**Experiment No 1**

Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers

**1. Python program File Name LinearProbing.py**

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

# Program to implement Hashing with Linear Probing

from Record import Record

class hashTable:

    # initialize hash Table

    def \_\_init\_\_(self):

        self.size = int(input("Enter the Size of the hash table : "))

        # initialize table with all elements 0

        self.table = list(None for i in range(self.size))

        self.elementCount = 0

        self.comparisons = 0

    # method that checks if the hash table is full or not

    def isFull(self):

        if self.elementCount == self.size:

            return True

        else:

            return False

    # method that returns position for a given element

    def hashFunction(self, element):

        return element % self.size

    # method that inserts element into the hash table

    def insert(self, record):

        # checking if the table is full

        if self.isFull():

            print("Hash Table Full")

            return False

        isStored = False

        position = self.hashFunction(record.get\_number())

        # checking if the position is empty

        if self.table[position] == None:

            self.table[position] = record

            print("Phone number of " + record.get\_name() + " is at position " + str(position))

            isStored = True

            self.elementCount += 1

        # collision occured hence we do linear probing

        else:

            print("Collision has occured for " + record.get\_name() + "'s phone number at position " + str(position) + " finding new Position.")

            while self.table[position] != None:

                position += 1

                if position >= self.size:

                    position = 0

            self.table[position] = record

            print("Phone number of " + record.get\_name() + " is at position " + str(position))

            isStored = True

            self.elementCount += 1

        return isStored

    # method that searches for an element in the table

    # returns position of element if found

    # else returns False

    def search(self, record):

        found = False

        position = self.hashFunction(record.get\_number())

        self.comparisons += 1

        if(self.table[position] != None):

            if(self.table[position].get\_name() == record.get\_name() and self.table[position].get\_number() == record.get\_number()):

                isFound = True

                print("Phone number found at position {} ".format(position) + " and total comparisons are " + str(1))

                return position

        # if element is not found at position returned hash function

            else:

                position += 1

                if position >= self.size-1:

                    position = 0

                while self.table[position] != None or self.comparisons <= self.size:

                    if(self.table[position].get\_name() == record.get\_name() and self.table[position].get\_number() == record.get\_number()):

                        isFound = True

                        #i=0

                        i = self.comparisons + 1

                        print("Phone number found at position {} ".format(position) + " and total comparisons are " + str(i) )

                        return position

                    position += 1

                    #print(position)

                    if position >= self.size-1:

                        position = 0

                    #print(position)

                    self.comparisons += 1

                    #print(self.comparisons)

                if isFound == False:

                    print("Record not found")

                    return false

    # method to display the hash table

    def display(self):

        print("\n")

        for i in range(self.size):

            print("Hash Value: "+str(i) + "\t\t" + str(self.table[i]))

        print("The number of phonebook records in the Table are : " + str(self.elementCount))

----------------------------------------------------------------.

**2. Python Program File Name : Record.py**

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

class Record:

    def \_\_init\_\_(self):

        self.\_name = None

        self.\_number = None

    def get\_name(self):

        return self.\_name

    def get\_number(self):

        return self.\_number

    def set\_name(self,name):

        self.\_name = name

    def set\_number(self,number):

        self.\_number = number

    def \_\_str\_\_(self):

        record = "Name: "+str(self.get\_name())+"\t"+"\tNumber: "+str(self.get\_number())

        return record

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

**3.  Python Program File Name : DoubleHashing.py**

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

from Record import Record

class doubleHashTable:

    # initialize hash Table

    def \_\_init\_\_(self):

        self.size = int(input("Enter the Size of the hash table : "))

        # initialize table with all elements 0

        self.table = list(None for i in range(self.size))

        self.elementCount = 0

        self.comparisons = 0

    # method that checks if the hash table is full or not

    def isFull(self):

        if self.elementCount == self.size:

            return True

        else:

            return False

    # First hash function

    def h1(self, element):

        return element % self.size

    # Second hash function

    def h2(self, element):

        return 5-(element % 5)

    # method to resolve collision by double hashing method

    def doubleHashing(self, record):

        posFound = False

        # limit variable is used to restrict the function from going into infinite loop

        # limit is useful when the table is 80% full

        limit = self.size

        i = 1

        # start a loop to find the position

        while i <= limit:

            # calculate new position by quadratic probing

            newPosition = (self.h1(record.get\_number()) + i\*self.h2(record.get\_number())) % self.size

            # if newPosition is empty then break out of loop and return new Position

            if self.table[newPosition] == None:

                posFound = True

                break

            else:

                # as the position is not empty increase i

                i += 1

        return posFound, newPosition

    # method that inserts element inside the hash table

    def insert(self, record):

        # checking if the table is full

        if self.isFull():

            print("Hash Table Full")

            return False

        posFound = False

        position = self.h1(record.get\_number())

        # checking if the position is empty

        if self.table[position] == None:

