//

// Human.h

// Project

//

// Created by Andrej Hurynovič on 27.10.20.

//

#ifndef Human\_h

#define Human\_h

#include <iostream>

#include "iomanip"

#include "Settings.h"

#include "Date.h"

#include "ExceptionEnter.h"

**using** **namespace** std;

**class** Human {

**protected**:

string privateID;

string name;

Date birthday;

**public**:

Human(string privateID = "UNDEFINE", string name = "UNDEFINED", Date birthday = Date()){

**this**->privateID = privateID;

**this**->name = name;

**this**->birthday = birthday;

}

**void** writeToFile(File&);

**void** readFromFile(File&);

string getPrivateID();

string getName();

Date getBirthday();

**void** setPrivateID(string);

**void** setName(string);

**void** setBirthday(Date);

**void** editMenu(**bool**);

**void** edit(**int**);

**static** **void** printSpreadsheet();

**friend** ostream& **operator** << (ostream &stream, Human human);

**friend** istream& **operator** >> (istream &stream, Human &human);

~Human(){};

};

#endif /\* Human\_h \*/

//

// Human.cpp

// Project

//

// Created by Andrej Hurynovič on 12.12.20.

//

#include "Human.h"

**void** Human::writeToFile(File& file){

file.write(privateID);

file.write(name);

file.write(birthday);

}

**void** Human::readFromFile(File& file){

file.read(&privateID);

file.read(&name);

file.read(&birthday);

}

string Human::getPrivateID(){

**return** **this**->privateID;

}

string Human::getName(){

**return** **this**->name;

}

Date Human::getBirthday(){

**return** **this**->birthday;

}

**void** Human::setPrivateID(string privateID){

**this**->privateID = privateID;

}

**void** Human::setName(string name){

**this**->name = name;

}

**void** Human::setBirthday(Date birthday){

**this**->birthday = birthday;

}

**void** Human::editMenu(**bool** getValue){

**do**{

cout << endl <<

"1. Private ID." << endl <<

"2. Name. " << endl <<

"3. Birthday. " << endl;

**if**(getValue){

cout << "0. Exit." << endl;

**int** value = ExceptionEnter::NumberData<**int**>(humanEditMenu);

**if**(!value){

**return**;

}

edit(value);

}**else**{

**return**;

}

}**while**(**true**);

}

**void** Human::edit(**int** key){

cout << "Enter new value: " << endl;

**switch** (key) {

**case** 1:

privateID = ExceptionEnter::stringData(DataType::privateID);

**break**;

**case** 2:

name = ExceptionEnter::stringData(DataType::name);

**break**;

**case** 3:

cin >> birthday;

**break**;

}

}

**void** Human::printSpreadsheet(){

cout <<

'|' << setw(Settings::getLength(DataType::privateID)) << "ID" << '|' <<

'|' << setw(Settings::getLength(DataType::name)) << "Name" << '|' <<

'|' << setw(Settings::getLength(DataType::year) + Settings::getLength(DataType::month) + Settings::getLength(DataType::day) + 2) << "Date" << '|';

}

ostream& **operator** << (ostream &outputStream, **const** Human human){

cout <<

'|' << setw(Settings::getLength(DataType::privateID)) << human.privateID << '|' <<

'|' << setw(Settings::getLength(DataType::name)) << human.name << '|' <<

'|' << human.birthday << '|';

**return** outputStream;

}

istream& **operator** >> (istream &inputStream, Human &human){

cout << "Enter name: ";

human.name = ExceptionEnter::stringData(name);

cout << "Enter birthday: " << endl;

cin >> human.birthday;

**return** inputStream;

}

//

// Bachelor.h

// Project

//

// Created by Andrej Hurynovič on 27.10.20.

//

#ifndef Bachelor\_h

#define Bachelor\_h

#include <vector>

#include "Human.h"

#include "Mark.h"

#include "Benefit.h"

**class** Bachelor: **public** Human {

**protected**:

string group;

**float** scholarship;

**bool** showScholarship;

vector<Mark> marks;

vector<Benefit> benefits;

**public**:

Bachelor(string group = "EMPTY ", **float** scholarship = -1, **bool** showScholarship = **true**){

**this**->group = group;

**this**->scholarship = scholarship;

**this**->showScholarship = showScholarship;

};

**void** writeToFile(File&);

**void** readFromFile(File&);

**void** calculateScholarship(**float** base = Settings::getBachelorBase());

string getGroup();

**float** getScholarship();

**bool** getShowScholarship();

Mark\* getMarkForID(**int**);

**unsigned** **long** getMarksSize();

Benefit\* getBenefitForID(**int**);

**unsigned** **long** getBenefitsSize();

**void** setGroup(string);

**void** setScholarship(**float**);

**void** setShowScholarship(**bool**);

**void** addMark(Mark);

**void** addBenefit(Benefit);

**void** removeMarkForID(**int**);

**void** removeBenefitForID(**int**);

**void** editMenu(**bool**);

**void** edit(**int**);

**static** **void** printSpreadsheet(**bool**);

**void** printMarksAndBenefits();

**friend** ostream& **operator** << (ostream &stream, Bachelor bachelor);

**friend** istream& **operator** >> (istream &stream, Bachelor &bachelor);

~Bachelor(){

marks.clear();

benefits.clear();

};

};

#endif /\* Bachelor\_h \*/

//

// Bachelor.cpp

// Project

//

// Created by Andrej Hurynovič on 13.12.20.

//

#include "Bachelor.h"

#include "Algorithms.h"

**void** Bachelor::writeToFile(File& file){

Algorithms::writeBachelorToFile(\***this**, file);

}

**void** Bachelor::readFromFile(File& file){

Human::readFromFile(file);

file.read(&group);

file.read(&scholarship);

**unsigned** **long** marksSize;

file.read(&marksSize);

Mark mark;

**for**(**int** i = 0; i < marksSize; i++){

file.read(&mark);

marks.push\_back(mark);

}

**unsigned** **long** benefitsSize;

file.read(&benefitsSize);

Benefit benefit;

**for**(**int** i = 0; i < benefitsSize; i++){

file.read(&benefit);

benefits.push\_back(benefit);

}

}

**void** Bachelor::calculateScholarship(**float** base){

scholarship = Algorithms::calclulateBachelorScholarship(\***this**, base);

}

string Bachelor::getGroup(){

**return** group;

}

**float** Bachelor::getScholarship(){

**return** scholarship;

}

**bool** Bachelor::getShowScholarship(){

**return** showScholarship;

}

Mark\* Bachelor::getMarkForID(**int** ID){

**if**(**this**->marks.size() <= ID){

**return** **NULL**;

}

**return** &marks[ID];

}

**unsigned** **long** Bachelor::getMarksSize(){

**return** marks.size();

}

Benefit\* Bachelor::getBenefitForID(**int** ID){

**if**(**this**->benefits.size() <= ID){

**return** **NULL**;

}

**return** &benefits[ID];

}

**unsigned** **long** Bachelor::getBenefitsSize(){

**return** benefits.size();

}

**void** Bachelor::setGroup(string group){

**this**->group = group;

}

**void** Bachelor::setScholarship(**float** scholarship){

**this**->scholarship = scholarship;

}

**void** Bachelor::setShowScholarship(**bool** showScholarship){

**this**->showScholarship = showScholarship;

}

**void** Bachelor::addMark(Mark mark){

marks.push\_back(mark);

}

**void** Bachelor::addBenefit(Benefit benefit){

benefits.push\_back(benefit);

}

**void** Bachelor::removeMarkForID(**int** ID){

**if**(**this**->marks.size() <= ID){

cout << "No such ID." << endl;

**return**;

}**else**{

cout << "The value was successfully removed." << endl;

}

marks.erase(marks.begin()+ID);

}

**void** Bachelor::removeBenefitForID(**int** ID){

**if**(**this**->benefits.size() <= ID){

cout << "No such ID." << endl;

**return**;

}**else**{

cout << "The value was successfully removed." << endl;

}

benefits.erase(benefits.begin()+ID);

}

**void** Bachelor::editMenu(**bool** getValue){

**do**{

Human::editMenu(**false**);

cout <<

"4. Group." << endl <<

"5. Mark. " << endl <<

"6. Benefit. " << endl;

**if**(getValue){

cout << "0. Exit." << endl;

**int** value = ExceptionEnter::NumberData<**int**>(bachelorEditMenu);

**if**(!value){

**return**;

}

edit(value);

}**else**{

**return**;

}

}**while**(**true**);

}

**void** Bachelor::edit(**int** key){

**if**(key < 4){

Human::edit(key);

}**else**{

**switch** (key) {

**case** 4:

cout << "Enter new value: " << endl;

group = ExceptionEnter::stringData(DataType::group);

**break**;

**case** 5:{

**if**(marks.empty()){

cout << "No marks, enter new one." << endl;

Mark mark;

cin >> mark;

addMark(mark);

}**else**{

**do**{

**if**(marks.empty()){

cout << "All marks was deleted." << endl;

**return**;

}

Mark::printSpreadsheet();

cout << endl;

**for**(**int** i = 0; i < marks.size(); i++){

cout << '|' << marks[i] << endl;

}

cout << endl <<

"1. Add." << endl <<

"2. Remove." << endl <<

"0. Exit." << endl;

**switch** (ExceptionEnter::NumberData<**int**>(markMenu)) {

**case** 0:

**return**;

**case** 1:{

cout << "Enter new value: " << endl;

Mark mark;

cin >> mark;

addMark(mark);

**break**;

}

**case** 2:

**if**(marks.size() == 1){

cout << "For the calculation, the student must have at least one mark." << endl;

**continue**;

}

cout << "Choose number: " << endl;

removeMarkForID(ExceptionEnter::NumberData<**int**>(markID));

}

}**while**(**true**);

}

calculateScholarship();

**break**;

}

**case** 6:{

**if**(benefits.empty()){

cout << "No benefits, enter new one." << endl;

Benefit benefit;

cin >> benefit;

addBenefit(benefit);

}**else**{

**do**{

**if**(benefits.empty()){

cout << "All benefits was deleted." << endl;

**return**;

}

cout << '|' << setw(Settings::getLength(DataType::benefitType)) << "Benefit" << '|' << endl;

**for**(**int** i = 0; i < benefits.size(); i++){

cout << '|' << benefits[i] << '|' << endl;

}

cout << endl <<

"1. Add." << endl <<

"2. Remove." << endl <<

"0. Exit." << endl;

**switch** (ExceptionEnter::NumberData<**int**>(benefitMenu)) {

**case** 0:

**return**;

**case** 1:{

Benefit benefit;

**bool** error = **false**;

cin >> benefit;

**for**(**int** i = 0; i < benefits.size(); i++){

**if**(benefit == benefits[i]){

cout << "You can't add two identical benefits." << endl;

error = **true**;

}

}

**if**(!error){

addBenefit(benefit);

}

**break**;

}

**case** 2:

cout << "Choose number: " << endl;

removeBenefitForID(ExceptionEnter::NumberData<**int**>(benefitID));

}

}**while**(**true**);

}

}

}

}

}

**void** Bachelor::printMarksAndBenefits(){

**if**(marks.size() <= 1 && benefits.size() <= 1){

**return**;

}

**int** markNumber = 1;

**int** benefitNumber = 1;

**do**{

cout << setw(Settings::getLength(DataType::privateID) +

Settings::getLength(DataType::name) +

Settings::getLength(DataType::year) +

Settings::getLength(DataType::month) +

Settings::getLength(DataType::day) + 2 + 6 +

Settings::getLength(DataType::group) + 2) << ' ';

**if**(marks.size() > markNumber){

cout << '|' << marks[markNumber++];

}**else**{

cout << setw(Settings::getLength(DataType::subject) + Settings::getLength(DataType::mark) + Settings::getLength(DataType::resit) + 7);

}

**if**(benefits.size() > benefitNumber){

cout << '|' << benefits[benefitNumber++] << '|';

}

cout << endl;

}**while**(marks.size() > markNumber || benefits.size() > benefitNumber);

}

**void** Bachelor::printSpreadsheet(**bool** scholarship){

Human::printSpreadsheet();

cout <<

'|' << setw(Settings::getLength(DataType::group)) << "Group" << '|' <<

'|' << setw(Settings::getLength(DataType::subject)) << "Subject" << '|' <<

'|' << setw(Settings::getLength(DataType::mark)) << "Mark" << '|' <<

'|' << setw(Settings::getLength(DataType::resit)) << "Resit" << '|' <<

'|' << setw(Settings::getLength(DataType::benefitType)) << "Benefit" << '|';

**if**(scholarship){

cout << '|' << setw(Settings::getLength(DataType::scholarship)) << "Scholarship" << '|';

