Program 1:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct Details
{
    char name[30];
    long num;
    Details()
    {
        name[0]='\0';
        num=0;
    }
};

class phoneDict{
    private:
        int n;
        char name[30];
        long phonum;
        int hashfunc;
        char searchkey[30];
        int rhashfunc;
        bool flag;

    public:
         void insert(Details arr[20])
        {
                cout<<"Enter the name of contact: ";
                cin>>name;
                cout<<"Enter the Contact number: ";
                cin>>phonum;
                hashfunc = int(name[0]) % 10;
                for(int i=0;i<20;i++)
            {

                    if(arr[hashfunc].name[0]=='\0')
                {
                    strcpy(arr[hashfunc].name,name);
                    arr[hashfunc].num=phonum;
                    cout<<"\nRecord created and inserted succesfully\n";
                    return;

                }
                else
                {
                    hashfunc=(hashfunc+1)%20;
                }
```

```
    }
}
 void search(Details arr[])
 {

    cout<<"\nEnter the name you want to search: ";
    cin>>searchkey;
    rhashfunc = int(searchkey[0]) % 10;
    cout<<"\nname\t\t\tphone num\n";
    for(int i=0;i<20;i++)
    {
        if(strcmp(arr[rhashfunc].name,searchkey)==0)
        {
            cout<<arr[rhashfunc].name<<"\t\t\t";
            cout<<arr[rhashfunc].num<<"\t\t\t\n";
            flag = true;
            return;
        }
        else
        {
            rhashfunc=(rhashfunc+1)%20;
        }
    }
    if(flag){
        cout<<"\nrecords are found\n";
    }
    else{
        cout<<"\nrecords  not found\n";
    }

 }
 void Delete(Details arr[])
 {
    cout<<"\nEnter the name you want to delete: ";
    cin>>searchkey;
    rhashfunc = int(searchkey[0]) % 10;
    for(int i=0;i<20;i++)
    {
        if(strcmp(arr[rhashfunc].name,searchkey)==0){
            arr[rhashfunc].name[0]='\0';
            cout<<"\nRecord Deleted";

            return;

        }
        {
            rhashfunc=(rhashfunc+1)%20;
```

```
            }

        }

            cout<<"\nRecord not found";
            return;
    }
    void Display(Details arr[])
    {
        cout<<"\nname\t\t\tphone num\n";
        for(int i=0;i<20;i++)
        {
          if(arr[i].name[0]!='\0')
          {
              cout<<arr[i].name<<"\t\t\t";
              cout<<arr[i].num<<"\t\t\t\n";
           }
        }
    }
};
int main(){
    Details arr[20];
    phoneDict a;
    int n;
    int ch;


do{
    cout<<"\n####MENU####\n";
    cout<<"1.Insert\n";
    cout<<"2.search\n";
    cout<<"3.delete\n";
    cout<<"4.Dispaly\n";
    cout<<"5.Exit\n";
    cout<<"Enter your choice: ";
    cin>>ch;
    switch(ch)
    {
      case 1:
        cout<<"Enter how many contacts you want to add: ";
        cin>>n;
        for(int i=0;i<n;i++)
        {
           a.insert(arr);
        }
        break;
      case 2:
        a.search(arr);
```

```
            break;
        case 3:
            a.Delete(arr);
            break;
        case 4:
            a.Display(arr);
            break;
        case 5:
            cout<<"Thanks for using\n";
            break;
    }
}while(ch!=5);
return 0;

}
```

```
####MENU####
1.Insert
2.search
3.delete
4.Dispaly
5.Exit
Enter your choice: 1
Enter how many contacts you want to add: 1
Enter the name of contact: Student1
Enter the Contact number: 9898989898

Record created and inserted succesfully

####MENU####
1.Insert
2.search
3.delete
4.Dispaly
5.Exit
Enter your choice: 2

Enter the name you want to search: Student1

name                    phone num
Student1                        9898989898

####MENU####
1.Insert
2.search
3.delete
4.Dispaly
5.Exit
Enter your choice: 4

name                    phone num
Student1                        9898989898
```

Program 2:

```
from ctypes import sizeof
print("hello world Welcome to sets")
a=set([])
MO=int(input("Enter the total number in set A"))
for i in range(MO):
	Me=int(input("Enter Elements "))
	a.add(Me);
b=set([])
NO=int(input("Enter the total number in set B"))
for i in range(NO):
	M=int(input("Enter Elements "))
	b.add(M);
print("set A:",a)
print("set B:",b)
while 1:
	print("\n\n\n*****MENU*****\n")
	print("1 :Union")
	print("2 :Intersection")
	print("3 :Set Difference")
	print("4 :Subset")
	print("5 :Is empty")
	print("6 :Len of A")
	print("7 :Len of B")
	print("8 :size of A and B")
	print("9 :searching")
	print("10:add element")
	print("11:Remove Element")
	print("12 for exit")
	ch=int(input("\n****Enter your choice****\n"))
	if ch==1:
		print("\nUnion of A and B is",a.union(b))
	if ch==2:
		print("Intersection of set A and B is",a.intersection(b))
	if ch==3:
		print("\nSet difference A-B as",a-b)
	if ch==4:
		if a.issubset(b):
			print("\nis subset")
		else:
			print("\n is not subset")
	if ch==5:
		if len(a)==0:
			print("\nSet A is empty")
		else:
			print("\nSet A is not empty")
	if ch==6:
```

