Федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича»

(СПбГУТ)

Кафедра Программной инженерии и вычислительной техники

**Лабораторная работа №1 по Объектно-ориентированному программированию**

**«Классы»**

**Работу выполнил:**

Студент 2-го курса

Группа ИКПИ-73

Сударев Андрей Владимирович

**Приняла:**

Петрова Ольга Борисовна

Санкт-Петербург

2018 уч. г.

**Цель работы**

Цель настоящей работы состоит в ознакомлении студента с правилами организации классов, принятыми при программировании на языке С++. В процессе выполнения настоящей работы каждый студент должен разработать два класса и написать тестовые программы для демонстрации их работоспособности.

**Вариант 20 (задания 8 и 14)**

Необходимо разработать класс, который может рассматриваться как улучшенный одномерный массив. Одним из “улучшений”, которые должны быть предусмотрены в разрабатываемых классах, является наличие так называемого “счетного” режима. При работе в счетном режиме производится подсчет количества элементов, которые хранятся в массиве. Для реализации такого режима в классе следует предусмотреть специальное поле.

Реализация такого класса предусматривает использование динамического массива. В **задаче № 8** следует создать одномерный массив. Конструктор такого массива должен задавать минимальное и максимальное значения индексов.

Разрабатываемый класс должен предусматривать выполнение следующих операций:

* копирование массивов,
* ввод элементов массива с клавиатуры,
* вывод элементов массива на экран дисплея,
* добавка элемента в конец занятой части массива,
* удаление части массивов; удаляемые элементы задаются значением индекса, с которого необходимо начать удаление и количеством удаляемых элементов,
* вычисление произведения элементов массива.

В **задаче 14** требуется разработать класс, обеспечивающий работу с рациональными числами.

Разработанный класс (Rational) должен обеспечить выполнение следующих операций:

* конструктор умолчания.
* конструктор с параметрами.
* сложение,
* вычитание,
* умножение,
* деление,
* сравнения на равенство и неравенство.
* отношения.
* вывод рационального числа на экран дисплея.

Замечание. После выполнения арифметических операций результат должен быть преобразован к редуцированной форме.

**Таблица идентификаторов**

**Задание №8**

|  |  |  |
| --- | --- | --- |
| Номер | Идентификатор | Назначение |
| 1 | Private: int \*Mass | Массив |
| 2 | Private: int Len | Длина массива |
| 3 | void cpy\_mass(const int \*, int) | Копирование массива с длинной len |
| 4 | void paste\_mass(int \*, int, int); | Вставить в массив \*a размером len len\_inp значений |
| 5 | void input\_mass(int); | Добавить в конец массива int len чисел с клавиатуры |
| 6 | void display\_mass(); | Вывести на экран массив |
| 7 | void appiend(int); | Добавить в конец массива int число |
| 8 | void erase(int, int); | удаление части массива «с» и «по» |
| 9 | int mul\_mass(); | Вернуть произведение всех элементов массива |
| 10 | int size\_mass(); | Вернуть размер массива |

**Задание №14**

|  |  |  |
| --- | --- | --- |
| Номер | Идентификатор | Назначение |
| 1 | int drob[2]; | Числитель [0] и знаменатель [1] |
| 2 | void raz(int \*a, int \*b) | Привести рациональные числа a и b к общему знаменателю |

**Программа на С++**

**Задача 8**

**Main.cpp**

#include "stdafx.h"

int main()

{

setlocale(0, "");

int mass[] = { 1,2,3,4,5,6,7,8,9,10,11,12 };

math\_mass mat(mass, 12);

mat.appiend(13);

mat.appiend(14);

mat.appiend(15);

mat.erase(3, 7);

mat.display\_mass();

cout << "Размер массива mat = " << mat.size\_mass() << endl;

cout << "Произведение всех элементов массива mat = " << mat.mul\_mass() << endl << endl;

math\_mass mat2;

mat2.input\_mass(9);

mat2.erase(3, 2);

mat2.display\_mass();

cout << "Размер массива mat2 = " << mat2.size\_mass() << endl;

cout << "Произведение всех элементов массива mat2 = " << mat2.mul\_mass() << endl;

math\_mass cpy(mat2);

cpy.erase(3, 15);

cpy.display\_mass();

return 0;

}

**Math\_mass.h**

#pragma once

class math\_mass

{

private:

int \*mass, len;

public:

math\_mass();

math\_mass(const int\*, int);

math\_mass(const math\_mass &);

void cpy\_mass(const int \*, int);

void paste\_mass(int \*, int, int);

void input\_mass(int);

void display\_mass();

void appiend(int);

void erase(int, int);

int mul\_mass();

int size\_mass();