}

}

ostream& **operator** << (ostream &stream, Bachelor bachelor){

cout << \***dynamic\_cast**<Human\*>(&bachelor) <<

'|' << bachelor.group << '|' <<

'|' << bachelor.marks[0] << '|';

**if**(bachelor.benefits.size()){

cout << bachelor.benefits[0] << '|';

}**else**{

cout << setw (Settings::getLength(benefitType)) << ' ' << '|';

}

**if**(bachelor.showScholarship){

cout << '|' << setw(Settings::getLength(DataType::scholarship)) << bachelor.scholarship << '|';

}

**return** stream;

}

istream& **operator** >> (istream &stream, Bachelor &bachelor){

cin >> \***dynamic\_cast**<Human\*>(&bachelor);

cout << endl << "Enter group (" << Settings::getLength(DataType::group) << " digits): " << endl;

bachelor.group = ExceptionEnter::stringData(DataType::group);

cout << endl << "Enter mark: " << endl;

Mark mark;

cin >> mark;

bachelor.addMark(mark);

cout << endl << "Add one more? (0/1): ";

**while** (ExceptionEnter::NumberData<**int**>(boolean)) {

cin >> mark;

bachelor.addMark(mark);

cout << endl << "Add one more? (0/1): ";

}

cout << endl << "Does the student have benefits? (0/1): ";

**if**(ExceptionEnter::NumberData<**int**>(boolean)){

Benefit benefit;

cin >> benefit;

bachelor.addBenefit(benefit);

cout << endl << "Add one more? (0/1): ";

**while** (ExceptionEnter::NumberData<**int**>(boolean)) {

**bool** error = **false**;

cin >> benefit;

**for**(**int** i = 0; i < bachelor.benefits.size(); i++){

**if**(benefit == bachelor.benefits[i]){

cout << "You can't add two identical benefits." << endl << endl << "Add one more? (0/1): ";

error = **true**;

**break**;

}

}

**if**(error){

**continue**;

}

bachelor.addBenefit(benefit);

cout << endl << "Add one more? (0/1): ";

}

}

**return** stream;

}

//

// Master.h

// Project

//

// Created by Andrej Hurynovič on 10.11.20.

//

#ifndef Master\_h

#define Master\_h

#include "Bachelor.h"

#include "Publication.h"

**class** Master: **public** Bachelor {

**protected**:

Publication\* publication;

**public**:

Master(Publication\* publication = **NULL**){

**this**->publication = publication;

};

Master(**const** Master& master){

**this**->privateID = master.privateID;

**this**->name = master.name;

**this**->birthday = master.birthday;

**this**->group = master.group;

**this**->scholarship = master.scholarship;

**this**->showScholarship = master.showScholarship;

**this**->marks = master.marks;

**this**->benefits = master.benefits;

publication = **new** Publication (\*master.publication);

}

**void** writeToFile(File&);

**void** readFromFile(File&);

**void** calculateScholarship(**float** base = Settings::getMasterBase());

Publication\* getPublication();

**void** setPublication(Publication\*);

**void** editMenu(**bool**);

**void** edit(**int** key);

**static** **void** printSpreadsheet(**bool**);

**friend** ostream& **operator** << (ostream &stream, Master master);

**friend** istream& **operator** >> (istream &stream, Master &master);

~Master(){

// delete publication;

};

};

#endif /\* Master\_h \*/

//

// Master.cpp

// Project

//

// Created by Andrej Hurynovič on 17.12.20.

//

#include "Master.h"

**void** Master::writeToFile(File& file){

Bachelor::writeToFile(file);

**bool** havePublication;

**if**(publication){

havePublication = **true**;

file.write(havePublication);

file.write(\*(publication));

}**else**{

havePublication = **false**;

file.write(havePublication);

}

}

**void** Master::readFromFile(File& file){

Bachelor::readFromFile(file);

**bool** havePublication;

file.read(&havePublication);

**if**(havePublication){

publication = **new** Publication();

file.read(publication);

}**else**{

publication = **NULL**;

}

}

**void** Master::calculateScholarship(**float** base){

Bachelor::calculateScholarship(base);

**if**(Settings::getPublication() && !**this**->publication){

scholarship = 0;

}

}

Publication\* Master::getPublication(){

**return** **this**->publication;

}

**void** Master::setPublication(Publication\* publication){

**this**->publication = publication;

}

**void** Master::editMenu(**bool** getValue){

**do**{

Bachelor::editMenu(**false**);

cout <<

"7. Publication. " << endl;

**if**(getValue){

cout << "0. Exit." << endl;

**int** value = ExceptionEnter::NumberData<**int**>(masterEditMenu);

**if**(!value){

**return**;

}

edit(value);

}**else**{

**return**;

}

}**while**(**true**);

}

**void** Master::edit(**int** key){

**if**(key < 7){

Bachelor::edit(key);

}**else**{

cout << "Enter new value: " << endl;

**switch** (key) {

**case** 7:

**if**(!publication){

publication = **new** Publication();

}

cin >> \*publication;

calculateScholarship();

**break**;

}

}

}

**void** Master::printSpreadsheet(**bool** scholarship){

Bachelor::printSpreadsheet(**false**);

cout <<

'|' << setw(Settings::getLength(DataType::publicationName)) << "Publication" << '|' <<

'|' << setw(Settings::getLength(DataType::journalName)) << "Journal name" << '|' <<

'|' << setw(Settings::getLength(DataType::accreditation)) << "Accreditation" << '|';

**if**(scholarship){

cout << '|' << setw(Settings::getLength(DataType::scholarship)) << "Scholarship" << '|';

}

}

ostream& **operator** << (ostream &stream, Master master){

**if**(master.showScholarship){

master.showScholarship = **false**;

cout << \***dynamic\_cast**<Bachelor\*>(&master);

master.showScholarship = **true**;

}**else**{

cout << \***dynamic\_cast**<Bachelor\*>(&master);

}

**if**(master.publication){

cout << \*(master.publication);

}**else**{

cout << setw(Settings::getLength(DataType::publicationName) + Settings::getLength(DataType::journalName) + Settings::getLength(DataType::accreditation) + 6) << '|';

}

**if**(master.showScholarship){

cout << '|' << setw(Settings::getLength(DataType::scholarship)) << master.scholarship << '|';

}

**return** stream;

}

istream& **operator** >> (istream &stream, Master &master){

cin >> \***dynamic\_cast**<Bachelor\*>(&master);

cout << endl << "Does the student have publication? (0/1): ";

**if**(ExceptionEnter::NumberData<**int**>(boolean)){

master.publication = **new** Publication();

cin >> \*(master.publication);

}

**return** stream;

}

//

// Doctor.h

// Project

//

// Created by Andrej Hurynovič on 10.11.20.

//

#ifndef Doctor\_h

#define Doctor\_h

#include"Master.h"

**class** Doctor: **public** Master{

**protected**:

**bool** dissertationComplited;

**public**:

Doctor(){

dissertationComplited = **false**;

};

**void** writeToFile(File&);

**void** readFromFile(File&);

**void** calculateScholarship(**float** base = Settings::getDoctorBase());

**bool** getDissertationComplited();

**void** setDissertationComplited(**bool**);

**void** editMenu(**bool**);

**void** edit(**int** key);

**static** **void** printSpreadsheet(**bool**);

**friend** ostream& **operator** << (ostream &stream, Doctor doctor);

**friend** istream& **operator** >> (istream &stream, Doctor &doctor);

~Doctor(){};

};

#endif /\* Doctor\_h \*/

//

// Doctor.cpp

// Project

//

// Created by Andrej Hurynovič on 17.12.20.

//

#include "Doctor.h"

**void** Doctor::writeToFile(File& file){

Master::writeToFile(file);

file.write(dissertationComplited);

}

**void** Doctor::readFromFile(File& file){

Master::readFromFile(file);

file.read(&dissertationComplited);

}

**void** Doctor::calculateScholarship(**float** base){

Master::calculateScholarship(base);

**if**(Settings::getJournalAccreditation() && (!**this**->publication || !**this**->publication->getAccreditation())){

scholarship = 0;

}

**if**(Settings::getDissertationComplited() && !**this**->dissertationComplited){

scholarship = 0;

}

}

**bool** Doctor::getDissertationComplited(){

**return** **this**->dissertationComplited;

}

**void** Doctor::setDissertationComplited(**bool** dissertationComplited){

**this**->dissertationComplited = dissertationComplited;

}

**void** Doctor::editMenu(**bool** getValue){

**do**{

Master::editMenu(**false**);

cout <<

"8. Dissertation complition. " << endl;

**if**(getValue){

cout << "0. Exit." << endl;

**int** value = ExceptionEnter::NumberData<**int**>(doctorEditMenu);

**if**(!value){

**return**;

}

edit(value);

}**else**{

**return**;

}

}**while**(**true**);

}

**void** Doctor::edit(**int** key){

**if**(key < 8){

Master::edit(key);

}**else**

cout << "Enter new value: " << endl;

**switch** (key) {

**case** 8:

dissertationComplited = ExceptionEnter::NumberData<**int**>(DataType::dissertationComplited);

calculateScholarship();

**break**;

}

}

**void** Doctor::printSpreadsheet(**bool** scholarship){

Master::printSpreadsheet(**false**);

cout << '|' << setw(Settings::getLength(DataType::dissertationComplited)) << "Dissertation" << '|';

**if**(scholarship){

cout << '|' << setw(Settings::getLength(DataType::scholarship)) << "Scholarship" << '|';

}

}

ostream& **operator** << (ostream &stream, Doctor doctor){

**if**(doctor.showScholarship){

doctor.showScholarship = **false**;

cout << \***dynamic\_cast**<Master\*>(&doctor);

doctor.showScholarship = **true**;

}**else**{

cout << \***dynamic\_cast**<Master\*>(&doctor);

}

cout << '|' << setw(Settings::getLength(DataType::dissertationComplited)) << doctor.dissertationComplited << '|';

**if**(doctor.showScholarship){

cout << '|' << setw(Settings::getLength(DataType::scholarship)) << doctor.scholarship << '|';

}

**return** stream;

}

istream& **operator** >> (istream &stream, Doctor &doctor){

cin >> \***dynamic\_cast**<Master\*>(&doctor);

cout << endl << "Does this student completed dissertation? (0/1): ";

doctor.dissertationComplited = ExceptionEnter::NumberData<**int**>(boolean);

**return** stream;

}

//

// ExceptionTypes.h

// LabWork 6. Exception

//

// Created by Andrej Hurynovič on 10.11.20.

//

#ifndef ExceptionTypes\_h

#define ExceptionTypes\_h

#include<iostream>

#include<exception>

**enum** DataType{

boolean,

//Human

name,

privateID,

humanEditMenu,

//Bachelor

group,

bachelorEditMenu,

scholarship,

//Master

masterEditMenu,

//Doctor

doctorEditMenu,

//Date

date,

year,

month,

day,

//Mark

mark,

resit,

subject,

markMenu,

markID,

//Benefit

benefitType,

benefitMenu,

benefitID,

//Publication

publicationName,

journalName,

accreditation,

//Settings

settingsMenu,

numberOfSettings,

bachelorBase,

masterBase,

doctorBase,

benefits,

disabledBenefit,

socialBenefit,

councilBenefit,

personalBenefit,

marks,

markFirst,

markSecond,

markThird,

publication,

journalAccreditation,

dissertationComplited,

//File

settingsFile,

//Menu

studnetTypes,

removeMenu,

searchMenu,

menuNumber,

scholarshipMenu,

scholarshipValue,

sortMenu,

};

**using** **namespace** std;

**class** ExceptionTypes {

**protected**:

**enum** DataType type;

string errorMessage;

string userMessage;

**public**:

ExceptionTypes(**enum** DataType type, string string = ""){

**this**->type = type;

**this**->userMessage = string;

**switch** (type) {

**case** boolean:

errorMessage = "IDKNRN";

**break**;

//Human

**case** name:

errorMessage = "name";

**break**;

**case** privateID:

errorMessage = "private id";

**break**;

**case** humanEditMenu:

errorMessage = "human edit menu";

**break**;

//Bachelor

**case** bachelorEditMenu:

errorMessage = "bachelor edit menu";

**break**;

**case** scholarship:

errorMessage = "scholarship";

**break**;

**case** markMenu:

errorMessage = "mark menu";

**break**;

**case** benefitMenu:

errorMessage = "benefit menu";

**break**;

**case** markID:

errorMessage = "mark ID";

**break**;

**case** benefitID:

errorMessage = "benefit ID";

**break**;

//Master

**case** masterEditMenu:

errorMessage = "master edit menu";

**break**;

//Doctor

**case** doctorEditMenu:

errorMessage = "doctor edit menu";