```
        print("\nLen A=",len(a))
    if ch==7:
        print("\nLen B=",len(b))
    if ch==8:
        print("\nsize of A is :",sizeof(a))
        print("\nsize of B is :",sizeof(b))
    if ch==9:
        Num=int(input("\nEnter Element to be searched "))
        if Num in a:
            print("\nNumber is found in set A")
        elif num in b:
            print("\nNumber is found in set B")
        else:
            print("\nSorry! Number not found")
    if ch==10:
        ad=int (input("Enter number to be added in set A"))
        a.add(ad)
        print("\nnumber Added in set A")
        print("\nUpdated set A is",a)
    if ch==11:
        rm==int(input("\nEnter a number to delete"))
        b.remove(rm)
        print("\nNumber removed")
        print("\nUpdated se B is",b)
    if ch==12:
        break;
```

Outputs:

```
hello world Welcome to sets
Enter the total number in set A3
Enter Elements 11
Enter Elements 12
Enter Elements 13
Enter the total number in set B3
Enter Elements 13
Enter Elements 14
Enter Elements 15
set A: {11, 12, 13}
set B: {13, 14, 15}
```

```
*****MENU*****

1 :Union
2 :Intersection
3 :Set Difference
4 :Subset
5 :Is empty
6 :Len of A
7 :Len of B
8 :size of A and B
9 :searching
10:add element
11:Remove Element
12 for exit

****Enter your choice****
1

Union of A and B is {11, 12, 13, 14, 15}
```

```
*****MENU*****

1 :Union
2 :Intersection
3 :Set Difference
4 :Subset
5 :Is empty
6 :Len of A
7 :Len of B
8 :size of A and B
9 :searching
10:add element
11:Remove Element
12 for exit

****Enter your choice****
2
Intersection of set A and B is {13}
```

```
*****MENU*****

1 :Union
2 :Intersection
3 :Set Difference
4 :Subset
5 :Is empty
6 :Len of A
7 :Len of B
8 :size of A and B
9 :searching
10:add element
11:Remove Element
12 for exit

****Enter your choice****
3

Set difference A-B as {11, 12}
```

```
*****MENU*****

1 :Union
2 :Intersection
3 :Set Difference
4 :Subset
5 :Is empty
6 :Len of A
7 :Len of B
8 :size of A and B
9 :searching
10:add element
11:Remove Element
12 for exit

****Enter your choice****
6

Len A= 3
```

Program 3:

```
#include <iostream>
#include <string.h>
using namespace std;

struct node // Node Declaration
{
    string label;
    //char label[10];
    int ch_count;
    struct node *child[10];
} * root;

class GT // Class Declaration
{
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.get();
    getline(cin, root->label);
    cout << "Enter number of chapters in book : ";
    cin >> tchapters;
    root->ch_count = tchapters;
    for (i = 0; i < tchapters; i++)
    {
        root->child[i] = new node;
        cout << "Enter the name of Chapter " << i + 1 << " : ";
        cin.get();
        getline(cin, root->child[i]->label);
        cout << "Enter number 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 Name of Section " << j + 1 << " : ";
            cin.get();
            getline(cin, root->child[i]->child[j]->label);
        }
    }
}

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 << "\nChapter " << i + 1;
            cout << " : " << r1->child[i]->label;
            cout << "\nSections : ";
            for (j = 0; j < r1->child[i]->ch_count; j++)
            {
                cout << "\n"<< r1->child[i]->child[j]->label;
            }
        }
    }
    cout << endl;
}

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:
            cout << "Thanks for using this program!!!";
            exit(1);
        default:
            cout << "Wrong choice!!!" << endl;
        }
    }
    return 0;

}
```

```
-----------------
Book Tree Creation
-----------------
1.Create
2.Display
3.Quit
Enter your choice : 1
Enter name of book : DSA
Enter number of chapters in book : 1
Enter the name of Chapter 1 : Hashing
Enter number of sections in Chapter : Hashing : 1
Enter Name of Section 1 : Hashing_Types

-----Book Hierarchy---
 Book title : DSA
Chapter 1 : Hashing
Sections :
Hashing_Types
----------------
Book Tree Creation
-----------------
1.Create
2.Display
3.Quit
Enter your choice :
```

Program 4:

```
/*

Author: Parag Ghorpade

PROBLEM STATEMENT:

Dictionary stores keywords & its meanings. Provide facility for adding new
keywords, deleting keywords, updating values of any entry.
Also,display whole data sorted in ascending or descending order.
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;
  }
```

```
=========================================

********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:
Practical4

Word found.
Practical4: data structures and algorithms practical number 4
Number of comparisons needed: 1
```

```
=========================================

********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:

Practical4 = data structures and algorithms practical number 4
```

```
=========================================

********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:
Practical4
Enter the meaning of the keyword:
data structures and algorithms practical number 4

Root node created successfully
```

Program 5:

```
#include<iostream>

#include<stdlib.h>
using namespace std;
struct node
{
    int data;
    node *left,*right;
    int lbit,rbit;
};
class tbt
{
   node *temp=NULL,*t1=NULL,*s=NULL,*head=NULL,*t=NULL;
   public:

   node *create();
   void insert();
   node *insuc(node*);
   node *inpre(node*);
   void dis();
   void display(node*);
   void thr();
   void thread(node*);
};
node *tbt::create()
{
   node *p=new(struct node);
   p->left=NULL;
   p->right=NULL;
   p->lbit=0;
   p->rbit=0;
   cout<<"\n enter the data";
   cin>>p->data;
   return p;
}
void tbt::insert()
{
    temp=create();
    if(head==NULL)
    {  node *p=new(struct node);
       head=p;
       head->left=temp;
       head->right=head;
       head->lbit=1;
```

