## ASSIGNMENT NO- 1

A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

Code:-

**# 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** BST

{ **public**:

### void create\_tree(); void display(node \* r1);

### BST()

{

root = NULL;

}

};

### void BST::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 BST::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;

}

} }

}

### int main()

{ **int** choice; BST bst; **while** (1)

{

cout<<"-----------------"<<**endl**; cout<<"Book Tree Creation"<<**endl**; cout<<"-----------------"<<**endl**; cout<<"1.Create"<<**endl**;

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

cout<<"Enter your choice : ";

cin>>choice;

**switch**(choice)

{

**case** 1: bst.create\_tree();

**break**; **case** 2: bst.display(root); **break**; **default**:

cout<<"Wrong choice"<<**endl**;

**break**;

} } }

**ASSIGNMENT NO-2**

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree

-i.insert new node ii.Find number of nodes in longest path iii.Minimum data value found in the tree

iv.Change a tree so that the roles of the left and right pointers are swapped at every node v.Seach tree

**#include**<iostream> **#include**<cstdlib> **using** **namespace** std;

**class** node

{

**public**: **int** data; node \*lchild,\*rchild; **node**();

};

**node::node**()

{

lchild=rchild=NULL;

}

**class** BST

{

node \*root; **public**:

### BST()

{

root=NULL;

}

### void create();

### void Inorder(node \*);

### void Smallest();

### int Height(node \*); void search(node \*,int); node\* getroot();

node\* **mirror**(node \*);

};

### node\* BST::getroot()

{

**return** root;

}

### void BST::create()

{

**char** ans;

node \*temp;

**do**

{

temp=**new** node(); cout<<"Enter data:"; cin>>temp->data;

**if**(root==NULL)

{

root=temp;

} **else**

{

node \*trav,\*parent; trav=root;

parent=trav;

**while**(trav!=NULL)

{

**if**(temp->data<trav->data)

{

parent=trav;

trav=trav->lchild;

} **else**

{

parent=trav;

trav=trav->rchild;

}

}

**if**(temp->data<parent->data)

{

parent->lchild=temp;

} **else**

{

parent->rchild=temp;

}

}cout<<"ADD ANOTHER..(y/n)"; cin>>ans; }**while**(ans=='y'||ans=='Y');

}

### void BST::Inorder(node \*n)

{

**if**(n!=NULL)

{

Inorder(n->lchild); cout<<**endl**<<n->data<<" ";

Inorder(n->rchild);

}

}

### void BST::Smallest()

{

node \*temp; temp=root;

**while**(temp->lchild!=NULL)

{

temp=temp->lchild;

}

cout<<"The Smallest element is:"<<temp->data<<"";

}

### int BST::Height(node \*n)

{

**if**(n==NULL)

{

**return** 0;

}

**int** lheight=Height(n->lchild);

**int** rheight=Height(n->rchild);

**if**(lheight>rheight) **return** (1+lheight); **else**

**return** (1+rheight);

}

### void BST::search(node \*n,int temp)

{

**if**(n!=NULL)

{

**if**(temp < n->data)

{

search(n->lchild,temp);

}

**else** **if**(temp > n->data)

{

search(n->rchild,temp);

} **else**

{

cout<<**endl**<<"Entered element found.";

}

} **else**

{

cout<<**endl**<<"Element not Found...!";

}

}

### node \*BST::mirror(node \*n)

{

**if**(n==NULL)

{

**return** 0;

} **else**

{

node \*temp; mirror(n->lchild); mirror(n->rchild);

temp=n->lchild; n->lchild=n->rchild; n->rchild=temp;

**return** n;}

}

### int main()

{

BST s; **int** ch,a,temp;

cout<<"\n\*\*\*\*\*\*\*\*\*\*TREE\*\*\*\*\*\*\*\*\*\*\*";

cout<<**endl**<<"CREATING A TREE "<<**endl**;

s.create();

**while**(1)

{ cout<<**endl**<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n1.INSERT\n2.NUMBERS OF NODES IN LONGEST PATH\n3.MINIMUM DATA VALUE IN A TREE\n4.SWAPPING\n5.SEARCH\n"

