ЗВІТ

Виконав:
Заночкин Є., КІТ-119а
Дата: 24 квітня 2020 р.

Лабораторна робота №5

АГРЕГАЦІЯ ТА КОМПОЗИЦІЯ

Мета роботи. отримати поняття агрегація та композиція; отримати знання про призначення ключових слів typedef та auto.

1.Завдання до роботи

Варіант 6.

Загальне завдання: Дослідити заздалегідь визначені типи даних з бібліотеки <cstddef> / <stddef.h>. Модернізувати розроблені у попередній роботі класи таким чином:

- замінити типи даних, що використовуються при індексуванні на типи з указаної бібліотеки;

- створити власний синонім типу, визначивши його необхідність;

- створити / оновити функцію сортування масиву, де крім поля, по якому виконується сортування, передається і вказівник на функцію, яка визначає напрям сортування;

- у базовий клас додати два поля, що мають кастомний тип даних (тип даних користувача) та які будуть відображати відношення «агрегація» та «композиція», при цьому оновити методи читання та запису об’єкта;

- ввести використання ключового слова auto як специфікатор зберігання типу змінної. Визначити плюси та мінуси цього використання.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: Task.

Клас, що має в собі масив базового класу та методи для роботи з ним: List.

Клас, що відображає агрегативні відносини з базовим класом: Student.

Клас, що відображає композитивні відносини з базовим класом: Date.

2.2 Опис змінних

int mark – оцінка за роботу (Task).

int countOfDoneExercises – кількість виконаних завдань (Task).

int studentIndex – індекс студента (Task).

string name – ім’я студента (Task).

int rgz – кількість ргз (Task).

Student age – вік студента (Student).

Date date – дата написання (Date).

int listSize – розмір масиву елементів класу List.

Task* stud – масив елементів класу Task.

List list – об’єкт класу List.

List test – об’єкт класу List.

Task var - об’єкт класу Task.

2.3 Опис методів

void createList() – створення масиву елементів і заповнення даними (List).

void printAll() const – виведення даних елементів у консоль (List).

void printOneStudent(stringstream&) const – виведення даних одного студента у консоль (List).

void addStudent(const Task) – додавання нового елементу в масив (List).

void deleteStudent(int) – видалення елемента з масиву (List).

int getStudentID(int)const – отримання даних елемента по індексу (List).

int getStudentRGZ(int)const - отримання даних елемента по кількості ргз (List).

void ReadFile(string, int) – читання даних з файлу (List).

int FileString(string) – кількість рядків у файлі (List).

void WriteFile(string) const – запис даних у файл (List).

stringstream getObj(int) const – повернення студента у вигляді рядка (List).

Task enterNewStudent() – введення нового студента з клавіатури (List).

void regexTask() – виведення на екран об’єктів, які в полі string мають 2 слова (List).

static bool sortAsc(const int&, const int&) – функція, що визначає напрям сортування (List).

static bool sortDesc(const int&, const int&) – функція, що визначає напрям сортування (List).

void sort(comp) – сортування масиву (List).

Task() – конструктор за змовчуванням (Task).

Task(int, string, int, int, int) – конструктор з аргументами (Task).

Task(const Task& stud) – конструктор копіювання (Task).

~Task() – деструктор (Task).

~List() – деструктор (List).

2.4 Опис функцій

void Menu() – функція меню.

int generateID() – функція, що надає кожному об’єкту унікальний індекс.

int generateRGZ() – функція, що надає кожному об’єкту унікальну кількість ргз.

Task CreateStudent() – стандартний об’єкт класу Task.

Task CreateStudent2() – стандартний об’єкт класу Task.

void TestAddStudent(List&, int) – тест функції додавання об’єкта до масиву об’єктів.

void TestDeleteStudent(List&, int) – тест функції видалення об’єкта з масиву об’єктів.

void TestGetStudenttID(List&, int) – тест функції повернення індексу студента.