            # empty position found , store the element and print the message

            self.table[position] = record

            print("Phone number of " + record.get\_name() + " is at position " + str(position))

            isStored = True

            self.elementCount += 1

        # If collision occured

        else:

            print("Collision has occured for " + record.get\_name() + "'s phone number at position " + str(position) + " finding new Position.")

            while not posFound:

                posFound, position = self.doubleHashing(record)

                if posFound:

                    self.table[position] = record

                    #print(self.table[position])

                    self.elementCount += 1

                    #print(position)

                    #print(posFound)

                    print("Phone number of " + record.get\_name() + " is at position " + str(position))

        return posFound

    # searches for an element in the table and returns position of element if found else returns False

    def search(self, record):

        found = False

        position = self.h1(record.get\_number())

        self.comparisons += 1

        if(self.table[position] != None):

            if(self.table[position].get\_name() == record.get\_name()):

                print("Phone number found at position {}".format(position) + " and total comparisons are " + str(1))

                return position

            # if element is not found at position returned hash function

            # then we search element using double hashing

            else:

                limit = self.size

                i = 1

                newPosition = position

                # start a loop to find the position

                while i <= limit:

                    # calculate new position by double Hashing

                    position = (self.h1(record.get\_number()) + i\*self.h2(record.get\_number())) % self.size

                    self.comparisons += 1

                    # if element at newPosition is equal to the required element

                    if(self.table[position] != None):

                        if self.table[position].get\_name() == record.get\_name():

                            found = True

                            break

                        elif self.table[position].get\_name() == None:

                            found = False

                            break

                        else:

                            # as the position is not empty increase i

                            i += 1

            if found:

                print("Phone number found at position {}".format(position) + " and total comparisons are " + str(i+1))

#return position

            else:

                print("Record not Found")

                return found

    # method to display the hash table

    def display(self):

        print("\n")

        for i in range(self.size):

            print("Hash Value: "+str(i) + "\t\t" + str(self.table[i]))

        print("The number of phonebook records in the Table are : " + str(self.elementCount))

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

**4.  Python Program File Name : main.py**

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

from LinearProbing import hashTable

from Record import Record

from DoubleHashing import doubleHashTable

def input\_record():

    record = Record()

    name = input("Enter Name:")

    number = int(input("Enter Number:"))

    record.set\_name(name)

    record.set\_number(number)

    return record

choice1 = 0

while(choice1 != 3):

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("1. Linear Probing      \*")

    print("2. Double Hashing      \*")

    print("3. Exit                \*")

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    choice1 = int(input("Enter Choice"))

    if choice1>3:

        print("Please Enter Valid Choice")

    if choice1 == 1:

        h1 = hashTable()

        choice2 = 0

        while(choice2 != 4):

            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

            print("1. Insert              \*")

            print("2. Search              \*")

            print("3. Display             \*")

            print("4. Back                \*")

            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

            choice2 = int(input("Enter Choice"))

            if choice2>4:

                print("Please Enter Valid Choice")

            if(choice2==1):

                record = input\_record()

                h1.insert(record)

            elif(choice2 == 2):

                record = input\_record()

                position = h1.search(record)

            elif(choice2 == 3):

                h1.display()

    elif choice1 == 2:

        h2 = doubleHashTable()

        choice2 = 0

        while(choice2 != 4):

            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

            print("1. Insert              \*")

            print("2. Search              \*")

            print("3. Display             \*")

            print("4. Back                \*")

            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

            choice2 = int(input("Enter Choice"))

            if choice2>4:

                print("Please Enter Valid Choice")

            if(choice2==1):

                record = input\_record()

                h2.insert(record)

            elif(choice2 == 2):

                record = input\_record()

                position = h2.search(record)

            elif(choice2 == 3):

                h2.display()

**Output:**

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

1. Linear Probing \*

2. Double Hashing \*

3. Exit \*

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

Enter Choice1

Enter the Size of the hash table : 5

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice1

Enter Name:abc

Enter Number:111111

Phone number of abc is at position 1

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice1

Enter Name:pqr

Enter Number:222222

Phone number of pqr is at position 2

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice1

Enter Name:xyz

Enter Number:33333

Phone number of xyz is at position 3

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice3

Hash Value: 0 None

Hash Value: 1 Name: abc Number: 111111

Hash Value: 2 Name: pqr Number: 222222

Hash Value: 3 Name: xyz Number: 33333

Hash Value: 4 None

The number of phonebook records in the Table are : 3

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice2

Enter Name:pqr

Enter Number:222222

Phone number found at position 2 and total comparisons are 1

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice4

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

1. Linear Probing \*

2. Double Hashing \*

3. Exit \*

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

Enter Choice2

Enter the Size of the hash table : 3

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice1

Enter Name:abc

Enter Number:11111

Phone number of abc is at position 2

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice1

Enter Name:pqr

Enter Number:222222

Phone number of pqr is at position 0

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice3

Hash Value: 0 Name: pqr Number: 222222

Hash Value: 1 None

Hash Value: 2 Name: abc Number: 11111

The number of phonebook records in the Table are : 2

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

1. Insert \*

2. Search \*

3. Display \*

4. Back \*

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

Enter Choice4

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

1. Linear Probing \*

2. Double Hashing \*

3. Exit \*

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

Enter Choice3

**Experiment no 2**

To create ADT that implement the "set" concept.

