Vilnius University
Kaunas Faculty

Institute of Social Sciences and Applied Informatics

Task No UzdII-15

Algorithm Theory and Data Structures

Assignment of:	*ISCsen8, Aniket Chauhan*
Assessed by:	Dilijonas D., dr. lekt.

Kaunas 2018

Contents

Task

There is a list of specialties, each with a separate list of students who graduated from this specialty (*sorted alphabetically*). There is also a list of companies that want employ young specialists, each of which has an additional list of preferred specialties and quantity of required specialists.

The system should provide the functionality of:

1. To add new specialty, as well as the list of its graduate students;
1. Print a list of the students of specified specialty;
1. Print the list of specialties;
1. To add a new company, as well as the list of specialties indicated by the company;
1. Print the list of companies;
1. Print the list of specialties of the specified firm;
1. Provide the specified firm by specialists upon request, adjusting the composition of the list of specialties and the list of companies accordingly (*move specialists from specialty list to company list*).

My approach to this was to implement a tree for the companies with struct connected to them (company requirements).
For the Specialties I have a different linked list and you can only use 7 specialties .Then there is sorted lists for all of them which come if they are called for a function.
the specialities required are entered in this way IT,MANAGMENT,Bussiness,etc.

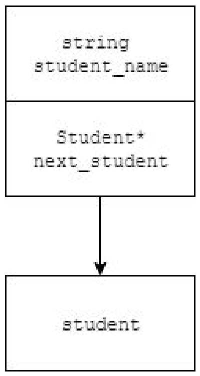
Description of Program Code

Class Description:

Class description C++ language

```
class Student{
public:
    string
student_name;

Student*next_stud
ent;

};
```

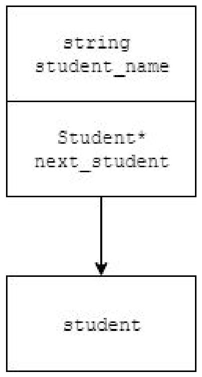
Graphical illustration of the class:

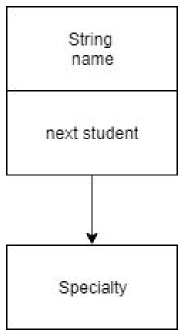


Class description C++ language

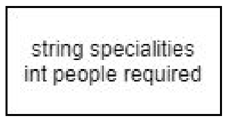
```
class Student{
public:
    string
student_name;

Student*next_stud
ent;

};
class Specialty{
public:
    string name;
    Specialty*
nextstudent;

}
```

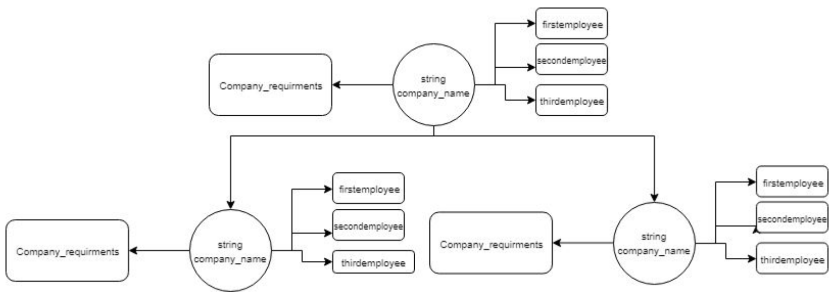
Graphical illustration of the class:





```
class Company_requirments{

public:

    string specialities;

    int people_required;

}

struct Company{

    string company_name;

                Company_requirments*
requirments;

    Student* employees;
```





```
Company* right_company;

Company* left_company;

Student* firstemployee;

Student* secondemployee;

Student* thridemployee;}
```

Description of the constants and created variables:

Name	Purpose	Data type
proot	It is the first element of the tree	Data structures (Specialty*)
pfirst	It is used to the speciality linked list	Data structure (Specialty*)
plast	It is used to link the nodes of the linked list	Data structures(Specialty*)
pfirst1	It is is used to make the sorted list.	Data structure (Student*)
pfirst2	It is is used to make the sorted list.	Data structure (Student*)
pfirst3	It is is used to make the sorted list.	Data structure (Student*)
pfirst4	It is is used to make the sorted list.	Data structure (Student*)

pfirst5	It is is used to make the sorted list.	Data structure (Student*)
pfirst6	It is is used to make the sorted list.	Data structure (Student*)
pfirst7	It is is used to make the sorted list.	Data structure (Student*)
Tree	It is used to create a tree for the companies and requirements to be stored.	Data structure (tree*)

Description of the variables used in the program:

