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

**Лабораторные работы №5 по Объектно-ориентированному программированию**

**«Шаблоны классов. Работа с исключительными ситуациями языка С++»**

**Работу выполнил:**

Студент 2-го курса

Группа ИКПИ-73

Сударев Андрей Владимирович

**Приняла:**

Петрова Ольга Борисовна

Санкт-Петербург

2018 уч. г.

**Цель работы**

В настоящей лабораторной работе необходимо решить две задачи, связанные с организацией шаблонов классов. Первая из задач состоит в преобразовании в шаблон класс того числового класса, который был разработан студентом в первой лабораторной работе по ООП. Вторая задача состоит в разработке шаблона контейнера. При решении второй задачи следует предусмотреть обработку исключительных ситуаций.

**Вариант 3**

Разработать шаблон класса для стека, построенного на основе массива с фиксированными размерами.

**Таблица идентификаторов**

**Задание №2**

|  |  |  |
| --- | --- | --- |
| Номер | Идентификатор | Назначение |
| 1 | Mass | Массив |
| 2 | size | Длина массива |
| 3 | void push(S) | Вставить элемент в конец стека |
| 4 | void pop(void) | Удалить 1 элемент из стека |
| 5 | void printStack(void) | Вывести стек на экран |
| 6 | int getSize(void) | Получить размер стека |
| 7 | S get(void) | Получить и удалить последний элемент из стека |

**Программа на С++**

**Задание №1**

**Main.cpp**

#include "stdafx.h"

int main()

{

Rational<int> numeric(4, 5), flying(6, 18);

Rational<long> rocket(11, 32), sun(99, 102);

numeric.sum(flying);

rocket.mul(sun);

cout << "Numeric: \n";

numeric.display();

cout << "Rocket: \n";

rocket.display();

cout << "Home: \n";

//home.gcd();

return 0;

}

**Rational.h**

#pragma once

template <typename T>

class Rational

{

private:

T drob[2];

public:

Rational();

Rational(T, T);

Rational(Rational &);

T getChisl();

T getZnam();

T gcd(T, T);

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

void display();

void input\_Chisl(T);

void input\_Znam(T);

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();

};

template <typename T>

Rational<T>::Rational()

{

drob[0] = 1;

drob[1] = 1;

}

template <typename T>

Rational<T>::Rational(T a, T b)

{

drob[0] = a;

drob[1] = b;

}

template <typename T>

Rational<T>::Rational(Rational & ch)

{

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

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

}

template <typename T>

Rational<T>::~Rational()

{

}

template <typename T>

T Rational<T>::gcd(T a, T b)

{

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

if (a > b) {

a = a % b;

}

else b = b % a;

}

return a + b;

}

template <typename T>

T Rational<T>::getChisl(void)

{

return drob[0];

}

template <typename T>

T Rational<T>::getZnam(void)

{

return drob[1];

}

template <typename T>

void Rational<T>::raz(T \*a, T \*b)

{

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

T buf1, buf2;

buf1 = a[1] / nok;

buf2 = b[1] / nok;

a[0] \*= buf2;

a[1] \*= buf2;

b[0] \*= buf1;

b[1] \*= buf1;

}

template <typename T>

void Rational<T>::display(void)

{

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

}

template <typename T>

void Rational<T>::input\_Chisl(T a)

{

drob[0] = a;

}

template <typename T>

void Rational<T>::input\_Znam(T b)

{

drob[1] = b;

}

template <typename T>

void Rational<T>::sum(const Rational drob2)

{

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

{

T buf[2];

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

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

raz(drob, buf);

drob[0] += buf[0];

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

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

template <typename T>

void Rational<T>::sub(const Rational drob2)

{

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

{

T buf[2];

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

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

raz(drob, buf);

drob[0] -= buf[0];

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

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

template <typename T>

void Rational<T>::mul(const Rational drob2)

{

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

{

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

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

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

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

template <typename T>

void Rational<T>::div(const Rational drob2)

{

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

{

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

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

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

drob[0] /= nok;

drob[1] /= nok;

}

else

{

drob[0] = 1;

drob[1] = 1;

}

}

template <typename T>

double Rational<T>::ratio()

{

if (drob[1] != 0)

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

else

return 0;

}

template <typename T>

Rational<T> Rational<T>::operator+(Rational drob2)

{

Rational Temp = drob2;

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

{

T buf[2];

buf[0] = drob[0];

buf[1] = drob[1];

raz(Temp.drob, buf);

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

T 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;

}

template <typename T>

Rational<T> Rational<T>::operator-(Rational drob2)

{

Rational Temp = drob2;

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

{

T buf[2];

buf[0] = drob[0];

buf[1] = drob[1];

raz(Temp.drob, buf);

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

T 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;

}

template <typename T>

Rational<T> Rational<T>::operator\*(Rational drob2)

{

Rational Temp = drob2;

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

{

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

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

T 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;

}

template <typename T>

Rational<T> Rational<T>::operator/(Rational drob2)

{

Rational Temp = drob2;

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

{

T buf;

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

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

Temp.drob[0] = buf;

T 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;

}

template <typename T>

bool Rational<T>::operator>(Rational drob2)

{

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

return true;

else return false;

}

template <typename T>

bool Rational<T>::operator<(Rational drob2)

{

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

return true;

else return false;

}

template <typename T>

bool Rational<T>::operator==(Rational drob2)

{

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

return true;

else return false;

}

template <typename T>

bool Rational<T>::operator>=(Rational drob2)

{

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

return true;

else return false;

}

template <typename T>

bool Rational<T>::operator<=(Rational drob2)

{

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

return true;

else return false;

}

template <typename T>

bool Rational<T>::operator!=(Rational drob2)

{

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

return true;

else return false;

}

**stdafx.h**

#pragma once

#include "targetver.h"

#include <iostream>

using namespace std;

#include "Rational.h"

**Задание №2**

**Main.cpp**

#include "stdafx.h"

int main()

{

try

{

Stack<int> one;

for (int i = 0; i < 100; i++)

one.push(i);

one.printStack();

for (int i = 0; i < 100; i++)

one.pop();

}

catch (const char\* msg)

{

cout << msg << endl;

}

return 0;

}

**stack.h**

#pragma once

template <class S>

class Stack

{

private:

S mass[64];

int size = 0;

public:

Stack();

~Stack();

void push(S);

void pop(void);

void printStack(void);

int getSize(void);

S get(void);

};

template <class S>

Stack<S>::Stack()

{

}

template <class S>

Stack<S>::~Stack()

{

}

template <class S>

void Stack<S>::push(S znach)

{

if (size < 63)

{

size++;

mass[size] = znach;

}

else throw "Stack is full\n";

}

template <class S>

void Stack<S>::pop(void)

{

if (size > -1)

{

mass[size] = 0;

size--;

}

else throw "Stack is empty\n";

}

template <class S>

void Stack<S>::printStack(void)

{

if (size > -1)

for (int i = size; i > 0; i--)

cout << mass[i] << endl;

}

template <class S>

S Stack<S>::get(void)

{

if (size > -1)

{

size--;

return mass[size + 1];

}

}

template <class S>

int Stack<S>::getSize(void)

{

return size;

}

**stdafx.h**

#pragma once

#include "targetver.h"

#include <iostream>

using namespace std;

#include "stack.h"