"7.DISPLAY\n8.EXIT \*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<**endl**

<<"ENTER YOUR CHOICE: "; cin>>ch;

**switch**(ch)

{

**case** 1:

cout<<**endl**<<"\nFOR INSERTION"<<**endl**;

s.create(); **break**;

**case** 2:a=s.Height(s.getroot());

cout<<**endl**<<"Longest path is of "<<a<<" nodes"<<**endl**; **break**;

**case** 3:s.Smallest();

**break**;

**case** 4:s.mirror(s.getroot()); s.Inorder(s.getroot());

**break**;

**case** 5:cout<<**endl**<<"Enter element to search:"; cin>>temp;

s.search(s.getroot(),temp);

**break**;

**case** 7:

cout<<**endl**<<"Displaying in inorder: "; s.Inorder(s.getroot());

**break**;

**case** 8: **exit**(0);

**break**;

}

}

**return** 0;

}

**ASSIGNMENT NO-3**

For given expression eg. a-b\*c-d/e+f construct inorder sequence and traverse it using postorder traversal(non recursive). #include**<string>** **#include** <iostream> **using** **namespace** std; **class** node;

**class** tree; **const** **int** MAX=50;

//===========Stack ==========================

**class** Stack

{

**int** top;

node \*info[MAX];

**public**: **Stack**() { top=-1;

}

**void** **push**(node \*cnode)

{

// cout<<"here\n";

// if(!isFull())

// {

// cout<<"in push\n"; top=top+1; info[top]=cnode;

// else

// {

// cout<<"\nStack Overflow\n";

// }

}

node \***Top**()

{

**return** info[top];

}

node \* **pop**()

{

**if**(!empty())

{

**return** info[top--];

}

**return** NULL;

}

### bool empty()

{

**if**(top==-1) **return** **true**; **else**

**return** **false**;

}

### bool isFull()

{

**if**(top==MAX-1) **return** **true**; **else**

**return** **false**;

}

};

//==================================================

**class** node

{ node \*left,\*right; **char** data; **public**: **node**()

{

left=right=NULL;

}

### node(char ch)

{

left=right=NULL;

data=ch;

}

**friend** **class** tree;

};

**class** tree

{

node \*root;

**public**: **tree**()

{

root=NULL;

### } void create(string str); void inorder\_rec(node \*rnode); void preorder\_rec(node \*rnode); void postorder\_rec(node \*rnode); void inorderNonRec();

### void preorderNonRec();

### void postorderNon();

### void postorder()

{

postorder\_rec(root);

}

### void inorder()

{

inorder\_rec(root);

}

### void preorder()

{

preorder\_rec(root);

}

### int priority(char ch);

};

### int tree::priority(char ch)

{

**switch**(ch)

{ **case** '+': **case** '-': **return** 0; **break**; **case** '\*': **case** '/': **return** 1; **break**; **case** '^': **return** 2;

**break**;

}

**return** -1;

}

### void tree::postorderNon()

{

Stack s1; node \*ptr=root; **int** arr[MAX],i=-1,flag; LOOP:**while**(ptr!=NULL)

{

s1.push(ptr); arr[++i]=0;

**if**(ptr->right!=NULL)

{

s1.push(ptr->right);

arr[++i]=1;

}

ptr=ptr->left;

}

ptr=s1.pop();

flag=arr[i--];

**while**(flag==0&& i>=0)

{

cout<<" "<<ptr->data; ptr=s1.pop(); flag=arr[i--];

}

**if**(flag==1 && i>=0)

{

**goto** LOOP;

}

cout<<" "<<ptr->data;

}

### void tree::preorderNonRec() {

Stack s1; node \*ptr=root;

**while**(ptr!=NULL)

{

cout<<" "<<ptr->data; **if**(ptr->right!=NULL) s1.push(ptr->right); **if**(ptr->left!=NULL) ptr=ptr->left; **else**

ptr=s1.pop();

}

}

### void tree::inorderNonRec()

{

node \*ptr=root; Stack s1;

X:**while**(ptr!=NULL)

