МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»

Лабораторная работа № 2

по дисциплине «Объектно-ориентированное программирование»

Направление подготовки: 230105 - «Программное обеспечение вычислительной техники и автоматизированных систем»

Выполнил слушатель: Бройтман Е.Д.
Вариант: №5
Дата сдачи:
Преподаватель: Силов Я.В.

Новосибирск, 2016г.

1. Задание

1) Программу, разработанную в лабораторной работе № 1, модифицировать таким образом, чтобы появилась возможность применять к объектам класса следующие операторы:

- Инкремент, декремент (можно только одну из префиксной и постфиксной форм).
- Сложение объектов класса.
- Операторы сравнения.
- Оператор присваивания.
- Оператор приведения к типу int (или другому численному).
- Операторы ввода и вывода в поток.

ПРИМЕЧАНИЕ: логика использования оператора выбирается самостоятельно. Неуместные для данного класса операторы можно не реализовывать, заменив их какими - то другими. Общее количество перегруженных в классе операторов – около 8 - 10.

2) Интерфейс пользователя должен обеспечивать :

- В начале работы программы создавать константный эталонный объект, и предоставлять пользователю возможность выбора операции сравнения. Эта операция применяется к эталонному объекту и всем остальным. Результаты выводить в виде таблицы. Таблица может выглядеть так (для класса время):
  [1] 00 : 00 : 00 > 12:00 : 00 false
  [2] 17 : 10 : 00 > 12:00 : 00 true
  [3] 11 : 11 : 00 > 12:00 : 00 false
  …..
  Но предпочтительней предусмотреть в этом цикле применение нескольких операторов сравнения с целью вывода корректного результата:
  [1] 00 : 00 : 00 < 12 : 00 : 00
  [2] 17 : 10 : 00 > 12 : 00 : 00
  [3] 12 : 00 : 00 = 12 : 00 : 00
- Предусмотреть возможность создания нового объекта, как результата выполнения математической операции над одним из имеющихся. Операндами являются уже существующие в массиве объекты, либо с фиксированными номерами (0-й для унарных операторов, 0-й и 1-й для бинарных), либо номера могут запрашиваться с клавиатуры.

2. Структурное описание

В данной лабораторной работе мы к классу **Fraction** добавляем перегруженные операторы.

Перегрузка операторов осуществляется 2-мя способами:

- С помощью методов класса.
- С помощью дружественных функций.

Для перегрузки операторов с помощью методов класса использован существующий файл «Fraction.cpp». Дружественные функции реализованы во вновь созданном файле «FractionFriends.cpp».

В заголовочном файле «Fraction.h» интерфейс (тело) класса дополнен методами перегрузки операторов**,** в файле «Fraction.cpp» реализованы методы класса для перегрузки операторов, в файле ”Menu.cpp” реализована проверка перегрузки операторов, в файле «FractionFriends.cpp» реализованы дружественные классу **Fraction** функции перегрузки операторов, первый операнд которых не является объектом класса. Функционал ос-

тальных файлов и перечисленных выше файлов, не связанный с перегрузкой операторов, не изменился.

Перегрузка математических операторов:
Fraction operator+(Fraction &Addendum) - перегрузка оператора сложения;

Перегрузка логических операторов:
bool operator==(Fraction &FractionRight)const - перегрузка оператора сравнения равно;
bool operator!=(Fraction &FractionRight)const - перегрузка оператора сравнения не равно;
bool operator>(Fraction &FractionRight)const - перегрузка оператора сравнения больше;
bool operator>=(Fraction &FractionRight)const - перегрузка оператора сравнения больше или равно;
bool operator<(Fraction &FractionRight)const - перегрузка оператора сравнения меньше;
bool operator<=(Fraction &FractionRight)const - перегрузка оператора сравнения меньше или равно;

Перегрузка специальных операторов:
Fraction & operator=(Fraction &FractionRight) - перегрузка оператора присваивания;
operator double() - приведение к типу double;
friend std::ostream & operator<<(std::ostream &out, Fraction &FractionOut) - перегрузка оператора вывода в поток;
friend std::istream& operator >> (std::istream &in, Fraction &FractionIn) - перегрузка оператора ввода из потока.