Name of variable	Purpose of variable	Data type	Dependencies
{The variable name is given}	*{Describes the purpose of the variable}*	*{Specifies the variable data type}*	*{Specifies which subroutines, the function, or the class, the structure of the variable belongs to}*
key	It is used to input the student name or speciality or to find the elements in the functions	String	sortedstudentlist1,sortedlist2,sortedlist3,sortedlist4,sortedlist5,sortedlist6,sortedlist7,insertspecialities,find
pfound	Is the element or the node found to insert values into them etc.	Company*	displayinfoaboutparticular,insertemployee,
pcurrent	Used to move to the next or find a certain node	Speciality*,Student*,Company*	sortedstudentlist1,sortedlist2,sortedlist3,sortedlist4,sortedlist5,sortedlist6,sortedlist7,insertspecialities,find,tree, etc.
globname	Used to enter information about the student that you want to add to the list	string	inputspecialitiesstudents();
globspec	used to insert the name of the speciality int he linked list	string	inputspecialitiesstudents()

globmodel	used to insert the value in the enqueue function /to insert the value in the linked list.(the value of car_model)	string	input();
			main();

companyname;	is used to insert the name of the company in to the tree	string	main();
			insertcompany();

specialitiesneeded;	Used to insert the specialties needed to the company into the list	string	main();
			insertcompany();

employeesreq;	Used to get the number of employees in the tree	int	main();
			insertcompany();

newnode	Is used to to make the linked list it is a MaxIterator and to enter the value of the variables and point to the next node	node	Enque();
yesorno='y';	used to loop the menu again anf again	char	main();
input1	used to enter the student name and used to search for the student	string	search1,2,3,4,5,6,7; insertemployee
input2	used to enter the name of the company that you want to add the student into	string	insertemployee;
input	to get a input to restart the stuff needed	int	main();inputaction
y	specifies the specialities you want to add	int	main();
x	specifies the number of companies you want to add to the list	int	main();

Description of subroutines and functions:

Purpose	Name and (function name to call it)	Variables of function	Returned variables	Notes
To make a sorted list of students name	sortedstudentlist1, sortedlist1,2,3,4,5,6,7	pNewlink pPrevious pCurrent	void pfirst(specificnumber)=pNewlink; (sorted lists)	used to make the sorted linked lists
To display all the sorted linked list	displaylist1,2,3,4,5,6,7	pCurrent	printed list void	used to print all the elemts out of that list
Displaying the items of the queue	display	temp	void	used to display elements of the queue
Getting input form the user	input	globmodel.globweight, globplate	globmodel.globweight,globplate	Used to enter the variables in the queue
Used to do certain operations by getting a input from the user	menu	inputnum	void	Used to do certain operations by getting a input from the user

to find the company	Company* find	pRoot	pCurrent	used to find the company

to insert a company	insertcompany	pNewnode,
		pNewReq
			proot=pNewnode
			rightcompnay=new node;
			leftcompany-=new node
				to insert a company into the tree

to display the tree	displaytree	stack<Company*>glob alstack stack<Company*>local stack	void	prints the tree
Destroy a node	destroy	destroyrec(proot)	void	deletes a node
Insert a employee to the tree node	insertemployee	pCurrent pfound	pcurrent	inserts a emloyee to the selcted company
input the specialities and students	inputspecialitiesa ndstudents	x,pfirst1,2,3,4,5,6,7	void	inserts students in to a sorted linked list and specailities into a normal linked list.
search function to find if there are any students	search(1,2,3,4,5,6, 7)	pcurrent,pfirst1,2,3,4,5, 6,7	void	tells if the key exists or not
To find a speciality and print its students	findkey	key,pcurrent,	void	finds the speciality and prints information about it.
performs action according to input	inputaction	input	void	does the function according to the input
Delete functions	destroyrec	pLocalRoot	delete pLocalRoot	delete the funcion
Used to execute all the functions	main	num	void	Used to execute all the certain functions specified in the main program

Menu structure description:

The menu in this occurs after the user inputs certain values .First the user has to enter two many specialities he wants to add (max 7)and then he is to add appropriate information for the specialities.Then enter how many companies he wants to add and then add information about the companies .Then the menu gives you 8 options

1)This option allows you to add speciality and add students (Sorted) list as will for the speciality(cant be done if there are 7 facilities or more)

2)This option allows you to print a list of students for the specified speciality

3)This option allows the print all the specialities.

4)This option allows to add a new company as well as information for that company

5)this option allows you to print the list of all the companies.

6)This allows you to print the list of requirments of the specified company.

7)This allows you to add a specialist to a company specified by the user.