{

s1.push(ptr);

ptr=ptr->left;

}

ptr=s1.pop();

**while**(ptr!=NULL)

{

cout<<" "<<ptr->data;

**if**(ptr->right!=NULL)

{

ptr=ptr->right;

**goto** X;

}

ptr=s1.pop();

}

}

### void tree::inorder\_rec(node \*rnode)

{

**if**(rnode)

{

inorder\_rec(rnode->left); cout<<" "<<rnode->data;

inorder\_rec(rnode->right);

}

}

### void tree::preorder\_rec(node \*rnode)

{

**if**(rnode)

{

cout<<" "<<rnode->data; preorder\_rec(rnode->left);

preorder\_rec(rnode->right);

}

}

**void** **tree::postorder\_rec**(node \*rnode) { **if**(rnode)

{

postorder\_rec(rnode->left); postorder\_rec(rnode->right);

cout<<" "<<rnode->data;

### } } void tree::create(string str) {

Stack s1,s2; **int** i=0; **char** ch;

**while**(str[i]!='\0')

{

//cout<<"in create()"; ch=str[i];

**if**(**isalpha**(ch)) //s1 operand stack and s2===operator

{

node \*temp=**new** node(ch);

s1.push(temp);

}

**else** //operator

{

//cout<<"in operator block()"; **if**(s2.empty())

{

node \*temp=**new** node(ch);

s2.push(temp);

}

**else** **if**(priority(ch)>priority(s2.Top()->data))

{

node \*temp=**new** node(ch);

s2.push(temp);

}

**else**

{

**while**(!s2.empty()&&priority(ch)<=priority(s2.Top()->data) )

{

node \*op=s2.pop(); node \*rchild=s1.pop();

node \*lchild=s1.pop();

op->right=rchild; op->left=lchild;

s1.push(op);

}

s2.push(**new** node(ch)); //push operand at last

}

}

i++; //cout<<" i "<<i;

}

**while**(!s2.empty()) //pop() until operator stack is not empty

{

node \*op=s2.pop(); node \*rchild=s1.pop(); node \*lchild=s1.pop();

op->right=rchild; op->left=lchild;

s1.push(op);

}

//set the root element to s1->top() root=s1.pop();

}

### int main() {

cout << "" << **endl**; // prints tree t1;

string exp="a-b\*c-d/e+f";

cout<<"\nOriginal Expression: "<<exp;

t1.create(exp);

cout<<"\nInorder Traversal Recursive: "; t1.inorder();

cout<<"\nInorder Non-Recursive: "; t1.inorderNonRec();

cout<<"\nPreorder Traversal Recursive: "; t1.preorder();

cout<<"\nPreorder traversal Non-Recursive: "; t1.preorderNonRec();

cout<<"\nPostorder Traversal recursive: "; t1.postorder();

cout<<"\nPostorder Non-Recursive: "; t1.postorderNon();

**return** 0; }

**ASSIGNMENT NO-4**

Write a function to get the number of vertices in an undirected graph and its edges. You may assume that no edge is input twice.

Use adjacency matrix representation of the graph and find runtime of the function

#include <iostream>

#include <cstdlib>

#include<stdlib.h> #include<list>

using namespace std;

int visited[20],visit[20],stk[20],top=0,qu[12],front,rear;

/\* Adjacency List Node \*/

struct AdjListNode

{

int dest;

struct AdjListNode\* next;

};

/\* Adjacency List \*/ struct AdjList

{ struct AdjListNode \*head;

};

/\* Class Graph \*/ class Graph

{

private: int V,k;

struct AdjList\* array; int adjM[10][10];

list<int>\* adj;

public:

Graph(int V)

{

this->V = V;

array = new AdjList [V];

for (int i=1;i<=V; ++i)

{

array[i].head = NULL; //In Adjacency list all vertices initially null

for(int j=1; j<=V; ++j)

{

adjM[i][j]=0; //nXn all values initially zero

}

}

adj = new list<int>[V]; for (int i=1; i<=V; ++i) visited[i]=0;

}

/\* Creating New Adjacency List Node \*/

AdjListNode\* newAdjListNode(int dest)