```
      head->rbit=0;
      temp->left=head;
      temp->right=head;
      temp->lbit=0;
      temp->rbit=0;
   }
   else
   {     t1=head;
         t1=t1->left;

         while(t1!=NULL)
         {   s=t1;
             if(((temp->data)>(t1->data))&&t1->rbit==1)
             {   t1=t1->right;   }
             else if(((temp->data)<(t1->data))&&t1->lbit==1)
             {   t1=t1->left;     }
             else
             { break;       }
         }
         if(temp->data>s->data)
         {
            s->right=temp;
            s->rbit=1;
            temp->left=inpre(head->left);
            temp->right=insuc(head->left);
         }
         else
         {
            s->left=temp;
            s->lbit=1;
            temp->left=inpre(head->left);
            temp->right=insuc(head->left);
         }
    }




}
node *tbt::inpre(node *m)
{
   if(m->lbit==1)
   {
     inpre(m->left);
   }
   if(m->data==temp->data&&t==NULL)
   {  return head;       }
   if(m->data==temp->data)
```

```
    {   return t;         }
    t=m;
    if(m->rbit==1)
    { inpre(m->right);
    }

}
node *tbt::insuc(node *m)
{
    if(m->lbit==1)
    {  t=m;
      insuc(m->left);
    }

    if(m->data==temp->data&&t==NULL)
    {  return head;       }
    if(m->data==temp->data)
    {   return t;        }

    if(m->rbit==1)
    { insuc(m->right);
    }
}
void tbt::dis()
{   display(head->left);
}
void tbt::display(node *m)
{

       if(m->lbit==1)
       {   display(m->left);              }
       cout<<"\n"<<m->data;
       if(m->rbit==1)
       {    display(m->right);             }



}
void tbt::thr()
{  cout<<"\n thread are";
   thread(head->left);
}
void tbt::thread(node *m)
{

       if(m->lbit==1)
       {   thread(m->left);              }
       if(m->lbit==0||m->rbit==0)
```

```
        {
        cout<<"\n"<<m->data;
        }
        if(m->rbit==1)
        {    thread(m->right);            }



}
int main()
{  tbt t;  int ch;
   while(1)
   {

   cout<<"\n enter the choice";
   cout<<"\n 1.insert data";
   cout<<"\n 2.display all data";
   cout<<"\n 3.display threaded node";
   cout<<"\n 4.exit";
   cin>>ch;
   switch(ch)
   {
      case 1:
               t.insert();
               break;
      case 2:
               t.dis();
               break;
      case 3:
               t.thr();
               break;
      case 4:  exit(0);

      default:
              cout<<"\n invalid entry";
    }
    }
   return 0;
}
```

```
student@student-OptiPlex-3010: ~/Documents/anamikadsl                    11:48 AM
(base) student@student-OptiPlex-3010:~$ cd /home/student/Documents/anamikadsl
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$ g++ assign5.cpp
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$ ./a.out

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit1

 enter the data34

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit1

 enter the data45

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit1

 enter the data90

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit1

 enter the data89

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit2

34
```

```
student@student-OptiPlex-3010: ~/Documents/anamikadsl                    11:49 AM
 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit1

 enter the data90

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit1

 enter the data89

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit2

34
45
89
90
 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit3

 thread are
34
45
89
90
 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit4
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$
```

Program 6:

```
#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;
}
```

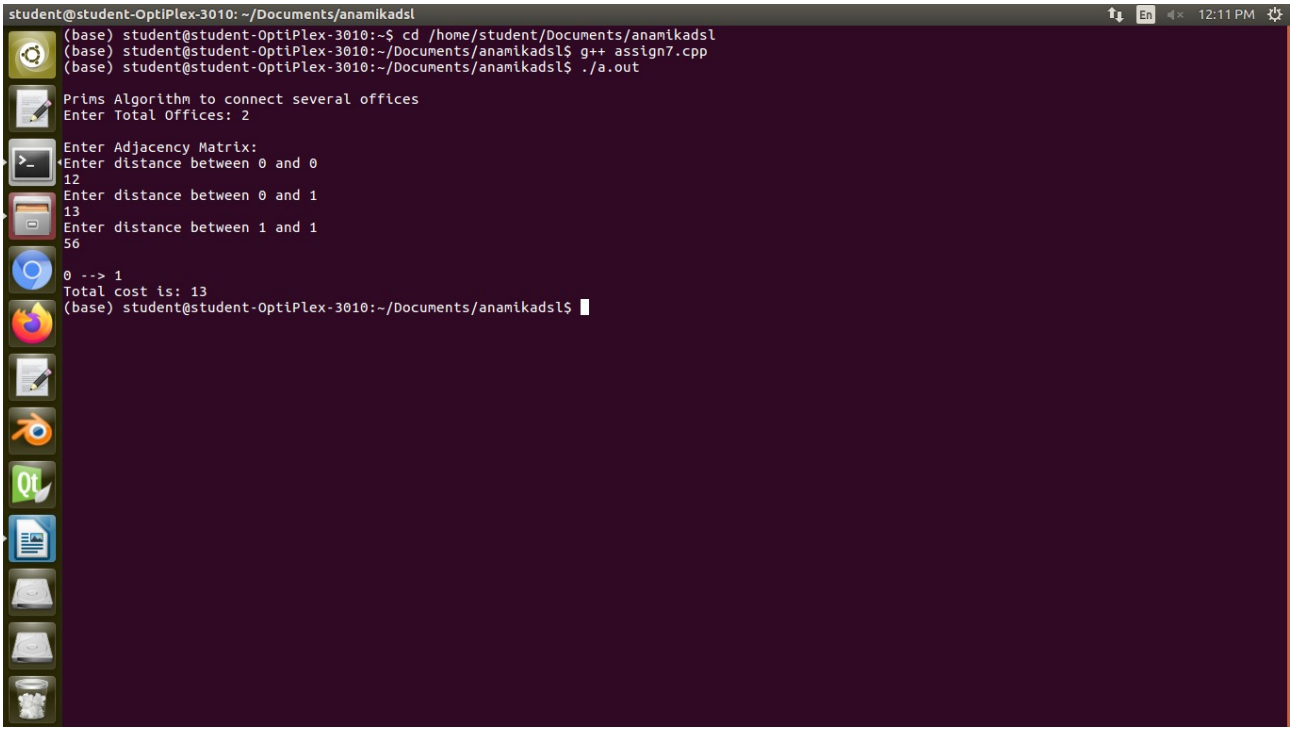
```
student@student-OptiPlex-3010: ~/Documents/anamikadsl
(base) student@student-OptiPlex-3010:~$ cd /home/student/Documents/anamikadsl
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$ g++ assign6.cpp
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$ ./a.out


