МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ “ХПІ”

Кафедра “Обчислювальна техніка та програмування”

Розрахункове завдання з дисципліни

«Програмування ч.2»

Пояснювальна записка

ЄСПД ГОСТ 19.404-79(СТЗВО-ХПІ-2.01-2018 ССОНП)

Розробники:

Виконав: студент групи КІТ-120Д

Руденко О. В

Перевірив: Пасько Д. А.

Харків 2021

ЗАТВЕРДЖЕНО

ЄСПД ГОСТ 19.404-79(СТЗВО-ХПІ-2.01-2018 ССОНП)

Розрахункове завдання з дисципліни

«Програмуваня ч.2»

Пояснювальна записка

Листів 52

ЄСПД ГОСТ 19.404-79(СТЗВО-ХПІ-2.01-2018 ССОНП)

Харків 2021

Вступ

Тема роботи:

Розробка інформаційно-довідкової системи.

Мета роботи:

Закріпити отримані знання з дисципліни «Програмування ч.2» шляхом використання типового комплексного завдання.

1. Призначення та галузь застосування;
2. Постановка завдання до розробки;
3. Опис вхідних та вихідних даних;
4. Опис складу технічних та програмних засобів;
5. Список джерел інформації;
6. Додаток, який складається з розробленого коду.

Призначення та галузь застосування

Об'єктно-орієнтоване програмування — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм.

Постановка завдання до розробки

При виконанні завдання з розробки інформаційно-довідкової системи необхідно виконати наступне:

1. З розділу «Розрахункове завдання/Індивідуальне завдання», відповідно до варіанта завдання, обрати прикладну галузь(варіант 3).
2. Для прикладної галузі розробити розгалужену ієрархію класів, що описана у завданні та складається з одного базового класу та двох спадкоємців. Клас повинен мати перевантажені оператори.
3. Розробити клас-список, що буде включати до себе масив вказівників до базового класу. А також базові методи роботи з списком: а) очистка списку б) відображення списку в) додавання/видалення/оновлення.
4. Розробити клас-контролер, що буде включати колекцію розроблених класів, та наступні методи роботи з колекцією: а) читання даних з файлу б) запис даних у файл.
5. Розробити клас меню, який має відображати діалогове меню для демонстрації реалізованих функцій класу контролера.
6. Виконати перевірку вхідних даних за допомогою регулярних виразів.
7. Оформити документацію: пояснювальну записку.

Опис вхідних та вихідних даних

Вхідні дані:

1. Бюджет (bool budget).
2. Прізвище студента (string nameStudent).
3. Прізвище куратора (string nameCurator).
4. Рік вступу (int year).
5. Група (string group).
6. Корпус кафедри (int departmentBuilding).

Вихідні дані:

1. Список даних про студента (Базовий клас).
2. Список даних про студента Магістратури. (Спадкоємець 1).
3. Список даних про студента Бакалаврату (Спадкоємець 2).

Опис складу технічних та програмних засобів

1. Створив базовий клас та двох спадкоємців відповідно до свого індивідуального завдання.
2. Створив функції читання даних з файлу та запис даних у файл .
3. Створив функції додавання та видалення елементу за вибором користувача.
4. Створив функцію меню для зручного користування програмою.
5. Створив функцію пошуку учня з страховкою.
6. Створив функцію зміни інформації про учня.
7. Створив функцію сортування учнів по класу (Зростанню і зменшенням).
8. Виконав перевірку вхідних даних за допомогою регулярних виразів.
9. Виконав перевантаження операторів.
10. Код программи.

Результат роботи програми

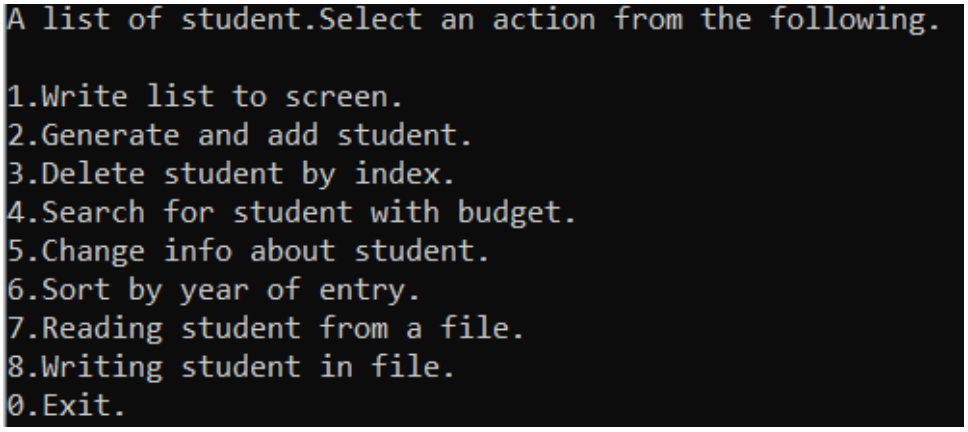

Рисунок 1 – Головне меню.

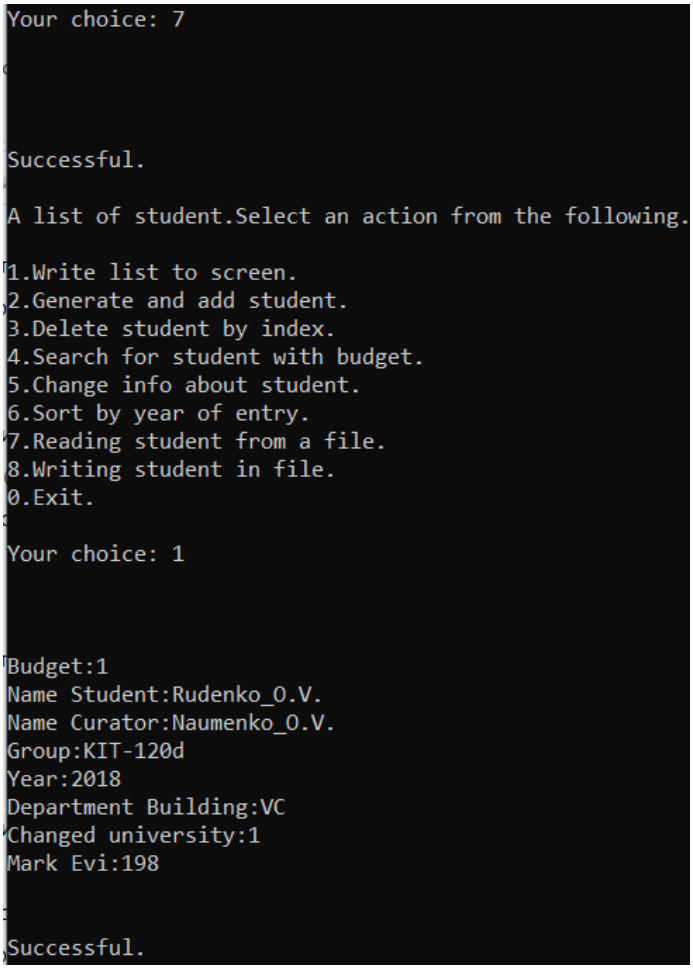

Рисунок 2 – Читання з файлу.

