

break;

case 3: a.displaya();

break;

case 4: exit(0);

default: cout<<"\n unknown choice";

}

}

return 0;

}

Practical No-7 Output

enter the choice

1.enter graph

2.display adjacency matrix for cities

3.display adjacency list for cities

4.exit1

enter no. of cities(max. 20)1

enter name of citiesmumbai

if path is present between city mumbai and mumbai then press enter y otherwise ny

enter time required to reach city mumbai from mumbai in minutes460

\*\*\*\*

mumbai

enter the choice

1.enter graph

2.display adjacency matrix for cities

3.display adjacency list for cities

4.exit2

mumbai

mumbai 460

enter the choice

1.enter graph

2.display adjacency matrix for cities

3.display adjacency list for cities

4.exit3

adjacency list is

mumbai-> mumbai

path and time required to reach cities is

mumbai-> mumbai

[time required: 460 min ]

enter the choice

1.enter graph

2.display adjacency matrix for cities

3.display adjacency list for cities

4.exit4

--------------------------------

Process exited after 27.25 seconds with return value 0

Press any key to continue . . .

Practical No-8

// A naive recursive implementation of

// optimal binary search tree problem

#include <bits/stdc++.h>

using namespace std;

// A utility function to get sum of

// array elements freq[i] to freq[j]

int sum(int freq[], int i, int j);

// A recursive function to calculate

// cost of optimal binary search tree

int optCost(int freq[], int i, int j)

{

// Base cases

if (j < i) // no elements in this subarray

return 0;

if (j == i) // one element in this subarray

return freq[i];

// Get sum of freq[i], freq[i+1], ... freq[j]

int fsum = sum(freq, i, j);

// Initialize minimum value

int min = INT\_MAX;

// One by one consider all elements

// as root and recursively find cost

// of the BST, compare the cost with

// min and update min if needed

for (int r = i; r <= j; ++r)

{

int cost = optCost(freq, i, r - 1) +

optCost(freq, r + 1, j);

if (cost < min)

min = cost;

}

// Return minimum value

return min + fsum;

}

// The main function that calculates

// minimum cost of a Binary Search Tree.

// It mainly uses optCost() to find

// the optimal cost.

int optimalSearchTree(int keys[],

int freq[], int n)

{

// Here array keys[] is assumed to be

// sorted in increasing order. If keys[]

// is not sorted, then add code to sort

// keys, and rearrange freq[] accordingly.

return optCost(freq, 0, n - 1);

}

// A utility function to get sum of

// array elements freq[i] to freq[j]

int sum(int freq[], int i, int j)

{

int s = 0;

for (int k = i; k <= j; k++)

s += freq[k];

return s;

}

// Driver Code

int main()

{

int keys[] = {10, 12, 20};

int freq[] = {34, 8, 50};

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

cout << "Cost of Optimal BST is "

<< optimalSearchTree(keys, freq, n);

return 0;

}

Practical No-8 Output

Cost of Optimal BST is 142

--------------------------------

Process exited after 5.059 seconds with return value 0

Press any key to continue . . .

Practical No-9

#include"iostream"

#include<string.h>

using namespace std;

//////////////////////////////////////////////////////////////

// //

// Name : Vivek S. Sharma //

// Title : BST //

// //

// //

//////////////////////////////////////////////////////////////

typedef struct node

{

char k[20];

char m[20];

class node \*left;

class node \* right;

}node;

class dict

{

public:

node \*root;

void create();

void disp(node \*);

void insert(node \* root,node \*temp);

int search(node \*,char []);

int update(node \*,char []);

node\* del(node \*,char []);

node \* min(node \*);

};

void dict :: create()

{

class node \*temp;

int ch;

do

{

temp = new node;

cout<<"\nEnter Keyword:";

cin>>temp->k;

cout<<"\nEnter Meaning:";

cin>>temp->m;

temp->left = NULL;

temp->right = NULL;

if(root == NULL)

{

root = temp;

}

else

{

insert(root, temp);