 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)2

 enter name of citiespune
jalgaon

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

 enter time required to reach city pune from pune in minutes0

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

 enter time required to reach city jalgaon from pune in minutes400

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

 enter time required to reach city pune from jalgaon in minutes500

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

 enter time required to reach city jalgaon from jalgaon in minutes23

 ****
pune
jalgaon

 enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit2

        pune    jalgaon
```

```
student@student-OptiPlex-3010: ~/Documents/anamikadsl
 if path is present between city jalgaon and jalgaon then press enter y otherwise ny

 enter time required to reach city jalgaon from jalgaon in minutes23

 ****
pune
jalgaon

 enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit2

        pune    jalgaon
 pune   0       400

 jalgaon        500     23


 enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit3

 adjacency list is
pune-> jalgaon
jalgaon-> pune-> jalgaon
 path and time required to reach cities is
pune-> jalgaon
   [time required: 400 min ]
jalgaon-> pune
   [time required: 500 min ]
jalgaon-> jalgaon
   [time required: 23 min ]

 enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit4
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$
```

Program 7:

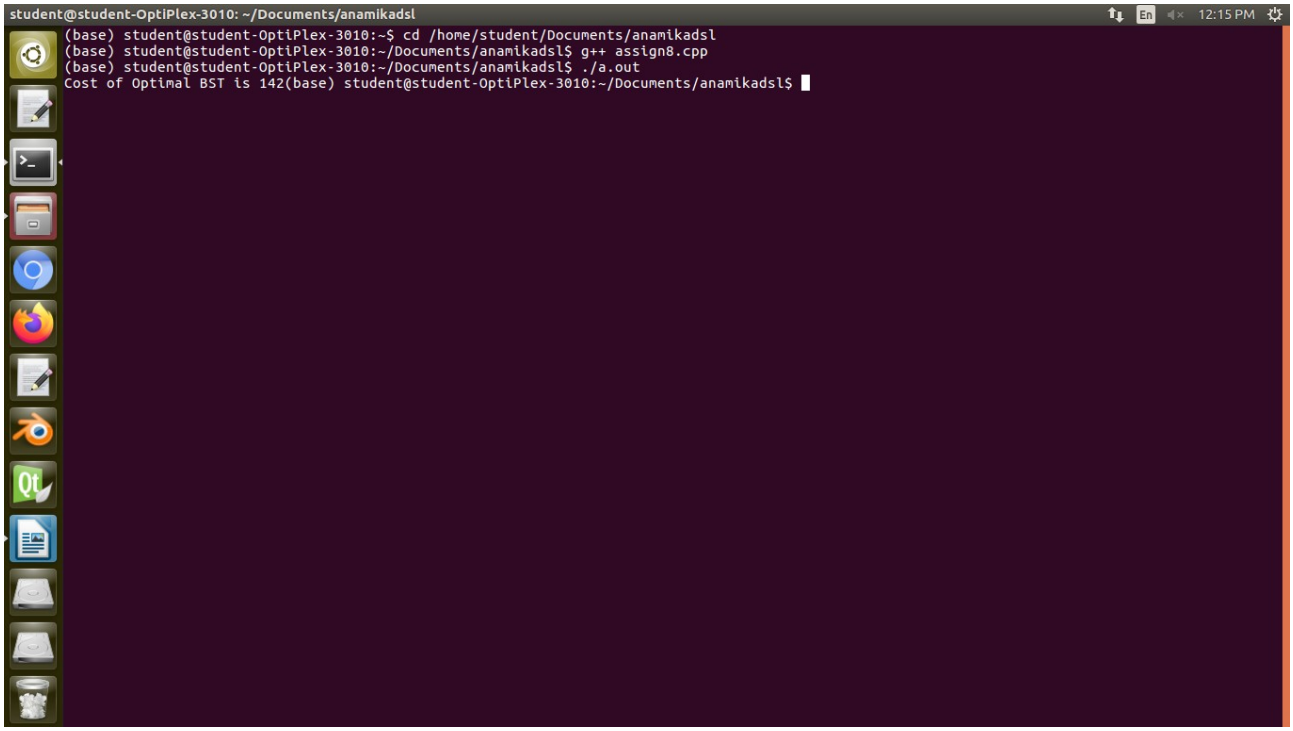
```
#include<iostream>

using namespace std;