**break**;

//Date

**case** group:

errorMessage = "group";

**break**;

**case** date:

errorMessage = "date";

**break**;

**case** year:

errorMessage = "year";

**break**;

**case** month:

errorMessage = "month";

**break**;

**case** day:

errorMessage = "day";

**break**;

//Mark

**case** mark:

errorMessage = "mark";

**break**;

**case** resit:

errorMessage = "resit";

**break**;

**case** subject:

errorMessage = "subject name";

**break**;

//Benefit

**case** benefitType:

errorMessage = "benefit type";

**break**;

//Publication

**case** publicationName:

errorMessage = "publication name";

**break**;

**case** journalName:

errorMessage = "journal name";

**break**;

**case** accreditation:

errorMessage = "accreditation";

**break**;

//Settings

**case** settingsMenu:

errorMessage = "settings menu";

**break**;

**case** numberOfSettings:

errorMessage = "number of settings";

**break**;

**case** bachelorBase:

errorMessage = "bachelor base scholarship";

**break**;

**case** masterBase:

errorMessage = "master base scholarship";

**break**;

**case** doctorBase:

errorMessage = "doctor base scholarship";

**break**;

**case** benefits:

errorMessage = "impact of benefits on the scholarship";

**break**;

**case** disabledBenefit:

errorMessage = "disabled";

**break**;

**case** socialBenefit:

errorMessage = "social benefit";

**break**;

**case** councilBenefit:

errorMessage = "council benefit";

**break**;

**case** personalBenefit:

errorMessage = "personal benefit";

**break**;

**case** marks:

errorMessage = "marks";

**break**;

**case** markFirst:

errorMessage = "marks first level (7 – 8)";

**break**;

**case** markSecond:

errorMessage = "marks second level (8 - 9)";

**break**;

**case** markThird:

errorMessage = "marks third level (9 - 10)";

**break**;

**case** publication:

errorMessage = "publication";

**break**;

**case** journalAccreditation:

errorMessage = "publication journal accreditation";

**break**;

**case** dissertationComplited:

errorMessage = "dissertation complited";

**break**;

//File

**case** settingsFile:

errorMessage = "settings file";

**break**;

//Menu

**case** studnetTypes:

errorMessage = "studnet types";

**break**;

**case** menuNumber:

errorMessage = "menu number";

**break**;

**case** removeMenu:

errorMessage = "remove menu";

**break**;

**case** searchMenu:

errorMessage = "search menu";

**break**;

**case** scholarshipMenu:

errorMessage = "scholarship menu";

**break**;

**case** scholarshipValue:

errorMessage = "scholarship value";

**break**;

**case** sortMenu:

errorMessage = "sort menu";

**break**;

}

}

string getErrorTypeString();

string getUserMessage();

**static** **void** Terminate();

~ExceptionTypes(){}

};

#endif /\* ExceptionTypes\_h \*/

//

// ExceptionTypes.cpp

// LabWork 6. Exception

//

// Created by Andrej Hurynovič on 10.11.20.

//

#include "ExceptionTypes.h"

string ExceptionTypes::getErrorTypeString(){

**return** errorMessage;

}

string ExceptionTypes::getUserMessage(){

**return** userMessage;

}

**void** ExceptionTypes::Terminate(){

cout << "Terminate called";

}

//

// ExceptionEnter.h

// LabWork 6. Exception

//

// Created by Andrej Hurynovič on 10.11.20.

//

#ifndef ExceptionEnter\_h

#define ExceptionEnter\_h

#include "ExceptionTypes.h"

#include "Date.h"

#include "BinaryTree.h"

**class** ExceptionEnter: **public** ExceptionTypes {

**public**:

**template** <**typename** numberType>

**static** numberType NumberData(**enum** DataType type){

numberType minimalValue = 0;

numberType maximalValue = 0;

numberType number;

**bool** temp;

**switch** (type) {

**case** boolean:

minimalValue = 0;

maximalValue = 1;

**break**;

//Human

**case** humanEditMenu:

minimalValue = 0;

maximalValue = 3;

**break**;

//Bachelor

**case** bachelorEditMenu:

minimalValue = 0;

maximalValue = 6;

**break**;

**case** markMenu:

minimalValue = 0;

maximalValue = 2;

**break**;

**case** benefitMenu:

minimalValue = 0;

maximalValue = 2;

**break**;

**case** markID:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** benefitID:

minimalValue = 0;

maximalValue = -1;

**break**;

//Master

**case** masterEditMenu:

minimalValue = 0;

maximalValue = 7;

**break**;

//Doctor

**case** doctorEditMenu:

minimalValue = 0;

maximalValue = 8;

**break**;

//Date

**case** year:

minimalValue = 1900;

maximalValue = 2020;

**break**;

**case** month:

minimalValue = 1;

maximalValue = 12;

**break**;

**case** day:

minimalValue = 1;

maximalValue = 31;

**break**;

//Mark

**case** mark:

minimalValue = 0;

maximalValue = 10;

**break**;

**case** resit:

minimalValue = 0;

maximalValue = 1;

**break**;

//Benefit

**case** benefitType:

minimalValue = 0;

maximalValue = 3;

**break**;

//Publication

**case** accreditation:

minimalValue = 0;

maximalValue = 1;

**break**;

//Settings

**case** settingsMenu:

minimalValue = 0;

maximalValue = 3;

**break**;

**case** numberOfSettings:

minimalValue = 0;

maximalValue = 15;

**break**;

**case** bachelorBase:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** masterBase:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** doctorBase:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** benefits:

minimalValue = 0;

maximalValue = 1;

**break**;

**case** disabledBenefit:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** socialBenefit:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** councilBenefit:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** personalBenefit:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** marks:

minimalValue = 0;

maximalValue = 1;

**break**;

**case** markFirst:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** markSecond:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** markThird:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** publication:

minimalValue = 0;

maximalValue = 1;

**break**;

**case** journalAccreditation:

minimalValue = 0;

maximalValue = 1;

**break**;

**case** dissertationComplited:

minimalValue = 0;

maximalValue = 1;

**break**;

//Menu

**case** studnetTypes:

minimalValue = 0;

maximalValue = 3;

**break**;

**case** menuNumber:

minimalValue = 0;

maximalValue = 8;

**break**;

**case** removeMenu:

minimalValue = 0;

maximalValue = 2;

**break**;

**case** searchMenu:

minimalValue = 0;

maximalValue = 5;

**break**;

**case** scholarshipMenu:

minimalValue = 0;

maximalValue = 2;

**break**;

**case** scholarshipValue:

minimalValue = 0;

maximalValue = -1;

**break**;

**case** sortMenu:

minimalValue = 0;

maximalValue = 2;

**break**;

**default**:

**break**;

}

**do**{

temp = **false**;

**try** {

set\_terminate(Terminate);

**try** {

cin >> number;

} **catch** (**const** std::overflow\_error& exception) {

cout << exception.exception::what();

}

**if**(cin.fail() || cin.get() != '\n'){

**throw** ExceptionTypes(type, "Stream error.");

}

**if**(minimalValue != maximalValue){

**if**(number < minimalValue){

**throw** ExceptionTypes(type, "The number is less than required.");

}

**if**(minimalValue < maximalValue){

**if**(number > maximalValue){

**throw** ExceptionTypes(type, "The number is greater than required.");

}

}

}

} **catch** (ExceptionTypes& exception) {

cin.clear();

rewind (stdin);

cout << "Please correct error in " << exception.getErrorTypeString() << ':' << endl << exception.getUserMessage() << endl;

temp = **true**;

}

**catch**(...){

cout << "Unknown error";

cin.clear();

rewind(stdin);

}

}**while**(temp);

**return** number;

}

**template** <**typename** studentType>

**static** **bool** checkThisID(string ID, BinaryTree<studentType, **int**> &tree){

**return** tree.checkKey(stoi(ID));

}

**static** string stringData(**enum** DataType);

**static** **bool** onlyLetters(string);

**static** **bool** onlyDigiths(string);

};

#endif /\* ExceptionEnter\_h \*/

//

// ExceptionEnter.cpp

// LabWork 6. Exception

//

// Created by Andrej Hurynovič on 10.11.20.

//

#include "ExceptionEnter.h"

#include "Settings.h"

string ExceptionEnter::stringData(**enum** DataType type){

**int** requiredLength = 0;

**int** maximalLength = 0;

string string;

**bool** temp;

**switch** (type) {

//Human

**case** name:

maximalLength = Settings::getLength(name);

**break**;

**case** privateID:

requiredLength = Settings::getLength(privateID);

**break**;

//Teacher

**case** subject:

maximalLength = Settings::getLength(subject);

**break**;

//Bachelor

**case** group:

requiredLength = Settings::getLength(group);

**break**;

//Publication

**case** publicationName:

maximalLength = Settings::getLength(publicationName);

**break**;

**case** journalName:

maximalLength = Settings::getLength(journalName);

**break**;

**default**:

**break**;

}

**do**{

temp = **false**;

**try** {

set\_terminate(Terminate);

**try** {

getline(cin, string);

} **catch** (**const** std::overflow\_error& exception) {

cout << exception.exception::what();

}

**if**(cin.fail()){

**throw** ExceptionTypes(type, "Stream fail.");

}

**if**(maximalLength){

**if**(string.length() > maximalLength){

**throw** ExceptionTypes(type, "The string is longer than required.");

}

}

**if**(requiredLength){

**if**(string.length() > requiredLength){

**throw** ExceptionTypes(type, "The string is longer than required.");

}

**if**(string.length() < requiredLength){

**throw** ExceptionTypes(type, "The string is shorter than required.");

}

}

**switch** (type) {

**case** name:

**if**(!onlyLetters(string)){

**throw** ExceptionTypes(type, "The name must consist only of letters.");

}

**break**;

**case** subject:

**if**(!onlyLetters(string)){

**throw** ExceptionTypes(type, "The name must consist only of letters.");

}

**break**;

**case** group:

**if**(!onlyDigiths(string)){

**throw** ExceptionTypes(type, "The name must consist only of digits.");

}

**break**;

**case** privateID:

**if**(!onlyDigiths(string)){

**throw** ExceptionTypes(type, "The name must consist only of digits.");

}

**break**;

**default**:

**break**;

}

} **catch** (ExceptionTypes& exception) {

cin.clear();

rewind (stdin);

cout << "Please correct error in " << exception.getErrorTypeString() << ':' << endl << exception.getUserMessage() << endl;

temp = **true**;

}

**catch**(...){

cout << "Unknown error";

cin.clear();

rewind(stdin);

}

}**while**(temp);

**return** string;

}

**bool** ExceptionEnter::onlyLetters(string string){

**for**(**int** i = 0; i < string.length(); i++){

**if**((string[i] < 'A' || string[i] > 'Z') && (string[i] < 'a' || string[i] > 'z')){

**return** **false**;

}

}

**return** **true**;

}

**bool** ExceptionEnter::onlyDigiths(string string){

**for**(**int** i = 0; i < string.length(); i++){

**if**((string[i] < '0' || string[i] > '9')){

**return** **false**;

}

}

**return** **true**;

}

//

// ExceptionFile.h

// Project

//

// Created by Andrej Hurynovič on 17.12.20.

//

#ifndef ExceptionFile\_h

#define ExceptionFile\_h

#include "ExceptionTypes.h"

#include <fstream>

**enum** FileType{

bachelors,

masters,

doctors,

settings,

};

**class** ExceptionFile: **public** ExceptionTypes{

**public**:

**static** fstream openFile(**enum** FileType, **bool**);

};

#endif /\* ExceptionFile\_h \*/

//

// Exception File.cpp

// Project

//

// Created by Andrej Hurynovič on 17.12.20.

//

#include "ExceptionFile.h"

#include "Settings.h"

#include "Algorithms.h"

fstream ExceptionFile::openFile(**enum** FileType type, **bool** write){

**return** Algorithms::openFile(type, write);

}

//

// Date.hpp

// LabWork 5. Inheritance

//

// Created by Andrej Hurynovič on 7.10.20.

//

#ifndef Date\_h

#define Date\_h

#include <iostream>

#include <iomanip>

**using** **namespace** std;

**class** Date{

**private**:

**int** day;

**int** month;

**int** year;

**public**:

Date(**int** day = 0, **int** month = 0, **int** year = 0){

**this**->day = day;

**this**->month = month;

**this**->year = year;

}

Date **operator** = (**const** Date&);

**friend** ostream& **operator** << (ostream &outputStream, **const** Date date);

**friend** istream& **operator** >> (istream &inputStream, Date &Date);

~Date(){};

};

#endif /\* Date\_h \*/

//

// Date.cpp

// LabWork 5. Inheritance

//

// Created by Andrej Hurynovič on 7.10.20.