a. Add (new Element) -Place a value into the set ,

b. Remove (element) Remove the value

c. Contains (element) Return true if element is in collection

**Code :**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

# define max 10

typedef struct list

{

int data;

struct list \*next;

} node\_type;

node\_type \*ptr[max],\*root[max],\*temp[max];

class Dictionary

{

public:

int index;

Dictionary();

void insert(int);

void search(int);

void delete\_ele(int);

};

Dictionary::Dictionary()

{

index=-1;

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

{

root[i]=NULL;

ptr[i]=NULL;

temp[i]=NULL;

}

}

void Dictionary::insert(int key)

{

index=int(key%max);

ptr[index]=(node\_type\*)malloc(sizeof(node\_type));

ptr[index]->data=key;

if(root[index]==NULL)

{

root[index]=ptr[index];

root[index]->next=NULL;

temp[index]=ptr[index];

}

else

{

temp[index]=root[index];

while(temp[index]->next!=NULL)

temp[index]=temp[index]->next;

temp[index]->next=ptr[index];

}

}

void Dictionary::search(int key)

{

int flag=0;

index=int(key%max);

temp[index]=root[index];

while(temp[index]!=NULL)

{

if(temp[index]->data==key)

{

cout<<"\nSearch key is found!!";

flag=1;

break;

}

else temp[index]=temp[index]->next;

}

if (flag==0)

cout<<"\nsearch key not found.......";

}

void Dictionary::delete\_ele(int key)

{

index=int(key%max);

temp[index]=root[index];

while(temp[index]->data!=key && temp[index]!=NULL)

{

ptr[index]=temp[index];

temp[index]=temp[index]->next;

}

ptr[index]->next=temp[index]->next;

cout<<"\n"<<temp[index]->data<<" has been deleted.";

temp[index]->data=-1;

temp[index]=NULL;

free(temp[index]);

}

int main()

{

int val,ch,n,num;

char c;

Dictionary d;

do

{

cout<<"\nMENU:\n1.Create";

cout<<"\n2.Search for a value\n3.Delete an value";

cout<<"\nEnter your choice:";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\nEnter the number of elements to be inserted:";

cin>>n;

cout<<"\nEnter the elements to be inserted:";

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

{

cin>>num;

d.insert(num);

}

break;

case 2:

cout<<"\nEnter the element to be searched:";

cin>>n;

d.search(n);

case 3:

cout<<"\nEnter the element to be deleted:";

cin>>n;

d.delete\_ele(n);

break;

default:

cout<<"\nInvalid Choice.";

}

cout<<"\nEnter y to Continue:";

cin>>c;

}

while(c=='y');

getch();

}

**Output**

MENU:

1.Create

2.Search for a value

3.Delete an value

Enter your choice:1

Enter the number of elements to be inserted:3

Enter the elements to be inserted:23

45

36

Enter y to Continue:y

MENU:

1.Create

2.Search for a value

3.Delete an value

Enter your choice:2

Enter the element to be searched:45

Search key is found!!

Enter the element to be deleted:36

36 has been deleted.

Enter y to Continue:n

**Experiment No 3**

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<<"\nEnter name of book : ";

cin>>root->label;

cout<<"\nEnter no. of chapters in book : ";

cin>>tchapters;

root->ch\_count = tchapters;

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

{

root->child[i] = new node;

cout<<"\nEnter Chapter name : ";

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

cout<<"\nEnter 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<<"\nEnter Section: "<<j+1<<" name\n";

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

}

}

}

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;

}

}

}

}

/\*

\* Main Contains Menu

\*/

int main()

{

int choice;

BST bst;

while (1)

{

cout<<"\n-----------------"<<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:

bst.create\_tree();

case 2:

bst.display(root);

break;

case 3:

exit(1);

default:

cout<<"Wrong choice \n"<<endl;

}

}

}

**Output:**

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

Book Tree Creation

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

1.Create

2.Display

3.Quit

Enter your choice : 1

Enter name of book : DSA

Enter no. of chapters in book : 2

Enter Chapter name : Trees

Enter no. of sections in Chapter: Trees 2

Enter Section: 1 name

BST

Enter Section: 2 name

OBST

Enter Chapter name : Files

Enter no. of sections in Chapter: Files 2

Enter Section: 1 name

Sequential

Enter Section: 2 name

Direct\_access

-----Book Hierarchy---

Book title : DSA

Chapter 1 Trees

Sections

BST

OBST

Chapter 2 Files

Sections

Sequential

Direct\_access

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

Book Tree Creation

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

1.Create

2.Display

3.Quit

Enter your choice : 2

-----Book Hierarchy---

Book title : DSA

Chapter 1 Trees

Sections

BST

OBST

Chapter 2 Files

Sections

Sequential

Direct\_access

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

Book Tree Creation

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

1.Create

2.Display

3.Quit

Enter your choice : 3

**Experiment no 4**

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 from root, 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. Search a value

