ЗВІТ

Виконав:

Заночкин Є., КІТ-119а

Дата: 8 березня 2020 р.

Лабораторна робота №2

Класи. Конструктори та деструктори. Перевантаження методів.

Мета роботи. отримати базові знання про класи, конструктори та деструктори. Дослідити механізм створення та видалення об’єктів.

1.Завдання до роботи

Варіант 6.

Прикладна галузь: Самостійні роботи студентів.

Базовий клас: Розрахунково-графічне завдання.

Загальне завдання: поширити попередню лабораторну роботу таким чином:

1) в базовому класі необхідно додати:

- мінімум одне поле типу char*;

- конструктор за замовчуванням, копіювання та конструктор з аргументами;

- деструктор;

2) у клас-список потрібно додати метод обходу масиву для виконання індивідуального завдання.

Індивідуальне завдання : визначити кількість РГЗ, що виконує студент за весь період навчання в інституті відповідно до навчального плану

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

2.2 Опис змінних

int mark – оцінка за роботу (Task).

int countOfDoneExercises – кількість виконаних завдань (Task).

int studentIndex – індекс студента (Task).

const char* maleOrFemale – стать студента (Task).

int rgz – кількість ргз (Task).

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.

2.3 Опис методів

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

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

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

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

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

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

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

Task() – конструктор за змовчуванням (Task).

Task(char*, int, 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.

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.

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 <cstdio>
#include <iostream>
using namespace std;

class Task
{
private:
	int mark;
	int countOfDoneExercises;
	int studentIndex;
	const char* maleOrFemale;
	int rgz;

public:
	int getMark() const;
	void setMark(int);

	int getCountOfDoneExercises() const;
	void setCountOfDoneExercises(int);

	int getStudentIndex() const;
	void setStudentIndex(int);

	const char* getMaleOrFemale() const;
	void setMaleOrFemale(const char*);

	int getRgz() const;
	void setRgz(int);

	Task();
	Task(char*, int, int, int, int);
	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(Task);
	void deleteStudent(int);
	void printAll() const;
	void printOneStudent(int) const;
	void createList();
	void getStudentID(int)const;
	void getStudentRGZ(int)const;
	~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; }

const char* Task::getMaleOrFemale() const { return maleOrFemale; }
```

```
	void Task::setMaleOrFemale(const char* male_or_female1) { maleOrFemale = male_or_female1; }

	int Task::getRgz() const { return rgz; }
	void Task::setRgz(int rgz1) { rgz = rgz1; }

	Task CreateStudent()
	{
		Task stud;
		stud.setMark(5);
		stud.setCountOfDoneExercises(5);
		stud.setMaleOrFemale("Male");
		return stud;
	}
	Task CreateStudent2()
	{
		Task stud;
		stud.setMark(2);
		stud.setCountOfDoneExercises(2);
		stud.setMaleOrFemale("Female");
		return stud;
	}

	Task::Task(char* male_or_female, int mark, int count_of_done_exercises, int student_index, int rgz) :mark(mark), countOfDoneExercises(count_of_done_exercises), studentIndex(student_index), maleOrFemale(male_or_female), rgz(rgz)
	{
		cout << "Constructor with parameter" << endl;
	}
	Task::Task() : mark(0), countOfDoneExercises(0), studentIndex(0), maleOrFemale("None"), rgz(0)
	{
		cout << "Default constructor" << endl;
	}
	Task::Task(const Task& stud) : mark(stud.mark), countOfDoneExercises(stud.countOfDoneExercises), studentIndex(stud.studentIndex), maleOrFemale(stud.maleOrFemale), rgz(stud.rgz)
	{
		cout << "Copy constructor" << endl;
	}
	Task::~Task()
	{
		cout << "Destructor Task" << endl;
	}
```

List.cpp

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

int List::getListSize() const { return listSize; }
void List::setListSize(int size) { listSize = size; }

void List::addStudent(const Task task)
{
	setListSize(getListSize() + 1);
	Task* newstud = new Task[listSize];
	for (int i = 0; listSize > i; i++)
		newstud[i] = stud[i];
	newstud[listSize - 1] = task;
	newstud[listSize - 1].setStudentIndex(stud[listSize - 2].getStudentIndex() +
1);
	newstud[listSize - 1].setRgz(stud[listSize - 2].getRgz() + 1);
	delete[] stud;
	stud = new Task[listSize];
	for (int i = 0; listSize > i; i++)
		stud[i] = newstud[i];
	delete[] newstud;
}
void List::createList()
{
	stud = new Task[listSize];
	for (int i = 0; listSize > i; i++)
	{
		stud[i] = CreateStudent();
		stud[i].setStudentIndex(i + 1);
		stud[i].setRgz(i + 1);
	}
}
void List::deleteStudent(int c)
{
	setListSize(getListSize() - 1);
	Task* newstud = new Task[listSize];
	int 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 (int i = 0; i < getListSize(); i++)
		stud[i] = newstud[i];
	delete[] newstud;
}
void List::printOneStudent(int number) const
{
	cout << "Index\t";
	cout << "Mark\t";
	cout << "Exercises\t";
	cout << "RGZ\t";
	cout << "Male/Female" << endl;
	printf("%-10d", stud[number].getStudentIndex());
	printf("%-10d", stud[number].getMark());
	printf("%-13d", stud[number].getCountOfDoneExercises());
	printf("%-10d", stud[number].getRgz());
	printf("%s", stud[number].getMaleOrFemale());
}
void List::printAll() const
{
	cout << "Index\t";
	cout << "Mark\t";
	cout << "Exercises\t";
	cout << "RGZ\t";
	cout << "Male/Female";
	for (int i = 0; List::getListSize() > i; i++)
	{
		cout << endl;
		printf("%-10d", stud[i].getStudentIndex());
		printf("%-10d", stud[i].getMark());
		printf("%-13d", stud[i].getCountOfDoneExercises());
		printf("%-10d", stud[i].getRgz());
		printf("%s", stud[i].getMaleOrFemale());
	}
}
List::~List()
{
	cout << "Destructor List" << endl;
	delete[] stud;
}

void List::getStudentID(int id) const
```

```
{
	for (int i = 0; i < listSize; i++)
		if (stud[i].getStudentIndex() == id)
		{
			printOneStudent(i);
			return;
		}
	cout << "Wrong ID" << endl;
}

void List::getStudentRGZ(int a) const
{
	for (int i = 0; i < listSize; i++)
		if (stud[i].getRgz() == a)
		{
			printOneStudent(i);
			return;
		}
	cout << "Wrong count of RGZ" << endl;
}
```