8)close the program

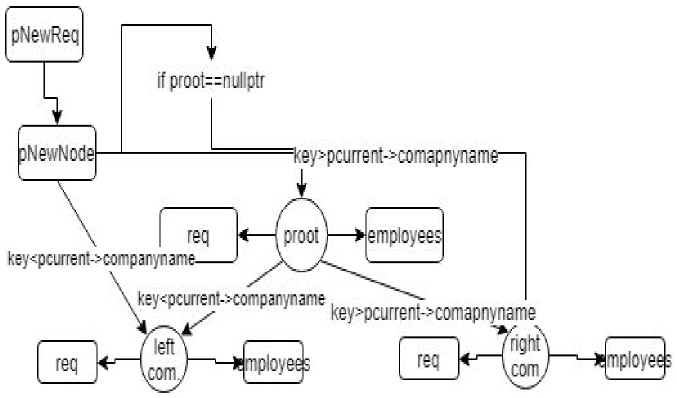
Description of main algorithms for data structure processing:

The following steps illustrate the operation of the insertcompanies(insert) procedure

Algorithm code in C++ language

```
void insertcompany(string
name,string specialities,int
employess){
        Company* pNewNode=new
Company(name);
        Company_requirments*
pNewReq=new
Company_requirments(specialities,em
ployess);
        pNewNode->requirments=
pNewReq;
        if(pRoot==nullptr)
            pRoot=pNewNode;
        else{
            Company* pCurrent=
pRoot;
            Company*pParent;
            while(true){
                pParent=pCurrent;

if(name<pCurrent->company_name){

pCurrent=pCurrent->left_company;
                    if(pCurrent==
nullptr){

pParent->left_company=pNewNode;
                        return;
                    }
                }
```

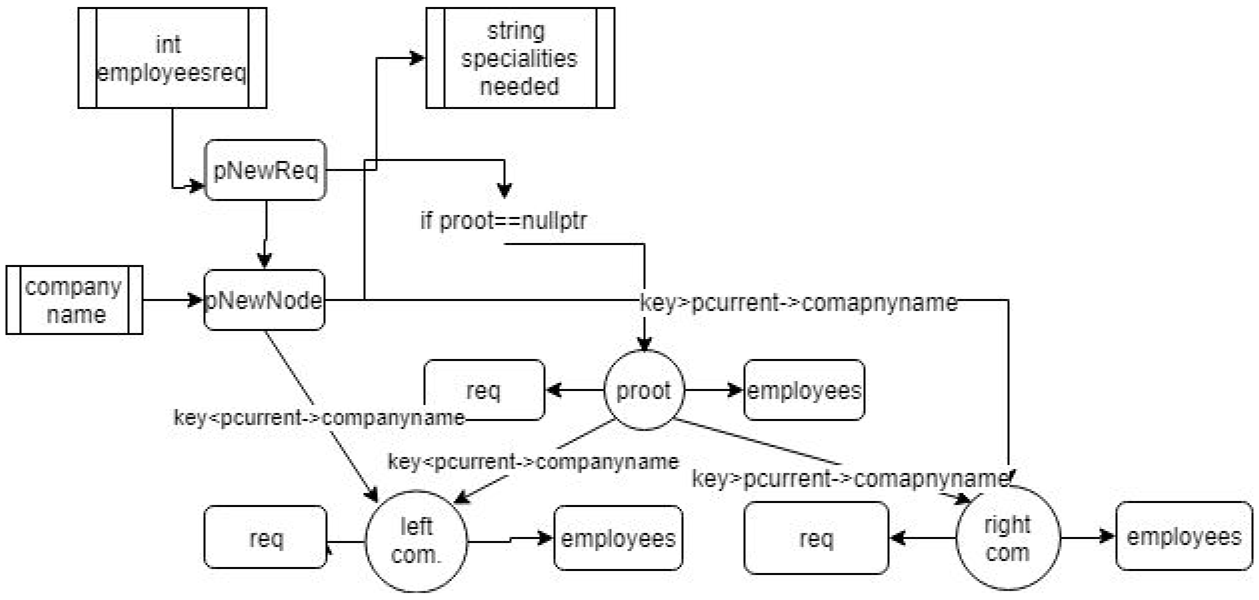
Graphical illustration of algorithm:


pNewReq
if proot==nullptr
pNewNode
key>pcurrent->comapnyname
req
proot
employees
key<pcurrent->companyname
key<pcurrent->companyname
key>pcurrent->comapnyname
req
left com.
employees
req
right com
employees

```
                else
                {

pCurrent=pCurrent->right_company;
                    if(pCurrent==
nullptr){

pParent->right_company=pNewNode;
                        return;
                    }
                }
            }
        }
    }
```