**Code :**

#include<iostream>

#include<math.h>

using namespace std;

struct Bstnode

{

int data;

Bstnode \*left = NULL;

Bstnode \*right = NULL;

};

class Btree

{

int n;

int x;

int flag=0;

public:

Bstnode \* root;

Btree()

{

root = NULL;

}

Bstnode \*GetNewNode(int in\_data)

{

Bstnode \* ptr = new Bstnode();

ptr->data = in\_data;

ptr->left = NULL;

ptr->right = NULL;

return ptr;

}

Bstnode \*insert( Bstnode \*temp , int in\_data)

{

if( temp == NULL )

{

temp = GetNewNode(in\_data);

}

else if( temp->data > in\_data)

{

temp->left = insert(temp->left , in\_data);

}

else

{

temp->right = insert( temp->right , in\_data);

}

return temp;

}

void input()

{

cout<<"ENTER NUMBER OF ELEMENTS IN THE BST : ";

cin>>n;

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

{

cout<<"NUMBER = ";

cin>>x;

root = insert(root , x);

}

}

void search(Bstnode \*temp ,int in\_data)

{

if( temp != NULL)

{

if(temp->data == in\_data)

{

cout<<"\nElement Found\n";

}

else if(in\_data < temp->data)

{

this->search(temp->left,in\_data);

}

else if(in\_data > temp->data)

{

this->search(temp->left,in\_data);

}

}

else

{

cout<<"\nElement not Found\n";

}

}

void minvalue(Bstnode \*temp)

{

while(temp->left != NULL)

{

temp = temp->left;

}

cout<<"\n MINIMUM VALUE IS = "<<temp->data<<endl;

}

void mirror(Bstnode \*temp)

{

if(temp == NULL)

{

return;

}

else

{

Bstnode \*ptr;

mirror(temp->left);

mirror(temp->right);

ptr = temp->left;

temp->left = temp->right;

temp->right = ptr;

}

}

void display()

{

cout<<endl<<"--- INORDER TRAVERSAL ---"<<endl;

inorder(root);

cout<<endl;

cout<<endl<<"--- POSTORDER TRAVERSAL ---"<<endl;

postorder(root);

cout<<endl;

cout<<endl<<"--- PREORDER TRAVERSAL ---"<<endl;

preorder(root);

cout<<endl;

}

void inorder(Bstnode \*temp)

{

if(temp != NULL)

{

inorder(temp->left);

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

inorder(temp->right);

}

}

void postorder(Bstnode \*temp)

{

if(temp != NULL)

{

postorder(temp->left);

postorder(temp->right);

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

}

}

void preorder(Bstnode \*temp)

{

if(temp != NULL)

{

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

preorder(temp->left);

preorder(temp->right);

}

}

int depth(Bstnode \*temp)

{

if(temp == NULL)

return 0;

return (max((depth(temp->left)),(depth(temp->right))) +1);

}

};

int main()

{

Btree obj;

obj.input();

obj.display();

int b;

obj.minvalue(obj.root);

obj.mirror(obj.root);

cout<<"\nEnter Element to be searched: ";

cin>>b;

obj.search(obj.root,b);

cout<<"\n Depth of Tree is : "<<obj.depth(obj.root);

return 0;

}

**Output:**

ENTER NUMBER OF ELEMENTS IN THE BST : 5

NUMBER = 50

NUMBER = 40

NUMBER = 60

NUMBER = 20

NUMBER = 80

--- INORDER TRAVERSAL ---

20 40 50 60 80

--- POSTORDER TRAVERSAL ---

20 40 80 60 50

--- PREORDER TRAVERSAL ---

50 40 20 60 80

MINIMUM VALUE IS = 20

Enter Element to be searched: 80

Element Found

Depth of Tree is : 3

**Experiment No5**

Construct an expression tree from the given prefix expression eg. +--a\*bc/def and

traverse it using post order traversal (non recursive) .