{

AdjListNode\* newNode = new AdjListNode;

newNode->dest = dest; newNode->next = NULL;

return newNode;

}

/\*Add Edge to Graph \*/

void addEdge()

{

int src,des;

cout<<"\n Enter Edge "; cout<<"\n From -> "; cin>>src; cout<<"\n To -> "; cin>>des;

adjM[src][des]=1; //value changes in matrix example src is 1 and dest is 0 so [1,0]=1

adjM[des][src]=1; //opposite [0,1]=1 adj[src].push\_back(des);// update adjlist ex 1->0 adj[des].push\_back(src);//update 0->1 AdjListNode\* newNode = newAdjListNode(des); newNode->next = array[src].head; array[src].head = newNode; newNode = newAdjListNode(src); newNode->next = array[des].head;

array[des].head = newNode;

}

/\* Print the graph\*/

void printGraph()

{ int v;

cout<<"\n Adjucency Matrix is : \n "; for (int i=1; i<=V; ++i)

{

for(int j=1; j<=V; ++j)

{

cout<<adjM[i][j]<<" ";

}

cout<<"\n";

}

for (v=1; v<=V;++v)

{

AdjListNode\* p = array[v].head;

cout<<"\n Adjacency list of vertex "<<v<<"\n head "; while(p)

{

cout<<"-> "<<p->dest; p = p->next;

}

cout<<endl;

}

}

};

/\* \* Main \*/

int main()

{

int ver,ed,ch,i,j,s;

cout<<"Enter Count of No of Vertices";

cin>>ver;

cout<<"Enter Count of No of Egdes"; cin>>ed; Graph gh(ed);

do

{

cout<<"\n Select option :";

cout<<"\n\t 1. Create graph \n \t2.Display graph \n \t 3.Exit"; cin>>ch;

switch(ch)

{

case 1:

for(i=0;i<ed;i++) gh.addEdge(); break; case 2:

// print the adjacency list representation of the above graph gh.printGraph(); break;

default : cout<<"\n Enter correct choice";

}

}while(ch!=3);

return 0;

}

**ASSIGNMENT NO-5**

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities.

The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used.

#include**<iostream>** **#define** MAX 10 **using** **namespace** std;

**class** airport

{

string city[MAX]; **int** distance[10][10];

**public** : **int** n; **airport**(); **void** **input\_g**();

### void output\_g();

};

### airport :: airport()

{

n = 0;

**for**(**int** i=0 ; i<MAX ; i++)

{

**for**(**int** j=0 ; j<MAX ; j++)

distance[i][j] = 0;

} }

### void airport :: input\_g()

{

**int** k; cout << "\nEnter the no. of cities: " ; cin >> n;

**for**(**int** k=0;k<n;k++)

{

cout<<"Enter "<<k+1<<" city name:"; cin>>city[k]; }

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

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

{

cout << "\nEnter Distance between " <<city[i]<< " to "

<<city[j]<<":\n"; cin >> distance[i][j];

distance[j][i] = distance[i][j];

} }

}

### void airport :: output\_g()

{

cout<<" ";

**for**(**int** k=0;k<n;k++)

{

cout<<city[k]<<" "; }cout<<"Enter city name:"; cout << "\n";

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

{ cout <<city[i] <<" ";

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

{

cout<<" "<< distance[i][j] ;

}

cout << " "; cout << "\n"; }

}

### int main()

{ airport obj; obj.input\_g(); obj.output\_g(); }

**ASSIGNMENT NO-6**

Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number (with Replacement)

#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: ";

cin>>ans;

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

}

int main()

{

long k;

int ch,index; char ans; HashFunction obj; do

{

cout<<"\n\t\*\*\*\*\* Dictionary (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: ";

cin>>ans;

}while(ans=='y'||ans=='Y'); return 0;

}

**ASSIGNMENT NO-7**

**For given set of elements create skip list. Find the element in the set that is closest to some given value.**

Code:-

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <cstring>

#define MAX\_LEVEL 6

const float P = 0.5;

using namespace std;