#define ROW 10
#define COL 10
#define infi 9999

class prims {
    int graph[ROW][COL], nodes;
public:
    void createGraph();
    void primsAlgo();
};

void prims::createGraph() {
    int i, j;
    cout << "Enter Total Offices: ";
    cin >> nodes;
    cout << "\nEnter Adjacency Matrix: \n";
    for (i = 0; i < nodes; i++) {
        for (j = i; j < nodes; j++) {
            cout << "Enter distance between " << i << " and " << j << endl;
            cin >> graph[i][j];
            graph[j][i] = graph[i][j];
        }
    }

    for (i = 0; i < nodes; i++) {
        for (j = 0; j < nodes; j++) {
            if (graph[i][j] == 0)
                graph[i][j] = infi;    //fill infinity where path is not present
        }
    }
}

void prims::primsAlgo() {
    int selected[ROW], i, j, ne=0;
    int zero = 0, one = 1, min = 0, x, y;
    int cost = 0;
    for (i = 0; i < nodes; i++)
        selected[i] = zero;

    selected[0] = one;        //starting vertex is always node-0

    while (ne < nodes - 1) {
```

```
        min = infi;

        for (i = 0; i < nodes; i++) {
            if (selected[i] == one) {
                for (j = 0; j < nodes; j++) {
                    if (selected[j] == zero) {
                        if (min > graph[i][j]) {
                            min = graph[i][j];
                            x = i;
                            y = j;
                        }
                    }
                }
            }
        }
        selected[y] = one;
        cout << "\n" << x << " --> " << y;
        cost += graph[x][y];
        ne++;
    }
    cout << "\nTotal cost is: " << cost << endl;
}

int main() {
    prims MST;
    cout << "\nPrims Algorithm to connect several offices\n";
    MST.createGraph();
    MST.primsAlgo();
}
```



Program 8:

```
#include <bits/stdc++.h>

using namespace std;
int sum(int frequency[], int i, int j)
{
    int sum = 0;
    for (int x = i; x <= j; x++)
        sum += frequency[x];
    return sum;
}
int optimalCost(int frequency[], int i, int j)
{
    if (j < i)
        return 0;
    if (j == i)
        return frequency[i];

    int frequencySum = sum(frequency, i, j);

    int min = INT_MAX;

    for (int r = i; r <= j; ++r)
    {
        int cost = optimalCost(frequency, i, r - 1) + optimalCost(frequency, r + 1, j);
        if (cost < min)
            min = cost;
    }

    return min + frequencySum;
}
int optimalSearchTree(int keys[], int frequency[], int n)
{
    return optimalCost(frequency, 0, n - 1);
}
int main()
{
    int keys[] = {10, 12, 20};
    int frequency[] = {34, 8, 50};
    int n = sizeof(keys) / sizeof(keys[0]);
    cout << "Cost of Optimal BST is " << optimalSearchTree(keys, frequency, n);
    return 0;
}
```


student@student-OptiPlex-3010: ~/Documents/anamikadsl
12:15 PM
(base) student@student-OptiPlex-3010:~$ cd /home/student/Documents/anamikadsl
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$ g++ assign8.cpp
(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$ ./a.out
Cost of Optimal BST is 142(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$

Program 9:

```
#include"iostream"

#include<string.h>
using namespace std;

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;
```

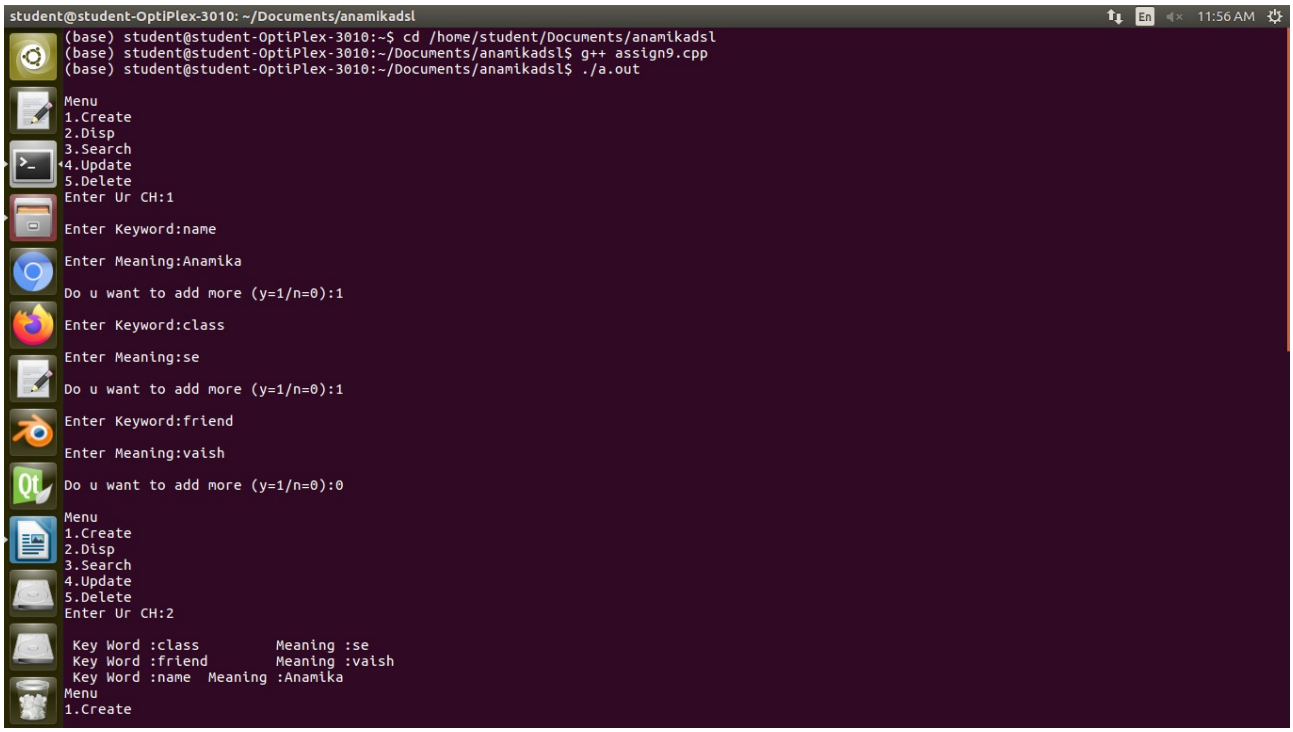
```
  }
  else
  {
   insert(root, temp);
  }
  cout<<"\nDo u want to add more (y=1/n=0):";
  cin>>ch;
 }
 while(ch == 1);