{Provide explanations of schema and algorithm.}; ***NEXT PAGE***

Block diagram illustrating theinput data into tree:


int employeesreq
string specialities needed
pNewReq
if proot==nullptr
company name
pNewNode
key>pcurrent->comapnyname
req
proot
employees
key<pcurrent->companyname
key<pcurrent->companyname
key>pcurrent->comapnyname
req
left com.
employees
req
right com
employees

Conclusions

1. {Present the conclusions and remarks on the work done – what is done fully, what was the easiest part and what most complex to implement? Provide comments on the general characteristics of the realization of data structures.};
2. I implemented a linked list which stores the speciality names as a linked list and then there are 7 sortedlists whcih each are attached(not linked) to a speciality so functions and i have a tree to insert company names which also has a attached node which stroes the requirments of speciality or the number of students.
3. The easiest one to implement was the linked list related stuff cause we already have done that long back so it was kind of very easy to do it.
4. The tree functions were very hard to implement as it was my first time implementing trees but because of the notes provided it was kind of easy and it was where i cleared the basics .
5. The enque function provides void output and is used to enque values.Deque function is used to deque values from the queue and the display function is used to display all the items in the list.
6. The sortedlists fucntion si used to nput sorted lists for students and they all have display functions for them as well.then the class tree has all the essentiail fuctions like inserting,displaying,finding,insertingemployees ,etc.then the input in the menu is controlled by inputactions which intakes the input and performs what is asked

7. The code works if all the inputs are done correctly like string values are added without spaces ,int is added for employees required,int is added for ow many companies or students are supposed to be added etc.
8. ALL FUNCTIONS WORK but 7 just links its doesnt display cause it wasnt specified.(limits to 3 employess)
9. For some reason i couldnt link them in non linear linked lists so i improvised my way through it and came with this solution i hope its okay.

Literature

1. *http://www.cplusplus.com*
2. *https://stackoverflow.com*
3. *www.codementor.io*
4. *www.tutorialspoint.com*