//

#include "Date.h"

#include "ExceptionEnter.h"

#include "Settings.h"

Date Date::**operator** = (**const** Date& otherDate){

**if**(&otherDate != **this**){

**this**->day = otherDate.day;

**this**->month = otherDate.month;

**this**->year = otherDate.year;

}

**return** \***this**;

}

ostream& **operator** << (ostream &outputStream, **const** Date date){

cout << setw(Settings::getLength(day)) << date.day <<

"." << setw(Settings::getLength(month)) << date.month <<

"." << setw(Settings::getLength(year)) << date.year;

**return** outputStream;

}

istream& **operator** >> (istream &inputStream, Date &date){

cout << "Enter day: ";

date.day = ExceptionEnter::NumberData<**int**>(day);

cout << "Enter month: ";

date.month = ExceptionEnter::NumberData<**int**>(month);

**do**{

**if**(date.month % 2 == 0 && date.day == 31){

cout << "Less than 31 days this month." << endl;

cout << "Enter day: ";

date.day = ExceptionEnter::NumberData<**int**>(day);

}**else**{

**break**;

}

}**while**(**true**);

**do**{

**if**(date.month == 2 && date.day > 29){

cout << "Less than 30 days this month." << endl;

cout << "Enter day: ";

date.day = ExceptionEnter::NumberData<**int**>(day);

}**else**{

**break**;

}

}**while**(**true**);

cout << "Enter year: ";

date.year = ExceptionEnter::NumberData<**int**>(year);

**do**{

**if**(!((date.year % 4 == 0) && !((date.year % 100 == 0) && (date.year % 400 != 0)))){

**if**(date.day > 28){

cout << "This is a non-leap year, there are less than 29 days in February." << endl;

cout << "Enter day: ";

date.day = ExceptionEnter::NumberData<**int**>(day);

}**else**{

**break**;

}

}**else**{

**break**;

}

}**while**(**true**);

**return** inputStream;

}

//

// Mark.h

// Project

//

// Created by Andrej Hurynovič on 27.10.20.

//

#ifndef Mark\_h

#define Mark\_h

#include "ExceptionEnter.h"

#include "Settings.h"

**using** **namespace** std;

**class** Mark {

**protected**:

**int** mark;

**bool** resit;

string subject;

**public**:

Mark(**int** mark = -1, **bool** resit = **false**, string subject = "UNDEF"){

**this**->mark = mark;

**this**->resit = resit;

**this**->subject = subject;

}

**int** getMark();

**bool** getResit();

string getSubject();

**void** setMark(**int**);

**void** setResit(**bool**);

**void** setSubject(string);

**static** **void** printSpreadsheet();

**friend** ostream& **operator** << (ostream &stream, Mark mark);

**friend** istream& **operator** >> (istream &stream, Mark &mark);

~Mark(){};

};

#endif /\* Mark\_h \*/

//

// Mark.cpp

// Project

//

// Created by Andrej Hurynovič on 13.12.20.

//

#include "Mark.h"

**int** Mark::getMark(){

**return** **this**->mark;

}

**bool** Mark::getResit(){

**return** **this**->resit;

}

string Mark::getSubject(){

**return** **this**->subject;

}

**void** Mark::setMark(**int** mark){

**this**->mark = mark;

}

**void** Mark::setResit(**bool** resit){

**this**->resit = resit;

}

**void** Mark::setSubject(string subject){

**this**->subject = subject;

}

**void** Mark::printSpreadsheet(){

cout <<

'|' << setw(Settings::getLength(DataType::subject)) << "Subject" << '|' <<

'|' << setw(Settings::getLength(DataType::mark)) << "Mark" << '|' <<

'|' << setw(Settings::getLength(DataType::resit)) << "Resit" << '|';

}

ostream& **operator** << (ostream &stream, Mark mark){

cout <<

setw(Settings::getLength(DataType::subject)) << mark.subject << '|' <<

'|' << setw(Settings::getLength(DataType::mark)) << mark.mark << '|' <<

'|' << setw(Settings::getLength(DataType::resit)) << mark.resit << '|';

;

**return** stream;

}

istream& **operator** >> (istream &stream, Mark &mark){

cout << "Enter subject: ";

mark.subject = ExceptionEnter::stringData(subject);

cout << "Enter mark: ";

mark.mark = ExceptionEnter::NumberData<**int**>(DataType::mark);

cout << "Did this student resit this subject? (0/1):" << endl;

mark.resit = ExceptionEnter::NumberData<**int**>(resit);

**return** stream;

}

//

// Benefit.h

// Project

//

// Created by Andrej Hurynovič on 27.10.20.

//

#ifndef Benefit\_h

#define Benefit\_h

#include <iostream>

#include "ExceptionEnter.h"

**using** **namespace** std;

**enum** BenefitType{

disabled,

social,

council,

personal

};

**class** Benefit {

**protected**:

**enum** BenefitType type;

**public**:

Benefit(BenefitType type = disabled){

**this**->type = type;

}

**enum** BenefitType getType();

**float** getModifier();

**void** setType(**enum** BenefitType);

**bool** **operator** == (**const** Benefit &benefit);

**friend** ostream& **operator** << (ostream &stream, Benefit benefit);

**friend** istream& **operator** >> (istream &stream, Benefit &benefit);

~Benefit(){};

};

#endif /\* Benefit\_h \*/

//

// Benefit.cpp

// Project

//

// Created by Andrej Hurynovič on 13.12.20.

//

#include "Benefit.h"

#include "Settings.h"

**enum** BenefitType Benefit::getType(){

**return** **this**->type;

}

**float** Benefit::getModifier(){

**return** Settings::getBenefitByType(**this**->type);

}

**void** Benefit::setType(**enum** BenefitType type){

**this**->type = type;

}

**bool** Benefit::**operator** == (**const** Benefit &benefit){

**if**(**this**->type == benefit.type){

**return** **true**;

}**else**{

**return** **false**;

}

}

ostream& **operator** << (ostream &stream, Benefit benefit){

cout << setw (Settings::getLength(benefitType));

**switch** (benefit.type) {

**case** disabled:

cout << "Disabled";

**break**;

**case** social:

cout << "Social";

**break**;

**case** council:

cout << "Council";

**break**;

**case** personal:

cout << "Personal";

**break**;

}

**return** stream;

}

istream& **operator** >> (istream &stream, Benefit &benefit){

cout << "Choose benefit type: type (mofifier)" << endl

<< "0. Disabled ("<< Settings::getBenefitByType(disabled) <<")" << endl

<< "1. Social ("<< Settings::getBenefitByType(social) <<")" << endl

<< "2. Concil ("<< Settings::getBenefitByType(council) <<")" << endl

<< "3. Personal ("<< Settings::getBenefitByType(personal) <<")" << endl;

**switch** (ExceptionEnter::NumberData<**int**>(benefitType)) {

**case** 0:

benefit.type = disabled;

**break**;

**case** 1:

benefit.type = social;

**break**;

**case** 2:

benefit.type = council;

**break**;

**case** 3:

benefit.type = personal;

**break**;

}

**return** stream;

}

//

// Publication.h

// Project

//

// Created by Andrej Hurynovič on 25.11.20.

//

#ifndef Publication\_h

#define Publication\_h

#include <iostream>

#include "Settings.h"

#include "ExceptionEnter.h"

**using** **namespace** std;

**class** Publication {

**protected**:

string name;

string journalName;

**bool** accreditation;

**public**:

Publication(string name = "UNDEFINED", string journalName = "UNDEFINED", **bool** accreditation = **false**){

**this**->name = name;

**this**->journalName = journalName;

**this**->accreditation = accreditation;

}

string getPublicationName();

string getJournalName();

**bool** getAccreditation();

**void** setPublicationName(string);

**void** setJournalName(string);

**void** setAccreditation(**bool**);

**friend** ostream& **operator** << (ostream &stream, Publication publication);

**friend** istream& **operator** >> (istream &stream, Publication &publication);

~Publication(){};

};

#endif /\* Publication\_h \*/

//

// Publication.cpp

// Project

//

// Created by Andrej Hurynovič on 13.12.20.

//

#include "Publication.h"

string Publication::getPublicationName(){

**return** **this**->name;

}

string Publication::getJournalName(){

**return** **this**->journalName;

}

**bool** Publication::getAccreditation(){

**return** **this**->accreditation;

}

**void** Publication::setPublicationName(string name){

**this**->name = name;

}

**void** Publication::setJournalName(string journalName){

**this**->journalName = journalName;

}

**void** Publication::setAccreditation(**bool** accreditation){

**this**->accreditation = accreditation;

}

ostream& **operator** << (ostream &outputStream, **const** Publication publication){

cout <<

'|' << setw(Settings::getLength(DataType::name)) << publication.name << '|' <<

'|' << setw(Settings::getLength(DataType::journalName)) << publication.journalName << '|' <<

'|' << setw(Settings::getLength(DataType::accreditation)) << publication.accreditation << '|';

**return** outputStream;

}

istream& **operator** >> (istream &inputStream, Publication &publication){

cout << "Enter publication name: ";

publication.name = ExceptionEnter::stringData(publicationName);

cout << "Enter journal name: ";

publication.journalName = ExceptionEnter::stringData(journalName);

cout << "Does this journal have accreditation? (0/1): " << endl;

publication.accreditation = ExceptionEnter::NumberData<**int**>(accreditation);

**return** inputStream;

}

//

// File.h

// Project

//

// Created by Andrej Hurynovič on 3.01.21.

//

#ifndef File\_h

#define File\_h

#include <iostream>

#include "ExceptionFile.h"

**class** File {

**public**:

fstream file;

FileType type;

//public:

File(FileType type, **bool** write){

**this**->type = type;

**if**(write){

**this**->file = ExceptionFile::openFile(type, **true**);

}**else**{

**this**->file = ExceptionFile::openFile(type, **false**);

}

}

**template** <**typename** dataType>

**void** read(dataType \*data){

file.read((**char**\*)data, **sizeof**(\*(data)));

}

**template** <**typename** dataType>

**void** write(dataType data){

file.write((**char**\*)&data, **sizeof**(dataType));

}

FileType getType();

**void** setType(FileType);

**bool** endOfFile();

~File(){

file.close();

}

};

#endif /\* File\_h \*/

//

// File.cpp

// Project

//

// Created by Andrej Hurynovič on 3.01.21.

//

#include "File.h"

FileType File::getType(){

**return** type;

}

**void** File::setType(FileType type){

**this**->type = type;

}

**bool** File::endOfFile(){

**long** now = file.tellp();

file.seekg(0, ios::end);

**long** fileLenght = file.tellp();

file.seekg(now, ios::beg);

**if**(now == fileLenght){

**return** **true**;

}**else**{

**return** **false**;

}

}

//

// Settings.h

// Project

//

// Created by Andrej Hurynovič on 13.12.20.

//

#ifndef Settings\_h

#define Settings\_h

#include "Benefit.h"

#include "File.h"

**class** Settings{

**protected**:

**static** **float** bachelorBase;

**static** **float** masterBase;

**static** **float** doctorBase;

**static** **bool** benefits;

**static** **float** disabledBenefit;

**static** **float** socialBenefit;

**static** **float** councilBenefit;

**static** **float** personalBenefit;

**static** **bool** marks;

**static** **float** markFirst;

**static** **float** markSecond;

**static** **float** markThird;

**static** **bool** resit;

**static** **bool** publication;

**static** **bool** journalAccreditation;

**static** **bool** dissertationComplited;

**public**:

**static** **void** getSettingsFromFile();

**static** **void** setSettingsToFile();

**static** **void** setTheDefaultSettings();

**static** **void** menu();

**static** **void** editSettings();

**static** **void** printSettings();

**static** **int** getLength(**enum** DataType);

**static** **float** getBachelorBase();

**static** **float** getMasterBase();

**static** **float** getDoctorBase();

**static** **bool** getBenefits();

**static** **float** getBenefitByType(**enum** BenefitType);

**static** **bool** getMarks();

**static** **float** getMarkFirst();

**static** **float** getMarkSecond();

**static** **float** getMarkThird();

**static** **bool** getResit();

**static** **bool** getPublication();

**static** **bool** getJournalAccreditation();

**static** **bool** getDissertationComplited();

};

#endif /\* Settings\_h \*/

//

// Settings.cpp

// Project

//

// Created by Andrej Hurynovič on 13.12.20.