3. Текст програми

Task.h

```
#pragma once
```

```
#define _CRT_SECURE_NO_WARNINGS
#define __CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#define DEBUG_NEW new(_NORMAL_BLOCK, __FILE__, __LINE__)
#define new DEBUG_NEW
#include <string>
#include <iostream>
#include <iomanip>
#include <sstream>
#include <fstream>
#include <regex>
#include <cstddef>
#include "Date.h"
#include "Student.h"
using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::setw;
using std::stringstream;
using std::regex;
typedef bool (comp)(const int&, const int&);
class Task
{
private:
int studentIndex;
string name;
int mark;
int countOfDoneExercises;
int rgz;
Date date;
Student age;
public:
int getMark() const;
void setMark(int);
int getCountOfDoneExercises() const;
void setCountOfDoneExercises(int);
int getStudentIndex() const;
void setStudentIndex(int);
int getRgz() const;
void setRgz(int);
string getName() const;
```

```
void setName(const string);
sint getDay() const;
void setDay(const sint);
sint getMonth() const;
void setMonth(const sint);
sint getYear() const;
void setYear(const sint);
int getAge() const;
void setAge(const int);
int generateID();
int generateRGZ();
Task();
Task(int, string, int, int, int, int, sint, sint, sint);
Task(const Task& stud);
~Task();
};
void Menu();
```

List.h

```
#pragma once
#include "Task.h"
class List
{
private:
int listSize;
public:
Task* stud;
int getListSize() const;
void setListSize(int);
void addStudent(const Task);
void deleteStudent(int);
void printAll() const;
void printOneStudent(stringstream&) const;
void createList();
int getStudentID(int)const;
int getStudentRGZ(int)const;
void ReadFile(string, int);
int FileString(string);
void WriteFile(string) const;
stringstream getObj(int i) const;
Task enterNewStudent();
void regexTask();
```

```
static bool sortAsc(const int&, const int&);
static bool sortDesc(const int&, const int&);
void sort(comp);
~List();
};
Task CreateStudent();
Task CreateStudent2();
```

Task.cpp

```
#include "Task.h"
#include "List.h"
int Task::getMark() const { return mark; }
void Task::setMark(int mark1) { mark = mark1; }
int Task::getCountOfDoneExercises() const { return countOfDoneExercises; }
void Task::setCountOfDoneExercises(int count_of_done_exercises1)
{ countOfDoneExercises = count_of_done_exercises1; }
int Task::getStudentIndex() const { return studentIndex; }
void Task::setStudentIndex(int exercises1) { studentIndex = exercises1; }
int Task::getRgz() const { return rgz; }
void Task::setRgz(int rgz1) { rgz = rgz1; }
string Task::getName() const { return name; }
void Task::setName(string name1) { name = name1; }
sint Task::getDay() const { return date.getDay(); }
void Task::setDay(sint day1) { date.setDay(day1); }
sint Task::getMonth() const { return date.getMonth(); }
void Task::setMonth(sint month1) { date.setMonth(month1); }
sint Task::getYear() const { return date.getYear();; }
void Task::setYear(sint year1) { date.setYear(year1); }
int Task::getAge() const { return age.getAge(); }
void Task::setAge(const int age1) { age.setAge(age1); }
Task CreateStudent()
{
Task stud;
stud.setMark(5);
stud.setCountOfDoneExercises(5);
stud.setName("Petrova Katya");
stud.setDay(5);
stud.setMonth(5);
stud.setYear(5555);
stud.setAge(5);
return stud;
}
```

```
Task CreateStudent2()
{
Task stud;
stud.setMark(2);
stud.setCountOfDoneExercises(2);
stud.setName("Ivanov Petya");
stud.setDay(2);
stud.setMonth(2);
stud.setYear(2222);
stud.setAge(2);
return stud;
}
Task::Task(int student_index, string name, int age, int mark, int countOfDoneExercises, int rgz, sint day, sint month, sint year) : studentIndex(student_index), name(name), age(age), mark(mark), countOfDoneExercises(countOfDoneExercises), rgz(rgz), date(day, month, year)
{ cout << "Constructor with parameter" << endl; }
Task::Task() : studentIndex(0), name(" "), age(0), mark(0), countOfDoneExercises(0), rgz(0), date(0, 0, 0000)
{ cout << "Default constructor" << endl; }
Task::Task(const Task& stud) : studentIndex(stud.studentIndex), name(stud.name), age(stud.age), mark(stud.mark), countOfDoneExercises(stud.countOfDoneExercises), rgz(stud.rgz),date(stud.date)
{ cout << "Copy constructor" << endl; }
Task::~Task()
{ cout << "Destructor Task" << endl; }
```