/\*

\* Skip Node Declaration

\*/

struct snode

{

int value;

snode \*\*forw;

snode(int level, int &value)

{

forw = new snode \* [level + 1];

memset(forw, 0, sizeof(snode\*) \* (level + 1));

this->value = value;

}

~snode()

{

delete [] forw;

}

};

/\*

\* Skip List Declaration

\*/

struct skiplist

{

snode \*header;

int value;

int level;

skiplist()

{

header = new snode(MAX\_LEVEL, value);

level = 0;

}

~skiplist()

{

delete header;

}

void display();

bool contains(int &);

void insert\_element(int &);

void delete\_element(int &);

};

/\*

\* Main: Contains Menu

\*/

int main()

{

skiplist ss;

int choice, n;

while (1)

{

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

cout<<endl<<"Operations on Skip list"<<endl;

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

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

cout<<"2.Delete Element"<<endl;

cout<<"3.Search Element"<<endl;

cout<<"4.Display List "<<endl;

cout<<"5.Exit "<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

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

cin>>n;

ss.insert\_element(n);

if(ss.contains(n))

cout<<"Element Inserted"<<endl;

break;

case 2:

cout<<"Enter the element to be deleted: ";

cin>>n;

if(!ss.contains(n))

{

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

break;

}

ss.delete\_element(n);

if(!ss.contains(n))

cout<<"Element Deleted"<<endl;

break;

case 3:

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

cin>>n;

if(ss.contains(n))

cout<<"Element "<<n<<" is in the list"<<endl;

else

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

case 4:

cout<<"The List is: ";

ss.display();

break;

case 5:

exit(1);

break;

default:

cout<<"Wrong Choice"<<endl;

}

}

return 0;

}

/\*

\* Random Value Generator

\*/

float frand()

{

return (float) rand() / RAND\_MAX;

}

/\*

\* Random Level Generator

\*/

int random\_level()

{

static bool first = true;

if (first)

{

srand((unsigned)time(NULL));

first = false;

}

int lvl = (int)(log(frand()) / log(1.-P));

return lvl < MAX\_LEVEL ? lvl : MAX\_LEVEL;

}

/\*

\* Insert Element in Skip List

\*/

void skiplist::insert\_element(int &value)

{

snode \*x = header;

snode \*update[MAX\_LEVEL + 1];

memset(update, 0, sizeof(snode\*) \* (MAX\_LEVEL + 1));

for (int i = level;i >= 0;i--)

{

while (x->forw[i] != NULL && x->forw[i]->value < value)

{

x = x->forw[i];

}

update[i] = x;

}

x = x->forw[0];

if (x == NULL || x->value != value)

{

int lvl = random\_level();

if (lvl > level)

{

for (int i = level + 1;i <= lvl;i++)

{

update[i] = header;

}

level = lvl;

}

x = new snode(lvl, value);

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

{

x->forw[i] = update[i]->forw[i];

update[i]->forw[i] = x;

}

}

}

/\*

\* Delete Element from Skip List

\*/

void skiplist::delete\_element(int &value)

{

snode \*x = header;

snode \*update[MAX\_LEVEL + 1];

memset (update, 0, sizeof(snode\*) \* (MAX\_LEVEL + 1));

for (int i = level;i >= 0;i--)

{

while (x->forw[i] != NULL && x->forw[i]->value < value)

{

x = x->forw[i];

}

update[i] = x;

}

x = x->forw[0];

if (x->value == value)

{

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

{

if (update[i]->forw[i] != x)

break;

update[i]->forw[i] = x->forw[i];

}

delete x;

while (level > 0 && header->forw[level] == NULL)

{

level--;

}

}

}

/\*

\* Display Elements of Skip List

\*/

void skiplist::display()

{

const snode \*x = header->forw[0];

while (x != NULL)

{

cout << x->value;

x = x->forw[0];

if (x != NULL)

cout << " - ";

}

cout <<endl;

}

/\*

\* Search Elemets in Skip List

\*/

bool skiplist::contains(int &s\_value)

{

snode \*x = header;

for (int i = level;i >= 0;i--)

{

while (x->forw[i] != NULL && x->forw[i]->value < s\_value)

{

x = x->forw[i];

}

}

x = x->forw[0];

return x != NULL && x->value == s\_value;

}

**ASSIGNMENT NO-8**

Given sequence k = k1 <k2 < ... < kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key.