//

#include "Settings.h"

**void** Settings::getSettingsFromFile(){

File file(settings, **false**);

file.read(&bachelorBase);

file.read(&masterBase);

file.read(&doctorBase);

file.read(&benefits);

file.read(&disabledBenefit);

file.read(&socialBenefit);

file.read(&councilBenefit);

file.read(&personalBenefit);

file.read(&marks);

file.read(&markFirst);

file.read(&markSecond);

file.read(&markThird);

file.read(&resit);

file.read(&publication);

file.read(&journalAccreditation);

file.read(&dissertationComplited);

}

**void** Settings::setSettingsToFile(){

File file(settings, **true**);

file.write(bachelorBase);

file.write(masterBase);

file.write(doctorBase);

file.write(benefits);

file.write(disabledBenefit);

file.write(socialBenefit);

file.write(councilBenefit);

file.write(personalBenefit);

file.write(marks);

file.write(markFirst);

file.write(markSecond);

file.write(markThird);

file.write(resit);

file.write(publication);

file.write(journalAccreditation);

file.write(dissertationComplited);

}

**void** Settings::setTheDefaultSettings(){

bachelorBase = 100;

masterBase = 200;

doctorBase = 400;

benefits = **true**;

disabledBenefit = 1.5;

socialBenefit = 2;

councilBenefit = 1.6;

personalBenefit = 3;

marks = **true**;

markFirst = 1.1;

markSecond = 1.2;

markThird = 1.3;

resit = **true**;

publication = **true**;

journalAccreditation = **true**;

dissertationComplited = **true**;

setSettingsToFile();

}

**void** Settings::menu(){

**do**{

cout << "0. Print settings." << endl << "1. Edit settings." << endl << "2. Set the default settings." << endl << "3. Exit." << endl;

**switch**(ExceptionEnter::NumberData<**int**>(DataType::settingsMenu)){

**case** 0:

printSettings();

**break**;

**case** 1:

editSettings();

**break**;

**case** 2:

setTheDefaultSettings();

**break**;

**case** 3:

**return**;

}

}**while**(**true**);

}

**void** Settings::editSettings(){

cout <<

"0. Bachelor base scholarship. (0...∞)." << endl <<

"1. Master base scholarship. (0...∞)." << endl <<

"2. Вoctor base scholarship. (0...∞)." << endl <<

"3. Benefits. (0/1):." << endl <<

"4. Disabled benif. (0...∞)." << endl <<

"5. Social benefit. (0...∞)." << endl <<

"6. Council benefit. (0...∞)." << endl <<

"7. Personal benefit. (0...∞)." << endl <<

"8. Marks. (0/1)." << endl <<

"9. Marks first level (7 – 8) (0...∞)." << endl <<

"10. Marks second level (8 - 9) (0...∞)." << endl <<

"11. Marks third level (9 - 10) (0...∞)." << endl <<

"12. Resit. (0/1)." << endl <<

"13. Publication. (0/1)." << endl <<

"14. Publication journal accreditation. (0/1)." << endl <<

"15. Dissertation complited (0/1)." << endl;

**int** temp = ExceptionEnter::NumberData<**int**>(numberOfSettings);

cout << "Enter new value: ";

**switch** (temp) {

**case** 0:

bachelorBase = ExceptionEnter::NumberData<**float**>(DataType::bachelorBase);

**break**;

**case** 1:

masterBase = ExceptionEnter::NumberData<**float**>(DataType::masterBase);

**break**;

**case** 2:

doctorBase = ExceptionEnter::NumberData<**float**>(DataType::doctorBase);

**break**;

**case** 3:

benefits = ExceptionEnter::NumberData<**float**>(DataType::benefits);

**break**;

**case** 4:

disabledBenefit = ExceptionEnter::NumberData<**float**>(DataType::disabledBenefit);

**break**;

**case** 5:

socialBenefit = ExceptionEnter::NumberData<**float**>(DataType::socialBenefit);

**break**;

**case** 6:

councilBenefit = ExceptionEnter::NumberData<**float**>(DataType::councilBenefit);

**break**;

**case** 7:

personalBenefit = ExceptionEnter::NumberData<**float**>(DataType::personalBenefit);

**break**;

**case** 8:

marks = ExceptionEnter::NumberData<**float**>(DataType::marks);

**break**;

**case** 9:

markFirst = ExceptionEnter::NumberData<**float**>(DataType::markFirst);

**break**;

**case** 10:

markSecond = ExceptionEnter::NumberData<**float**>(DataType::markSecond);

**break**;

**case** 11:

markThird = ExceptionEnter::NumberData<**float**>(DataType::markThird);

**break**;

**case** 12:

resit = ExceptionEnter::NumberData<**float**>(DataType::resit);

**break**;

**case** 13:

publication = ExceptionEnter::NumberData<**float**>(DataType::publication);

**break**;

**case** 14:

journalAccreditation = ExceptionEnter::NumberData<**float**>(DataType::journalAccreditation);

**break**;

**case** 15:

dissertationComplited = ExceptionEnter::NumberData<**float**>(DataType::dissertationComplited);

}

setSettingsToFile();

cout << "New value was saved." << endl;

}

**void** Settings::printSettings(){

cout <<

"0. Bachelor base scholarship = " << bachelorBase << endl <<

"1. Master base scholarship = " << masterBase << endl <<

"2. Вoctor base scholarship = " << doctorBase << endl <<

"3. Impact of benefits on the scholarship = " << benefits << endl <<

"4. Disabled benefit " << disabledBenefit << endl <<

"5. Social benefit = " << socialBenefit << endl <<

"6. Council benefit = " << councilBenefit << endl <<

"7. Personal benefit = " << personalBenefit << endl <<

"8. Marks = " << marks << endl <<

"9. Marks first level (7 – 8) = " << markFirst << endl <<

"10. Marks second level (8 - 9) = " << markSecond << endl <<

"11. Marks third level (9 - 10) = " << markThird << endl <<

"12. Resit = " << resit << endl <<

"13. Publication = " << publication << endl <<

"14. Publication journal accreditation = " << journalAccreditation << endl <<

"15. Dissertation complited = " << dissertationComplited << endl << endl;

}

**float** Settings::getBachelorBase(){

**return** bachelorBase;

}

**float** Settings::getMasterBase(){

**return** masterBase;

}

**float** Settings::getDoctorBase(){

**return** doctorBase;

}

**bool** Settings::getBenefits(){

**return** benefits;

}

**float** Settings::getBenefitByType(**enum** BenefitType type){

**switch** (type) {

**case** disabled:

**return** disabledBenefit;

**break**;

**case** social:

**return** socialBenefit;

**break**;

**case** council:

**return** councilBenefit;

**break**;

**case** personal:

**return** personalBenefit;

**break**;

}

}

**bool** Settings::getMarks(){

**return** marks;

}

**float** Settings::getMarkFirst(){

**return** markFirst;

}

**float** Settings::getMarkSecond(){

**return** markSecond;

}

**float** Settings::getMarkThird(){

**return** markThird;

}

**bool** Settings::getResit(){

**return** resit;

}

**bool** Settings::getPublication(){

**return** publication;

}

**bool** Settings::getJournalAccreditation(){

**return** journalAccreditation;

}

**bool** Settings::getDissertationComplited(){

**return** dissertationComplited;

}

**int** Settings::getLength(**enum** DataType type){

**switch** (type) {

**case** name:

**return** 16;

**break**;

**case** privateID:

**return** 8;

**break**;

//Teacher

**case** subject:

**return** 16;

**break**;

//Bachelor

**case** group:

**return** 6;

**break**;

**case** scholarship:

**return** 11;

**break**;

//Date

**case** date:

**return** 2;

**break**;

**case** year:

**return** 4;

**break**;

**case** month:

**return** 2;

**break**;

**case** day:

**return** 2;

//Mark

**case** mark:

**return** 4;

**break**;

**case** DataType::resit:

**return** 5;

**break**;

//Benefit

**case** benefitType:

**return** 8;

**break**;

//Publication

**case** publicationName:

**return** 16;

**break**;

**case** journalName:

**return** 16;

**break**;

**case** accreditation:

**return** 13;

**break**;

//Doctor

**case** DataType::dissertationComplited:

**return** 12;

**break**;

**default**:

**return** 0;

**break**;

}

}

**float** Settings::bachelorBase = 100;

**float** Settings::masterBase = 200;

**float** Settings::doctorBase = 400;

**bool** Settings::benefits = **true**;

**float** Settings::disabledBenefit = 1.5;

**float** Settings::socialBenefit = 2;

**float** Settings::councilBenefit = 1.6;

**float** Settings::personalBenefit = 3;

**bool** Settings::marks = **true**;

**float** Settings::markFirst = 1.1;

**float** Settings::markSecond = 1.2;

**float** Settings::markThird = 1.3;

**bool** Settings::resit = **true**;

**bool** Settings::publication = **true**;

**bool** Settings::journalAccreditation = **true**;

**bool** Settings::dissertationComplited = **true**;

//

// Menu.h

// Project

//

// Created by Andrej Hurynovič on 17.12.20.

//

#ifndef Menu\_h

#define Menu\_h

#include "Doctor.h"

#include "BinaryTree.h"

#include "Undo.h"

**class** Menu {

BinaryTree<Bachelor, **int**> bachelors;

BinaryTree<Master, **int**> masters;

BinaryTree<Doctor, **int**> doctors;

stack<Undo<Bachelor>> bachelorsUndoStack;

stack<Undo<Master>> mastersUndoStack;

stack<Undo<Doctor>> doctorsUndoStack;

**int** chooseStudentType();

**bool** noStudents();

**template** <**typename** studentType>

**bool** noStudentsIn(BinaryTree<studentType, **int**>&);

**bool** checkPrivateID(string privateID, **bool** printMessage = **true**);

**template** <**typename** studentType>

BinaryTree<studentType, **int**>sortBinaryTree(BinaryTree<studentType, **int**>);

**template** <**typename** studentType>

studentType createStudent();

**template** <**typename** studentType>

**void** editStudent(BinaryTree<studentType, **int**>&);

**template** <**typename** studentType>

**bool** editStudentByID(BinaryTree<studentType, **int**>&, string);

**template** <**typename** studentType>

**void** removeStudent(BinaryTree<studentType, **int**>&);

**template** <**typename** studentType>

**bool** removeStudentByID(BinaryTree<studentType, **int**>&, string);

**template** <**typename** studentType>

**void** printStudent(BinaryTree<studentType, **int**>);

**template** <**typename** studentType>

**void** searchStudent(BinaryTree<studentType, **int**>&);

**template** <**typename** studentType>

**void** undoStudent(stack<Undo<studentType>> &stack, BinaryTree<studentType, **int**>& tree);

**template** <**typename** studentType>

**void** addUndo(stack<Undo<studentType>>&, Undo<studentType>);

**template** <**typename** studentType>

**void** writeStudent(File&, BinaryTree<studentType, **int**>&);

**void** readStudents();

**template** <**typename** studentType>

**void** readStudent(File&, BinaryTree<studentType, **int**>&);

**public**:

Menu(){

readStudents();

}

**void** call();

**void** enter();

**void** edit();

**void** remove();

**void** print();

**void** search();

**void** undo();

**void** settings();

**void** write();

~Menu(){

bachelors.clear();

masters.clear();

doctors.clear();

}

};

**template** <**typename** studentType>

**bool** Menu::noStudentsIn(BinaryTree<studentType, **int**> &tree){

**return** tree.empty();

}

**template** <**typename** studentType>

BinaryTree<studentType, **int**> Menu::sortBinaryTree(BinaryTree<studentType, **int**> tree){

vector<string> usedIDs;

BinaryTree<studentType, **int**> sortedTree;

cout << endl <<

"Choose sort way: " << endl <<

"0. Private ID." << endl <<

"1. Group." << endl <<

"2. Scholarship." << endl;

**switch** (ExceptionEnter::NumberData<**int**>(sortMenu)) {

**case** 0:

sortedTree = tree;

**break**;

**case** 1:{

**auto** iterator = tree.begin();

**int** a = 0;

**int** b = 0;

**int** c;

**int** minimalGroup;

**int** previousGroup = 1000000;

**do**{

minimalGroup = 999999;

iterator = tree.begin();

**do**{

c = 0;

**if**((&iterator)->getKey() && stoi((&iterator)->getData().getGroup()) < minimalGroup && stoi((&iterator)->getData().getGroup()) != previousGroup){

**for**(**int** g = 0; g < usedIDs.size(); g++){

**if**((&iterator)->getData().getPrivateID() == usedIDs[g]){

c = 1;

}

}

**if**(!c){

a = 0;

minimalGroup = stoi((&iterator)->getData().getGroup());

}

}

iterator++;

}**while**(iterator != **NULL**);

**if**(a == -1){

**break**;

}

a = 0;

iterator = tree.begin();

**do**{

**if**(stoi((&iterator)->getData().getGroup()) == minimalGroup){

usedIDs.push\_back((&iterator)->getData().getPrivateID());

sortedTree.add((&iterator)->getData(), b);

b++;

}

iterator++;