}

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;

}
```



```
student@student-OptiPlex-3010: ~/Documents/anamikadsl
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:2

 Key Word :class         Meaning :se
 Key Word :friend        Meaning :vaish
 Key Word :name  Meaning :Anamika
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:3

Enter Keyword which u want to search:friend

No of Comparisons:2
Keyword Found
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:4

Enter Keyword which meaning  want to update:name

Enter New Meaning ofKeywordnameutkarsh

Meaning Updated
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:5
```

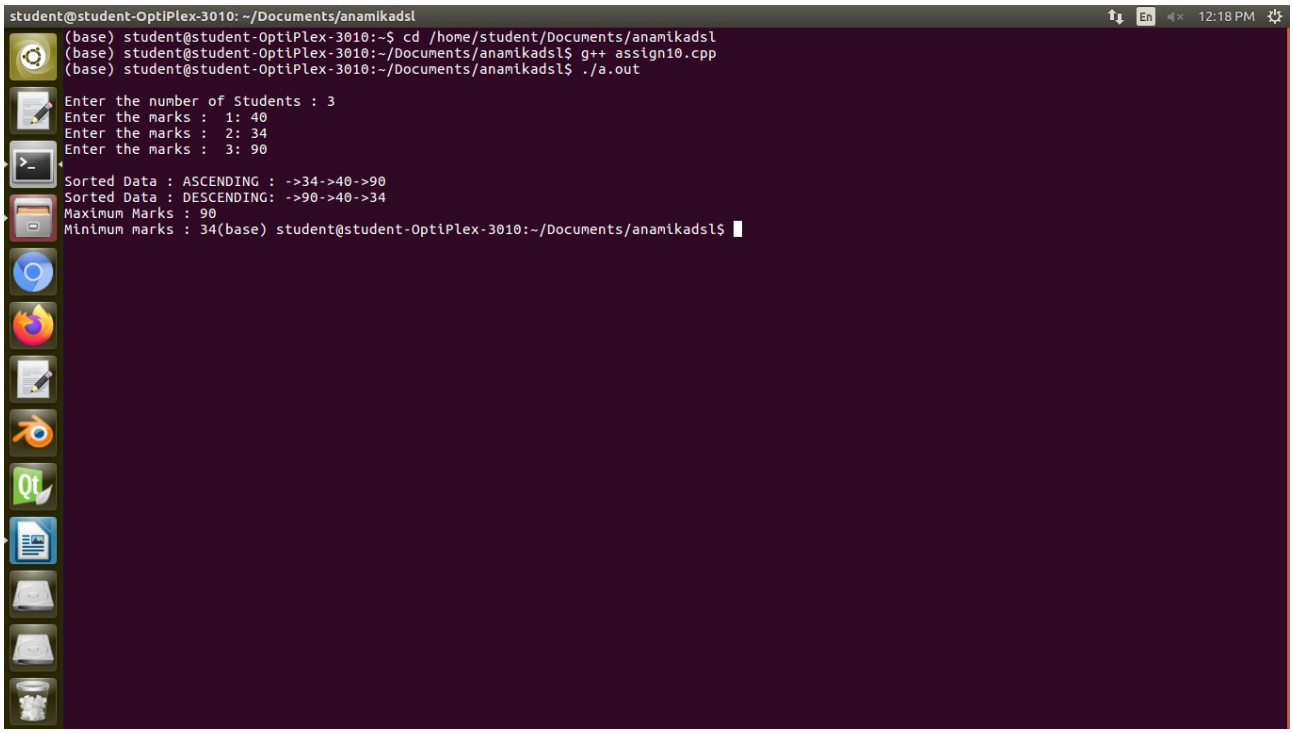
```
student@student-OptiPlex-3010: ~/Documents/anamikadsl
No of Comparisons:2
Keyword Found
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:4

Enter Keyword which meaning  want to update:name

Enter New Meaning ofKeywordnameutkarsh

Meaning Updated
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:5

Enter Keyword which u want to delete:name

Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:2

 Key Word :class         Meaning :se
 Key Word :friend        Meaning :vaish
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:
```

Program 10:

```
#include <iostream>

using namespace std;
void MaxHeapify(int a[], int i, int n)
{
 int j, temp;
 temp = a[i];
 j = 2*i;

  while (j <= n)
 {
  if (j < n && a[j+1] > a[j])
  j = j+1;

  if (temp > a[j])
   break;
  else if (temp <= a[j])
  {
   a[j/2] = a[j];
   j = 2*j;
  }
 }
 a[j/2] = temp;
 return;
}
void MinHeapify(int a[], int i, int n)
{
 int j, temp;
 temp = a[i];
 j = 2*i;

  while (j <= n)
 {
  if (j < n && a[j+1] < a[j])
  j = j+1;
  if (temp < a[j])
   break;
  else if (temp >= a[j])
  {
   a[j/2] = a[j];
   j = 2*j;
  }
 }
 a[j/2] = temp;
 return;
}
```