Program:-

**#include<iostream>** **using** **namespace** std; **void** **con\_obst**(**void**); **void** **print**(**int**,**int**);

**float** a[20],b[20],wt[20][20],c[20][20]; **int** r[20][20],n;

### int main()

{

**int** i;

cout<<"\n\*\*\*\*\*\* PROGRAM FOR OBST \*\*\*\*\*\*\n";

cout<<"\nEnter the no. of nodes : ";

cin>>n;cout<<"\nEnter the probability for successful search :: ";

cout<<"\n————————————————\n";

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

{

cout<<"p["<<i<<"]"; cin>>a[i];

}

cout<<"\nEnter the probability for unsuccessful search :: ";

cout<<"\n————————————————–\n";

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

{

cout<<"q["<<i<<"]"; cin>>b[i];

}

con\_obst(); print(0,n); cout<<**endl**; }

### void con\_obst(void)

{

**int** i,j,k,l,min; **for**(i=0;i<n;i++) { //Initialisation

c[i][i]=0.0; r[i][i]=0; wt[i][i]=b[i];

// for j-i=1 can be j=i+1 wt[i][i+1]=b[i]+b[i+1]+a[i+1];

c[i][i+1]=b[i]+b[i+1]+a[i+1]; r[i][i+1]=i+1;

}

c[n][n]=0.0; r[n][n]=0;

wt[n][n]=b[n];

//for j-i=2,3,4....,n

**for**(i=2;i<=n;i++)

{

**for**(j=0;j<=n-i;j++)

{

wt[j][j+i]=b[j+i]+a[j+i]+wt[j][j+i-1];

c[j][j+i]=9999;

**for**(l=j+1;l<=j+i;l++)

{

**if**(c[j][j+i]>(c[j][l-1]+c[l][j+i]))

{

c[j][j+i]=c[j][l-1]+c[l][j+i];

r[j][j+i]=l;

}

}

c[j][j+i]+=wt[j][j+i];

}

cout<<**endl**;

}

cout<<"\n\nOptimal BST is :: "; cout<<"\nw[0]["<<n<<"] :: "<<wt[0][n]; cout<<"\nc[0]["<<n<<"] :: "<<c[0][n];

cout<<"\nr[0]["<<n<<"] :: "<<r[0][n];

}

**void** **print**(**int** l1,**int** r1)

{

**if**(l1>=r1)

**return**;

**if**(r[l1][r[l1][r1]-1]!=0)

cout<<"\n Left child of "<<r[l1][r1]<<" :: "<<r[l1][r[l1][r1]-1]; **if**(r[r[l1][r1]][r1]!=0)

cout<<"\n Right child of "<<r[l1][r1]<<" :: "<<r[r[l1][r1]][r1]; print(l1,r[l1][r1]-1);

print(r[l1][r1],r1); **return**;

}

## ASSIGNMENT NO-9

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords.

Provide facility to display whole data sorted in ascending/ Descending order

\*/

# Program:-

#include<iostream>

#include<stdio.h> #include<string.h> using namespace std;

class Tree

{

typedef struct node

{

char key[10]; char meaning[10]; struct node \*left; struct node \* right;

}btree; public: btree \*New,\*root; Tree(); void create();

void insert(btree \*root,btree \*New);

void inorder(); void inorder\_rec(btree \*root); void postorder(); void postorder\_rec(btree \*root);

};

Tree::Tree()

{

root=NULL;

}

void Tree::inorder()

{

inorder\_rec(root);

}

void Tree::inorder\_rec(btree \*root)

{

if(root!=NULL)

{

inorder\_rec(root->left); cout<<"\n\t"<<root->key<<"\t"<<root->meaning; inorder\_rec(root->right);

}

}

void Tree::postorder()

{

postorder\_rec(root);

}

void Tree::postorder\_rec(btree \*root)

{

if(root!=NULL)