List.cpp

```
#include "List.h"
#include "Task.h"
#include "Date.h"
int List::getListSize() const { return listSize; }
void List::setListSize(int size) { listSize = size; }
int Task::generateID()
{
static int id = 1;
return id++;
}
int Task::generateRGZ()
{
static int RGZ = 5;
return RGZ++;
}
void List::addStudent(const Task task)
```

```
{
if (task.getMark() == 0)
{
cout << "Empty object. Error";
return;
}
setListSize(getListSize() + 1);
Task* newstud = new Task[listSize];
for (size_t i = 0; i < listSize - 1; i++)
newstud[i] = stud[i];
newstud[listSize - 1] = task;
if (newstud[listSize - 1].getStudentIndex() == 0)
{
newstud[listSize - 1].setStudentIndex(stud->generateID());
newstud[listSize - 1].setRgz(stud->generateRGZ());
}
delete[] stud;
stud = new Task[listSize];
for (size_t i = 0; listSize > i; i++)
stud[i] = newstud[i];
delete[] newstud;
}
void List::createList()
{
stud = new Task[listSize];
for (size_t i = 0; listSize > i; i++)
{
stud[i] = CreateStudent();
stud[i].setStudentIndex(stud->generateID());
stud[i].setRgz(stud->generateRGZ());
}
}
void List::deleteStudent(int c)
{
setListSize(getListSize() - 1);
Task* newstud = new Task[listSize];
size_t i = 0;
for (; i < getListSize(); i++)
{
if (stud[i].getStudentIndex() == c)
break;
newstud[i] = stud[i];
```

```
}
for (; i < getListSize(); i++)
newstud[i] = stud[i + 1];
delete[] stud;
stud = new Task[listSize];
for (size_t i = 0; i < getListSize(); i++)
stud[i] = newstud[i];
delete[] newstud;
}
void List::printOneStudent(stringstream& ss) const
{
int index;
string name, name2;
int mark, age;
int exercises;
int rgz;
sint day, month, year;
cout << std::left << setw(10) << "Index" << setw(13) << "Name" << setw(8) << "Age" << setw(10) <<
"Mark" << setw(16) << "Exercises" << setw(13) << "RGZ" << "Date" << endl;
ss >> index;
ss >> name;
ss >> name2;
ss >> age;
ss >> mark;
ss >> exercises;
ss >> rgz;
ss >> day;
ss >> month;
ss >> year;
if (name2 == "")
name = name + " ";
else (name = name + " " + name2);
cout << std::left;
cout << setw(6) << index;
cout << setw(18) << name;
cout << setw(8) << age;
cout << setw(13) << mark;
cout << setw(13) << exercises;
cout << setw(10) << rgz;
cout << setw(3) << day << setw(3) << month << year;
}
void List::printAll() const
```

```
{
cout << std::left << setw(10) << "Index" << setw(13) << "Name" << setw(8) << "Age" << setw(10) << "Mark" << setw(16) << "Exercises" << setw(13) << "RGZ" << "Date" << endl;
for (size_t i = 0; List::getListSize() > i; i++)
cout << std::left << setw(6) << stud[i].getStudentIndex() << setw(18) << stud[i].getName() << setw(8) << stud[i].getAge() << setw(13) << stud[i].getMark() << setw(13) << stud[i].getCountOfDoneExercises() <<
setw(10) << stud[i].getRgz() << setw(3) << stud[i].getDay() << setw(3) << stud[i].getMonth() << stud[i].getYear() << endl;
}
List::~List()
{
cout << "Destructor List" << endl;
delete[] stud;
}
int List::getStudentID(int id) const
{
for (size_t i = 0; i < listSize; i++)
if (stud[i].getStudentIndex() == id)
return i;
cout << "Wrong ID" << endl;
return -1;
}
int List::getStudentRGZ(int a) const
{
for (size_t i = 0; i < listSize; i++)
if (stud[i].getRgz() == a)
return i;
cout << "Wrong count of RGZ" << endl;
return -1;
}
void List::ReadFile(string filename, int c)
{
std::ifstream fin(filename);
if (!fin.is_open())
return;
string line;
regex regular("([\\d]* [A-ZA-Я]+[\\wA-Яа-я,.;:-]* [\\wA-Яа-я,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
int i = 0, a = 0, b = 0;
delete[] stud;
stud = new Task[c];
while (getline(fin, line) && i < c)
```

```
{
if (regex_match(line.c_str(), regular))
{
int studentIndex;
string name, name2;
int mark, age;
int countOfDoneExercises;
int rgz;
sint day, month, year;
std::istringstream fin(line);
fin >> studentIndex;
fin >> name;
fin >> name2;
fin >> age;
fin >> mark;
fin >> countOfDoneExercises;
fin >> rgz;
fin >> day;
fin >> month;
fin >> year;
if (name2 == "")
name = name + " ";
else (name = name + " " + name2);
do
{
b = 0;
a = name.find("--");
if (a != -1)
{
name.erase(a, 1);
b = 1;
}
a = name.find(" ");
if (a != -1)
{
name.erase(a, 1);
b = 1;