#include <iostream>

#include <string.h>

using namespace std;

struct node

{

char data;

node \*left;

node \*right;

};

class tree

{

char prefix[20];

public:

node \*top;

void expression(char[]);

void display(node \*);

void non\_rec\_postorder(node \*);

void del(node \*);

};

class stack1

{

node \*data[30];

int top;

public:

stack1()

{

top = -1;

}

int empty()

{

if (top == -1)

return 1;

return 0;

}

void push(node \*p)

{

data[++top] = p;

}

node \*pop()

{

return (data[top--]);

}

};

void tree::expression(char prefix[])

{

char c;

stack1 s;

node \*t1, \*t2;

int len, i;

len = strlen(prefix);

for (i = len - 1; i >= 0; i--)

{

top = new node;

top->left = NULL;

top->right = NULL;

if (isalpha(prefix[i]))

{

top->data = prefix[i];

s.push(top);

}

else if (prefix[i] == '+' || prefix[i] == '\*' || prefix[i] == '-' || prefix[i] == '/')

{

t2 = s.pop();

t1 = s.pop();

top->data = prefix[i];

top->left = t2;

top->right = t1;

s.push(top);

}

}

top = s.pop();

}

void tree::display(node \*root)

{

if (root != NULL)

{

cout << root->data;

display(root->left);

display(root->right);

}

}

void tree::non\_rec\_postorder(node \*top)

{

stack1 s1, s2; /\*stack s1 is being used for flag . A NULL data implies that the right subtree has not been visited \*/

node \*T = top;

cout << "\n";

s1.push(T);

while (!s1.empty())

{

T = s1.pop();

s2.push(T);

if (T->left != NULL)

s1.push(T->left);

if (T->right != NULL)

s1.push(T->right);

}

while (!s2.empty())

{

top = s2.pop();

cout <<top->data;

}

}

void tree::del(node \*node)

{

if (node == NULL)

return;

/\* first delete both subtrees \*/

del(node->left);

del(node->right);

free(node);

}

int main()

{

char expr[20];

tree t;

cout <<"Enter prefix Expression : ";

cin >> expr;

t.expression(expr);

cout <<"Postfix experession is : ";

t.non\_rec\_postorder(t.top);

t.del(t.top);

}

**Output:**

Enter prefix Expression : +--a\*bc/def

Postfix experession is :

abc\*-de/-f+

**Experiment No 6**

Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

#include <iostream>

#include <stdlib.h>

using namespace std;

int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10], visited[10];

int stk[10], top, visit1[10], visited1[10];

int main()

{

int m;

cout << "Enter number of vertices : ";

cin >> n;

cout << "Enter number of edges : ";

cin >> m;

cout << "\nEDGES :\n";

for (k = 1; k <= m; k++)

{

cin >> i >> j;

cost[i][j] = 1;

cost[j][i] = 1;

}

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

{

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

{

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

}

cout << endl;

}

//display function

cout << "Enter initial vertex : ";

cin >> v;

cout << "The BFS of the Graph is: \n";

cout <<v<<" ";

visited[v] = 1;

k = 1;

while (k < n)

{

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

if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)

{

visit[j] = 1;

qu[rear++] = j;

}

v = qu[front++];

cout << v << " ";

k++;

visit[v] = 0;

visited[v] = 1;

}

cout <<endl<<"Enter initial vertex : ";

cin >> v;

cout << "The DFS of the Graph is :\n";

visited[v] = 1;

k = 1;

while (k <= n)

{

for (j = n; j >= 1; j--)

if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)

{

visit1[j] = 1;

stk[top] = j;

top++;

}

v = stk[--top];

cout << v << " ";

k++;

visit1[v] = 0;

visited1[v] = 1;

}

return 0;

}

Output:

Enter number of vertices : 3

Enter number of edges : 3

EDGES :

1 2

2 3

1 3

0 1 1

1 0 1

1 1 0

Enter initial vertex : 3

The BFS of the Graph is:

3 1 2

Enter initial vertex : 3

The DFS of the Graph is :

1 3 2

**Experiment No 7**

You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

#include<iostream>

#include<climits>

using namespace std;

template <class T>

class Graph

{ int \*\* AM,num;

T \* data;

public:

Graph(int n)

{ AM=new int\*[n];

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

AM[i]=new int[n];

num=n;

data=new T[n];

cout<<"Enter names of all cities \n: ";

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

cin>>data[i];

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

cout<<data[j]<<" ";

cout<<endl;

cout<<"Enter cost if you want to connect cities else enter 0: \n";

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

{

cout<<"Nodes connected to "<<data[i]<<" :\n";

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

cout<<AM[i][j]<<"\t";

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

if(j==i)

{

cout<<"0\t";

AM[i][j]=AM[j][i]=0;

}

else {

cin>>cost;

AM[i][j]=AM[j][i]=cost;

}

}

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

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

if(AM[i][j]==0)AM[i][j]=INT\_MAX;

}

void prims()

{

cout<<"\nCities that we need to connect:\n";

int \*visited=new int[num](),\*distance=new int[num],\*from=new int[num](),cost=0;

visited[0]=1;

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

distance[i]=AM[0][i];

int u,v;

for(int count=num-1;count>0;count--)

{ int min=INT\_MAX;

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

if(visited[j]==0&&distance[j]<min)

{v=j;min=distance[j];}

u=from[v];

cout<<data[u]<<"==>"<<data[v]<<"\tcost: "<<AM[u][v]<<endl;

visited[v]=1;

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

if(visited[j]==0&&AM[j][v]<distance[j])