~math\_mass();

};

**Math\_mass.h**

#include "stdafx.h"

#include "math\_mass.h"

math\_mass::math\_mass()

{

mass = new int[1];

len = 1;

}

math\_mass::math\_mass(const int \*a, int len)

{

this->len = len;

mass = new int[len];

for (int i = 0; i < len; i++)

mass[i] = a[i];

}

math\_mass::math\_mass(const math\_mass & a)

{

len = a.len;

mass = new int[len];

for (int i = 0; i < len; i++)

mass[i] = a.mass[i];

}

void math\_mass::cpy\_mass(const int \*a, int len)

{

delete[] mass;

int \*buf = new int[len];

for (int i = 0; i < len; i++)

buf[i] = a[i];

mass = buf;

this->len = len;

}

void math\_mass::paste\_mass(int \*a, int len, int inp\_len)

{

for (int i = 0; i < len && i < this->len && i < inp\_len; i++)

a[i] = mass[i];

}

void math\_mass::input\_mass(int len) {

delete[] mass;

this->len = len;

int \*buf = new int[len];

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

cout << "Элемент[" << i << "] = ";

cin >> buf[i];

}

mass = buf;

}

void math\_mass::display\_mass() {

for (int i = 0; i < len; i++)

cout << "math\_mass[" << i << "] = " << mass[i] << endl;

}

void math\_mass::appiend(int sim)

{

len += 1;

int \*buf = new int[len];

for (int i = 0; i < len - 1; i++)

buf[i] = mass[i];

buf[len - 1] = sim;

cpy\_mass(buf, len);

}

void math\_mass::erase(int pos, int l)

{

if (pos + l >= len)

l = len - pos;

int \*buf = new int[len - l];

int i = 0;

for (; i < pos; i++)

buf[i] = mass[i];

for (int j = pos + l; j < len; j++, i++)

buf[i] = mass[j];

len -= l;

cpy\_mass(buf, len);

}

int math\_mass::mul\_mass() {

int res = 1;

for (int i = 0; i < len; i++)

res \*= mass[i];

return res;

}

int math\_mass::size\_mass() {

return len;

}

math\_mass::~math\_mass()

{

delete[] mass;

}

**stdafx.h**

#pragma once

#include "targetver.h"

#include <stdio.h>

#include <tchar.h>

#include <iostream>

#include <string>

#include <clocale>

#include "math\_mass.h"

using namespace std;

**Задача 14**

**Main.cpp**

#include "stdafx.h"

int main()

{

setlocale(0, "");

Rational a(3, 10), b(2, 10), c;

c = a + b + a + b;

cout << "C:\n";

c.display();

cout << "\nA:\n";

a.display();

cout << "\nB:\n";

b.display();

return 0;

}

**Rational.h**

#pragma once

class Rational

{

private:

int drob[2];

public:

Rational();

Rational(int, int);

Rational(Rational &);

int getChisl();

int getZnam();

int gcd(int, int);

void raz(int\*, int\*);

void display();

void input\_Chisl(int);

void input\_Znam(int);

void sum(const Rational);

void sub(const Rational);

void mul(const Rational);

void div(const Rational);

double ratio();

Rational operator+(Rational);

Rational operator-(Rational);

Rational operator\*(Rational);

Rational operator/(Rational);

bool operator>(Rational);

bool operator<(Rational);

bool operator==(Rational);

bool operator!=(Rational);

bool operator>=(Rational);

bool operator<=(Rational);

~Rational();

};

**Rational.cpp**

#include "stdafx.h"

#include "Rational.h"

Rational::Rational()

{

drob[0] = 1;

drob[1] = 1;

}

Rational::Rational(int a, int b)

{

drob[0] = a;

drob[1] = b;

}

Rational::Rational(Rational & ch)

{

drob[0] = ch.drob[0];

drob[1] = ch.drob[1];

}

Rational::~Rational()

{

}

int Rational::gcd(int a, int b)

{

while (a != 0 && b != 0) {

if (a > b) {

a = a % b;

}

else b = b % a;

}

return a + b;

}

int Rational::getChisl(void)

{

return drob[0];

}

int Rational::getZnam(void)

{

return drob[1];

}

void Rational::raz(int \*a, int \*b)

{

int nok = gcd(a[1], b[1]);

int buf1, buf2;

buf1 = a[1] / nok;

buf2 = b[1] / nok;

a[0] \*= buf2;

a[1] \*= buf2;

b[0] \*= buf1;

b[1] \*= buf1;

}

void Rational::display(void)

{

cout << drob[0] << endl << "-\n" << drob[1] << endl;