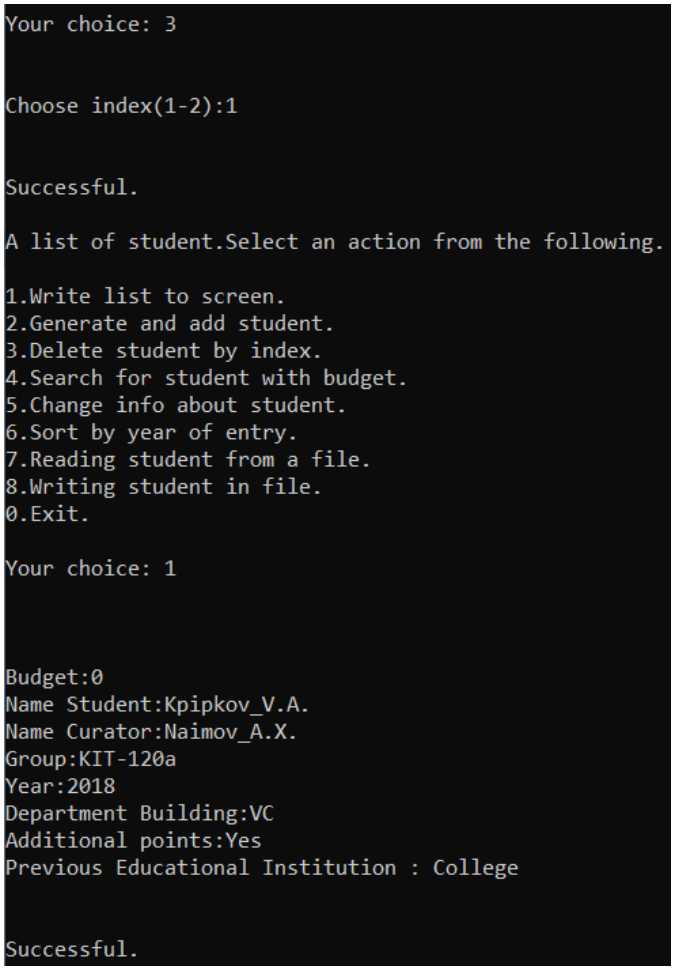

Рисунок 3.1 – Видалення з файлу.

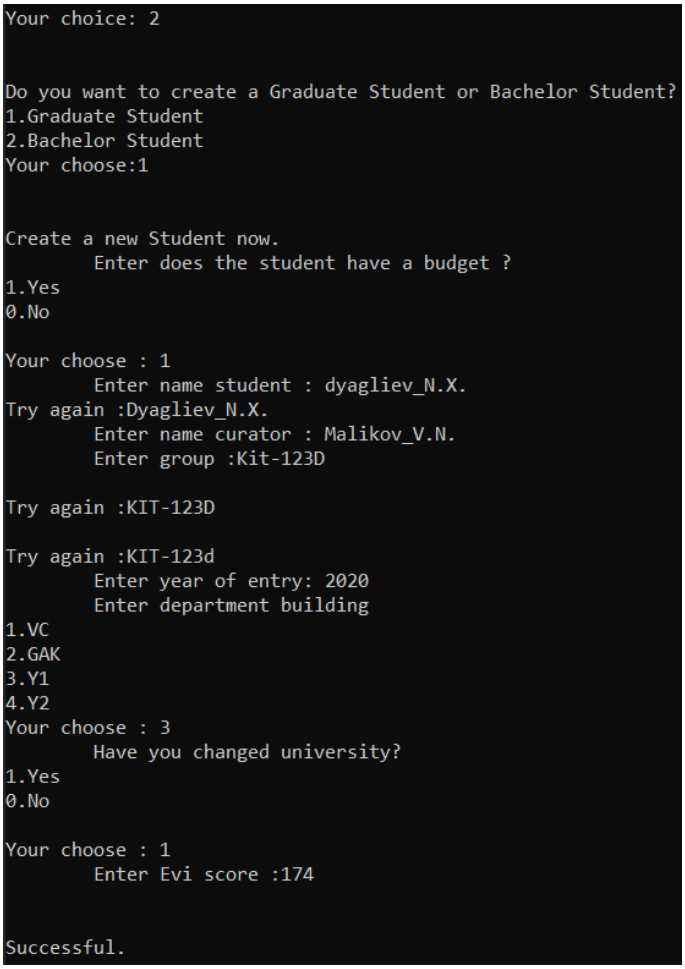

Рисунок 3.2 – Додавання елементу.

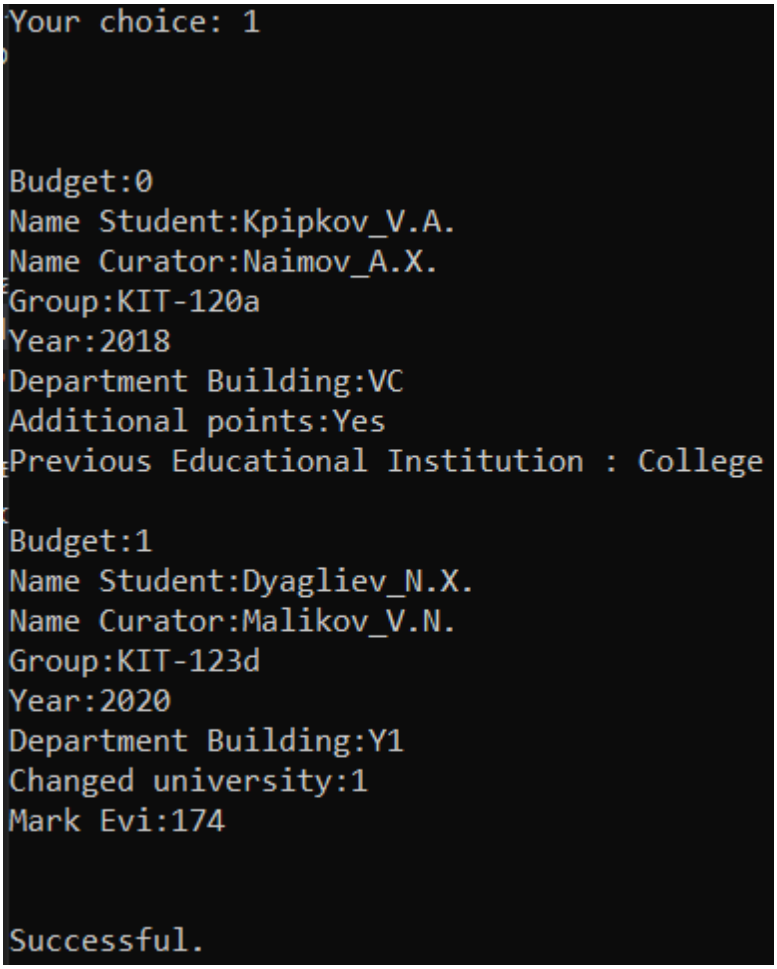

Рисунок 3.3 – Список після додавання елементу.