{distance[j]=AM[j][v];from[j]=v;}

cost+=AM[u][v];

}

cout<<"Total cost of connecting all cities : "<<cost<<endl;

}

};

int main()

{ int n;

cout<<"Enter number of cities: ";

cin>>n;

Graph<string> gr(n);

gr.prims();

return 0;

}

**Output**

Enter number of cities: 3

Enter names of all cities

: m

p

n

m p n

Enter cost if you want to connect cities else enter 0:

Nodes connected to m :

0 200 100

Nodes connected to p :

200 0 800

Nodes connected to n :

100 800 0

Cities that we need to connect:

m==>n cost: 100

m==>p cost: 200

Total cost of connecting all cities : 300

**Experiment 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?

#include<iostream>

using namespace std;

#define SIZE 10

class OBST

{

int p[SIZE]; // Probabilities with which we search for an element(frequency)

int q[SIZE];//Probabilities that an element is not found (frequency)

int a[SIZE];//Elements from which OBST is to be built

int w[SIZE][SIZE];//Weight ‘w[i][j]’ of a tree having root ’r[i][j]’

int c[SIZE][SIZE];//Cost ‘c[i][j] of a tree having root ‘r[i][j]

int r[SIZE][SIZE];//represents root

int n; // number of nodes

public:

/\* This function accepts the input data \*/

void get\_data()

{

int i;

cout<<"\n Optimal Binary Search Tree \n";

cout<<"\n Enter the number of nodes";

cin>>n;

cout<<"\n Enter the data as…\n";

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

{

cout<<"\n a["<<i<<"]";

cin>>a[i];

}

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

{

cout<<"\n p["<<i<<"]";

cin>>p[i];

}

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

{

cout<<"\n q["<<i<<"]";

cin>>q[i];

}

}

/\* This function returns a value in the range ‘r[i][j-1]’ to ‘r[i+1][j]’so

that the cost ‘c[i][k-1]+c[k][j]’is minimum \*/

int Min\_Value(int i,int j)

{

int m,k;

int minimum=32000;

for(m=r[i][j-1];m<=r[i+1][j];m++)

{

if((c[i][m-1]+c[m][j])<minimum)

{

minimum=c[i][m-1]+c[m][j];

k=m;

}

}

return k;

}

/\* This function builds the table from all the given probabilities It

basically computes C,r,W values \*/

void build\_OBST()

{

int i,j,k,l,m;

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

{

//initialize

w[i][i]=q[i];

r[i][i]=c[i][i]=0;

//Optimal trees with one node

w[i][i+1]=q[i]+q[i+1]+p[i+1];

r[i][i+1]=i+1;

c[i][i+1]=q[i]+q[i+1]+p[i+1];

}

w[n][n]=q[n];

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

//Find optimal trees with ‘m’ nodes

for(m=2;m<=n;m++)

{

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

{

j=i+m;

w[i][j]=w[i][j-1]+p[j]+q[j];

k=Min\_Value(i,j);

c[i][j]=w[i][j]+c[i][k-1]+c[k][j];

r[i][j]=k;

}

}

}

/\* This function builds the tree from the tables made by the OBST function \*/

void build\_tree()

{

int i,j,k;

int queue[20],front=-1,rear=-1;

cout<<"The Optimal Binary Search Tree For the Given Node Is…\n";

cout<<"\n The Root of this OBST is ::"<<r[0][n];

cout<<"\nThe Cost of this OBST is::"<<c[0][n];

cout<<"\n\n\t NODE \t LEFT CHILD \t RIGHT CHILD ";

cout<<"\n";

queue[++rear]=0;

queue[++rear]=n;

while(front!=rear)

{

i=queue[++front];

j=queue[++front];

k=r[i][j];

cout<<"\n\t"<<k;

if(r[i][k-1]!=0)

{

cout<<"\t\t"<<r[i][k-1];

queue[++rear]=i;

queue[++rear]=k-1;

}

else

cout<<"\t\t";

if(r[k][j]!=0)

{

cout<<"\t"<<r[k][j];

queue[++rear]=k;

queue[++rear]=j;

}

else

cout<<"\t";

}//end of while

cout<<"\n";

}

};//end of the class

/\*This is the main function \*/

int main()

{

OBST obj;

obj.get\_data();

obj.build\_OBST();

obj.build\_tree();

return 0;

}

**Output**

Optimal Binary Search Tree

Enter the number of nodes3

Enter the data as…

a[1]5

a[2]2

a[3]8

p[1]1

p[2]1

p[3]1

q[0]0

q[1]0

q[2]0

q[3]

0

The Optimal Binary Search Tree For the Given Node Is…

The Root of this OBST is ::2

The Cost of this OBST is::5

NODE LEFT CHILD RIGHT CHILD

2 1 3

1

3

**Experiment No :9**

A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.