}**while**(iterator != **NULL**);

a = -1;

previousGroup = minimalGroup;

}**while**(**true**);

**break**;

}

**case** 2:{

**auto** iterator = tree.begin();

**int** a = 0;

**int** b = 0;

**int** c;

**int** minimalScholarship;

**int** previousScholarship = 1000000;

**do**{

minimalScholarship = 999999;

iterator = tree.begin();

**do**{

c = 0;

**if**((&iterator)->getKey() && (&iterator)->getData().getScholarship() < minimalScholarship && (&iterator)->getData().getScholarship() != previousScholarship){

**for**(**int** g = 0; g < usedIDs.size(); g++){

**if**((&iterator)->getData().getPrivateID() == usedIDs[g]){

c = 1;

}

}

**if**(!c){

a = 0;

minimalScholarship = (&iterator)->getData().getScholarship();

}

}

iterator++;

}**while**(iterator != **NULL**);

**if**(a == -1){

**break**;

}

a = 0;

iterator = tree.begin();

**do**{

**if**((&iterator)->getData().getScholarship() == minimalScholarship){

usedIDs.push\_back((&iterator)->getData().getPrivateID());

sortedTree.add((&iterator)->getData(), b);

b++;

}

iterator++;

}**while**(iterator != **NULL**);

a = -1;

previousScholarship = minimalScholarship;

}**while**(**true**);

**break**;

}

}

**return** sortedTree;

}

**template** <**typename** studentType>

studentType Menu::createStudent(){

studentType student;

string privateID;

cout << "Enter private ID ("<< Settings::getLength(DataType::privateID) << " digits): ";

**do**{

privateID = ExceptionEnter::stringData(DataType::privateID);

}**while**(checkPrivateID(privateID));

student.setPrivateID(privateID);

cin >> student;

student.calculateScholarship();

**return** student;

}

**template** <**typename** studentType>

**void** Menu::editStudent(BinaryTree<studentType, **int**> &tree){

**if**(noStudentsIn(tree)){

cout << "No students of this type." << endl;

**return**;

}

printStudent(tree);

**do**{

cout << "Enter student ID: ";

}**while**(!editStudentByID(tree, ExceptionEnter::stringData(privateID)));

}

**template** <**typename** studentType>

**bool** Menu::editStudentByID(BinaryTree<studentType, **int**> &tree, string ID){

Node<studentType, **int**>\* node = tree.findNodeKey(stoi(ID));

**if**(!node){

**return** **false**;

}

studentType \*student = **new** studentType(node->getData());

Doctor\* tempStudent = **static\_cast**<Doctor\*>(student);

**if**(**typeid**(studentType) == **typeid**(Bachelor)){

addUndo(bachelorsUndoStack, Undo<Bachelor>(UndoType::edit, **dynamic\_cast**<Bachelor\*>(tempStudent), stoi(student->getPrivateID())));

}

**if**(**typeid**(studentType) == **typeid**(Master)){

addUndo(mastersUndoStack, Undo<Master>(UndoType::edit, **dynamic\_cast**<Master\*>(tempStudent), stoi(student->getPrivateID())));

}

**if**(**typeid**(studentType) == **typeid**(Doctor)){

addUndo(doctorsUndoStack, Undo<Doctor>(UndoType::edit, tempStudent, stoi(student->getPrivateID())));

}

node->getData().editMenu(**true**);

**if**(node->getData().getPrivateID() != ID){

**if**(checkPrivateID(node->getData().getPrivateID())){

node->getData().setPrivateID(ID);

cout << "This private ID is already exist, returned to the original value." << endl;

}**else**{

tree.add(node->getData(), stoi(node->getData().getPrivateID()));

tree.remove(stoi(ID));

**if**(**typeid**(studentType) == **typeid**(Bachelor)){

bachelorsUndoStack.top().setPrivateID(stoi(node->getData().getPrivateID()));

}

**if**(**typeid**(studentType) == **typeid**(Master)){

mastersUndoStack.top().setPrivateID(stoi(node->getData().getPrivateID()));

}

**if**(**typeid**(studentType) == **typeid**(Doctor)){

doctorsUndoStack.top().setPrivateID(stoi(node->getData().getPrivateID()));

}

}

}

**return** **true**;

}

**template** <**typename** studentType>

**void** Menu::removeStudent(BinaryTree<studentType, **int**> &tree){

**if**(noStudentsIn(tree)){

cout << "No students of this type." << endl;

**return**;

}

printStudent(tree);

cout << endl <<

"1. Choose student to remove." << endl <<

"2. Remove all students of this type. (This action cannot be undone!)" << endl <<

"0. Exit." << endl;

**switch** (ExceptionEnter::NumberData<**int**>(removeMenu)){

**case** 0:

**return**;

**case** 1:

**do**{

cout << "Enter student ID: ";

}**while**(!removeStudentByID(tree, ExceptionEnter::stringData(privateID)));

cout << "This student was removed." << endl;

**break**;

**case** 2:

tree.clear();

cout << "All students of this type was removed." << endl;

**default**:

**break**;

}

}

**template** <**typename** studentType>

**bool** Menu::removeStudentByID(BinaryTree<studentType, **int**> &tree, string ID){

**if**(!tree.checkKey(stoi(ID))){

**return** **false**;

}

studentType \*student = **new** studentType(tree.findKey(stoi(ID)));

Doctor\* tempStudent = **static\_cast**<Doctor\*>(student);

**if**(**typeid**(studentType) == **typeid**(Bachelor)){

addUndo(bachelorsUndoStack, Undo<Bachelor>(UndoType::remove, **dynamic\_cast**<Bachelor\*>(tempStudent)));

}

**if**(**typeid**(studentType) == **typeid**(Master)){

addUndo(mastersUndoStack, Undo<Master>(UndoType::remove, **dynamic\_cast**<Master\*>(tempStudent)));

}

**if**(**typeid**(studentType) == **typeid**(Doctor)){

addUndo(doctorsUndoStack, Undo<Doctor>(UndoType::remove, tempStudent));

}

tree.remove(stoi(ID));

**return** **true**;

}

**template** <**typename** studentType>

**void** Menu::

searchStudent(BinaryTree<studentType, **int**> &students){

**if**(students.empty()){

cout << "No students of this type." << endl;

}

BinaryTree<studentType, **int**> tree;

**do**{

cout << endl <<

"1. Private ID." << endl <<

"2. Group." << endl <<

"3. Name." << endl <<

"4. Scholarship size." << endl <<

"5. Benifits." << endl <<

"0. Exit." << endl;

tree.clear();

**switch** (ExceptionEnter::NumberData<**int**>(searchMenu)){

**case** 0:

**return**;

**case** 1:{

cout << "Enter private ID ("<< Settings::getLength(DataType::privateID) << " digits): ";

string ID = ExceptionEnter::stringData(privateID);

Node<studentType, **int**> \*node = students.findNodeKey(stoi(ID));

**if**(node){

tree.add(node->getData(), stoi(node->getData().getPrivateID()));

}

}

**break**;

**case** 2:{

cout << endl << "Enter group (" << Settings::getLength(DataType::group) << " digits): " << endl;

string group = ExceptionEnter::stringData(DataType::group);

**auto** iterator = students.begin();

**do**{

**if**((&iterator)->getData().getGroup() == group){

tree.add((&iterator)->getData(), stoi((&iterator)->getData().getPrivateID()));

}

iterator++;

}**while**(iterator != **NULL**);

}

**break**;

**case** 3:{

cout << endl << "Enter name (" << Settings::getLength(DataType::name) << " digits): " << endl;

string name = ExceptionEnter::stringData(DataType::name);

**auto** iterator = students.begin();

**do**{

**if**((&iterator)->getData().getName() == name){

tree.add((&iterator)->getData(), stoi((&iterator)->getData().getPrivateID()));

}

iterator++;

}**while**(iterator != **NULL**);

}

**break**;

**case** 4:

cout << endl <<

"1. Integer value." << endl <<

"2. Range." << endl <<

"0. Exit." << endl;

**switch** (ExceptionEnter::NumberData<**int**>(scholarshipMenu)) {

**case** 0:

**return**;

**break**;

**case** 1:{

cout << endl << "Enter value: " << endl;

**int** value = ExceptionEnter::NumberData<**int**>(scholarshipValue);

**auto** iterator = students.begin();

**do**{

**if**(**int**((&iterator)->getData().getScholarship()) == value){

tree.add((&iterator)->getData(), stoi((&iterator)->getData().getPrivateID()));

}

iterator++;

}**while**(iterator != **NULL**);

**break**;

}

**case** 2:

cout << endl << "Enter first value: " << endl;

**int** firstValue = ExceptionEnter::NumberData<**int**>(scholarshipValue);

cout << endl << "Enter second value: " << endl;

**int** secondValue = ExceptionEnter::NumberData<**int**>(scholarshipValue);

**if**(firstValue > secondValue){

**int** tempValue = secondValue;

secondValue = firstValue;

firstValue = tempValue;

}

**auto** iterator = students.begin();

**do**{

**if**(firstValue <= **int**((&iterator)->getData().getScholarship()) && secondValue >= **int**((&iterator)->getData().getScholarship())){

tree.add((&iterator)->getData(), stoi((&iterator)->getData().getPrivateID()));

}

iterator++;

}**while**(iterator != **NULL**);

}

**break**;

**case** 5:

Benefit benefit;

cin >> benefit;

**auto** iterator = students.begin();

**do**{

**int** i = 0;

**while**((&iterator)->getData().getBenefitForID(i) != **NULL**){

**if**(\*(&iterator)->getData().getBenefitForID(i) == benefit){

tree.add((&iterator)->getData(), stoi((&iterator)->getData().getPrivateID()));

}

i++;

}

iterator++;

}**while**(iterator != **NULL**);

**break**;

}

**if**(!tree.empty()){

**this**->printStudent(tree);

}**else**{

cout << endl << "Can't find students by this parameters." << endl;

}

}**while**(**true**);

}

**template** <**typename** studentType>

**void** Menu::undoStudent(stack<Undo<studentType>> &stack, BinaryTree<studentType, **int**>& tree){

**if**(stack.empty()){

cout << "No undo actions." << endl;

}**else**{

stack.top().execute(tree);

stack.pop();

cout << "Action complited." << endl;

}

}

**template** <**typename** studentType>

**void** Menu::addUndo(stack<Undo<studentType>> &stack, Undo<studentType> undo){

stack.push(undo);

}

**template** <**typename** studentType>

**void** Menu::printStudent(BinaryTree<studentType, **int**> tree){

**if**(tree.empty()){

cout << "No students of this type." << endl;

**return**;

}

tree = sortBinaryTree(tree);

studentType::printSpreadsheet(**true**);

cout << endl;

**auto** iterator = tree.begin();

**do**{

cout << (&iterator)->getData() << endl;

(&iterator)->getData().printMarksAndBenefits();

iterator++;

}**while**(iterator != **NULL**);

cout << endl;

}

**template** <**typename** studentType>

**void** Menu::readStudent(File &file, BinaryTree<studentType, **int**> &students){

**while**(!file.endOfFile()){

studentType student;

student.readFromFile(file);

students.add(student, stoi(student.getPrivateID()));

}

}

**template** <**typename** studentType>

**void** Menu::writeStudent(File &file, BinaryTree<studentType, **int**> &tree){

**auto** iterator = tree.begin();

**while**(iterator != **NULL**){

(&iterator)->getData().writeToFile(file);

iterator++;

}

}

#endif /\* Menu\_h \*/

//

// Menu.cpp

// Project

//

// Created by Andrej Hurynovič on 17.12.20.

