ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | А. В. Устинов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| КЛАССЫ, КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ, ОДИНОЧНОЕ НАСЛЕДОВАНИЕ, ПРАВА ДОСТУПА |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**1. Цель работы**

Изучить принципы создания классов, наследования и ограничения прав доступа к полям и методам класса, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Закрепить знания по теме: Классы, конструкторы и деструкторы, одиночное наследование, права доступа.

**Вариант 8**

Создать класс «Vector» для определения одномерных массивов целых чисел (векторов). Должны быть реализованы методы:

* Обращения к отдельному элементу массива с контролем выхода за пределы массива;
* Возможность задания произвольных границ индексов при создании объекта;
* Возможность выполнения операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов;
* Умножения и деления всех элементов массива на скаляр;

**2. Листинг программы**

#define \_CRTDBG\_MAP\_ALLOC **MAIN.GO**

#include <stdlib.h>

#include <crtdbg.h>

#ifdef \_DEBUG

#ifndef DBG\_NEW

#define DBG\_NEW new ( \_NORMAL\_BLOCK , \_\_FILE\_\_ , \_\_LINE\_\_ )

#define newDBG\_NEW

#endif

#endif

#include <iostream>

#include "Vector.h"

using namespace std;

void check() {

Vector vec3(3);

Vector vecl(2);

vecl[0] = 1;

vecl[1] = 2;

Vector vecr(3, 5);

vecr[3] = 5;

vecr[4] = 7;

// vecr[5] = 5; // out of range

cout << "vecl: "; vecl.print();

cout << "vecr: "; vecr.print();

cout << "vecl len: "<< vecl.getLen() << endl;

Vector sum = vecl + vecr;

//sum = vecl + vec3; // different length

cout << "sum: "; sum.print();

Vector subtraction = vecr - vecl;

cout << "subtraction: "; subtraction.print();

Vector multipl = vecr \* 4;

cout << "multipl: "; multipl.print();

Vector division = multipl / 2;

cout << "division: "; division.print();

}

int main() {

check();

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT);

\_CrtDumpMemoryLeaks();

system("pause");

return 0;

}

#pragma once **VECTOR.H**

#include <iostream>

#include <memory>

class Vector

{

public:

Vector(int l);

Vector(int st, int ed);

int & operator[](int i);

Vector operator+(const Vector & rv);

Vector operator-(const Vector & rv);

Vector operator\*(int i);

Vector operator/(const int m);

Vector(Vector const &);

int getLen();

void print();

private:

std::unique\_ptr<int[]> arr;

std::unique\_ptr<int> start, end, len;

};

#include "Vector.h" **VECTOR.CPP**

Vector::Vector(int l) : Vector(0, l) {}

Vector::Vector(int st, int ed) : start(new int(st)), end(new int(ed)) {

len.reset(new int(ed - st));

arr.reset(new int[\*len]);

}

int& Vector::operator[] (int i) {

if (i < \*start || i >= \*end) {

std::cout << std::endl << "Wrong index of massive: " << i << std::endl;

system("pause");

exit(1);

}

return arr[i - \*start];

}

Vector Vector::operator+(const Vector& rv) {

if (\*len != \*rv.len) {

std::cout << std::endl << "Different length of vectors." << std::endl;

system("pause");

exit(1);

}

Vector out(\*len);

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

out.arr[i] = arr[i] + rv.arr[i];

}

return out;

}

Vector Vector::operator-(const Vector& rv) {

if (\*len != \*rv.len) {

std::cout << std::endl << "Different length of vectors." << std::endl;

system("pause");

exit(1);

}

Vector out(\*len);

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

out.arr[i] = arr[i] - rv.arr[i];

}

return out;

}

Vector Vector::operator\* (const int m) {

Vector out(\*len);

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

out.arr[i] = arr[i] \* m;

}

return out;

}

Vector Vector::operator/ (const int m) {

if (m == 0) {

std::cout << std::endl << "Division by zero." << std::endl;

system("pause");

exit(1);

}

Vector out(\*len);

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

out.arr[i] = arr[i] / m;

}

return out;

}

Vector::Vector(Vector const & obj) {

start.reset(new int(\*obj.start));

end.reset(new int(\*obj.end));

len.reset(new int(\*obj.len));

arr.reset(new int[\*len]);

memcpy(arr.get(), obj.arr.get(), sizeof(int) \* \*len);

}

int Vector::getLen() {

return \*len;

}

void Vector::print() {

std::cout << "[";

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

if (i != 0)

std::cout << ", ";

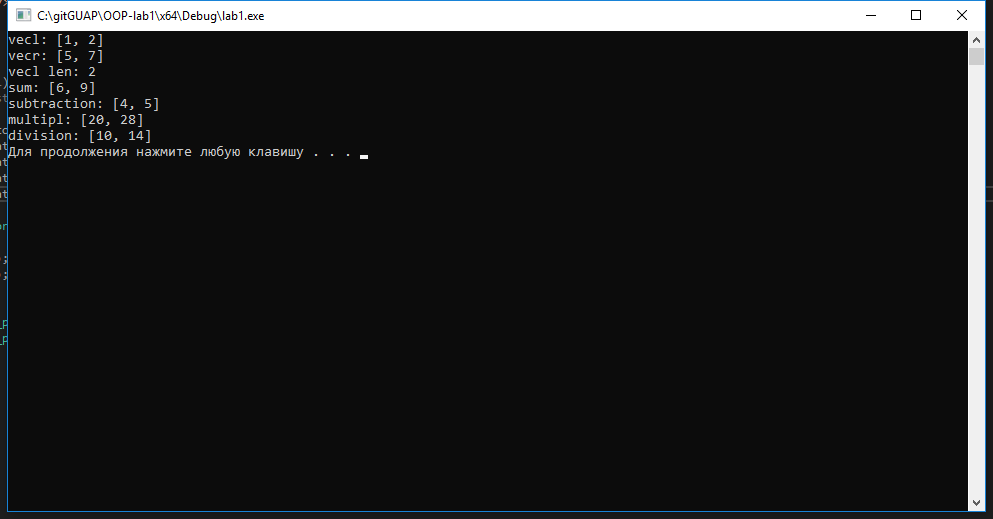
std::cout << arr[i];

}

std::cout << "]" << std::endl;

}

**3. Пример работы**



**4. Вывод**

Во время работы выполнения лабораторной работы я познакомился AVL-деревьями. Научился добавлять или удалять элементы из него. Научился балансировать дерево.