}

cout<<"\nDo u want to add more (y=1/n=0):";

cin>>ch;

}

while(ch == 1);

}

void dict :: insert(node \* root,node \*temp)

{

if(strcmp (temp->k, root->k) < 0 )

{

if(root->left == NULL)

root->left = temp;

else

insert(root->left,temp);

}

else

{ if(root->right == NULL)

root->right = temp;

else

insert(root->right,temp);

}

}

void dict:: disp(node \* root)

{

if( root != NULL)

{

disp(root->left);

cout<<"\n Key Word :"<<root->k;

cout<<"\t Meaning :"<<root->m;

disp(root->right);

}

}

int dict :: search(node \* root,char k[20])

{

int c=0;

while(root != NULL)

{

c++;

if(strcmp (k,root->k) == 0)

{

cout<<"\nNo of Comparisons:"<<c;

return 1;

}

if(strcmp (k, root->k) < 0)

root = root->left;

if(strcmp (k, root->k) > 0)

root = root->right;

}

return -1;

}

int dict :: update(node \* root,char k[20])

{

while(root != NULL)

{

if(strcmp (k,root->k) == 0)

{

cout<<"\nEnter New Meaning ofKeyword"<<root->k;

cin>>root->m;

return 1;

}

if(strcmp (k, root->k) < 0)

root = root->left;

if(strcmp (k, root->k) > 0)

root = root->right;

}

return -1;

}

node\* dict :: del(node \* root,char k[20])

{

node \*temp;

if(root == NULL)

{

cout<<"\nElement No Found";

return root;

}

if (strcmp(k,root->k) < 0)

{

root->left = del(root->left, k);

return root;

}

if (strcmp(k,root->k) > 0)

{

root->right = del(root->right, k);

return root;

}

if (root->right==NULL&&root->left==NULL)

{

temp = root;

delete temp;

return NULL;

}

if(root->right==NULL)

{

temp = root;

root = root->left;

delete temp;

return root;

}

else if(root->left==NULL)

{

temp = root;

root = root->right;

delete temp;

return root;

}

temp = min(root->right);

strcpy(root->k,temp->k);

root->right = del(root->right, temp->k);

return root;

}

node \* dict :: min(node \*q)

{

while(q->left != NULL)

{

q = q->left;

}

return q;

}

int main()

{

int ch;

dict d;

d.root = NULL;

do

{

cout<<"\nMenu\n1.Create\n2.Disp\n3.Search\n4.Update\n5.Delete\nEnter Ur CH:";

cin>>ch;

switch(ch)

{

case 1: d.create();

break;

case 2: if(d.root == NULL)

{

cout<<"\nNo any Keyword";

}

else

{

d.disp(d.root);

}

break;

case 3: if(d.root == NULL)

{

cout<<"\nDictionary is Empty. First add keywords then try again ";

}

else

{

cout<<"\nEnter Keyword which u want to search:";

char k[20];

cin>>k;

if( d.search(d.root,k) == 1)

cout<<"\nKeyword Found";

else

cout<<"\nKeyword Not Found";

}

break;

case 4:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty. First add keywords then try again ";

}

else

{

cout<<"\nEnter Keyword which meaning want to update:";

char k[20];

cin>>k;

if(d.update(d.root,k) == 1)

cout<<"\nMeaning Updated";

else

cout<<"\nMeaning Not Found";

}

break;

case 5:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty. First add keywords then try again ";

}

else

{

cout<<"\nEnter Keyword which u want to delete:";

char k[20];

cin>>k;

if(d.root == NULL)

{

cout<<"\nNo any Keyword";

}

else

{

d.root = d.del(d.root,k);

}

}

}

}

while(ch<=5);

return 0;