3. Функциональное описание

Рассмотрим реализацию методов и функций перегрузки операторов.
1. Fraction operator+(Fraction &Addendum) - перегрузка оператора сложения. Первый операнд указатель this. В качестве параметров принимает по ссылке объект класса Fraction &Addendum. Создается новый объект класса Fraction **Temp.** В объект **Temp** записывается результат сложения двух операндов. Далее вызывается метод сокращения дробей **Reduction()**. В качестве результата оператор **return** возвращает значение объекта **Temp**.
2. bool operator==(Fraction &FractionRight)const - перегрузка оператора сравнения равно. Результат логическое значение true или false. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд). В условии if проверяем равенство значений левого и правого операндов. Дроби с учетом знаков приводятся к общему знаменателю и сравниваются их числители. Если условие выполняется, возвращаем значение true, иначе false.
3. bool operator!=(Fraction &FractionRight)const - перегрузка оператора сравнения не равно. Результат логическое значение true или false. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд).Используем перегруженный оператор ==. Если условие выполняется, возвращаем значение true, иначе false.
4. bool operator>(Fraction &FractionRight)const - перегрузка оператора сравнения больше. Результат логическое значение true или false. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд). Дроби с учетом знаков

приводятся к общему знаменателю и сравниваются их числители по условию >. Если условие выполняется возвращаем значение true, иначе false.
5. bool operator>=(Fraction &FractionRight)const - перегрузка оператора сравнения больше или равно. Результат логическое значение true или false. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд). Дроби с учетом знаков приводятся к общему знаменателю и сравниваются их числители по условию >=. Если условие выполняется возвращаем значение true, иначе false.
6. bool operator<(Fraction &FractionRight)const - перегрузка оператора сравнения меньше. Результат логическое значение true или false. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд).Используем перегруженный оператор >=. Если условие выполняется, возвращаем значение true, иначе false.
7. bool operator<=(Fraction &FractionRight)const - перегрузка оператора сравнения меньше или равно. Результат логическое значение true или false. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд).Используем перегруженный оператор >. Если условие выполняется, возвращаем значение true, иначе false.
8. Fraction & operator=(Fraction &FractionRight) - перегрузка оператора присваивания. В качестве параметров принимает по ссылке объект класса Fraction &FractionRight (второй операнд). Выполняем операцию присваивания для числителя и знаменателя и возвращаем значение указателя this.
9. operator double() – приведение объекта к типу double. Производится явное преобразование числителя и знаменателя к типу double. Результатом является частное от числителя и знаменателя.
10. friend std::ostream & operator<<(std::ostream &out, Fraction &FractionOut) - перегрузка оператора вывода данных в поток. Выполняется как дружественная функция. В качестве параметров принимает по ссылке (первый операнд) – класс потока вывода (ostream), объект класса Fraction &FractionOut (второй операнд). В качестве результата возвращается значение аргумента out.
11. friend std::istream& operator >> (std::istream &in, Fraction &FractionIn) - перегрузка оператора ввода данных из потока. Выполняется как дружественная функция. В качестве параметров принимает по ссылке (первый операнд) – класс потока ввода (istream), объект класса Fraction &FractionIn (второй операнд). В качестве результата возвращается значение аргумента in.

4. Описание работы программы

По методике лабораторной работы №1 добавляем в массив 3 объекта дроби. Рис. 1.