{

postorder\_rec(root->right); cout<<"\n\t"<<root->key<<"\t"<<root->meaning; postorder\_rec(root->left);

}

} void Tree::create() {

New=new btree;

New->left=New->right=NULL; cout<<"\n\tEnter the Keyword: "; cin>>New->key;

cout<<"\n\tEnter the Meaning of "<<New->key<<" : "; cin>>New->meaning; if(root==NULL)

{

|  |  |
| --- | --- |
| }  else { | root=New; |
|  | insert(root,New); |

}

}

void Tree::insert(btree \*root,btree \*New)

{

if(strcmp(root->key,New->key)>0)

{

if(root->left==NULL)

root->left=New;

else

insert(root->left,New);

}

else

{

if(root->right==NULL) root->right=New;

else

insert(root->right,New);

}

}

int main()

{ Tree tr; int ch;

char ans;

do {

cout<<"\n\t\*\*\*\*\* BST Operations \*\*\*\*\*"; cout<<"\n\t1. Create\n\t2. Display\n\t3. Exit"; cout<<"\n\t.....Enter Your Choice: "; cin>>ch; switch(ch)

{

case 1:

do

{

tr.create();

cout<<"......Do You Want To Continue: ";

cin>>ans;

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

break; case 2: cout<<"\n\t\t1. Ascending\n\t\t2. Descending\n\t\t.....Enter Your Choice:

";

cin>>ch;

cout<<"\n\tKeyword\tMeaning";

switch(ch)

{

case 1:

tr.inorder();

break; case 2:

tr.postorder();

break;

}

break; case 3: break;

}

cout<<"\n\t\t..... Do You Want to Continue: ";

cin>>ans;

}while(ans=='y'||ans=='Y'); return 0;

}

**ASSIGNMENT NO-10**

Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

#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;

**ASSIGNMENT NO-11**

Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

#include<iostream>

#include<fstream> #include<string.h> using namespace std;

class student

{

typedef struct stud

{

int roll; char name[10]; char div; char add[10];

}stud;

stud rec; public:

void create(); void display(); int search();

void Delete();

};

void student::create()

{

char ans;

ofstream fout;

fout.open("stud.dat",ios::out|ios::binary);

do

{

cout<<"\n\tEnter Roll No of Student : ";

cin>>rec.roll;

cout<<"\n\tEnter a Name of Student : ";

cin>>rec.name;

cout<<"\n\tEnter a Division of Student : ";

cin>>rec.div;

cout<<"\n\tEnter a Address of Student : ";

cin>>rec.add;

fout.write((char \*)&rec,sizeof(stud))<<flush;

cout<<"\n\tDo You Want to Add More Records: ";

cin>>ans;

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

fout.close();

}

void student::display()

{

ifstream fin; #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;

}fin.open("stud.dat",ios::in|ios::binary); fin.seekg(0,ios::beg); cout<<"\n\tThe Content of File are:\n"; cout<<"\n\tRoll\tName\tDiv\tAddress";

while(fin.read((char \*)&rec,sizeof(stud)))

{

if(rec.roll!=-1)

cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;

}

fin.close();

}

int student::search()

{

int r,i=0;

ifstream fin;

fin.open("stud.dat",ios::in|ios::binary);

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

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

cin>>r;

while(fin.read((char \*)&rec,sizeof(stud)))

{

if(rec.roll==r)

{

cout<<"\n\tRecord Found...\n"; cout<<"\n\tRoll\tName\tDiv\tAddress";

cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;

return i; } i++;

}

fin.close();

return 0;

}

void student::Delete()

{

int pos; pos=search();

fstream f;

f.open("stud.dat",ios::in|ios::out|ios::binary);

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

if(pos==0)

{

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

return;

}

int offset=pos\*sizeof(stud); f.seekp(offset); rec.roll=-1; strcpy(rec.name,"NULL"); rec.div='N'; strcpy(rec.add,"NULL");

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

f.seekg(0);

f.close();

cout<<"\n\tRecord Deleted";

}

int main()