}

Practical No-9 Output

Menu

1.Create

2.Disp

3.Search

4.Update

5.Delete

Enter Ur CH:1

Enter Keyword:ch

Enter Meaning:character

Do u want to add more (y=1/n=0):2

Menu

1.Create

2.Disp

3.Search

4.Update

5.Delete

Enter Ur CH:3

Enter Keyword which u want to search:ch

No of Comparisons:1

Keyword Found

Menu

1.Create

2.Disp

3.Search

4.Update

5.Delete

Enter Ur CH:4

Enter Keyword which meaning want to update:ch

Enter New Meaning ofKeywordchword

Meaning Updated

Menu

1.Create

2.Disp

3.Search

4.Update

5.Delete

Enter Ur CH:5

Enter Keyword which u want to delete:ch

Menu

1.Create

2.Disp

3.Search

4.Update

5.Delete

Enter Ur CH:

Practical No-10

#include<iostream>

using namespace std;

class hp

{

int heap[20],heap1[20],x,n1,i;

public:

hp()

{ heap[0]=0; heap1[0]=0;

}

void getdata();

void insert1(int heap[],int);

void upadjust1(int heap[],int);

void insert2(int heap1[],int);

void upadjust2(int heap1[],int);

void minmax();

};

void hp::getdata()

{

cout<<"\n enter the no. of students";

cin>>n1;

cout<<"\n enter the marks";

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

{ cin>>x;

insert1(heap,x);

insert2(heap1,x);

}

}

void hp::insert1(int heap[20],int x)

{

int n;

n=heap[0];

heap[n+1]=x;

heap[0]=n+1;

upadjust1(heap,n+1);

}

void hp::upadjust1(int heap[20],int i)

{

int temp;

while(i>1&&heap[i]>heap[i/2])

{

temp=heap[i];

heap[i]=heap[i/2];

heap[i/2]=temp;

i=i/2;

}

}

void hp::insert2(int heap1[20],int x)

{

int n;

n=heap1[0];

heap1[n+1]=x;

heap1[0]=n+1;

upadjust2(heap1,n+1);

}

void hp::upadjust2(int heap1[20],int i)

{

int temp1;

while(i>1&&heap1[i]<heap1[i/2])

{

temp1=heap1[i];

heap1[i]=heap1[i/2];

heap1[i/2]=temp1;

i=i/2;

}

}

void hp::minmax()

{

cout<<"\n max marks"<<heap[1];

cout<<"\n##";

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

{ cout<<"\n"<<heap[i]; }

cout<<"\n min marks"<<heap1[1];

cout<<"\n##";

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

{ cout<<"\n"<<heap1[i]; }

}

int main()

{

hp h;

h.getdata();

h.minmax();

return 0;

}

Practical No-10 Output

enter the no. of students2

enter the marks40

45

max marks45

##

2

45

40

min marks40

##

2

40

45

--------------------------------

Process exited after 25.48 seconds with return value 0

Practical No-11

#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

class tel

{

public:

int rollNo,roll1;

char name[10];

char div;

char address[20];

void accept()

{

cout<<"\n\tEnter Roll Number : ";

cin>>rollNo;

cout<<"\n\tEnter the Name : ";

cin>>name;

cout<<"\n\tEnter the Division:";

cin>>div;

cout<<"\n\tEnter the Address:";

cin>>address;

}

void accept2()

{

cout<<"\n\tEnter the Roll No. to modify : ";

cin>>rollNo;

}

void accept3()

{

cout<<"\n\tEnter the name to modify : ";

cin>>name;

}

int getRollNo()

{

return rollNo;

}

void show()

{

cout<<"\n\t"<<rollNo<<"\t\t"<<name<<"\t\t"<<div<<"\t\t"<<address;

}

};

int main()

{

int i,n,ch,ch1,rec,start,count,add,n1,add2,start2,n2,y,a,b,on,oname,add3,start3,n3,y1,add4,start4,n4;

char name[20],name2[20];

tel t1;

count=0;

fstream g,f;

do

{

cout<<"\n>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<";

cout<<"\n1.Insert and overwrite\n2.Show\n3.Search & Edit(number)\n4.Search & Edit(name)\n5.Search & Edit(onlynumber)\n6.Search & edit(only name)\n 7.Delete a Student Record\n 8.Exit\n\tEnter the Choice\t:";

cin>>ch;

switch(ch)