}

void Rational::input\_Chisl(int a)

{

drob[0] = a;

}

void Rational::input\_Znam(int b)

{

drob[1] = b;

}

void Rational::sum(const Rational drob2)

{

if (drob[1] != 0 && drob2.drob[1] != 0)

{

int buf[2];

buf[0] = drob2.drob[0];

buf[1] = drob2.drob[1];

raz(drob, buf);

drob[0] += buf[0];

int nok = gcd(drob[0], drob[1]);

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

void Rational::sub(const Rational drob2)

{

if (drob[1] != 0 && drob2.drob[1] != 0)

{

int buf[2];

buf[0] = drob2.drob[0];

buf[1] = drob2.drob[1];

raz(drob, buf);

drob[0] -= buf[0];

int nok = gcd(drob[0], drob[1]);

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

void Rational::mul(const Rational drob2)

{

if (drob[1] != 0 && drob2.drob[1] != 0)

{

drob[0] \*= drob2.drob[0];

drob[1] \*= drob2.drob[1];

int nok = gcd(drob[0], drob[1]);

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

void Rational::div(const Rational drob2)

{

if (drob[1] != 0 && drob2.drob[0] != 0)

{

drob[0] \*= drob2.drob[1];

drob[1] \*= drob2.drob[0];

int nok = gcd(drob[0], drob[1]);

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

double Rational::ratio()

{

if (drob[1] != 0)

return drob[0] / (double)drob[1];

else

return 0;

}

Rational Rational::operator+(Rational drob2)

{

Rational Temp = drob2;

if (drob[1] != 0 && Temp.drob[1] != 0)

{

int buf[2];

buf[0] = drob[0];

buf[1] = drob[1];

raz(Temp.drob, buf);

Temp.drob[0] += buf[0];

int nok = gcd(Temp.drob[0], Temp.drob[1]);

Temp.drob[0] /= nok;

Temp.drob[1] /= nok;

}

else

{

Temp.drob[0] = 1;

Temp.drob[1] = 1;

}

return Temp;

}

Rational Rational::operator-(Rational drob2)

{

Rational Temp = drob2;

if (drob[1] != 0 && Temp.drob[1] != 0)

{

int buf[2];

buf[0] = drob[0];

buf[1] = drob[1];

raz(Temp.drob, buf);

Temp.drob[0] = buf[0] - Temp.drob[0];

int nok = gcd(Temp.drob[0], Temp.drob[1]);

Temp.drob[0] /= nok;

Temp.drob[1] /= nok;

}

else

{

Temp.drob[0] = 1;

Temp.drob[1] = 1;

}

return Temp;

}

Rational Rational::operator\*(Rational drob2)

{

Rational Temp = drob2;

if (drob[1] != 0 && Temp.drob[1] != 0)

{

Temp.drob[0] \*= drob[0];

Temp.drob[1] \*= drob[1];

int nok = gcd(Temp.drob[0], Temp.drob[1]);

Temp.drob[0] /= nok;

Temp.drob[1] /= nok;

}

else

{

Temp.drob[0] = 1;

Temp.drob[1] = 1;

}

return Temp;

}

Rational Rational::operator/(Rational drob2)

{

Rational Temp = drob2;

if (drob[1] != 0 && Temp.drob[0] != 0)

{

int buf;

buf = drob[0] \* Temp.drob[1];

Temp.drob[1] = drob[1] \* Temp.drob[0];

Temp.drob[0] = buf;

int nok = gcd(Temp.drob[0], Temp.drob[1]);

Temp.drob[0] /= nok;

Temp.drob[1] /= nok;

}

else

{

Temp.drob[0] = 1;

Temp.drob[1] = 1;

}

return Temp;

}

bool Rational::operator>(Rational drob2)

{

if (ratio() > drob2.ratio())

return true;

else return false;

}

bool Rational::operator<(Rational drob2)

{

if (ratio() < drob2.ratio())

return true;

else return false;

}

bool Rational::operator==(Rational drob2)

{

if (ratio() == drob2.ratio())

return true;

else return false;

}

bool Rational::operator>=(Rational drob2)

{

if (ratio() >= drob2.ratio())

return true;

else return false;

}

bool Rational::operator<=(Rational drob2)

{

if (ratio() <= drob2.ratio())

return true;

else return false;

}

bool Rational::operator!=(Rational drob2)

{

if (ratio() != drob2.ratio())

return true;

else return false;

}

**stdafx.h**

#pragma once

#include "targetver.h"

#include <stdio.h>

#include <tchar.h>

#include <iostream>

#include <clocale>

#include "Rational.h"

using namespace std;