```
void MaxHeapSort(int a[], int n)
{
 int i, temp;
 for (i = n; i >= 2; i--)
 {
 temp = a[i];
  a[i] = a[1];
  a[1] = temp;
  MaxHeapify(a, 1, i - 1);
 }
}
void MinHeapSort(int a[], int n)
{
 int i, temp;
 for (i = n; i >= 2; i--)
 {
  temp = a[i];
  a[i] = a[1];
  a[1] = temp;
  MinHeapify(a, 1, i - 1);
 }
}
void Build_MaxHeap(int a[], int n)
{
 int i;
 for(i = n/2; i >= 1; i--)
  MaxHeapify(a, i, n);
}
void Build_MinHeap(int a[], int n)
{
 int i;
 for(i = n/2; i >= 1; i--)
  MinHeapify(a, i, n);
}
int main()
{
 int n, i;
 cout<<"\nEnter the number of Students : ";
 cin>>n;
 n++;
 int arr[n];
 for(i = 1; i < n; i++)
 {
  cout<<"Enter the marks :  "<<i<<": ";
  cin>>arr[i];
 }
 Build_MaxHeap(arr, n-1);
 MaxHeapSort(arr, n-1);
```

```
int max,min;
cout<<"\nSorted Data : ASCENDING : ";
for (i = 1; i < n; i++)
 cout<<"->"<<arr[i];
min=arr[1];
Build_MinHeap(arr, n-1);
  MinHeapSort(arr, n-1);
  cout<<"\nSorted Data : DESCENDING: ";
max=arr[1];
  for (i = 1; i < n; i++)
    cout<<"->"<<arr[i];
  cout<<"\nMaximum Marks : "<<max<<"\nMinimum marks : "<<min;
return 0;
}
```



Program 11:

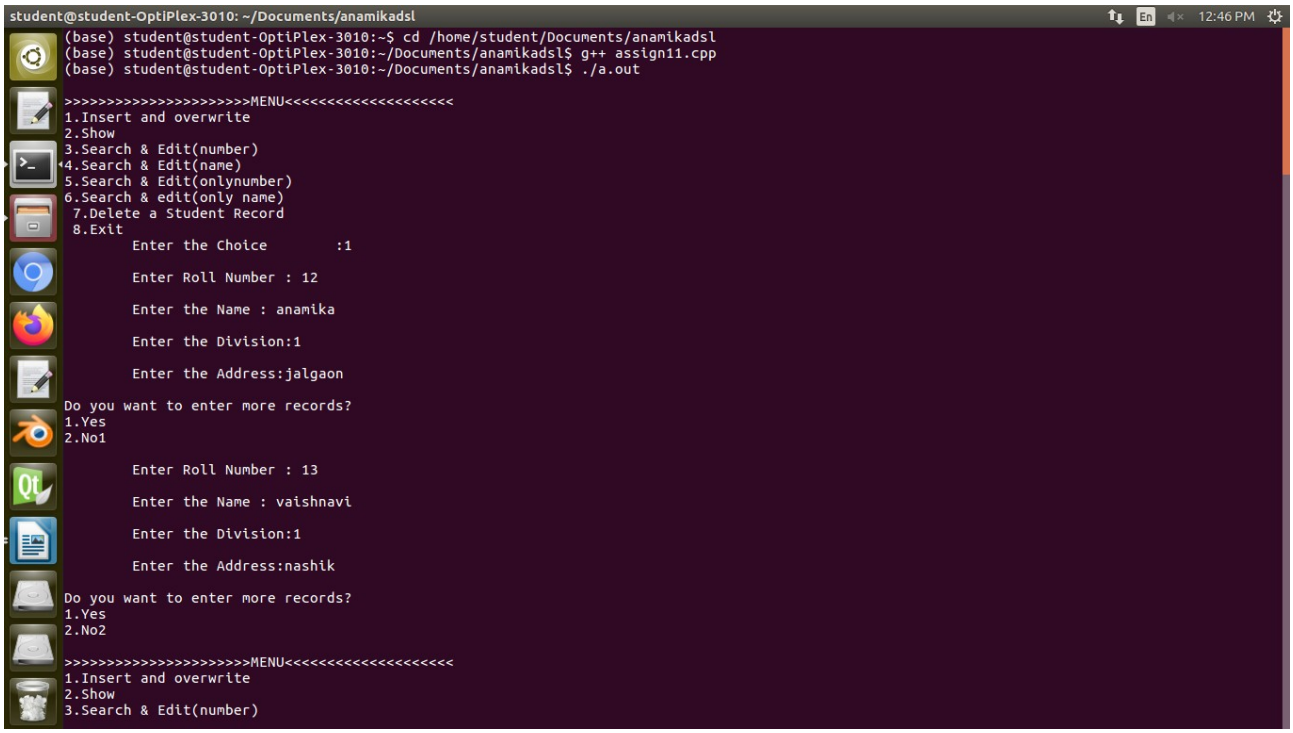
```
#include<iostream>

#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);
}
```



```
student@student-OptiPlex-3010: ~/Documents/anamikadsl                                12:46 PM
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
        Enter the Choice        :2

        12              anamika         1               jalgaon
        13              vaishnavi               1               nashik
>>>>>>>>>>>>>>>>>>>>>>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 find12