Annexes

```
#include <iostream>

#include <string>

#include <stack>

using namespace std;




class Student{

public:

   string student_name;

   Student* next_student;

public:

   Student(string studentname):

         student_name(studentname),next_student(nullptr)

   {}

   void DisplayListElement()

   {

      cout<<"["<<student_name<<"];";

   }

};
```

```
class Specialty{
public:
    string name;
    Specialty* nextstudent;

public:
    Specialty(string specname):
            name(specname),nextstudent(nullptr)
    {}
    void DisplayListElemnt(){
        cout<<"Speciality Name: ["<<name<<"];";
    }

};
Specialty* pfirst;
Specialty* plast;
class Company_requirments{
public:
    string specialities;
    int people_required;

public: Company_requirments(string specalitie,int required):
            specialities(specalitie),people_required(required)
    {}
    void DisplayListElement(){
        cout<<" ["<<specialities<<"] ["<<people_required<<"];";
    }

};
```

```
struct Company{
    string company_name;
    Company_requirments* requirments;
    Student* employees;
    Company* right_company;
    Company* left_company;
    Student* firstemployee;
    Student* secondemployee;
    Student* thridemployee;
    Company(string name):

company_name(name),right_company(nullptr),left_company(nullptr),employees(nullptr),requirments(nullptr),firstemployee(
            nullptr),secondemployee(nullptr),thridemployee(nullptr)
            {}


    Company(){
        cout<<"Name- "<<company_name<<"  ";
    }
    void displayNode(){
        cout<<"{ "<< company_name<<" } ";
    }


};
Student* pfirst1= nullptr;
Student* pfirst2= nullptr;
Student* pfirst3= nullptr;
Student* pfirst4= nullptr;
Student* pfirst5= nullptr;
Student* pfirst6= nullptr;
Student* pfirst7= nullptr;
```

```
void sortedstudentlist1(string key){

    Student* pNewLink = new Student(key); //make new link

    Student* pPrevious = NULL; //start at first

    Student* pCurrent = pfirst1;

//until end of list,

    while (pCurrent != NULL && key > pCurrent->student_name)

    { //or key > current,

        pPrevious = pCurrent;

        pCurrent = pCurrent->next_student; //go to next item

    }

    if (pPrevious == NULL) //at beginning of list

        pfirst1 = pNewLink; //first --> newLink

    else //not at beginning

        pPrevious->next_student = pNewLink; //old prev -->

    pNewLink->next_student=pCurrent;


}

void displaylist1(){

    cout << "List(first-->last) : ";

    Student* pCurrent = pfirst1; //start at beginning of list

    while (pCurrent != NULL) //until end of list,

    {

        pCurrent->DisplayListElement(); //print data

        pCurrent = pCurrent->next_student; //move to next link

    }

    cout << endl;

}

void sortedlist2(string key) {


    Student *pNewLink = new Student(key); //make new link
    Student *pPrevious = NULL; //start at first

    Student *pCurrent = pfirst2;

//until end of list,

    while (pCurrent != NULL && key > pCurrent->student_name) { //or key > current,

        pPrevious = pCurrent;

        pCurrent = pCurrent->next_student; //go to next item

    }

    if (pPrevious == NULL) //at beginning of list

        pfirst2 = pNewLink; //first --> newLink

    else //not at beginning

        pPrevious->next_student = pNewLink; //old prev -->

    pNewLink->next_student = pCurrent;

}

void displaylist2(){

    cout << "List(first-->last) : ";

    Student* pCurrent = pfirst2; //start at beginning of list

    while (pCurrent != NULL) //until end of list,

    {

        pCurrent->DisplayListElement(); //print data

        pCurrent = pCurrent->next_student; //move to next link

    }

    cout << endl;

}

void sortedlist3(string key){

    Student *pNewLink = new Student(key);

    Student *pPrevious = NULL;

    Student *pCurrent = pfirst3;


    while (pCurrent != NULL && key > pCurrent->student_name) {

        pPrevious = pCurrent;
        pCurrent = pCurrent->next_student;
    }
    if (pPrevious == NULL)
        pfirst3 = pNewLink;
    else
        pPrevious->next_student = pNewLink;
    pNewLink->next_student = pCurrent;
}
void displaylist3(){
    cout << "List(first-->last) : ";
    Student* pCurrent = pfirst3; //start at beginning of list
    while (pCurrent != NULL) //until end of list,
    {
        pCurrent->DisplayListElement(); //print data
        pCurrent = pCurrent->next_student; //move to next link
    }
    cout << endl;
}
void sortedlist4(string key){
    Student *pNewLink = new Student(key);
    Student *pPrevious = NULL;
    Student *pCurrent = pfirst4;

    while (pCurrent != NULL && key > pCurrent->student_name) {
        pPrevious = pCurrent;
        pCurrent = pCurrent->next_student;
    }
    if (pPrevious == NULL)
        pfirst4 = pNewLink;
    else
```

```
        pPrevious->next_student = pNewLink;
    pNewLink->next_student = pCurrent;
}
void displaylist4(){
    cout << "List(first-->last) : ";
    Student* pCurrent = pfirst4; //start at beginning of list
    while (pCurrent != NULL) //until end of list,
    {
        pCurrent->DisplayListElement(); //print data
        pCurrent = pCurrent->next_student; //move to next link
    }
    cout << endl;
}
void sortedlist5(string key){
    Student *pNewLink = new Student(key);
    Student *pPrevious = NULL;
    Student *pCurrent = pfirst5;

    while (pCurrent != NULL && key > pCurrent->student_name) {