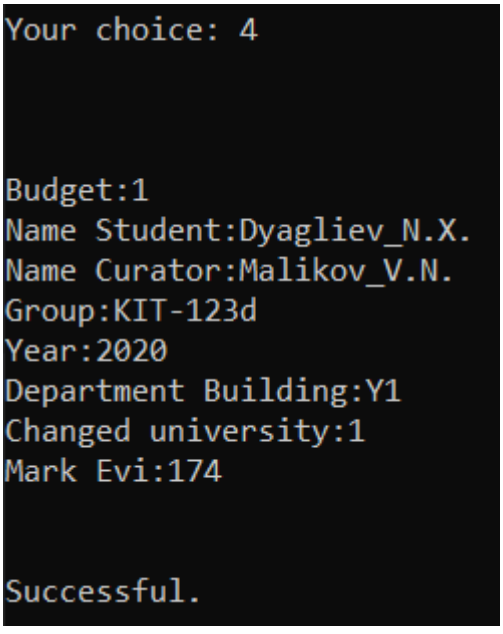

Рисунок 4 – Пошук учня з бюджетною формою.

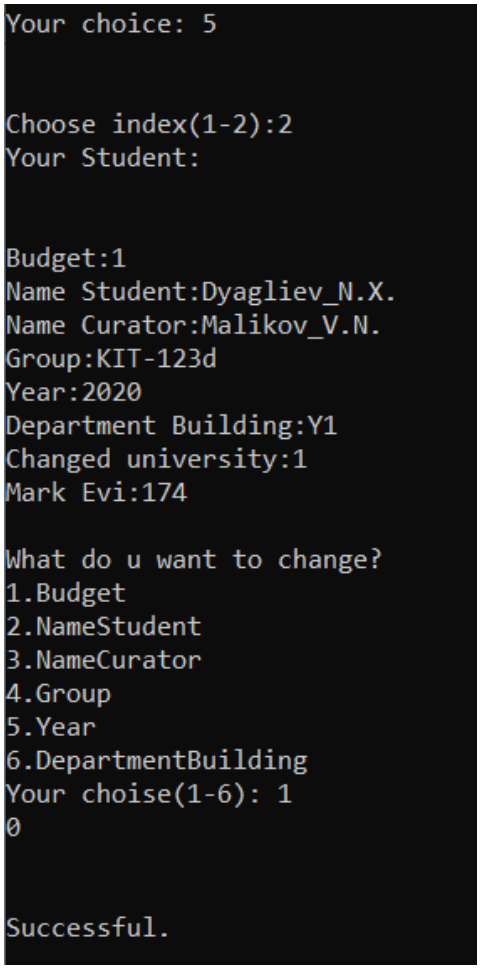

Рисунок 5.1 – Функція для змінення інформації про студента.

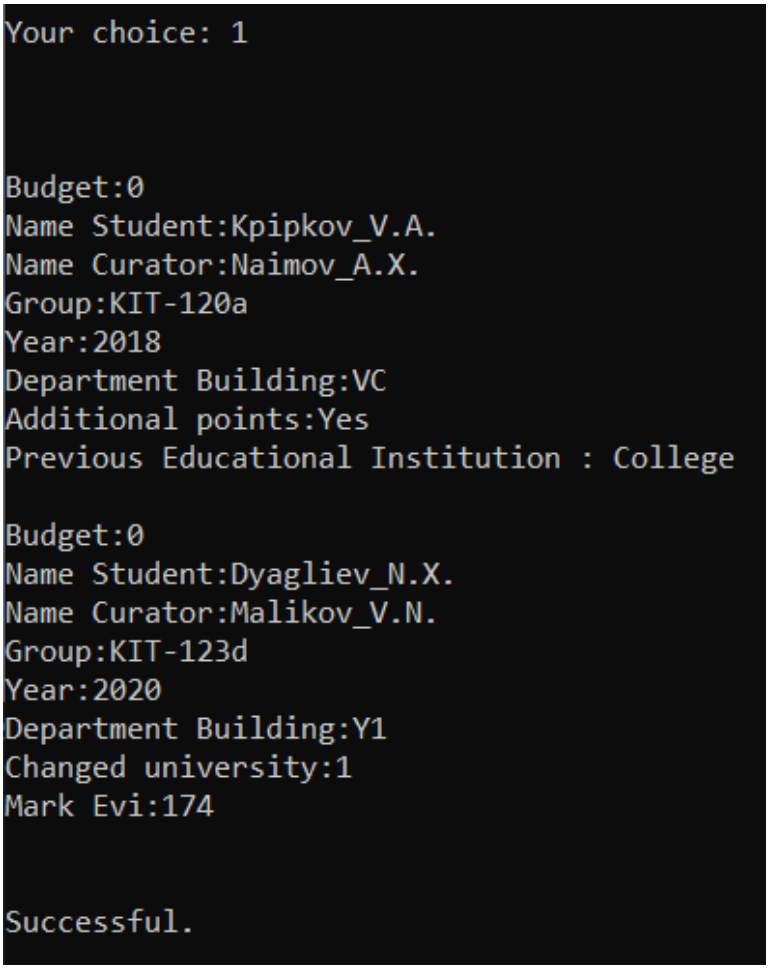

Рисунок 5.2 – Список після змінення інформації про студента.