//

#include "Menu.h"

**int** Menu::chooseStudentType(){

cout << "1. Bachelor." << endl << "2. Master." << endl << "3. Doctor." << endl << "0. Exit." << endl;

**return** ExceptionEnter::NumberData<**int**>(studnetTypes);

}

**bool** Menu::noStudents(){

**if**(noStudentsIn(bachelors) && noStudentsIn(masters) && noStudentsIn(doctors)){

cout << "No students." << endl;

**return** **true**;

}

**return** **false**;

}

**bool** Menu::checkPrivateID(string privateID, **bool** printMessage){

**if**(ExceptionEnter::checkThisID(privateID, bachelors) || ExceptionEnter::checkThisID(privateID, masters) || ExceptionEnter::checkThisID(privateID, doctors)){

**if**(printMessage){

cout << "This private ID is already exist." << endl;

}

**return** **true**;

}

**return** **false**;

}

**void** Menu::call(){

**do**{

cout << endl <<

"1. Add studnet." << endl <<

"2. Edit studnet." << endl <<

"3. Remove students." << endl <<

"4. Print students." << endl <<

"5. Search students." << endl <<

"6. Undo." << endl <<

"7. Settings." << endl <<

"8. Save students." << endl <<

"0. Exit." << endl;

**switch** (ExceptionEnter::NumberData<**int**>(menuNumber)) {

**case** 0:

**return**;

**case** 1:

enter();

**break**;

**case** 2:

edit();

**break**;

**case** 3:

remove();

**break**;

**case** 4:

print();

**break**;

**case** 5:

search();

**break**;

**case** 6:

undo();

**break**;

**case** 7:

settings();

**break**;

**case** 8:

write();

**break**;

}

}**while**(**true**);

}

**void** Menu::enter(){

**switch** (chooseStudentType()) {

**case** 0:{

**return**;

}

**case** 1:{

Bachelor bachelor = createStudent<Bachelor>();

addUndo(bachelorsUndoStack, Undo<Bachelor>(add, **NULL**, stoi(bachelor.getPrivateID())));

bachelors.add(bachelor, stoi(bachelor.getPrivateID()));

**break**;

}

**case** 2:{

Master master(createStudent<Master>());

addUndo(mastersUndoStack, Undo<Master>(add, **NULL**, stoi(master.getPrivateID())));

masters.add(master, stoi(master.getPrivateID()));

**break**;

}

**case** 3:{

Doctor doctor = createStudent<Doctor>();

addUndo(doctorsUndoStack, Undo<Doctor>(add, **NULL**, stoi(doctor.getPrivateID())));

doctors.add(doctor, stoi(doctor.getPrivateID()));

**break**;

}

}

}

**void** Menu::edit(){

**if**(noStudents()){

**return**;

}

**switch** (chooseStudentType()) {

**case** 0:

**return**;

**case** 1:

editStudent(bachelors);

**break**;

**case** 2:

editStudent(masters);

**break**;

**case** 3:

editStudent(doctors);

**break**;

}

}

**void** Menu::remove(){

**if**(noStudents()){

**return**;

}

**switch** (chooseStudentType()){

**case** 0:

**return**;

**case** 1:

removeStudent(bachelors);

**break**;

**case** 2:

removeStudent(masters);

**break**;

**case** 3:

removeStudent(doctors);

**break**;

}

}

**void** Menu::print(){

**if**(noStudents()){

**return**;

}

**switch** (chooseStudentType()) {

**case** 0:

**return**;

**case** 1:

printStudent(bachelors);

**break**;

**case** 2:

printStudent(masters);

**break**;

**case** 3:

printStudent(doctors);

**break**;

}

}

**void** Menu::search(){

**if**(noStudents()){

**return**;

}

**switch** (chooseStudentType()) {

**case** 0:

**return**;

**case** 1:

searchStudent(bachelors);

**break**;

**case** 2:

searchStudent(masters);

**break**;

**case** 3:

searchStudent(doctors);

**break**;

}

}

**void** Menu::undo(){

**if**(bachelorsUndoStack.empty() && mastersUndoStack.empty() && doctorsUndoStack.empty()){

cout << "No undo." << endl;

**return**;

}

**switch** (chooseStudentType()){

**case** 0:

**return**;

**case** 1:

undoStudent(bachelorsUndoStack, bachelors);

**break**;

**case** 2:

undoStudent(mastersUndoStack, masters);

**break**;

**case** 3:

undoStudent(doctorsUndoStack, doctors);

**break**;

}

}

**void** Menu::settings(){

Settings::menu();

**auto** iterator = bachelors.begin();

**while**(iterator != **NULL**){

(&iterator)->getData().calculateScholarship();

iterator++;

};

**auto** iterator2 = masters.begin();

**while**(iterator2 != **NULL**){

(&iterator2)->getData().calculateScholarship();

iterator2++;

};

**auto** iterator3 = doctors.begin();

**while**(iterator3 != **NULL**){

(&iterator3)->getData().calculateScholarship();

iterator3++;

};

}

**void** Menu::readStudents(){

File file(FileType::bachelors, **false**);

readStudent(file, bachelors);

File file2(FileType::masters, **false**);

readStudent(file2, masters);

File file3(FileType::doctors, **false**);

readStudent(file3, doctors);

}

**void** Menu::write(){

File file(FileType::bachelors, **true**);

writeStudent(file, bachelors);

File file2(FileType::masters, **true**);

writeStudent(file2, masters);

File file3(FileType::doctors, **true**);

writeStudent(file3, doctors);

cout << "Students was saved successfully." << endl;

}

//

// Undo.h

// Project

//

// Created by Andrej Hurynovič on 8.01.21.

//

#ifndef Undo\_h

#define Undo\_h

#include "BinaryTree.h"

**enum** UndoType{

add,

edit,

remove,

};

**template** <**typename** studentTypeUndo>

**class** Undo {

**private**:

**enum** UndoType type;

studentTypeUndo \*student;

**int** privateID;

**public**:

Undo(**enum** UndoType type = add, studentTypeUndo \*student = **NULL**, **int** privateID = 0){

**this**->type = type;

**this**->student = student;

**this**->privateID = privateID;

}

**void** execute(BinaryTree<studentTypeUndo, **int**>&);

**enum** UndoType getType();

studentTypeUndo\* getStudent();

**int** getPrivateID();

**void** setType(**enum** UndoType);

**void** setStudent(studentTypeUndo\*);

**void** setPrivateID(**int**);

};

**template** <**typename** studentTypeUndo>

**void** Undo<studentTypeUndo>::execute(BinaryTree<studentTypeUndo, **int**> &tree){

**switch** (type) {

**case** add:

tree.remove(privateID);

**break**;

**case** edit:

tree.remove(privateID);

tree.add(\*student, stoi(student->getPrivateID()));

**break**;

**case** UndoType::remove:

tree.add(\*student, stoi(student->getPrivateID()));

**break**;

}

}

**template** <**typename** studentTypeUndo>

**enum** UndoType Undo<studentTypeUndo>::getType(){

**return** type;

}

**template** <**typename** studentTypeUndo>

studentTypeUndo\* Undo<studentTypeUndo>::getStudent(){

**return** student;

}

**template** <**typename** studentTypeUndo>

**int** Undo<studentTypeUndo>::getPrivateID(){

**return** privateID;

}

**template** <**typename** studentTypeUndo>

**void** Undo<studentTypeUndo>::setType(**enum** UndoType type){

**this**->type = type;

}

**template** <**typename** studentTypeUndo>

**void** Undo<studentTypeUndo>::setStudent(studentTypeUndo \*student){

**this**->student = student;

}

**template** <**typename** studentTypeUndo>

**void** Undo<studentTypeUndo>::setPrivateID(**int** privateID){

**this**->privateID = privateID;

}

#endif /\* Undo\_h \*/

//

// Algorithms.h

// Project

//

// Created by Andrej Hurynovič on 10.01.21.

//

#ifndef Algorithms\_h

#define Algorithms\_h

#include "Bachelor.h"

#include "ExceptionFile.h"

#include "Undo.h"

**class** Algorithms {

**public**:

**static** **void** writeBachelorToFile(Bachelor&, File&);

**static** fstream openFile(**enum** FileType, **bool**);

**static** **float** calclulateBachelorScholarship(Bachelor&, **float**);

**template** <**typename** studentType>

**static** **void** executeUndo(Undo<studentType>&, BinaryTree<studentType, **int**>&);

};

**template** <**typename** studentType>

**void** Algorithms::executeUndo(Undo<studentType>& undo, BinaryTree<studentType, **int**>& tree){

**switch** (undo.getType) {

**case** add:

tree.remove(undo.getPrivateID);

**break**;

**case** edit:

tree.remove(undo.getPrivateID);

tree.add(\*undo.getStudent, stoi(undo.student->getPrivateID()));

**break**;

**case** UndoType::remove:

tree.add(\*undo.getStudent, stoi(undo.student->getPrivateID()));

**break**;

}

}

#endif /\* Algorithms\_h \*/

//

// Algorithms.cpp

// Project

//

// Created by Andrej Hurynovič on 10.01.21.

//

#include "Algorithms.h"

**void** Algorithms::writeBachelorToFile(Bachelor& bachelor, File& file){

bachelor.Human::writeToFile(file);

file.write(bachelor.getGroup());

file.write(bachelor.getScholarship());

**unsigned** **long** marksSize = bachelor.getMarksSize();

file.write(marksSize);

**for**(**int** i = 0; i < marksSize; i++){

file.write(bachelor.getMarkForID(i));

}

**unsigned** **long** benefitsSize = bachelor.getBenefitsSize();

file.write(benefitsSize);

**for**(**int** i = 0; i < benefitsSize; i++){

file.write(bachelor.getBenefitForID(i));

}

}

fstream Algorithms::openFile(**enum** FileType type, **bool** write){

string fileName;

**switch** (type) {

**case** bachelors:

fileName = "Bachelors";

**break**;

**case** masters:

fileName = "Masters";

**break**;

**case** doctors:

fileName = "Doctors";

**break**;

**case** settings:

fileName = "Settings";

**break**;

}

fstream file;

**if**(write){

file.open(fileName, ios::trunc | ios::binary | ios::out);

}**else**{

file.open(fileName, ios::binary | ios::in);

}

**if**(!file.is\_open()){

ofstream tempFile(fileName, ios::binary);

tempFile.close();

**if**(type == settings){

Settings::setTheDefaultSettings();

}

file.open(fileName, ios::binary | ios::in);

**return** file;

}

**return** file;

}

**float** Algorithms::calclulateBachelorScholarship(Bachelor& bachelor, **float** base){

**float** scholarship = base;

**if**(Settings::getMarks()){

**float** marksModifier = 0;

**for** (**int** i = 0; i < bachelor.getMarksSize(); i++){

marksModifier += bachelor.getMarkForID(i)->getMark();

}

marksModifier /= bachelor.getMarksSize();

**if**(marksModifier >= 7){

**if**(marksModifier >= 7 && marksModifier < 8){

marksModifier = Settings::getMarkFirst();

}

**if**(marksModifier >= 8 && marksModifier < 9){

marksModifier = Settings::getMarkSecond();

}

**if**(marksModifier >= 9 && marksModifier <= 10){

marksModifier = Settings::getMarkThird();

}

scholarship += ((base \* marksModifier) - base);

}

}

**if**(Settings::getResit()){

**for** (**int** i = 0; i < bachelor.getMarksSize(); i++){

**if**(bachelor.getMarkForID(i)->getResit()){

scholarship = 0;

**return** scholarship;

}

}

}

**if**(Settings::getBenefits()){

**for**(**int** i = 0; i < bachelor.getBenefitsSize(); i++){

scholarship += base \* bachelor.getBenefitForID(i)->getModifier() - base;

}

}

**return** scholarship;

}

//

// BinaryTree.h

// BinaryTreeIterator Test

//

// Created by Andrej Hurynovič on 7.01.21.

//

#ifndef BinaryTree\_h

#define BinaryTree\_h

#include "BinaryTreeBinaryTreeIterator.h"

**template** <**typename** DataType, **typename** KeyDataType>

**class** BinaryTree {

**private**:

Node<DataType, KeyDataType>\* root;

BinaryTreeIterator<DataType, KeyDataType> beginConst;

BinaryTreeIterator<DataType, KeyDataType> endConst;

**public**:

BinaryTree(Node<DataType, KeyDataType>\* root = **NULL**){

**this**->root = root;

**this**->beginConst = root;

**this**->endConst = root;

}

BinaryTree(DataType data, KeyDataType key){

**this**->root = **new** Node<DataType, KeyDataType>(data, key);

**this**->beginConst = root;

**this**->endConst = root;

}

**bool** empty();

**bool** checkKey(KeyDataType);

Node<DataType, KeyDataType>\* findNodeKey(**int**);

DataType& findKey(**int**);

**int** size();

**void** add(DataType , KeyDataType);

**void** remove(KeyDataType);

**void** clear();

**void** print();

DataType& **operator** [] (**int**);

BinaryTreeIterator<DataType, KeyDataType> begin();

BinaryTreeIterator<DataType, KeyDataType> end();

};

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTree<DataType, KeyDataType>::empty(){

**if**(!root){

**return** **true**;

}**else**{

**return** **false**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTree<DataType, KeyDataType>::checkKey(KeyDataType key){

BinaryTreeIterator<DataType, KeyDataType> iterator = begin();

**while**(iterator != **NULL**){

**if**((&iterator)->getKey() == key){

**return** **true**;

}

iterator++;

}

**return** **false**;

}

**template** <**typename** DataType, **typename** KeyDataType>

Node<DataType, KeyDataType>\* BinaryTree<DataType, KeyDataType>::findNodeKey(**int** key){