}
a = name.find(",,");
if (a != -1)
{
name.erase(a, 1);
```

```
b = 1;
}
a = name.find("::");
if (a != -1)
{
name.erase(a, 1);
b = 1;
}
a = name.find(";;");
if (a != -1)
{
name.erase(a, 1);
b = 1;
}
a = name.find("_");
if (a != -1)
{
name.erase(a, 1);
b = 1;
}
} while (b == 1);
Task ex(studentIndex, name, age, mark, countOfDoneExercises, rgz,day,month,year);
stud[i++] = ex;
}
}
setListSize(c);
fin.close();
cout << "Data from file have written";
return;
}
int List::FileString(string filename)
{
int c = 0;
string line;
regex regular("([\\d]* [A-ZA-Я]+[\\wA-Яа-я,.;:-]* [\\wA-Яа-я,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*
[\\d]*)");
std::ifstream fin(filename);
if (!fin.is_open())
{
cout << "Error open file";
return 0;
}
```

```
while (getline(fin, line))
{
if (regex_match(line, regular))
c++;
else cout << "String is not correct" << endl;
}
fin.close();
return c;
}
void List::WriteFile(string filename) const
{
std::ofstream fout(filename);
if (!fout.is_open())
{
cout << "Error open file";
return;
}
for (size_t i = 0; i < listSize; i++)
{
fout << std::left;
fout << setw(6) << stud[i].getStudentIndex();
fout << setw(18) << stud[i].getName();
fout << setw(8) << stud[i].getAge();
fout << setw(13) << stud[i].getMark();
fout << setw(13) << stud[i].getCountOfDoneExercises();
fout << setw(10) << stud[i].getRgz();
fout << setw(3) << stud[i].getDay() << setw(3) << stud[i].getMonth() << stud[i].getYear() << endl;
}
cout << "Write to file - correct" << endl;
return;
}
stringstream List::getObj(int i) const
{
stringstream ss;
ss << stud[i].getStudentIndex();
ss << " " << stud[i].getName();
ss << " " << stud[i].getAge();
ss << " " << stud[i].getMark();
ss << " " << stud[i].getCountOfDoneExercises();
ss << " " << stud[i].getRgz();
ss << " " << stud[i].getDay();
ss << " " << stud[i].getMonth();
```

```
ss << " " << stud[i].getYear();
return ss;
}
Task List::enterNewStudent()
{
Task add, error;
int index, mark, rgz, exercises, age;
string name, surname, data;
sint day, month, year;
regex regular("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
cout << "Enter student data (ID, Surname, Name, Age, Mark, Exercises, RGZ, Date(day,month,year)):" << endl;
cin.ignore();
getline(cin, data);
std::istringstream temp(data);
temp >> index;
temp >> surname;
temp >> name;
temp >> age;
temp >> mark;
temp >> exercises;
temp >> rgz;
temp >> day;
temp >> month;
temp >> year;
if (name == "")
surname = surname + " ";
else (surname = surname + " " + name);
if (!regex_match(data, regular))
{
cout << "You enter wrong data";
return error;
}
add.setStudentIndex(index);
add.setName(surname);
add.setAge(age);
add.setMark(mark);
add.setCountOfDoneExercises(exercises);
add.setRgz(rgz);
add.setDay(day);
add.setMonth(month);
```

```
add.setYear(year);
return add;
}
void List::regexTask()
{
stringstream ss;
int index, mark, rgz, exercises, age;
string name, name2;
sint day, month, year;
regex regular("(^[A-ZA-Я]+[\\wA-Яа-я,.;:-]* [\\wA-Яа-я,.;:-]+)");
int listSize = getListSize();
cout << std::left << setw(10) << "Index" << setw(13) << "Name" << setw(8) << "Age" << setw(10) <<
"Mark" << setw(16) << "Exercises" << setw(13) << "RGZ" << "Date" << endl;
for (size_t i = 0; i < listSize; i++)
if (regex_match(stud[i].getName(), regular))
{
ss = getObj(i);
ss >> index >> name >> name2 >> age >> mark >> exercises >> rgz >> day >> month >> year;
if (name2 == "")
name = name + " ";
else (name = name + " " + name2);
cout << std::left;
cout << setw(6) << index;
cout << setw(18) << name;
cout << setw(8) << age;
cout << setw(13) << mark;
cout << setw(13) << exercises;
cout << setw(10) << rgz;
cout << setw(3) << day << setw(3) << month << year << endl;
}
}
bool List::sortAsc(const int& a, const int& b) { return a > b; }
bool List::sortDesc(const int& a, const int& b) { return a < b; }
void List::sort(comp condition)
{
Task temp;
int pr;
do
{
pr = 0;
for (size_t i = 0; i < getListSize()-1; i++)
{
```

```
if (condition(stud[i].getMark(), stud[i + 1].getMark()))
{
temp = stud[i];
stud[i] = stud[i + 1];
stud[i + 1] = temp;
pr = 1;
}
}
} while (pr == 1);
}
```