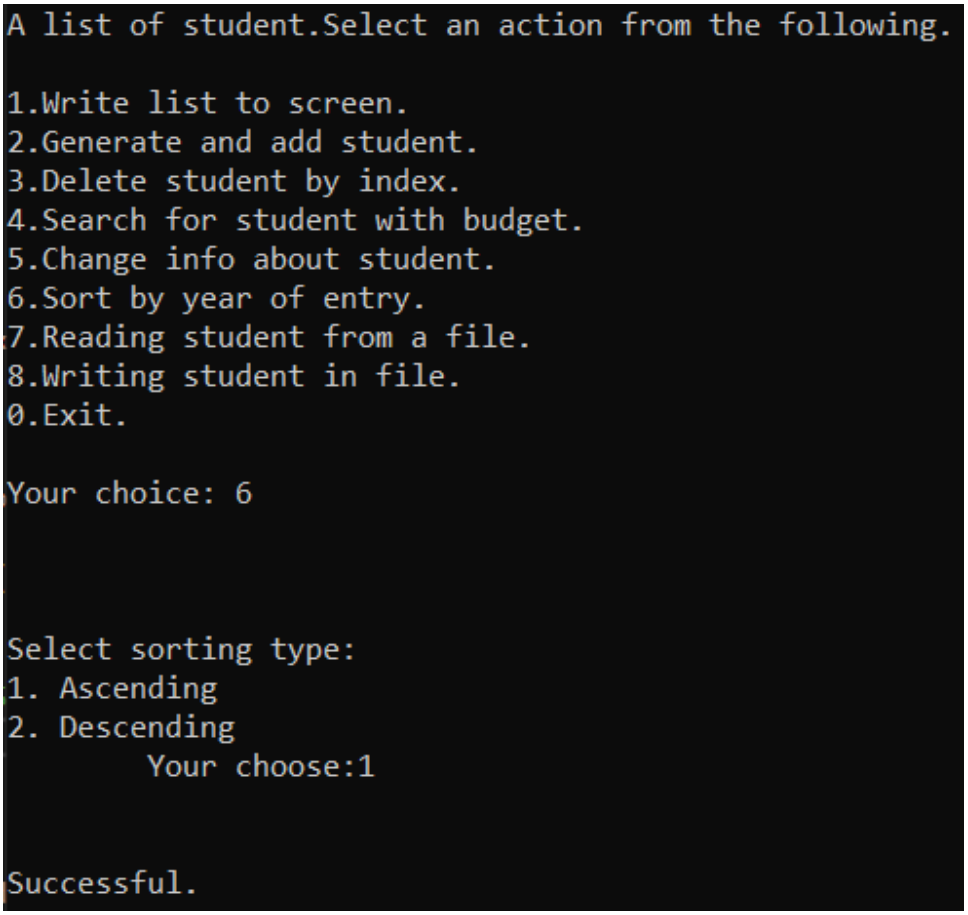

Рисунок 6.1 – Функція для сортування студентів за роком вступу(Зростання).

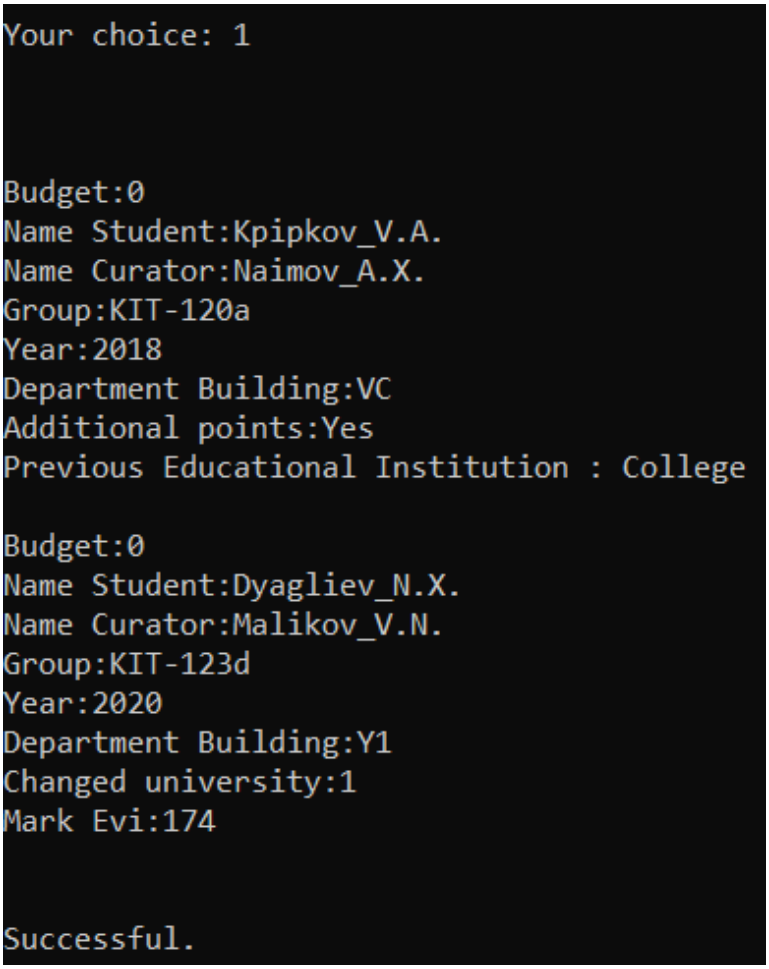

Рисунок 6.2 – Список після сортування студентів за роком вступу (Зростання).

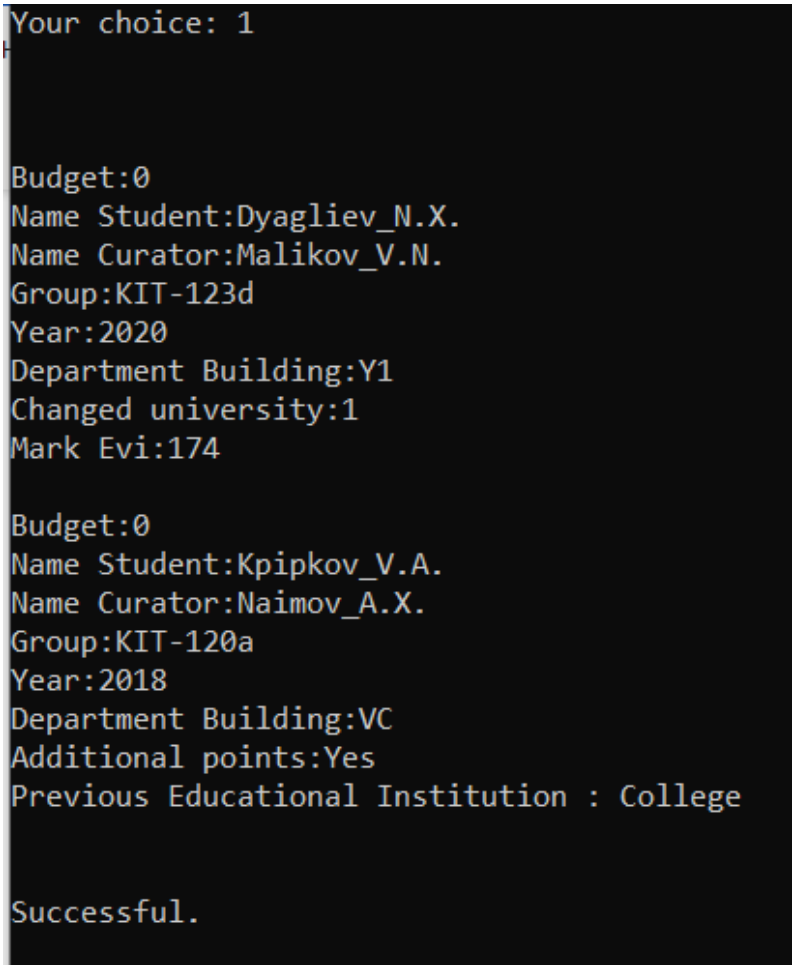

Рисунок 6.3 – Список після сортування студентів за роком вступу (Убування).

Висновок

Закріпив отримані знання з дисципліни «Програмування ч.2» шляхом використання типового комплексного завдання.

Список джерел інформації

1. Ашарина, И.В. Объектно-ориентированное программирование в C++: лекции и упражнения: Учебное пособие для вузов / И.В. Ашарина. - М.: РиС, 2015. - 336 с.
2. Ашарина, И.В. Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум: Учебное пособие для вузов / И.В. Ашарина, Ж.Ф. Крупская. - М.: ГЛТ , 2015. - 232 с.
3. Лафоре, Р. Объектно-ориентированное программирование в C++. Классика Computer Science / Р. Лафоре. - СПб.: Питер, 2013. - 928
4. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. - СПб.: Питер, 2018. - 928 с.