{

case 1:

f.open("StuRecord.txt",ios::out);

x:t1.accept();

f.write((char\*) &t1,(sizeof(t1)));

cout<<"\nDo you want to enter more records?\n1.Yes\n2.No";

cin>>ch1;

if(ch1==1)

goto x;

else

{

f.close();

break;

}

case 2:

f.open("StuRecord.txt",ios::in);

f.read((char\*) &t1,(sizeof(t1)));

//cout<<"\n\tRoll No.\t\tName \t\t Division \t\t Address";

while(f)

{

t1.show();

f.read((char\*) &t1,(sizeof(t1)));

}

f.close();

break;

case 3:

cout<<"\nEnter the roll number you want to find";

cin>>rec;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*)&t1,(sizeof(t1)));

while(f)

{

if(rec==t1.rollNo)

{

cout<<"\nRecord found";

add=f.tellg();

f.seekg(0,ios::beg);

start=f.tellg();

n1=(add-start)/(sizeof(t1));

f.seekp((n1-1)\*sizeof(t1),ios::beg);

t1.accept();

f.write((char\*) &t1,(sizeof(t1)));

f.close();

count++;

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

if(count==0)

cout<<"\nRecord not found";

f.close();

break;

case 4:

cout<<"\nEnter the name you want to find and edit";

cin>>name;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*)&t1,(sizeof(t1)));

while(f)

{

y=(strcmp(name,t1.name));

if(y==0)

{

cout<<"\nName found";

add2=f.tellg();

f.seekg(0,ios::beg);

start2=f.tellg();

n2=(add2-start2)/(sizeof(t1));

f.seekp((n2-1)\*sizeof(t1),ios::beg);

t1.accept();

f.write((char\*) &t1,(sizeof(t1)));

f.close();

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

break;

case 5:

cout<<"\n\tEnter the roll number you want to modify";

cin>>on;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*) &t1,(sizeof(t1)));

while(f)

{

if(on==t1.rollNo)

{

cout<<"\n\tNumber found";

add3=f.tellg();

f.seekg(0,ios::beg);

start3=f.tellg();

n3=(add3-start3)/(sizeof(t1));

f.seekp((n3-1)\*(sizeof(t1)),ios::beg);

t1.accept2();

f.write((char\*)&t1,(sizeof(t1)));

f.close();

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

break;

case 6:

cout<<"\nEnter the name you want to find and edit";

cin>>name2;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*)&t1,(sizeof(t1)));

while(f)

{

y1=(strcmp(name2,t1.name));

if(y1==0)

{

cout<<"\nName found";

add4=f.tellg();

f.seekg(0,ios::beg);

start4=f.tellg();

n4=(add4-start4)/(sizeof(t1));

f.seekp((n4-1)\*sizeof(t1),ios::beg);

t1.accept3();

f.write((char\*) &t1,(sizeof(t1)));

f.close();

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

break;

case 7:

int roll;

cout<<"Please Enter the Roll No. of Student Whose Info You Want to Delete: ";

cin>>roll;

f.open("StuRecord.txt",ios::in);

g.open("temp.txt",ios::out);

f.read((char \*)&t1,sizeof(t1));

while(!f.eof())

{

if (t1.getRollNo() != roll)

g.write((char \*)&t1,sizeof(t1));

f.read((char \*)&t1,sizeof(t1));

}

cout << "The record with the roll no. " << roll << " has been deleted " << endl;

f.close();

g.close();

remove("StuRecord.txt");

rename("temp.txt","StuRecord.txt");

break;

case 8:

cout<<"\n\tThank you";

break;

}

}while(ch!=8);

}

Practical No-11 Output

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :1

Enter Roll Number : 46

Enter the Name : divya

Enter the Division:a

Enter the Address:junner

Do you want to enter more records?