Menu.cpp

```
#include "Task.h"
#include "List.h"
void Menu()
{
List list;
Student age;
int c = 0, a = 0, b = 0, value = 0;
auto count_of_students = 1;
int menu_number = 1;
int delete_number;
string fileName;
stringstream ss;
Task var;
int file;
int* studentAge;
list.setListSize(count_of_students);
studentAge = age.createList(count_of_students);
list.createList();
while (menu_number)
{
menu_number = 0;
cout << endl << "Menu:" << endl;
cout << "1.Add a new student" << endl;
cout << "2.Add a new student (enter from keyboard)" << endl;
cout << "3.Delete one student" << endl;
cout << "4.Show all student" << endl;
cout << "5.Show student via his index" << endl;
cout << "6.Show student via his count of RGZ" << endl;
cout << "7.Read array from file" << endl;
cout << "8.Write to file" << endl;
```

```
cout << "9.Surname+Name in object" << endl;
cout << "10.Sort (mark)" << endl;
cout << "11.End program" << endl;
cin >> menu_number;
switch (menu_number)
{
case 1:
var = CreateStudent2();
list.addStudent(var);
break;
case 2:
list.addStudent(list.enterNewStudent());
break;
case 3:
cout << "Enter a index of student who you want to delete:" << endl;
cin >> delete_number;
if (delete_number < 1)
{
cout << "Wrong student index" << endl;
break;
}
for (size_t i = 0; list.getListSize() > i; i++)
if (delete_number == list.stud[i].getStudentIndex())
{
list.deleteStudent(delete_number);
break;
}
break;
case 4:
list.printAll();
break;
case 5:
cout << "Enter a index of student:";
cin >> c;
b = list.getStudentID(c);
if (b == -1)
break;
ss = list.getObj(b);
list.printOneStudent(ss);
break;
case 6:
cout << "Enter a count of RGZ";
```

```
cin >> a;
b = list.getStudentRGZ(a);
if (b == -1)
break;
ss = list.getObj(b);
list.printOneStudent(ss);
break;
case 7:
cout << "Enter file name:";
cin >> fileName;
list.ReadFile(fileName, list.FileString(fileName));
break;
case 8:
cout << "Enter file name:";
cin >> fileName;
list.WriteFile(fileName);
break;
case 9:
list.regexTask();
break;
case 10:
cout << "1) Increasing" << endl;
cout << "2) Decreasing" << endl;
cin >> value;
cout << endl;
if (value == 1) list.sort(list.sortAsc);
else if (value == 2) list.sort(list.sortDesc);
else cout << "Wrong number." << endl;
break;
case 11:
menu_number = 0;
break;
default:
cout << "You have chosen the wrong number of the menu";
break;
}
}
age.deleteList();
return;
}
```