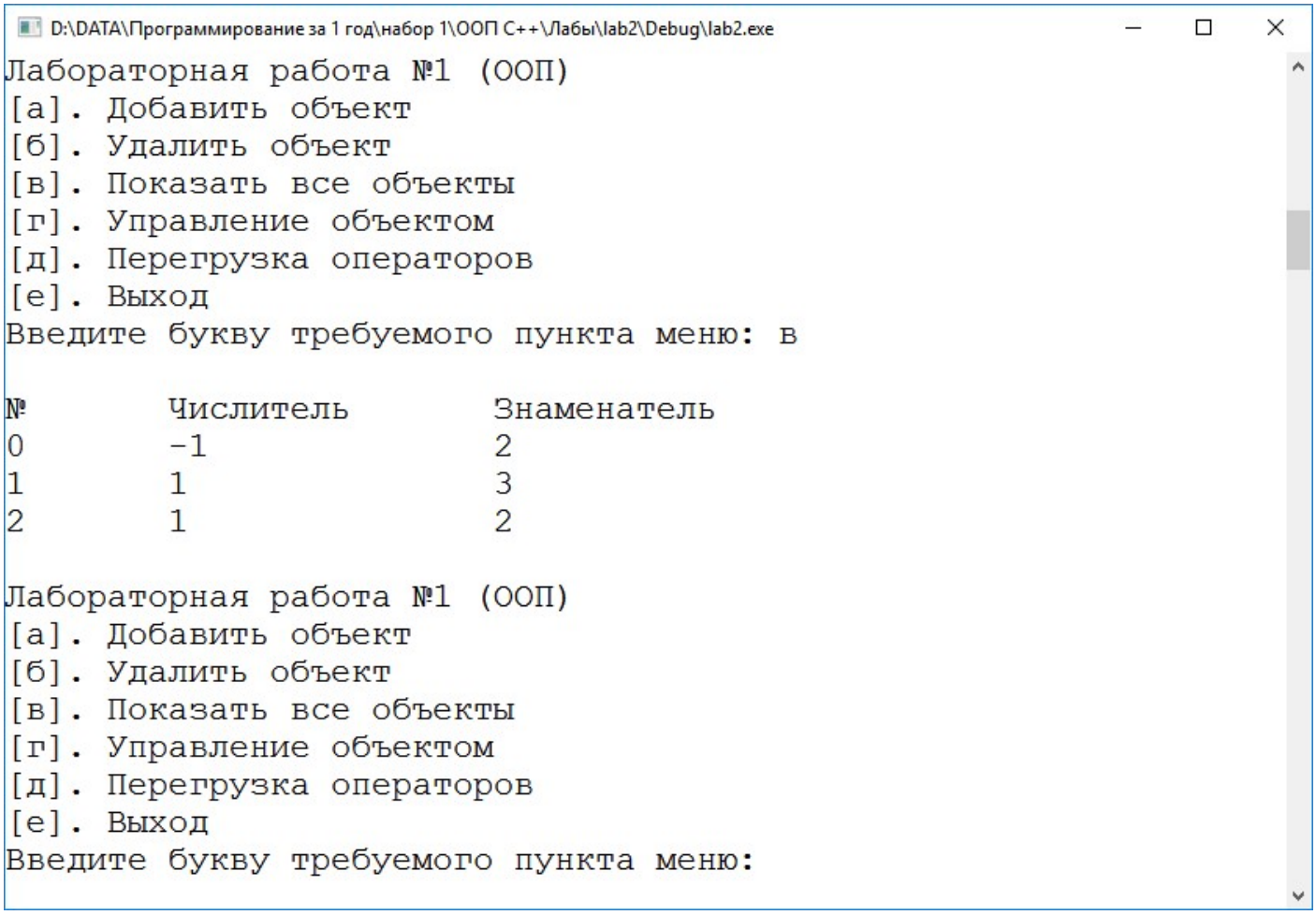

Рис. 1 Значения в массиве дробей для тестирования перегрузки операторов.

При выборе пункта д на экран будут выведены имеющиеся в массиве объекты-дроби и меню второго уровня перегрузки операторов. Рис. 2.