#include<iostream>

#include<string.h>

using namespace std;

class dict

{

dict \*root,\*node,\*left,\*right,\*tree1;

string s1,s2;

int flag,flag1,flag2,flag3,cmp;

public:

dict()

{

flag=0,flag1=0,flag2=0,flag3=0,cmp=0;

root=NULL;

}

void input();

void create\_root(dict\*,dict\*);

void check\_same(dict\*,dict\*);

void input\_display();

void display(dict\*);

void input\_remove();

dict\* remove(dict\*,string);

dict\* findmin(dict\*);

void input\_find();

dict\* find(dict\*,string);

void input\_update();

dict\* update(dict\*,string);

};

void dict::input()

{

node=new dict;

cout<<"\nEnter the keyword:\n";

cin>>node->s1;

cout<<"Enter the meaning of the keyword:\n";

cin.ignore();

getline(cin,node->s2);

create\_root(root,node);

}

void dict::create\_root(dict \*tree,dict \*node1)

{

int i=0,result;

char a[20],b[20];

if(root==NULL)

{

root=new dict;

root=node1;

root->left=NULL;

root->right=NULL;

cout<<"\nRoot node created successfully"<<endl;

return;

}

for(i=0;node1->s1[i]!='\0';i++)

{

a[i]=node1->s1[i];

}

for(i=0;tree->s1[i]!='\0';i++)

{

b[i]=tree->s1[i];

}

result=strcmp(b,a);

check\_same(tree,node1);

if(flag==1)

{

cout<<"The word you entered already exists.\n";

flag=0;

}

else

{

if(result>0)

{

if(tree->left!=NULL)

{

create\_root(tree->left,node1);

}

else

{

tree->left=node1;

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

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

cout<<"Node added to left of "<<tree->s1<<"\n";

return;

}

}

else if(result<0)

{

if(tree->right!=NULL)

{

create\_root(tree->right,node1);

}

else

{

tree->right=node1;

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

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

cout<<"Node added to right of "<<tree->s1<<"\n";

return;

}

}

}

}

void dict::check\_same(dict \*tree,dict \*node1)

{

if(tree->s1==node1->s1)

{

flag=1;

return;

}

else if(tree->s1>node1->s1)

{

if(tree->left!=NULL)

{

check\_same(tree->left,node1);

}

}

else if(tree->s1<node1->s1)

{

if(tree->right!=NULL)

{

check\_same(tree->right,node1);

}

}

}

void dict::input\_display()

{

if(root!=NULL)

{

cout<<"The words entered in the dictionary are:\n\n";

display(root);

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

void dict::display(dict \*tree)

{

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

{

cout<<tree->s1<<" = "<<tree->s2<<"\n\n";

}

else

{

if(tree->left!=NULL)

{

display(tree->left);

}

cout<<tree->s1<<" = "<<tree->s2<<"\n\n";

if(tree->right!=NULL)

{

display(tree->right);

}

}

}

void dict::input\_remove()

{

char t;

if(root!=NULL)

{

cout<<"\nEnter a keyword to be deleted:\n";

cin>>s1;

remove(root,s1);

if(flag1==0)

{

cout<<"\nThe word '"<<s1<<"' has been deleted.\n";

}

flag1=0;

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

dict\* dict::remove(dict \*tree,string s3)

{

dict \*temp;

if(tree==NULL)

{

cout<<"\nWord not found.\n";

flag1=1;

return tree;

}

else if(tree->s1>s3)

{

tree->left=remove(tree->left,s3);

return tree;

}

else if(tree->s1<s3)

{

tree->right=remove(tree->right,s3);

return tree;

}

else

{

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

{

delete tree;

tree=NULL;

}

else if(tree->left==NULL)

{

temp=tree;

tree=tree->right;

delete temp;

}

else if(tree->right==NULL)

{

temp=tree;

tree=tree->left;

delete temp;

}

else

{

temp=findmin(tree->right);

tree=temp;

tree->right=remove(tree->right,temp->s1);

}

}

return tree;

}

dict\* dict::findmin(dict \*tree)

{

while(tree->left!=NULL)

{

tree=tree->left;

}

return tree;

}

void dict::input\_find()

{

flag2=0,cmp=0;

if(root!=NULL)

{

cout<<"\nEnter the keyword to be searched:\n";

cin>>s1;

find(root,s1);

if(flag2==0)

{

cout<<"Number of comparisons needed: "<<cmp<<"\n";

cmp=0;

}

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

dict\* dict::find(dict \*tree,string s3)

{

if(tree==NULL)

{

cout<<"\nWord not found.\n";

flag2=1;

flag3=1;

cmp=0;

}

else

{

if(tree->s1==s3)

{

cmp++;

cout<<"\nWord found.\n";

cout<<tree->s1<<": "<<tree->s2<<"\n";

tree1=tree;

return tree;

}

else if(tree->s1>s3)

{

cmp++;

find(tree->left,s3);

}

else if(tree->s1<s3)

{

cmp++;

find(tree->right,s3);

}

}

return tree;

}

void dict::input\_update()