main.cpp

```
#include "Task.h"
int main()
{
Menu();
if (_CrtDumpMemoryLeaks())
cout << endl << "WARNING! Memory leak" << endl;
else
cout << endl << "There is no memory leak" << endl;
return 0;
}
```

Student.h

```
#pragma once
#include <iostream>
#include <string>
using std::string;
class Student {
private:
int age;
int listSize;
int* list;
public:
int getAge()const;
void setAge(const int);
void deleteList();
int* createList(int size);
int students(int value);
Student();
Student(int age);
Student(const Student& other);
~Student();
};
```

Date.h

```
#pragma once
typedef short sint;
class Date
{
private:
sint day;
sint month;
```

```
sint year;
public:
sint getDay() const;
void setDay(sint day);
sint getMonth() const;
void setMonth(sint month);
sint getYear() const;
void setYear(sint year);
Date();
Date(sint, sint, sint);
Date(const Date& date);
~Date();
};
```

Student.cpp

```
#include "Student.h"
int Student::getAge() const { return age; }
void Student::setAge(const int age1) { age = age1; }
int* Student::createList(int size)
{
listSize = size;
list = new int[size];
for (size_t i = 0; i < size; i++)
list[i] = students(i);
return list;
}
int Student::students(int value)
{
int age;
switch (value)
{
case 1:
age = 5;
return age;
case 2:
age = 15;
return age;
}
}
void Student::deleteList()
{
delete[] list;
```

```
}
Student::Student() : age(0) {}
Student::Student(int age) : age(age) {}
Student::Student(const Student& student) : age(student.age) {}
Student::~Student() {}
```

Date.cpp

```
#include "Date.h"
sint Date::getDay() const { return day; }
void Date::setDay(sint day1) { day = day1; }
sint Date::getMonth() const { return month; }
void Date::setMonth(sint month1) { month = month1; }
sint Date::getYear() const { return year; }
void Date::setYear(sint year1) { year = year1; }
Date::Date() : day(2), month(2), year(2002) {}
Date::Date(sint day, sint month, sint year) : day(day), month(month), year(year) {}
Date::Date(const Date& date) : day(date.day), month(date.month), year(date.year) {}
Date::~Date() {}
```