1.Yes

2.No1

Enter Roll Number : 56

Enter the Name : hrutu

Enter the Division:g

Enter the Address:murbad

Do you want to enter more records?

1.Yes

2.No2

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :2

46 divya a junner

56 hrutu g murbad

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :3

Enter the roll number you want to find46

Record found

Enter Roll Number : 56

Enter the Name : hrutu

Enter the Division:g

Enter the Address:murbad

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :4

Enter the name you want to find and edithrutu

Name found

Enter Roll Number : 56

Enter the Name : hrutu

Enter the Division:g

Enter the Address:murbad

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :5

Enter the roll number you want to modify56

Number found

Enter the Roll No. to modify : 56

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :

7

Please Enter the Roll No. of Student Whose Info You Want to Delete:

7

The record with the roll no. 7 has been deleted

>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<

1.Insert and overwrite

2.Show

3.Search & Edit(number)

4.Search & Edit(name)

5.Search & Edit(onlynumber)

6.Search & edit(only name)

7.Delete a Student Record

8.Exit

Enter the Choice :8

Thank you

--------------------------------

Process exited after 188.7 seconds with return value 0

Press any key to continue . . .

Practical No-12

// C++ program to implement

// external sorting using

// merge sort

#include <bits/stdc++.h>

using namespace std;

struct MinHeapNode {

// The element to be stored

int element;

// index of the array from which

// the element is taken

int i;

};

// Prototype of a utility function

// to swap two min heap nodes

void swap(MinHeapNode\* x, MinHeapNode\* y);

// A class for Min Heap

class MinHeap {

// pointer to array of elements in heap

MinHeapNode\* harr;

// size of min heap

int heap\_size;

public:

// Constructor: creates a min

// heap of given size

MinHeap(MinHeapNode a[], int size);

// to heapify a subtree with

// root at given index

void MinHeapify(int);

// to get index of left child

// of node at index i

int left(int i) { return (2 \* i + 1); }

// to get index of right child

// of node at index i

int right(int i) { return (2 \* i + 2); }

// to get the root

MinHeapNode getMin() { return harr[0]; }

// to replace root with new node

// x and heapify() new root

void replaceMin(MinHeapNode x)

{

harr[0] = x;

MinHeapify(0);

}

};

// Constructor: Builds a heap from

// a given array a[] of given size

MinHeap::MinHeap(MinHeapNode a[], int size)

{

heap\_size = size;

harr = a; // store address of array

int i = (heap\_size - 1) / 2;

while (i >= 0) {

MinHeapify(i);

i--;

}

}

// A recursive method to heapify

// a subtree with root

// at given index. This method

// assumes that the

// subtrees are already heapified

void MinHeap::MinHeapify(int i)

{

int l = left(i);

int r = right(i);

int smallest = i;

if (l < heap\_size && harr[l].element < harr[i].element)

smallest = l;

if (r < heap\_size && harr[r].element < harr[smallest].element)

smallest = r;

if (smallest != i) {

swap(&harr[i], &harr[smallest]);

MinHeapify(smallest);

}

}

// A utility function to swap two elements

void swap(MinHeapNode\* x, MinHeapNode\* y)

{

MinHeapNode temp = \*x;

\*x = \*y;

\*y = temp;

}

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

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

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

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

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

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

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

/\* Merge the temp arrays back into arr[l..r]\*/

// Initial index of first subarray

i = 0;

// Initial index of second subarray

j = 0;

// Initial index of merged subarray

k = l;

while (i < n1 && j < n2) {

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

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

else

arr[k++] = R[j++];

}

/\* Copy the remaining elements of L[],

if there are any \*/

while (i < n1)

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

/\* Copy the remaining elements of R[],

if there are any \*/

while (j < n2)

arr[k++] = R[j++];

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

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

{

if (l < r) {

// Same as (l+r)/2, but avoids overflow for

// large l and h

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

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

FILE\* openFile(char\* fileName, char\* mode)