**if**(empty()){

**return** **NULL**;

}

**return** root->find(key);

}

**template** <**typename** DataType, **typename** KeyDataType>

DataType& BinaryTree<DataType, KeyDataType>::findKey(**int** key){

**return** root->find(key)->getData();

}

**template** <**typename** DataType, **typename** KeyDataType>

**int** BinaryTree<DataType, KeyDataType>::size(){

**int** counter = 0;

BinaryTreeIterator<DataType, KeyDataType> iterator = begin();

**while**(iterator != **NULL**){

iterator++;

counter++;

}

**return** counter;

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** BinaryTree<DataType, KeyDataType>::add(DataType data, KeyDataType key){

**if**(checkKey(key)){

**return**;

}

Node<DataType, KeyDataType> \*node;

node = **new** Node<DataType, KeyDataType>(data, key);

**if**(empty()){

root = node;

beginConst = root;

endConst = root;

}**else**{

root->add(node);

**if**(beginConst > node->getKey()){

beginConst = node;

}

**if**(endConst < node->getKey()){

endConst = node;

}

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** BinaryTree<DataType, KeyDataType>::remove(KeyDataType key){

**if**(key == (&beginConst)->getKey()){

**if**((&beginConst)->getNext((&beginConst)->getKey()) == **NULL**){

beginConst = **NULL**;

}**else**{

beginConst = (&beginConst)->getNext((&beginConst)->getKey());

}

}

root->deleteKey(&root, key);

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** BinaryTree<DataType, KeyDataType>::clear(){

**if**(root){

root->deleteAll();

root = **NULL**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** BinaryTree<DataType, KeyDataType>::print(){

BinaryTreeIterator<DataType, KeyDataType> iterator = begin();

**while**(iterator != **NULL**){

cout << (&iterator) << endl;

iterator++;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

DataType& BinaryTree<DataType, KeyDataType>::**operator** [] (**int** temp){

**if**(temp > size()){

**return** \*(begin() + (size() - 1));

}

**return** \*(begin() + temp);

}

**template** <**typename** DataType, **typename** KeyDataType>

BinaryTreeIterator<DataType, KeyDataType> BinaryTree<DataType, KeyDataType>::begin(){

**return** beginConst;

}

**template** <**typename** DataType, **typename** KeyDataType>

BinaryTreeIterator<DataType, KeyDataType> BinaryTree<DataType, KeyDataType>::end(){

**return** endConst;

}

#endif /\* BinaryTree\_h \*/

//

// BinaryTreeIterator.h

// LabWork 8. BinaryTreeIterators

//

// Created by Andrej Hurynovič on 14.12.20.

//

#include "Node.h"

#ifndef BinaryTreeIterator\_h

#define BinaryTreeIterator\_h

**template** <**typename** DataType, **typename** KeyDataType>

**class** BinaryTreeIterator{

**private**:

Node<DataType, KeyDataType>\* current;

**public**:

BinaryTreeIterator(Node<DataType, KeyDataType>\* current = **NULL**){

**this**->current = current;

}

DataType& **operator** \* ();

Node<DataType, KeyDataType>\* **operator** & ();

**void** **operator** = (Node<DataType, KeyDataType>\* current);

**void** **operator** ++ (**int**);

BinaryTreeIterator& **operator** + (**int** temp);

**void** **operator** -- (**int**);

BinaryTreeIterator& **operator** - (**int** temp);

**bool** **operator** > (KeyDataType key);

**bool** **operator** >= (KeyDataType key);

**bool** **operator** < (KeyDataType key);

**bool** **operator** <= (KeyDataType key);

**bool** **operator** == (**const** BinaryTreeIterator& iterator);

**bool** **operator** != (**const** BinaryTreeIterator& iterator);

};

**template** <**typename** DataType, **typename** KeyDataType>

**void** BinaryTreeIterator<DataType, KeyDataType>::**operator** = (Node<DataType, KeyDataType>\* current) {

**this**->current = current;

}

**template** <**typename** DataType, **typename** KeyDataType>

DataType& BinaryTreeIterator<DataType, KeyDataType>::**operator** \* () {

**return** (current->getData());

}

**template** <**typename** DataType, **typename** KeyDataType>

Node<DataType, KeyDataType>\* BinaryTreeIterator<DataType, KeyDataType>::**operator** & (){

**return** current;

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** BinaryTreeIterator<DataType, KeyDataType>::**operator** ++ (**int**){

**this**->current = current->getNext(current->getKey());

}

**template** <**typename** DataType, **typename** KeyDataType>

BinaryTreeIterator<DataType, KeyDataType>& BinaryTreeIterator<DataType, KeyDataType>::**operator** + (**int** temp){

**do**{

(\***this**)++;

temp--;

}**while**(temp && \***this** != **NULL**);

**return** \***this**;

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTreeIterator<DataType, KeyDataType>::**operator** > (KeyDataType key){

**if**(current->getKey() > key){

**return** **true**;

}**else**{

**return** **false**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTreeIterator<DataType, KeyDataType>::**operator** >= (KeyDataType key){

**if**(current->getKey() >= key){

**return** **true**;

}**else**{

**return** **false**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTreeIterator<DataType, KeyDataType>::**operator** < (KeyDataType key){

**if**(current->getKey() < key){

**return** **true**;

}**else**{

**return** **false**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTreeIterator<DataType, KeyDataType>::**operator** <= (KeyDataType key){

**if**(current->getKey() <= key){

**return** **true**;

}**else**{

**return** **false**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTreeIterator<DataType, KeyDataType>::**operator** == (**const** BinaryTreeIterator& iterator) {

**if**(**this**->current == iterator.current){

**return** **true**;

}**else**{

**return** **false**;

}

}

**template** <**typename** DataType, **typename** KeyDataType>

**bool** BinaryTreeIterator<DataType, KeyDataType>::**operator** != (**const** BinaryTreeIterator& iterator) {

**if**(**this**->current != iterator.current){

**return** **true**;

}**else**{

**return** **false**;

}

}

#endif /\* BinaryTreeIterator\_h \*/

//

// Node.h

// BinaryTreeIterator Test

//

// Created by Andrej Hurynovič on 7.01.21.

//

#ifndef Node\_h

#define Node\_h

**template** <**typename** DataType, **typename** KeyDataType>

**class** Node {

**public**:

DataType data;

KeyDataType key;

Node\* left;

Node\* right;

Node\* parent;

**public**:

Node(DataType data, KeyDataType key){

**this**->data = data;

**this**->key = key;

**this**->left = **NULL**;

**this**->right = **NULL**;

**this**->parent = **NULL**;

}

**void** add(Node\*);

Node\* find(**int**);

Node\* getNext(KeyDataType);

**void** deleteKey(Node\*\* root, KeyDataType);

Node\* supportDeleteFunction(Node\*);

**void** deleteAll();

DataType& getData();

KeyDataType getKey();

**void** setData(DataType);

**void** setKey(KeyDataType);

};

**template** <**typename** DataType, **typename** KeyDataType>

**void** Node<DataType, KeyDataType>::add(Node\* node){

**if**(node->key < **this**->key){

**if**(**this**->left){

**this**->left->add(node);

}**else**{

node->parent = **this**;

**this**->left = node;

}

}

**if**(node->key > **this**->key){

**if**(**this**->right){

**this**->right->add(node);

}**else**{

node->parent = **this**;

**this**->right = node;

}

}

}

**template** <**typename** DataType, **typename** KeyDataType>

Node<DataType, KeyDataType>\* Node<DataType, KeyDataType>::find(**int** key){

**if**(**this**->key == key){

**return** **this**;

}

**if**(key < **this**->key){

**if**(left){

**return** left->find(key);

}

}

**if**(key > **this**->key){

**if**(right){

**return** right->find(key);

}

}

**return** **NULL**;

}

**template** <**typename** DataType, **typename** KeyDataType>

Node<DataType, KeyDataType>\* Node<DataType, KeyDataType>::getNext(KeyDataType key){

**if**(!parent && !right && !left){

**return** **NULL**;

}

**if**(left && right){

**if**(left->key > key){

**return** left->getNext(key);

}**else**{

**if**(**this**->key <= key){

**if**(right->key > key){

**return** right->getNext(key);

}

}

}

}

**if**(left && left->key > key){

**return** left->getNext(key);

}

**if**(right){

**if**(**this**->key > key){

**return** **this**;

}

**if**(right->key > key){

**return** right->getNext(key);

}**else**{

**if**(parent){

**return** parent->getNext(key);

}**else**{

**return** **NULL**;

}

}

}

**if**(**this**->key > key){

**return** **this**;

}

**if**(parent){

Node\* temp = parent->getNext(key);

**if**(temp && temp->getKey() > key){

**return** temp;

}**else**{

**return** **NULL**;

}

}

**return** **NULL**;

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** Node<DataType, KeyDataType>::deleteKey(Node\*\* root, KeyDataType key){

**if**(\*root){

**if**((\*root)->key == key){

**if**(key == 0){

}

**if**(!((\*root)->parent) && !((\*root)->left) && !((\*root)->right)){

**delete** (\*root);

(\*root) = **NULL**;

**return**;

}

**if**(!(\*root)->left && !(\*root)->right){

**if**((\*root)->parent->left == (\*root)){

(\*root)->parent->left = **NULL**;

**delete** (\*root);

**return**;

}

**if**((\*root)->parent->right == (\*root)){

(\*root)->parent->right = **NULL**;

**delete** (\*root);

**return**;

}

}

**if**(!(\*root)->left && (\*root)->right){

Node\* tempNode = (\*root)->right;

(\*root)->right->parent = (\*root)->parent;

**delete** (\*root);

(\*root) = tempNode;

**return**;

}

**if**((\*root)->left && !(\*root)->right){

(\*root)->left->parent = (\*root)->parent;

Node\* tempNode = (\*root)->left;

**delete** (\*root);

(\*root) = tempNode;

**return**;

}

**if**((\*root)->left && (\*root)->right){

Node\* tempNode = supportDeleteFunction((\*root)->right);

Node\* temp = tempNode->parent;

**if**(!temp){

temp = \*root;

}

**if**(!tempNode->left && !tempNode->right){

**if**(temp->left && temp->left == tempNode){

**if**(temp->left->key == tempNode->key){

temp->left = **NULL**;

}

}

**if**(temp->right && temp->right == tempNode){

**if**(temp->right->key == tempNode->key){

temp->right = **NULL**;

}

}

**if**((\*root)->left && (\*root)->left != tempNode){

tempNode->left = (\*root)->left;

(\*root)->left->parent = tempNode;

}

**if**((\*root)->right && (\*root)->right != tempNode){

tempNode->right = (\*root)->right;

(\*root)->right->parent = tempNode;

}

tempNode->parent = (\*root)->parent;

**delete** (\*root);

(\*root) = tempNode;

}

**if**(!tempNode->left && tempNode->right){

**if**(temp->left){

**if**(temp->left->key == tempNode->key){

tempNode->right->parent = temp;

temp->left = tempNode->right;

}

}

**if**(temp->right){

**if**(temp->right->key == tempNode->key){

tempNode->right->parent = temp;

temp->right = tempNode->right;

}

}

**if**((\*root)->left != tempNode){

tempNode->left = (\*root)->left;

(\*root)->left->parent = tempNode;

}

**if**((\*root)->right != tempNode){

tempNode->right = (\*root)->right;

(\*root)->right->parent = tempNode;

}

tempNode->parent = (\*root)->parent;

**delete** (\*root);

(\*root) = tempNode;

}

}

}**else**{

**if**(key < (\*root)->key){

deleteKey(&(\*root)->left, key);

}**else**{

deleteKey(&(\*root)->right, key);

}

}

}

}

**template** <**typename** DataType, **typename** KeyDataType>

Node<DataType, KeyDataType>\* Node<DataType, KeyDataType>::supportDeleteFunction(Node\* root){

**if**((!root->left && root->right) || (!root->left && !root->right)){

**return** root;

}

**if**(root->left){

**return** supportDeleteFunction(root->left);

}

**return** **NULL**;

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** Node<DataType, KeyDataType>::deleteAll(){

**if**(left){

left->deleteAll();

left = **NULL**;

}

**if**(right){

right->deleteAll();

right = **NULL**;

}

**delete** **this**;

}

**template** <**typename** DataType, **typename** KeyDataType>

DataType& Node<DataType, KeyDataType>::getData(){

**return** data;

}

**template** <**typename** DataType, **typename** KeyDataType>

KeyDataType Node<DataType, KeyDataType>::getKey(){

**return** key;

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** Node<DataType, KeyDataType>::setData(DataType data){

**this**->data = data;

}

**template** <**typename** DataType, **typename** KeyDataType>

**void** Node<DataType, KeyDataType>::setKey(KeyDataType key){

**this**->key = key;

}

#endif /\* Node\_h \*/