        pPrevious = pCurrent;
        pCurrent = pCurrent->next_student;
    }
    if (pPrevious == NULL)
        pfirst5 = pNewLink;
    else
        pPrevious->next_student = pNewLink;
    pNewLink->next_student = pCurrent;
}
void displaylist5(){
    cout << "List(first-->last) : ";
    Student* pCurrent = pfirst5; //start at beginning of list

    while (pCurrent != NULL) //until end of list,

    {

        pCurrent->DisplayListElement(); //print data

        pCurrent = pCurrent->next_student; //move to next link

    }

    cout << endl;

}

void sortedlist6(string key){

    Student *pNewLink = new Student(key);

    Student *pPrevious = NULL;

    Student *pCurrent = pfirst6;


    while (pCurrent != NULL && key > pCurrent->student_name) {

        pPrevious = pCurrent;

        pCurrent = pCurrent->next_student;

    }

    if (pPrevious == NULL)

        pfirst6 = pNewLink;

    else

        pPrevious->next_student = pNewLink;

    pNewLink->next_student = pCurrent;

}

void displaylist6(){

    cout << "List(first-->last) : ";

    Student* pCurrent = pfirst6; //start at beginning of list

    while (pCurrent != NULL) //until end of list,

    {

        pCurrent->DisplayListElement(); //print data

        pCurrent = pCurrent->next_student; //move to next link
    }
    cout << endl;
}
void sortedlist7(string key){
    Student *pNewLink = new Student(key);
    Student *pPrevious = NULL;
    Student *pCurrent = pfirst7;

    while (pCurrent != NULL && key > pCurrent->student_name) {
        pPrevious = pCurrent;
        pCurrent = pCurrent->next_student;
    }
    if (pPrevious == NULL)
        pfirst7 = pNewLink;
    else
        pPrevious->next_student = pNewLink;
    pNewLink->next_student = pCurrent;
}
void displaylist7(){
    cout << "List(first-->last) : ";
    Student* pCurrent = pfirst7; //start at beginning of list
    while (pCurrent != NULL) //until end of list,
    {
        pCurrent->DisplayListElement(); //print data
        pCurrent = pCurrent->next_student; //move to next link
    }
    cout << endl;
}
void insertSpecialities(string key){
    Specialty* temp =new Specialty(key);
    if(pfirst== nullptr)
    {
        pfirst=temp;
        plast=temp;
    }
    else
    {
        plast->nextstudent=temp;
        plast=temp;
    }
}

void printspecialities(){
    Specialty* pCurrent=pfirst;
    while(pCurrent!=NULL){
        pCurrent->DisplayListElemnt();
        pCurrent=pCurrent->nextstudent;
    }


}
string globemployee="not found";
class tree{
private:
    Company* pRoot;
public:
    tree():
            pRoot(nullptr)
    {}
    Company* find(string key){
        Company* pCurrent=pRoot;
        while (pCurrent->company_name!=key)

        {

            if(key<pCurrent->company_name)

                pCurrent=pCurrent->left_company;

            else

                pCurrent=pCurrent->right_company;



            if(pCurrent==NULL){

                return NULL;

            };

        }

        if(pCurrent!=NULL){

            pCurrent->displayNode();

            pCurrent->requirments->DisplayListElement();

        }



        return pCurrent;

    }

    void insertcompany(string name,string specialities,int employess){

        Company* pNewNode=new Company(name);

        Company_requirments* pNewReq=new Company_requirments(specialities,employess);

        pNewNode->requirments= pNewReq;

        if(pRoot==nullptr)

            pRoot=pNewNode;

        else{

            Company* pCurrent= pRoot;

            Company*pParent;

            while(true){

                pParent=pCurrent;
                if(name<pCurrent->company_name){
                    pCurrent=pCurrent->left_company;
                    if(pCurrent== nullptr){
                        pParent->left_company=pNewNode;
                        return;
                    }
                }
                else
                {
                    pCurrent=pCurrent->right_company;
                    if(pCurrent== nullptr){
                        pParent->right_company=pNewNode;
                        return;
                    }
                }
            }
        }
    }

    void displaytree()
    {
        stack<Company*>globalStack;
        globalStack.push(pRoot);
        int nBlanks =32;
        bool isRowEmpty= false;
        cout<<"...";
        cout<<endl;
        while(isRowEmpty==false){
            stack<Company*> localStack;
            isRowEmpty=true;
        for(int j =0;j<nBlanks;j++)
            cout<<" ";
        while(globalStack.empty()== false){
            Company* temp=globalStack.top();
            globalStack.pop();
            if (temp!=NULL)
            {
                cout<<temp->company_name;
                temp->requirments->DisplayListElement();
                localStack.push(temp->left_company);
                localStack.push(temp->right_company);
                if(temp->left_company!=NULL||temp->right_company!=NULL)
                    isRowEmpty=false;
            }
            else {
                cout << "----------";
                localStack.push(NULL);
                localStack.push(NULL);
            }
            for (int j=0;j<nBlanks*2-2;j++)
                cout<<" ";
        }
        cout<<endl;
        nBlanks /=2;
        while (localStack.empty()== false){
            globalStack.push(localStack.top());
            localStack.pop();