Test.cpp

```
#include "List.h"
#include "Task.h"
int TestAddStudent(List&, int);
int TestDeleteStudent(List&, int);
int TestGetStudentID(List&, int);
int TestReadFile(List&, int);
int main()
{
{
List test;
int count = 0;
test.setListSize(2);
test.createList();
count = TestAddStudent(test, count);
count = TestDeleteStudent(test, count);
count = TestGetStudentID(test, count);
count = TestReadFile(test, count);
if (count == 4)
cout << endl << "All tests are successful" << endl;
else
cout << endl << "Not all tests are successful" << endl;
```

```
}
if (_CrtDumpMemoryLeaks())
cout << endl << "WARNING! Memory leak" << endl;
else
cout << endl << "There is no memory leak" << endl;
return 0;
}
int TestAddStudent(List& test, int count)
{
test.addStudent(CreateStudent2());
if (test.getListSize() == 3)
{
cout << endl << "Test: Add_student - successful" << endl;
count++;
}
else
cout << endl << "Test: Add_student - unsuccessful" << endl;
return count;
}
int TestDeleteStudent(List& test, int count)
{
test.deleteStudent(test.getListSize());
if (test.getListSize() == 2)
{
cout << endl << "Test: Delete_student - successful" << endl;
count++;
}
else
cout << endl << "Test: Delete_student - unsuccessful" << endl;
return count;
}
int TestGetStudentID(List& test, int count)
{
int num = test.getStudentID(2);
if (num == 1)
{
cout << endl << "Test: GetStudentId - successful" << endl;
count++;
}
else
cout << endl << "Test: GetStudentId - unsuccessful" << endl;
return count;
```

```
}
int TestReadFile(List& test, int count)
{
string filename = "Text.txt";
int a = 3;
test.ReadFile(filename,a);
string expected = "Ivanov Vasya";
string real = test.stud[0].getName();
if (expected == real)
{
cout << endl << "Test: Readfile - successful" << endl;
count++;
}
else
cout << endl << "Test: Readfile - unsuccessful" << endl;
return count;
}
```

4. Результати роботи програми

```
Index     Name          Age      Mark      Exercises        RGZ          Date
1     Petrova Katya      5        5            5             5          5  5  5555
```

Рисунок 1 – Додано 2 поля до базового класу (Age – агрегація, Date – композиція)

```
Index     Name          Age      Mark      Exercises        RGZ          Date
1     Petrova Katya      5        5            5             5          5  5  5555
2     Ivanov Petya       2        2            2             6          2  2  2222
3     Ivanov Petya       2        2            2             7          2  2  2222
10    Zanochkin Egor     18       3            5             8          2  2  2002

Menu:
1.Add a new student
2.Add a new student (enter from keyboard)
3.Delete one student
4.Show all student
5.Show student via his index
6.Show student via his count of RGZ
7.Read array from file
8.Write to file
9.Surname+Name in object
10.Sort (mark)
11.End program
10
1) Incrising
2) Decrising
2

Default constructor
Destructor Task

Menu:
1.Add a new student
2.Add a new student (enter from keyboard)
3.Delete one student
4.Show all student
5.Show student via his index
6.Show student via his count of RGZ
7.Read array from file
8.Write to file
9.Surname+Name in object
10.Sort (mark)
11.End program
4
Index     Name          Age      Mark      Exercises        RGZ          Date
1     Petrova Katya      5        5            5             5          5  5  5555
10    Zanochkin Egor     18       3            5             8          2  2  2002
2     Ivanov Petya       2        2            2             6          2  2  2222
3     Ivanov Petya       2        2            2             7          2  2  2222
```

Рисунок 2 – Сортування масиву

5. Висновки

Під час виконання лабораторної роботи було замінено тип даних при індексуванні на тип size_t, створено власний синонім типу, створено функцію сортування масиву, додано 2 поля у базовий клас (агрегація та композиція) та використано ключове слово auto.

Програма протестована, виконується без помилок, витоків пам’яті немає.