Код програми

У файлі main.cpp

```
#include "Helper.h"

int main() {
List list;
Helper helper;
helper.menu(list);

}
```

У файлі Helper.cpp

```
#include "Helper.h"



void Helper::menu(List& list) {
cout << "\n\nA list of student.Select an action from the following.\n\n";
cout << "1.Write list to screen.\n";
cout << "2.Generate and add student.\n";
cout << "3.Delete student by index.\n";
cout << "4.Search for student with budget.\n";
cout << "5.Change info about student.\n";
cout << "6.Sort by year of entry.\n";
cout << "7.Reading student from a file.\n";
cout << "8.Writing student in file.\n";
cout << "0.Exit.\n\n";
```

```
cout << "Your choice: ";

int choice = -1;

while (choice < 0 || choice>9) {
cin >> choice;
if (choice < 0 || choice>9)
cout << "You entered an invalid value, please retry\n\nYour choice: ";
}
cout << endl << endl;

switch (choice) {
case 0:
list.clear();
return;
case 1:
list.showAll();
break;
case 2:
list.addStudent();
break;
case 3:
choice = 0;
cout<<"Choose index(1-" << list.getSize() << "):";
while (choice <= 0 || choice > list.getSize()) {
cin >> choice;
if (choice < 0 || choice > list.getSize())
```

```
cout << "You entered an index that is larger than the size of the list, please retry\n\nYour choice: ";
}
list.removeStudent(choice - 1);
break;
case 4:
cout << list.getStudentBudget(10).print();
break;
case 5:
choice = 0;
cout << "Choose index(1-" << list.getSize() << "):";
while (choice <= 0 || choice > list.getSize()) {
cin >> choice;
if (choice < 0 || choice > list.getSize())
cout << "You entered an index that is larger than the size of the list, please retry\n\nYour choice: ";
}
list.setStudent(choice - 1);
break;
case 6:
list.sortYearOfEntry();
break;
case 7:
list.readFromFile();
break;
case 8:
list.writeToFile();
break;
```

```
}
cout << "\n\nSuccessful.";
menu(list);
}
```

У файлі Helper.h

```
#pragma once
#include "List.h"

class Helper
{
public:
void menu(List& list);
};
```

У файлі List.cpp

```
#include "List.h"

void List::addStudent() {
cout << "Do you want to create a Graduate Student or Bachelor Student?
\n1.Graduate Student\n2.Bachelor Student\nYour choose:";
int choose = chooseInt(1, 2);
Student* p = NULL;
int digit;
getchar;

if (choose == 1) {
GraduateStudent tmp;
```

```
cout << "\n\nCreate a new Student now.\n    Enter does the student have a budget ? \n1.Yes\n0.No\n\nYour choose : ";


tmp.setBudget(chooseInt(0, 1));
cout << "      Enter name student : ";
tmp.setNameStudent(check(regName));
cout << "      Enter name curator : ";
tmp.setNameCurator(check(regName));
cout << "      Enter group :";
tmp.setGroup(check(regGroup));
cout << "      Enter year of entry: ";
tmp.setYearOfEntry(chooseInt(2000, 2020));
cout << "      Enter department building \n1.VC\n2.GAK\n3.Y1\n4.Y2\nYour choose : ";
tmp.setDepartmentBuilding(chooseInt(1, 4));
cout << "      Have you changed university? \n1.Yes\n0.No\n\nYour choose : ";
tmp.setChangeUniversity(chooseInt(0, 1));
cout << "      Enter Evi score :";
tmp.setMarkEvi(chooseInt(1, 200));
p = &tmp;


}
else if (choose == 2) {
BachelorStudent tmp;
cout << "\n\nCreate a new Student now.\n    Enter does the student have a budget ? \n1.Yes\n0.No\n\nYour choose : ";


tmp.setBudget(chooseInt(0, 1));
```

```
cout << "     Enter name student : ";
tmp.setNameStudent(check(regName));
cout << "     Enter name curator : ";
tmp.setNameCurator(check(regName));
cout << "     Enter group :";
tmp.setGroup(check(regGroup));
cout << "     Enter year of entry: ";
tmp.setYearOfEntry(chooseInt(2000, 2020));
cout << "     Enter department building \n1.VC\n2.GAK\n3.Y1\n4.Y2\nYour choose : ";
tmp.setDepartmentBuilding(chooseInt(1, 4));
cout << "     Do you have additional points? \n1.Yes\n0.No\nYour choose :";
tmp.setAdditionalPoints(chooseInt(0, 1));
cout << "     Sports activities\n1.School\n2.Technical school\n3.College\n\nYour choose :";
tmp.setPreviousEducationalInstitution(chooseInt(1, 3));

p = &tmp;
}
list.push_back       (p);
}
void ListofGraduateStudent::get2018YearOfEntry() {
for (GraduateStudent& emp : list) {
if (emp.getYearOfEntry() == 2018)
cout << emp.print() << endl;
}
}
void ListofBachelorStudent::get2018YearOfEntry() {
```

```
for (BachelorStudent& emp : list) {
if (emp.getYearOfEntry() == 2018)
cout << emp.print() << endl;
}
}
void ListofBachelorStudent::getWasCollege() {
for (BachelorStudent& emp : list) {
if (emp.getPreviousEducationalInstitution() == 3)
cout << emp.print() << endl;
}
}

void ListofGraduateStudent::getWasAtAnotherUniversity() {
for (GraduateStudent& emp : list) {
if (emp.getChangeUniversity() == 1)
cout << emp.print() << endl;
}
}
void List::removeStudent(const int index) {
if (!list.empty())
delete* (list.begin() + index);
list.erase(list.begin() + index);
}
void List::showAll() {
for (int i = 0; i < list.size(); i++) {

cout << list[i]->print() << endl;
```

```
}
}
void List::setStudent(const int index) {
int choice = 0, type = 0;
type = list[index]->whoIAm();

cout << "Your Student:\n\n";
cout << list[index]->print() << endl;
cout << "\nWhat do u want to
change?\n1.Budget\n2.NameStudent\n3.NameCurator\n4.Group\n5.Year\n6.De
partmentBuilding\n";

cout << "Your choise(1-6): ";
choice = chooseInt(1, 6);

switch (choice)
{
case 1:
list[index]->setBudget(chooseInt(0, 1));
break;
case 2:
list[index]->setNameStudent(check(regName));
break;
case 3:
list[index]->setNameCurator(check(regName));
break;
case 4:
list[index]->setGroup(check(regGroup));
```

```
break;
case 5:
list[index]->setYearOfEntry(chooseInt(2000, 2020));
break;
case 6:
list[index]->setDepartmentBuilding(!list[index]->getDepartmentBuilding());
break;

}
}
void List::clear() {
if (!list.empty()) {
for (int i = list.size(); i > 0; i--) {
removeStudent(i - 1);
}
}
}
Student& List::getStudent(const int index) {
return *(list[index]);
}
Student& List::getStudentYearOfEntry(const int yearOfEntry) {
for (int i = 0; i < list.size(); i++) {
if (list[i]->getYearOfEntry() == yearOfEntry)
return *(list[i]);
}
}
void List::sortYearOfEntry() {
```

```
cout << "\nSelect sorting type: \n1. Ascending \n2. Descending \n      Your choose:";
int type = chooseInt(1, 2);

if (type == 1) {
std::sort(list.begin(), list.end());
}
else {
std::sort(list.begin(), list.end());
std::reverse(list.begin(), list.end());
}
}
int List::getSize() {
return list.size();
}
Student& List::operator[](const int index) {
return *list[index];
}

string List::check(regex reg) {
string word;
do {
getline(cin, word);
if (!std::regex_match(word, reg))
cout << "\nTry again :";
} while (!std::regex_match(word, reg));
return word;
}
```

```
int List::chooseInt(const int start, const int end) {
int digit = 0;
do {
cin >> digit;
if (digit >= start && digit <= end) {
return digit;
}
else
cout << "\nTry again : ";
} while (true);
}
void List::readFromFile() {
bool budget;
string nameStudent;
string nameCurator;
string group;
int yearOfEntry;
int departmentBuilding;
bool changeUniversity;
int markEvi;
bool additionalPoints;
int previousEducationalInstitution;

std::ifstream in(filename);
if (in.is_open())
{
int size = 0;
```

```
int type = 0;
list.clear();


in >> size;
list.reserve(size);
for (int i = 0; i < size; i++) {
Student* p = NULL;
in >> type;
if (type == 1) {
in >> budget >> nameStudent >> nameCurator >> group >> yearOfEntry >> departmentBuilding >> changeUniversity >> markEvi;
GraduateStudent* tmp = new GraduateStudent(budget, nameStudent, nameCurator, group, yearOfEntry, departmentBuilding, changeUniversity, markEvi);
p = *(&tmp);
list.push_back(p);
}
else if (type == 2) {
in >> budget >> nameStudent >> nameCurator >> group >> yearOfEntry >> departmentBuilding >> additionalPoints >> previousEducationalInstitution;
BachelorStudent* tmp = new BachelorStudent(budget, nameStudent, nameCurator, group, yearOfEntry, departmentBuilding, additionalPoints, previousEducationalInstitution);
p = *(&tmp);
list.push_back(p);
}


}
in.close();
```

```
}
}
void List::writeToFile() {
std::ofstream out(filename);
if (out.is_open()) {
out << list.size() << endl;
for (int i = 0; i < list.size(); i++)
out << list[i]->whoIAm() << " " << list[i]->getBudget() << " " << list[i]->getNameStudent() << " " << list[i]->getNameCurator() << " " << list[i]->getGroup() << " " << list[i]->getYearOfEntry() << " " << list[i]->getDepartmentBuilding() << endl;
}
out.close();
}
```