        }
    }
    cout<<"----------------------------------------------";
```

```
        cout<<endl;


    }
    void destroy()
    {
        destroyrec(pRoot);
    }
    void destroyrec(Company* pLocalRoot){
        if(pLocalRoot!=NULL){
            destroyrec(pLocalRoot->left_company);
            destroyrec(pLocalRoot->right_company);
            delete pLocalRoot;
        }
    }
Company* insertemployee(string key1,string key2){
        Company* pcurrent=pRoot;
        Company *pfound=pcurrent;
    while(1) {
        if (pcurrent->company_name==key1){
            pfound= pcurrent;
            break;
        }
        if (key1 < pcurrent->company_name) {
            pcurrent = pcurrent->left_company;
        } else if (key1 > pcurrent->company_name) {
            pcurrent = pcurrent->right_company;
        }
        if (pcurrent == nullptr)
            return NULL;
    }
```

```
while(1) {
    Student* pNewnode= new Student(key2);
    if (pfound->firstemployee == nullptr) {
        pfound->firstemployee=pNewnode;
        break;
    }
    else if (pfound->secondemployee == nullptr){
        pfound->secondemployee = pNewnode;
        break;
    }
    else if (pfound->thridemployee == nullptr)
        pfound->thridemployee= pNewnode;
        break;
}}
Company* displayinfoaboutparticular(string key1){
    Company* pcurrent=pRoot;
    Company* pfound=pcurrent;
 while(1) {
     if (pcurrent->company_name==key1){
         pfound= pcurrent;
     break;
     }
       if (key1 < pcurrent->company_name) {
                pcurrent = pcurrent->left_company;
            } else if (key1 > pcurrent->company_name) {
                pcurrent = pcurrent->right_company;
            }
            if (pcurrent == nullptr)
                return NULL;
         }
        pfound->displayNode();

        pfound->requirments->DisplayListElement();

        return pcurrent;

    }

};

tree Tree;


void menu(){

    cout<<"What do you want to do:  "<<endl;

    cout<<"[1]To add new speciality as well as the list of its graduate students "<<endl;

    cout<<"[2]Print a list of the students of specified speciality"<<endl;

    cout<<"[3]Print the list of Specialities"<<endl;

    cout<<"[4]To add a new company,as well as the list of Specialities indicated by the company"<<endl;

    cout<<"[5]Print the list of Companies"<<endl;

    cout<<"[6]Print the list of Specialites of the Company"<<endl;

    cout<<"[7]Add a Specialist you want to add to a Company"<<endl;

    cout<<"[8]Close the program"<<endl;

}

string companyname;

string specialitiesneeded;

int employeesreq;

char yesorno='y';



string globname;

string globspec;


void inputspecialitesandstudents(){

    int x;

    if(pfirst1==nullptr) {
```

```
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedstudentlist1(globname);
    }
    insertSpecialities(globspec);
    return;
}
if(pfirst2==nullptr) {
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedlist2(globname);
    }
    insertSpecialities(globspec);
    return;
}
if(pfirst3==nullptr) {
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedlist3(globname);
    }
    insertSpecialities(globspec);
    return;
}
if(pfirst4==nullptr) {
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedlist4(globname);
    }
    insertSpecialities(globspec);
    return;
}
if(pfirst5==nullptr) {
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedlist5(globname);
    }
    insertSpecialities(globspec);
    return;
}
if(pfirst6==nullptr) {
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedlist6(globname);
    }
    insertSpecialities(globspec);
    return;
}
if(pfirst7==nullptr) {
    cout << "What is the name of the speciality you want to add" << endl;
    cin >> globspec;
    cout << "How many students do you want to add" << endl;
    cin >> x;
    for (int i = 0; i < x; ++i) {
        cout << "What is the name of the student? " << endl;
        cin >> globname;
        sortedlist7(globname);
    }
    insertSpecialities(globspec);
    return;
}
```

```
}


int check=0;

string null="removed";

Student* temp= new Student(null);


void search1(string key) {

    Student *pcurrent = pfirst1;


    while (pcurrent != nullptr) {

        if (pcurrent->student_name == key) {

            cout<<"found"<<endl;

            pcurrent=temp;

            check=1;

            return;

        }

        if (pcurrent->student_name != key){

            pcurrent = pcurrent->next_student;

    }}

}

    void search2(string key) {

      Student*  pcurrent = pfirst2;

        while (pcurrent != nullptr) {

            if (pcurrent->student_name == key) {

                cout<<"found"<<endl;

                pcurrent=temp;

                 check=1;

                return;

            }

            if (pcurrent->student_name != key)
            pcurrent = pcurrent->next_student;
        }

        }

    void search3(string key){
     Student* pcurrent = pfirst3;
    while (pcurrent != nullptr) {
        if (pcurrent->student_name == key) {
            cout<<"found"<<endl;
            pcurrent=temp;
             check=1;
            return;
        }
        if(pcurrent->student_name!=key){
            pcurrent=pcurrent->next_student;
        }
    }
}
void search4(string key) {
    Student *pcurrent = pfirst4;

    while (pcurrent != nullptr) {
        if (pcurrent->student_name == key) {
            cout<<"found"<<endl;