{

FILE\* fp = fopen(fileName, mode);

if (fp == NULL) {

perror("Error while opening the file.\n");

exit(EXIT\_FAILURE);

}

return fp;

}

// Merges k sorted files. Names of files are assumed

// to be 1, 2, 3, ... k

void mergeFiles(char\* output\_file, int n, int k)

{

FILE\* in[k];

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

char fileName[2];

// convert i to string

snprintf(fileName, sizeof(fileName),

"%d", i);

// Open output files in read mode.

in[i] = openFile(fileName, "r");

}

// FINAL OUTPUT FILE

FILE\* out = openFile(output\_file, "w");

// Create a min heap with k heap

// nodes. Every heap node

// has first element of scratch

// output file

MinHeapNode\* harr = new MinHeapNode[k];

int i;

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

// break if no output file is empty and

// index i will be no. of input files

if (fscanf(in[i], "%d ", &harr[i].element) != 1)

break;

// Index of scratch output file

harr[i].i = i;

}

// Create the heap

MinHeap hp(harr, i);

int count = 0;

// Now one by one get the

// minimum element from min

// heap and replace it with

// next element.

// run till all filled input

// files reach EOF

while (count != i) {

// Get the minimum element

// and store it in output file

MinHeapNode root = hp.getMin();

fprintf(out, "%d ", root.element);

// Find the next element that

// will replace current

// root of heap. The next element

// belongs to same

// input file as the current min element.

if (fscanf(in[root.i], "%d ",

&root.element)

!= 1) {

root.element = INT\_MAX;

count++;

}

// Replace root with next

// element of input file

hp.replaceMin(root);

}

// close input and output files

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

fclose(in[i]);

fclose(out);

}

// Using a merge-sort algorithm,

// create the initial runs

// and divide them evenly among

// the output files

void createInitialRuns(

char\* input\_file, int run\_size,

int num\_ways)

{

// For big input file

FILE\* in = openFile(input\_file, "r");

// output scratch files

FILE\* out[num\_ways];

char fileName[2];

for (int i = 0; i < num\_ways; i++) {

// convert i to string

snprintf(fileName, sizeof(fileName),

"%d", i);

// Open output files in write mode.

out[i] = openFile(fileName, "w");

}

// allocate a dynamic array large enough

// to accommodate runs of size run\_size

int\* arr = (int\*)malloc(

run\_size \* sizeof(int));

bool more\_input = true;

int next\_output\_file = 0;

int i;

while (more\_input) {

// write run\_size elements

// into arr from input file

for (i = 0; i < run\_size; i++) {

if (fscanf(in, "%d ", &arr[i]) != 1) {

more\_input = false;

break;

}

}

// sort array using merge sort

mergeSort(arr, 0, i - 1);

// write the records to the

// appropriate scratch output file

// can't assume that the loop

// runs to run\_size

// since the last run's length

// may be less than run\_size

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

fprintf(out[next\_output\_file],

"%d ", arr[j]);

next\_output\_file++;

}

// close input and output files

for (int i = 0; i < num\_ways; i++)

fclose(out[i]);

fclose(in);

}

// For sorting data stored on disk

void externalSort(

char\* input\_file, char\* output\_file,

int num\_ways, int run\_size)

{

// read the input file,

// create the initial runs,

// and assign the runs to

// the scratch output files

createInitialRuns(input\_file,

run\_size, num\_ways);

// Merge the runs using

// the K-way merging

mergeFiles(output\_file, run\_size, num\_ways);

}

// Driver program to test above

int main()

{

// No. of Partitions of input file.

int num\_ways = 10;

// The size of each partition

int run\_size = 1000;

char input\_file[] = "input.txt";

char output\_file[] = "output.txt";

FILE\* in = openFile(input\_file, "w");

srand(time(NULL));

// generate input

for (int i = 0; i < num\_ways \* run\_size; i++)

fprintf(in, "%d ", rand());

fclose(in);

externalSort(input\_file, output\_file, num\_ways,

run\_size);

return 0;

}