Menu.cpp

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

void Menu()
{
	List list;
	int c = 0, a = 0;
	int count_of_students = 2;
	int menu_number = 1;
	int delete_number;
	list.setListSize(count_of_students);
	list.createList();
	while (menu_number)
	{
		menu_number = 0;
		cout << endl << "Menu:" << endl;
		cout << "1.Add a new student" << endl;
		cout << "2.Delete one student" << endl;
		cout << "3.Show all student" << endl;
		cout << "4.Show student via his index" << endl;
		cout << "5.End program" << endl;
		cout << "6.Individual task" << endl;
		cin >> menu_number;
```

```
                switch (menu_number)
                {
                case 1:
                        list.addStudent(CreateStudent2());
                        break;
                case 2:
                        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 (int i = 0; list.getListSize() > i; i++)
                                if (delete_number == list.stud[i].getStudentIndex())
                                {
                                        list.deleteStudent(delete_number);
                                        break;
                                }
                        break;
                case 3:
                        list.printAll();
                        break;
                case 4:
                        cout << "Enter a index of student:";
                        cin >> c;
                        list.getStudentID(c);
                        break;
                case 5:
                        menu_number = 0;
                        break;
                case 6:
                        cout << "Enter a count of RGZ";
                        cin >> a;
                        list.getStudentRGZ(a);
                        break;
                default:
                        cout << "You have chosen the wrong number of the menu";
                        break;
                }
        }
        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;
}
```

Test.cpp

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

int TestAddStudent(List&, int);
int TestDeleteStudent(List&, int);

int main()
{
	{
		List test;
		int count = 0;
		test.setListSize(2);
		test.createList();

		count = TestAddStudent(test, count);
		count = TestDeleteStudent(test, count);

		if (count == 2)
			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;
}
```

4. Результати роботи програми

```
Default constructor
Default constructor
Default constructor
Copy constructor
Destructor Task
Destructor Task
Default constructor
Copy constructor
Destructor Task
Destructor Task

Menu:
1.Add a new student
2.Delete one student
3.Show all student
4.Show student via his index
5.End program
6.Individual task
3
Index   Mark    Exercises       RGZ     Male/Female
1         5         5             1         Male
2         5         5             2         Male
Menu:
1.Add a new student
2.Delete one student
3.Show all student
4.Show student via his index
5.End program
6.Individual task
6
Enter a count of RGZ2
Index   Mark    Exercises       RGZ     Male/Female
2         5         5             2         Male
Menu:
1.Add a new student
2.Delete one student
3.Show all student
4.Show student via his index
5.End program
6.Individual task
5
Destructor List
Destructor Task
Destructor Task

There is no memory leak
Для продолжения нажмите любую клавишу . . .
```

```
Default constructor
Default constructor
Default constructor
Copy constructor
Destructor Task
Destructor Task
Default constructor
Copy constructor
Destructor Task
Destructor Task
Default constructor
Copy constructor
Destructor Task
Default constructor
Default constructor
Default constructor
Destructor Task
Destructor Task
Default constructor
Default constructor
Default constructor
Destructor Task
Destructor Task
Destructor Task
Destructor Task

Test: Add_student - successful
Default constructor
Default constructor
Destructor Task
Destructor Task
Destructor Task
Default constructor
Default constructor
Destructor Task
Destructor Task

Test: Delete_student - successful

All tests are successful
Destructor List
Destructor Task
Destructor Task

There is no memory leak
Для продолжения нажмите любую клавишу . . .
```

5. Висновки

Було поширено попередню лабораторну роботу, додано поле char*, конструктори(за змовчуванням, з аргументами, копіювання), реалізація конструкторів показана за допомогою списків ініціалізації, деструктори, а також метод для виконання індивідуального завдання.

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