            pcurrent=temp;
            check=1;
            break;
        }
        if (pcurrent->student_name != key)
            pcurrent = pcurrent->next_student;
    }


}
void search5(string key) {
    Student *pcurrent = pfirst5;
    while (pcurrent != nullptr) {
        if (pcurrent->student_name == key) {
            cout<<"found"<<endl;
            pcurrent=temp;
            check=1;
            break;
        }
        if (pcurrent->student_name != key)
            pcurrent = pcurrent->next_student;
    }


}


void search6(string key) {
  Student*  pcurrent = pfirst6;
    while (pcurrent != nullptr) {
        if (pcurrent->student_name == key) {
            cout<<"found"<<endl;
            pcurrent=temp;
            check=1;
            break;
        }
        if (pcurrent->student_name != key)
```

```
            pcurrent = pcurrent->next_student;
    }


}
void search7(string key){
   Student*  pcurrent = pfirst7;
    while (pcurrent != nullptr) {
        if (pcurrent->student_name == key) {
            cout<<"found"<<endl;
            pcurrent=temp;
            check=1;
            break;
        }
        if (pcurrent->student_name != key)
            pcurrent = pcurrent->next_student;
    }
}
void findkey(string key){
    if(pfirst->name==key){
        displaylist1();
        return;
    }
    if(pfirst->nextstudent->name==key){
        displaylist2();
        return;
    }
    if(pfirst->nextstudent->nextstudent->name==key){
        displaylist3();
        return;
    }

    if(pfirst->nextstudent->nextstudent->nextstudent->name==key){

        displaylist4();

        return;

    }

    if(pfirst->nextstudent->nextstudent->nextstudent->nextstudent->name==key){

        displaylist5();

        return;

    }

    if(pfirst->nextstudent->nextstudent->nextstudent->nextstudent->nextstudent->name==key){

        displaylist6();

        return;

    }

    if(pfirst->nextstudent->nextstudent->nextstudent->nextstudent->nextstudent->nextstudent->name==key){

        displaylist7();

        return;

    }

    else{

        cout<<"not found"<<endl;

    }

}

void inputaction(int input)

{

    if (input == 4) {

        string companyname;

        string specialitiesneeded;

        int employeesreq;


        cout << "What is the name of Company you want to add?" << endl;

        cin >> companyname;
        cout << "What are the specialities the company is looking for? " << endl;

        cin >> specialitiesneeded;

        cout << "How many number of employees is the company looking for? " << endl;

        cin >> employeesreq;

        Tree.insertcompany(companyname, specialitiesneeded, employeesreq);

    }

    if (input == 6) {

        string key;

        cout << "Which Company do you want to print information about?" << endl;

        cin >> key;

        Tree.displayinfoaboutparticular(key);

    }

    if (input == 8) {

        exit(0);

    }

    if (input == 5) {

        Tree.displaytree();

    }

    if(input==1){

        inputspecialitesandstudents();

    }

    if(input==2){

        string key;

        cout<<"Enter the Speciality that you want print information about"<<endl;

        cin>>key;

        findkey(key);

    }

    if(input==3){

        printspecialities();

    }
```

```
    if(input==7){
        string input1;
        string input2;
    cout<<"Which student do you want to add to the company?"<<endl;
    cin>>input1;
    search1(input1);
       if(check==0)
       search2(input1);
        if(check==0)
       search3(input1);
        if(check==0)
        search4(input1);
        if(check==0)
            search5(input1);
        if(check==0)
        search6(input1);
        if(check==0);
        search7(input1);
        if(check==0)
        cout<<"not found";
        if(check==1) {
        cout << "Which company do you want to add it?" << endl;
        cin >> input2;
        Tree.insertemployee(input2, input1);
    }
    }
}



int main() {
```

```
int input;

int x;

int y;

cout<<"RULES: "<<endl;

cout<<"1. The program can only support 7 specialities"<<endl;

cout<<"2. You have to enter the Specialities required by the company in this format a speciality,speciality "<<endl;

cout<<"3. You can only add 3 students to the Company"<<endl;

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


cout<<endl;

cout<<"How many specialities do you want to add? "<<endl;

cin>>y;

if(y<=7) {

    for (int j = 0; j < y; ++j) {

        inputspecialitesandstudents();

    }

}

else if(y>=8){

    cout<<"Sorry wrong input"<<endl;

}

    cout << "How many Companies do you want add?" << endl;

    cin >> x;

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

        cout << "What is the name of Company you want to add?" << endl;

        cin >> companyname;

        cout << "What are the specialities the company is looking for? " << endl;

        cin >> specialitiesneeded;

        cout << "How many number of employees is the company looking for? " << endl;

        cin >> employeesreq;
```

```
            Tree.insertcompany(companyname, specialitiesneeded, employeesreq);

        }


    while (yesorno == 'y')

    {

        menu();

        cin >> input;

        inputaction(input);

        cout<<endl;

        cout << "Do you want to do anything else press [y] for yes and [n] for no"<<endl;

        cin >> yesorno;

    }

}
```