{

student obj; int ch,key; char ans;

do

{

cout<<"\n\t\*\*\*\*\* Student Information \*\*\*\*\*";

cout<<"\n\t1. Create\n\t2. Display\n\t3. Delete\n\t4. Search\n\t5. Exit";

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

switch(ch)

{

case 1: obj.create();

break;

case 2: obj.display();

break;

case 3: obj.Delete();

break;

case 4: key=obj.search();

if(key==0)

cout<<"\n\tRecord Not Found...\n";

break; case 5: break;

}

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

cin>>ans;

}while(ans=='y'||ans=='Y'); return 1;

}

**ASSIGNMENT NO-12**

Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language

**import** java.util.Scanner;

**public** **class** HeapSort

{

**private** **static** **int** *N*;

**public** **static** **void** sort(**int** arr[])

{

*heapify*(arr);

**for** (**int** i = *N*; i > 0; i--)

{

*swap*(arr,0, i);

*N* = *N*-1;

*maxheap*(arr, 0);

}

}

/\* Function to build a heap \*/

**public** **static** **void** heapify(**int** arr[])

{

*N* = arr.length-1;

**for** (**int** i = *N*/2; i >= 0; i--)

*maxheap*(arr, i);

}

/\* Function to swap largest element in heap \*/

**public** **static** **void** maxheap(**int** arr[], **int** i)

{

**int** left = 2\*i ;

**int** right = 2\*i + 1;

**int** max = i;

**if** (left <= *N* && arr[left] > arr[i])

max = left;

**if** (right <= *N* && arr[right] > arr[max])

max = right;

**if** (max != i)

{

*swap*(arr, i, max);

*maxheap*(arr, max);

}

}

/\* Function to swap two numbers in an array \*/

**public** **static** **void** swap(**int** arr[], **int** i, **int** j)

{

**int** tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

}

/\* Main method \*/

**public** **static** **void** main(String[] args)

{

Scanner scan = **new** Scanner( System.***in*** );

System.***out***.println("Heap Sort Test\n");

**int** n, i;

/\* Accept number of elements \*/

System.***out***.println("Enter number of integer elements");

n = scan.nextInt();

/\* Make array of n elements \*/

**int** arr[] = **new** **int**[ n ];

/\* Accept elements \*/

System.***out***.println("\nEnter "+ n +" integer elements");

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

arr[i] = scan.nextInt();

/\* Call method sort \*/

*sort*(arr);

/\* Print sorted Array \*/

System.***out***.println("\nElements after sorting ");

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

System.***out***.print(arr[i]+" ");

System.***out***.println();

}

}

**ASSIGNMENT NO-13**

Write a Java program which will demonstrate a concept of Interfaces and packages: In this assignment design and use of customized interfaces and packages for a specific application are expected.

\*\*\*\*\*\*\*\*\*\*\* circle.java

package shape; import java.io.\*; import java.util.\*;

import java.util.Scanner;

public class circle implements perimeter {

private double r;

public void calculate()

{

System.*out*.print("Enter radius"); Scanner input=new Scanner(System.*in*);

double r=input.nextDouble(); double p=2\*3.14\*r;

System.*out*.println("Perimeter of circle:"+p);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* perimeter.java

package shape;

public interface perimeter {

void calculate();

}

\*\*\*\*\*\*\*\*\*

rectangle.java

package shape; import java.io.\*; import java.util.\*; import java.util.Scanner;

public class rectangle implements perimeter

{

private double length, breadth;

public void calculate()

{

System.*out*.print("Enter length"); Scanner input=new Scanner(System.*in*); double length=input.nextDouble(); System.*out*.print("Enter breadth"); double breadth=input.nextDouble(); double p=2\*(length+breadth);

System.*out*.println("Perimeter of rectangle:"+p);

}

}

main.java

package shape;

public class main{

public static void main(String[] args)

{

perimeter obj;

circle obj1=new circle(); rectangle obj2=new rectangle();

System.*out*.println("\n\t calculating Perimeter of circle\n");

obj=obj1;

obj.calculate();

System.*out*.println("calculating Perimeter of rectangle:\n");

obj=(perimeter) obj2;

obj.calculate();

}

}