Record found
        Enter Roll Number : 17

        Enter the Name : anamika

        Enter the Division:1

        Enter the Address:jalgaon

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

        17              anamika         1               jalgaon
        13              vaishnavi               1               nashik
>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
```

```
student@student-OptiPlex-3010: ~/Documents/anamikadsl                                12:47 PM
        17              anamika         1               jalgaon
        13              vaishnavi               1               nashik
>>>>>>>>>>>>>>>>>>>>>>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 find13

Record found
        Enter Roll Number : 13

        Enter the Name : utkarsh

        Enter the Division:3

        Enter the Address:muzaffarpur

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

        17              anamika         1               jalgaon
        13              utkarsh         3               muzaffarpur
>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
```

```
Terminal  File  Edit  View  Search  Terminal  Help                                   12:47 PM
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 modify17

        Number found
        Enter the Roll No. to modify : 12

>>>>>>>>>>>>>>>>>>>>>>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 editutkarsh

Name found
        Enter the name to modify : manish

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

        12              anamika         1               jalgaon
        13              manish          3               muzaffarpur
>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<
```

```
student@student-OptiPlex-3010: ~/Documents/anamikadsl                                  12:48 PM
 7.Delete a Student Record
 8.Exit
        Enter the Choice        :2

        12              anamika         1               jalgaon
        13              manish          3               muzaffarpur
>>>>>>>>>>>>>>>>>>>>>>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: 13
The record with the roll no. 13 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

        12              anamika         1               jalgaon
>>>>>>>>>>>>>>>>>>>>>>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(base) student@student-OptiPlex-3010:~/Documents/anamikadsl$
```

Program 12:

```
#include <bits/stdc++.h>

#define max 20
using namespace std;
// Structure of Employee
struct employee {
    string name;
    long int code;
    string designation;
    int exp;
    int age;
};

int num;
void showMenu();

// Array of Employees to store the
// data in the form of the Structure
// of the Array
employee emp[max], tempemp[max],
    sortemp[max], sortemp1[max];

// Function to build the given datatype
void build()
{
    cout << "Build The Table\n";
    cout << "Maximum Entries can be "
         << max << "\n";

    cout << "Enter the number of "
         << "Entries required";
    cin >> num;

    if (num > 20) {
        cout << "Maximum number of "
             << "Entries are 20\n";
        num = 20;
    }
    cout << "Enter the following data:\n";

    for (int i = 0; i < num; i++) {
        cout << "Name ";
        cin >> emp[i].name;

        cout << "Employee ID ";
        cin >> emp[i].code;
```

```
        cout << "Designation ";
        cin >> emp[i].designation;

        cout << "Experience ";
        cin >> emp[i].exp;

        cout << "Age ";
        cin >> emp[i].age;
    }

    showMenu();
}

// Function to insert the data into
// given data type
void insert()
{
    if (num < max) {
        int i = num;
        num++;

        cout << "Enter the information "
             << "of the Employee\n";
        cout << "Name ";
        cin >> emp[i].name;

        cout << "Employee ID ";
        cin >> emp[i].code;

        cout << "Designation ";
        cin >> emp[i].designation;

        cout << "Experience ";
        cin >> emp[i].exp;

        cout << "Age ";
        cin >> emp[i].age;
    }
    else {
        cout << "Employee Table Full\n";
    }

    showMenu();
}

// Function to delete record at index i
void deleteIndex(int i)
```

```
{
    for (int j = i; j < num - 1; j++) {
        emp[j].name = emp[j + 1].name;
        emp[j].code = emp[j + 1].code;
        emp[j].designation
            = emp[j + 1].designation;
        emp[j].exp = emp[j + 1].exp;
        emp[j].age = emp[j + 1].age;
    }
    return;
}

// Function to delete record
void deleteRecord()
{
    cout << "Enter the Employee ID "
         << "to Delete Record";

    int code;

    cin >> code;
    for (int i = 0; i < num; i++) {
        if (emp[i].code == code) {
            deleteIndex(i);
            num--;
            break;
        }
    }
    showMenu();
}

void searchRecord()
{
    cout << "Enter the Employee"
         << " ID to Search Record";

    int code;
    cin >> code;

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

        // If the data is found
        if (emp[i].code == code) {
            cout << "Name "
                 << emp[i].name << "\n";

            cout << "Employee ID "
                 << emp[i].code << "\n";
```

```
            cout << "Designation "
                << emp[i].designation << "\n";

            cout << "Experience "
                << emp[i].exp << "\n";

            cout << "Age "
                << emp[i].age << "\n";
            break;
        }
    }

    showMenu();
}

// Function to show menu
void showMenu()
{

    cout << "-------------------------"
         << "GeeksforGeeks Employee"
         << " Management System"
         << "-------------------------\n\n";

    cout << "Available Options:\n\n";
    cout << "Build Table         (1)\n";
    cout << "Insert New Entry    (2)\n";
    cout << "Delete Entry        (3)\n";
    cout << "Search a Record     (4)\n";
    cout << "Exit                (5)\n";

    int option;

    // Input Options
    cin >> option;

    // Call function on the bases of the
    // above option
    if (option == 1) {
        build();
    }
    else if (option == 2) {
        insert();
    }
    else if (option == 3) {
        deleteRecord();
    }
```

```
    else if (option == 4) {
        searchRecord();
    }
    else if (option == 5) {
        return;
    }
    else {
        cout << "Expected Options"
            << " are 1/2/3/4/5";
        showMenu();
    }
}
// Driver Code
int main()
{
    showMenu();
    return 0;
}
```