У файлі List.h

```
#pragma once
#include "Student.h"
class List
{
private:
std::vector<Student*> list;

public:
void addStudent();
void removeStudent(const int index);
void setStudent(const int index);
void clear();
void showAll();
```

```
int getSize();
Student& getStudent(const int index);
Student& getStudentYearOfEntry(const int grade);

void sortYearOfEntry();

string check(regex reg);
int chooseInt(int start, int end);

Student& operator[](const int index);
friend std::ostream& operator << (std::ostream& out, const List& obj) {
for (int i = 0; i < obj.list.size(); i++)
out << obj.list[i];

return out;
}

void readFromFile();
void writeToFile();
};

class ListofGraduateStudent {
private:
std::vector<GraduateStudent> list;
```

```
public:
void addStudent(GraduateStudent& student);
void addObjects();
void removeStudent(const int index);
void showAll();

int getSize();
GraduateStudent& operator[](const int index);
GraduateStudent& getStudent(const int index);
GraduateStudent& getStudentBudget(const int grade);
string check(regex reg);
int chooseInt(int start, int end);
void get2018YearOfEntry();
	void getWasAtAnotherUniversity();



};

class ListofBachelorStudent {
private:
std::vector<BachelorStudent> list;

public:
void addStudent(BachelorStudent& pupils);
void addObjects();
void removeStudent(const int index);
void showAll();
```

```
int getSize();
BachelorStudent& operator[](const int index);
BachelorStudent& getStudent(const int index);
BachelorStudent& getStudentBudget(const int grade);
string check(regex reg);
int chooseInt(int start, int end);
void get2018YearOfEntry();
void getWasCollege();
};
```

У файлі Student.cpp

```
#include "Student.h"

Student::Student() :budget(1), nameStudent("exampleName"),
nameCurator("exampleName"), group("exampleGroup"), yearOfEntry(0),
departmentBuilding(0)
{
//cout << "\n\t\tThere was a DEFAULT constructor here\n";
}

Student::Student(const bool budget, const string nameStudent, const string
nameCurator, const string group, const int yearOfEntry, const int
departmentBuilding) :
budget(budget), nameStudent(nameStudent), nameCurator(nameCurator),
group(group), yearOfEntry(yearOfEntry),
departmentBuilding(departmentBuilding)
{
//cout << "\n\t\tHere was a constructor with PARAMETRS\n";
}
```

```
Student::Student(const Student& obj) :budget(obj.budget),
nameStudent(obj.nameStudent), nameCurator(obj.nameCurator),
group(obj.group), yearOfEntry(obj.yearOfEntry),
departmentBuilding(obj.departmentBuilding)
{
//cout << "\n\t\tThere was a COPY constructor here\n";
}

Student& Student::operator= (const Student& obj) {
budget = obj.budget;
nameStudent = obj.nameStudent;
nameCurator = obj.nameCurator;
group = obj.group;
yearOfEntry = obj.yearOfEntry;
departmentBuilding = obj.departmentBuilding;

return *this;
}

string GraduateStudent::print() {
std::stringstream ss;
ss << "\nBudget:" << budget ? "Yes" : "No";
ss << "\nName Student:" << nameStudent;
ss << "\nName Curator:" << nameCurator;
ss << "\nGroup:" << group;
ss << "\nYear:" << yearOfEntry;
ss << "\nDepartment Building:";
```

```
switch (departmentBuilding)
{
case 1:
ss << "VC";
break;
case 2:
    ss << "GAK";
break;
case 3:
ss << "Y1";
break;
case 4:
ss << "Y2";
break;
}
ss << "\nChanged university:" << changeUniversity ? "Yes" : "No";
ss << "\nMark Evi:" << markEvi;
return ss.str();
}
string BachelorStudent::print() {
std::stringstream ss;
ss << "\nBudget:" << budget ? "Yes" : "No";
ss << "\nName Student:" << nameStudent;
ss << "\nName Curator:" << nameCurator;
ss << "\nGroup:" << group;
ss << "\nYear:" << yearOfEntry;
ss << "\nDepartment Building:";
```

```
switch (departmentBuilding)
{
case 1:
ss << "VC";
break;
case 2:
ss << "GAK";
break;
case 3:
ss << "Y1";
break;
case 4:
ss << "Y2";
break;
}
ss << "\nAdditional points:" << (additionalPoints ? "Yes" : "No");
ss << "\nPrevious Educational Institution : ";
switch (previousEducationalInstitution)
{
case 1:
ss << "School";
break;
case 2:
ss << "Technical school";
break;
case 3:
ss << "College";
```

```
break;
}

return ss.str();
}
int GraduateStudent::whoIAm() { return 1; };
int BachelorStudent::whoIAm() { return 2; };

GraduateStudent::GraduateStudent(bool budget, string nameStudent, string nameCurator, string group, int yearOfEntry, int departmentBuilding, bool changeUniversity, int markEvi) {
 setBudget(budget);
 setNameStudent(nameStudent);
 setNameCurator(nameCurator);
 setGroup(group);
 setYearOfEntry(yearOfEntry);
    setDepartmentBuilding(departmentBuilding);
 setChangeUniversity(changeUniversity);
 setMarkEvi(markEvi);

}

BachelorStudent::BachelorStudent(bool budget, string nameStudent, string nameCurator, string group, int yearOfEntry, int departmentBuilding, bool additionalPoints, int previousEducationalInstitution) {
setBudget(budget);
setNameStudent(nameStudent);
setNameCurator(nameCurator);
setGroup(group);
```

```
setYearOfEntry(yearOfEntry);
setDepartmentBuilding(departmentBuilding);
setAdditionalPoints(additionalPoints);
setPreviousEducationalInstitution(previousEducationalInstitution);
}


void Student::setBudget(bool budget) { this->budget = budget; }
void Student::setNameStudent(string nameStudent) { this->nameStudent = nameStudent; }
void Student::setNameCurator(string nameCurator) { this->nameCurator = nameCurator; }
void Student::setGroup(string group) { this->group = group; }
void Student::setYearOfEntry(int yearOfEntry) { this->yearOfEntry = yearOfEntry; }
void Student::setDepartmentBuilding(int departmentBuilding) { this->departmentBuilding = departmentBuilding; }


bool Student::getBudget() { return budget; }
string Student::getNameStudent() { return nameStudent; }
string Student::getNameCurator() { return nameCurator; }
string Student::getGroup() { return group; }
int Student::getYearOfEntry() { return yearOfEntry; }
int Student::getDepartmentBuilding() { return departmentBuilding; }
```