{

if(root!=NULL)

{

cout<<"\nEnter the keyword to be updated:\n";

cin>>s1;

update(root,s1);

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

dict\* dict::update(dict \*tree,string s3)

{

flag3=0;

find(tree,s3);

if(flag3==0)

{

cout<<"\nEnter the updated meaning of the keyword:\n";

cin.ignore();

getline(cin,tree1->s2);

cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";

}

return tree;

}

int main()

{

int ch;

dict d;

do

{

cout<<"\n==========================================\n"

"\n\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:\n"

"\nEnter your choice:\n"

"1.Add new keyword.\n"

"2.Display the contents of the Dictionary.\n"

"3.Delete a keyword.\n"

"4.Find a keyword.\n"

"5.Update the meaning of a keyword.\n"

"6.Exit.\n"

"===============================================\n";

cin>>ch;

switch(ch)

{

case 1:d.input();

break;

case 2:d.input\_display();

break;

case 3:d.input\_remove();

break;

case 4:d.input\_find();

break;

case 5:d.input\_update();

break;

default:cout<<"\nPlease enter a valid option!\n";

break;

}

}while(ch!=6);

return 0;

}

**Output**

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

1

Enter the keyword:

OS

Enter the meaning of the keyword:

Operating System

Root node created successfully

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

1

Enter the keyword:

DS

Enter the meaning of the keyword:

Data Structures

Node added to left of OS

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

1

Enter the keyword:

SE

Enter the meaning of the keyword:

Software Engineering

Node added to right of OS

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

1

Enter the keyword:

MP

Enter the meaning of the keyword:

Microprocessor

Node added to right of DS

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

2

The words entered in the dictionary are:

DS = Data Structures

MP = Microprocessor

OS = Operating System

SE = Software Engineering

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

3

Enter a keyword to be deleted:

OS

Word not found.

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

2

The words entered in the dictionary are:

DS = Data Structures

MP = Microprocessor

OS = Operating System

SE = Software Engineering

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

4

Enter the keyword to be searched:

DS

Word found.

DS: Data Structures

Number of comparisons needed: 2

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

5

Enter the keyword to be updated:

MP

Word found.

MP: Microprocessor

Enter the updated meaning of the keyword:

Microprocessor and Tech

The meaning of 'MP' has been updated.

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

2

The words entered in the dictionary are:

DS = Data Structures

MP = Microprocessor and Tech

OS = Operating System

SE = Software Engineering

==========================================

\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:

Enter your choice:

1.Add new keyword.

2.Display the contents of the Dictionary.

3.Delete a keyword.

4.Find a keyword.

5.Update the meaning of a keyword.

6.Exit.

===============================================

6

**Experiment 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.

**Code:**

#include<iostream>

using namespace std;

class hp

{

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

public:

string sub;

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 Subject : \n";

cin>>sub;

cout<<"\n Enter the marks of students: \n";

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 Maximum marks in "<<sub<<" are : \n"<<heap[1];

cout<<"\n Minimum marks in "<<sub<< " are : \n" <<heap1[1];

}

int main()

{

hp h;

h.getdata();

h.minmax();

return 0;

}

**Output**

Enter the no. of students : 5

Enter the Subject :

DSA

Enter the marks of students:

20

17

29

15

10

Maximum marks in DSA are :

29

Minimum marks in DSA are :

10

**Experiment 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<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);

}

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 : 1

Enter the Name : ABC

Enter the Division:A

Enter the Address:abcde

Do you want to enter more records?

1.Yes

2.No1

Enter Roll Number : 2

Enter the Name : DEF

Enter the Division:B

Enter the Address:defg

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

1 ABC A abcde

2 DEF B defg

>>>>>>>>>>>>>>>>>>>>>>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 find2

Record found

Enter Roll Number : 3

Enter the Name : PQR

Enter the Division:C

Enter the Address:pqrs

>>>>>>>>>>>>>>>>>>>>>>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

1 ABC A abcde

3 PQR C pqrs

>>>>>>>>>>>>>>>>>>>>>>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 editPQR

Name found

Enter Roll Number : 4

Enter the Name : DEF

Enter the Division:D

Enter the Address:defg

>>>>>>>>>>>>>>>>>>>>>>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

1 ABC A abcde

4 DEF D defg

>>>>>>>>>>>>>>>>>>>>>>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 modify4

Number found

Enter the Roll No. to modify : 5

>>>>>>>>>>>>>>>>>>>>>>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

1 ABC A abcde

5 DEF D defg

>>>>>>>>>>>>>>>>>>>>>>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 :6

Enter the name you want to find and editABC

Name found

Enter the name to modify : XYZ

>>>>>>>>>>>>>>>>>>>>>>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

1 XYZ A abcde

5 DEF D defg

>>>>>>>>>>>>>>>>>>>>>>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: 1

The record with the roll no. 1 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 :2

5 DEF D defg

>>>>>>>>>>>>>>>>>>>>>>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