У файлі Student.h

```
#pragma once
#include <regex>
#include <string>
#include <iostream>
```

```
#include <fstream>
#include <vector>
#include <sstream>
#include <algorithm>

using std::string;
using std::cout;
using std::cin;
using std::endl;
using std::regex;


#define filename "D:\\text.txt"
static regex regName("([A-Z]{1}[a-z]{1,}[_]{1}[A-Z]{1}\.[A-Z]{1}\.)");
static regex regGroup("([A-Z]{1,4}[-]{1}[0-9]{3}[a-z]{1})");


class Student
{
public:
bool budget;
string nameStudent;
string nameCurator;
string group;
int yearOfEntry;
    int departmentBuilding;
```

```
Student();
Student(const bool budget, const string nameStudent, const string nameCurator, const string group, const int yearOfEntry, const int departmentBuilding);
Student(const Student& obj);
virtual ~Student() = 0 {};

virtual string print() = 0;
virtual int whoIAm() = 0;

Student& operator=(const Student& obj);
friend bool operator==(const Student& obj1, const Student& obj2) {
return (obj1.yearOfEntry == obj2.yearOfEntry);
}
friend bool operator>(const Student& obj1, const Student& obj2) {
return (obj1.yearOfEntry > obj2.yearOfEntry);
}
friend bool operator<(const Student& obj1, const Student& obj2) {
return (obj1.yearOfEntry < obj2.yearOfEntry);
}
friend bool operator!=(const Student& obj1, const Student& obj2) {
return !(obj1 == obj2);
}
friend std::ostream& operator<< (std::ostream& out, const Student& obj) {
out << "\nBudget:" << obj.budget ? "Yes" : "No";
out << "\nName Student:" << obj.nameStudent;
out << "\nName Curator:" << obj.nameCurator;
```

```
out << "\nGroup:" << obj.group;
out << "\nYear:" << obj.yearOfEntry;
out << "\nDepartment Building:";
switch (obj.departmentBuilding)
{
case 1:
out << "VC";
break;
case 2:
out << "GAK";
break;
case 3:
out << "Y1";
break;
case 4:
out << "Y2";
break;
}
return out;
}
friend std::istream& operator>> (std::istream& in, Student& obj) {
in >> obj.budget;
in >> obj.nameStudent;
in >> obj.nameCurator;
in >> obj.group;
in >> obj.yearOfEntry;
in >> obj.departmentBuilding;
```

```
return in;
}

void setBudget(bool budget);
void setNameStudent(string nameStudent);
void setNameCurator(string nameCurator);
void setGroup(string group);
void setYearOfEntry(int yearOfEntry);
void setDepartmentBuilding(int departmentBuilding);

bool getBudget();
string getNameStudent();
string getNameCurator();
string getGroup();
int getYearOfEntry();
int getDepartmentBuilding();
};

class GraduateStudent : public Student {
private:
bool changeUniversity;
int markEvi;

public:
GraduateStudent() :markEvi(0), changeUniversity(false) {};
```

```
GraduateStudent(bool budget, string nameStudent, string nameCurator, string group, int yearOfEntry, int departmentBuilding, bool changeUniversity, int markEvi);
~GraduateStudent() {}


friend std::ostream& operator<< (std::ostream& out, const GraduateStudent& obj) {


out << "\nBudget:" << obj.budget ? "Yes" : "No";
out << "\nName Student:" << obj.nameStudent;
out << "\nName Curator:" << obj.nameCurator;
out << "\nGroup:" << obj.group;
out << "\nYear:" << obj.yearOfEntry;
out << "\nDepartment Building:";
switch (obj.departmentBuilding)
{
case 1:
out << "VC";
break;
case 2:
out << "GAK";
break;
case 3:
out << "Y1";
break;
case 4:
out << "Y2";
break;
```

```
}
out << "\nChanged university:" << obj.changeUniversity ? "Yes" : "No";
out << "\nMark Evi:" << obj.markEvi;



return out;
}

friend std::istream& operator>> (std::istream& in, GraduateStudent& obj) {
in >> obj.budget >> obj.nameStudent >> obj.nameCurator >> obj.group >> obj.yearOfEntry >> obj.departmentBuilding >> obj.changeUniversity >> obj.markEvi;
return in;
}
friend bool operator==(const GraduateStudent& obj1, const GraduateStudent& obj2) {
return (obj1.changeUniversity == obj2.changeUniversity);
}

virtual string print()override final;
virtual int whoIAm()override final;

int getChangeUniversity() { return changeUniversity; }
   int getMarkEvi() { return markEvi; }

void setChangeUniversity(int s) { changeUniversity = s; }
void setMarkEvi(int l) { markEvi = l; }
};
```

```
class BachelorStudent : public Student {

private:

bool additionalPoints;

int previousEducationalInstitution;


public:

BachelorStudent() : additionalPoints(false), previousEducationalInstitution(0) {};

BachelorStudent(bool budget, string nameStudent, string nameCurator, string group, int yearOfEntry, int departmentBuilding, bool additionalPoints, int previousEducationalInstitution);

~BachelorStudent() {};


friend std::ostream& operator<< (std::ostream& out, const BachelorStudent& obj) {


out << "\nBudget:" << obj.budget ? "Yes" : "No";

out << "\nName Student:" << obj.nameStudent;

out << "\nName Curator:" << obj.nameCurator;

out << "\nGroup:" << obj.group;

out << "\nYear:" << obj.yearOfEntry;

out << "\nDepartment Building:";

switch (obj.departmentBuilding)

{

case 1:

out << "VC";

break;

case 2:
```

```
out << "GAK";
break;
case 3:
out << "Y1";
break;
case 4:
out << "Y2";
break;
}
out << "\nAdditional points:" << (obj.additionalPoints ? "Yes" : "No");
out << "\nPrevious Educational Institution : ";
switch (obj.previousEducationalInstitution)
{
case 1:
out << "School";
break;
case 2:
out << "Technical school";
break;
case 3:
out << "College";
break;
}


 return out;
}
```

```
friend std::istream& operator>> (std::istream& in, BachelorStudent& obj) {

in >> obj.budget >> obj.nameStudent >> obj.nameCurator >> obj.group >>
obj.yearOfEntry >> obj.departmentBuilding >>obj.additionalPoints >>
obj.previousEducationalInstitution;

return in;

}

friend bool operator==(const BachelorStudent& obj1, const BachelorStudent&
obj2) {

return (obj1.previousEducationalInstitution ==
obj2.previousEducationalInstitution);

}

virtual string print()override final;

virtual int whoIAm()override final;


bool getAdditionalPoints() { return additionalPoints; }

int getPreviousEducationalInstitution() { return previousEducationalInstitution;
}


void setAdditionalPoints(bool s) { additionalPoints = s; }

void setPreviousEducationalInstitution(int l) { previousEducationalInstitution =
l; }

};
```

У файлі ListofGraduateStudent.cpp

```
#include "List.h"


void ListofGraduateStudent::addStudent(GraduateStudent& student) {

list.push_back(student);

}
```

```
void ListofGraduateStudent::removeStudent(const int index) {
if (!list.empty())
list.erase(list.begin() + index);
}
void ListofGraduateStudent::showAll() {
for (GraduateStudent& emp : list)
cout << emp.print() << endl;
}
GraduateStudent& ListofGraduateStudent::getStudent(const int index) { return list[index]; }
GraduateStudent& ListofGraduateStudent::getStudentBudget(const int yearOfEntry) {
for (GraduateStudent& emp : list) {
if (emp.getYearOfEntry() == yearOfEntry)
return emp;
}
}


int ListofGraduateStudent::getSize() { return list.size(); }

GraduateStudent& ListofGraduateStudent::operator[](const int index) {
return list[index];
}

void ListofGraduateStudent::addObjects() {
bool choice = true;
GraduateStudent tmp;
```

```
int digit;


while (choice) {
getchar;
cout << "\n\nCreate a new Student now.\n    Enter does the student have a budget ? \n1.Yes\n0.No\n\nYour choose : ";


tmp.setBudget(chooseInt(0, 1));
cout << "     Enter name student : ";
tmp.setNameStudent(check(regName));
cout << "     Enter name curator : ";
tmp.setNameCurator(check(regName));
cout << "     Enter group :";
tmp.setGroup(check(regGroup));
cout << "     Enter year of entry : ";
tmp.setYearOfEntry(chooseInt(2000, 2020));
cout << "     Enter department building \n1.VC\n2.GAK\n3.Y1\n4.Y2\nYour choose : ";
tmp.setDepartmentBuilding(chooseInt(1, 4));
cout << "     Have you changed university? \n1.Yes\n0.No\n\nYour choose : ";
tmp.setChangeUniversity(chooseInt(0, 1));
cout << "     Enter Evi score :";
tmp.setMarkEvi(chooseInt(1, 200));


list.push_back(tmp);
cout << "A new employee has been added to the list. Would you like to add another one? \n Your choice (1 - yes, 0 - no):";
cin >> choice;
```

```
}
}

string ListofGraduateStudent::check(regex reg) {
string word;
do {
getline(cin, word);
if (!std::regex_match(word, reg))
cout << "\nTry again :";
} while (!std::regex_match(word, reg));
return word;
}

int ListofGraduateStudent::chooseInt(int start, int end) {
int digit = 0;
do {
cin >> digit;
if (digit >= start && digit <= end) {
return digit;
}
else
cout << "\nTry again : ";
} while (true);
}
```

У файлі ListofBachelorStudent.cpp

```
#include "List.h"
```

```
void ListofBachelorStudent::addStudent(BachelorStudent& student) {
list.push_back(student);
}
void ListofBachelorStudent::removeStudent(const int index) {
if (!list.empty())
list.erase(list.begin() + index);
}
void ListofBachelorStudent::showAll() {
for (BachelorStudent& emp : list)
cout << emp.print() << endl;
}
BachelorStudent& ListofBachelorStudent::getStudent(const int index) { return list[index]; }
BachelorStudent& ListofBachelorStudent::getStudentBudget(const int yearOfEntry) {
for (BachelorStudent& emp : list) {
if (emp.getYearOfEntry() == yearOfEntry)
return emp;
}
}


int ListofBachelorStudent::getSize() { return list.size(); }
BachelorStudent& ListofBachelorStudent::operator[](const int index) {
return list[index];
}
void ListofBachelorStudent::addObjects() {
bool choice = true;
```

```
BachelorStudent tmp;
int digit;

while (choice) {
getchar;
cout << "\n\nCreate a new Student now.\n    Enter does the student have a budget ? \n1.Yes\n0.No\n\nYour choose : ";

tmp.setBudget(chooseInt(0, 1));
cout << "     Enter name student :";
tmp.setNameStudent(check(regName));
cout << "     Enter name curator : ";
tmp.setNameCurator(check(regName));
cout << "     Enter group :";
tmp.setGroup(check(regGroup));
cout << "     Enter year of entry : ";
tmp.setYearOfEntry(chooseInt(2000, 2020));
cout << "     Enter department building \n1.VC\n2.GAK\n3.Y1\n4.Y2\nYour choose : ";
tmp.setDepartmentBuilding(chooseInt(1, 4));
cout << "     Do you have additional points? \n1.Yes\n0.No\nYour choose :";
tmp.setAdditionalPoints(chooseInt(0, 1));
cout << "     Previous Educational Institution\n1.School\n2.Technical school\n3.College\n\nYour choose :";
tmp.setPreviousEducationalInstitution(chooseInt(1, 3));

list.push_back(tmp);
```

```
cout << "A new employee has been added to the list. Would you like to add another one? \n Your choice (1 - yes, 0 - no):";
cin >> choice;
}
}

string ListofBachelorStudent::check(regex reg) {
string word;
do {
getline(cin, word);
if (!std::regex_match(word, reg))
cout << "\nTry again :";
} while (!std::regex_match(word, reg));
return word;
}


int ListofBachelorStudent::chooseInt(int start, int end) {
int digit = 0;
do {
cin >> digit;
if (digit >= start && digit <= end) {
return digit;
}
else
cout << "\nTry again : ";
} while (true);
}
```