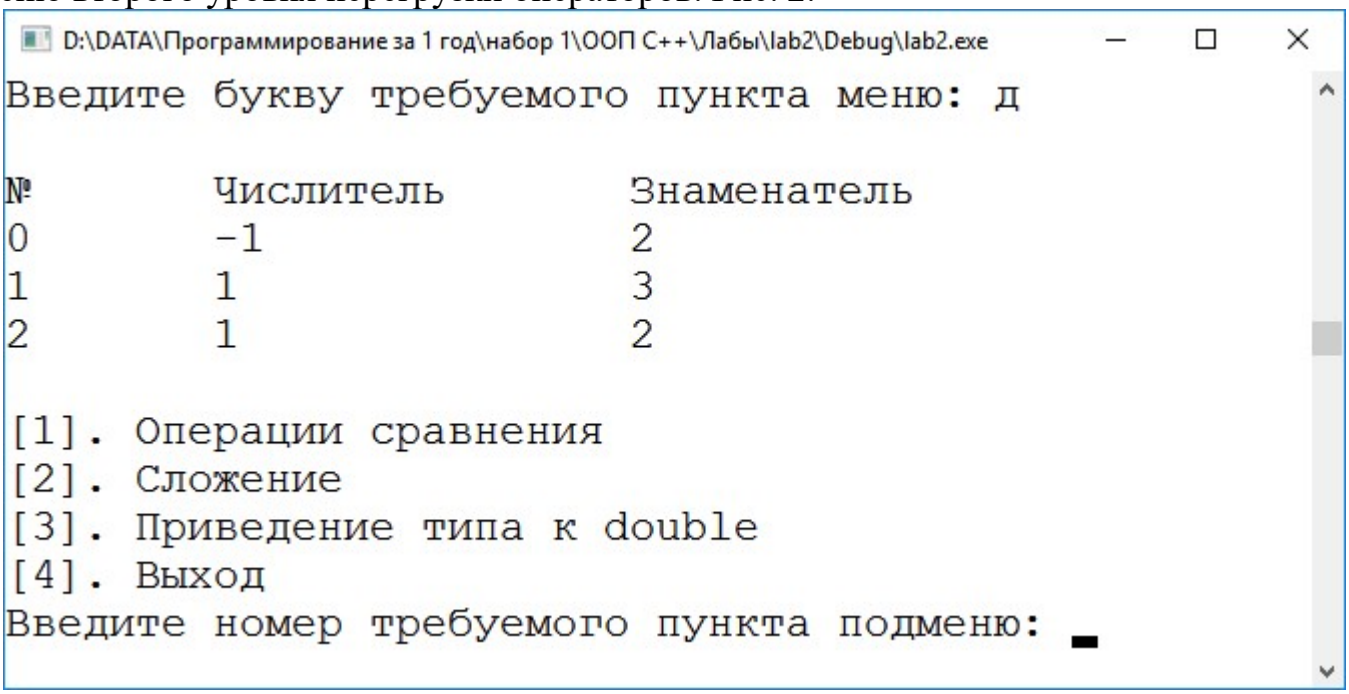

Рис. 2 Меню второго уровня перегрузки операторов.

При выборе пункта 1 меню второго уровня перегрузки операторов будут выведены имеющиеся в массиве объекты-дроби и предложено ввести индекс эталонного объекта. Рис. 3.

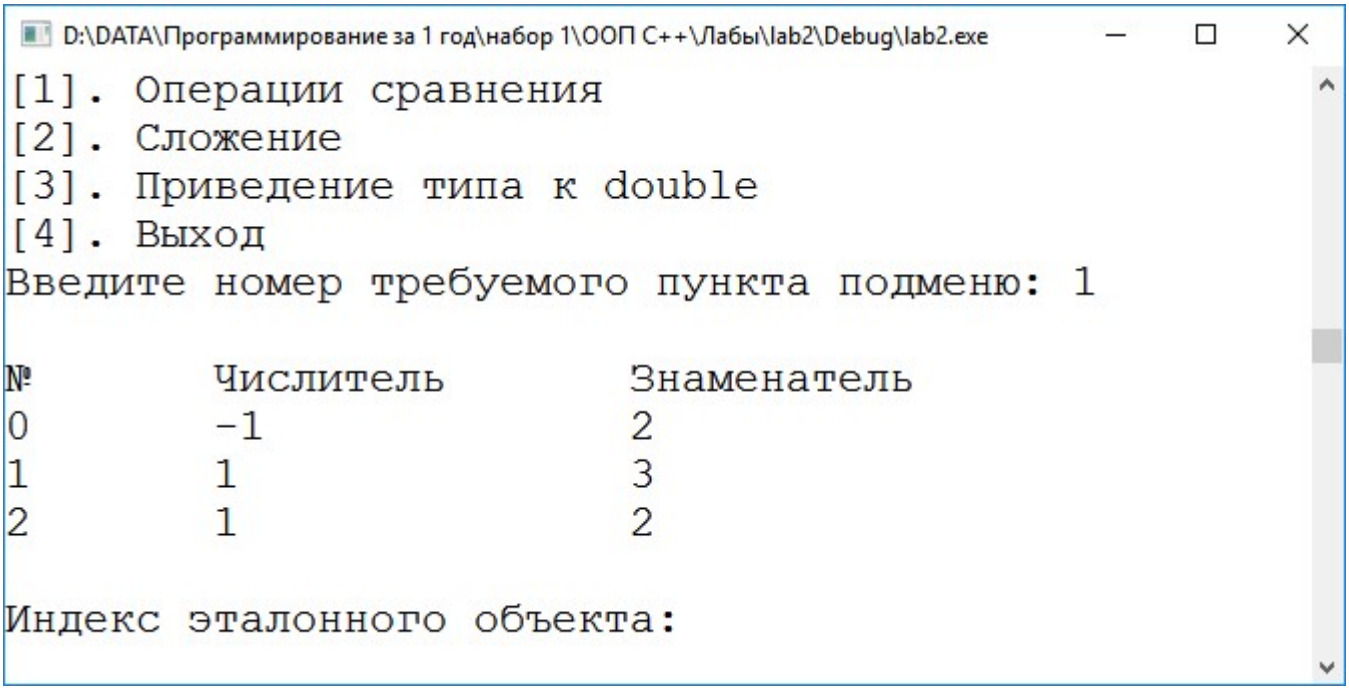

Рис. 3 Результат выбора пункта 1 меню второго уровня перегрузки операторов.

После ввода индекса эталонного объекта на экран будет выведено меню третьего уровня операций сравнения. Рис. 4.

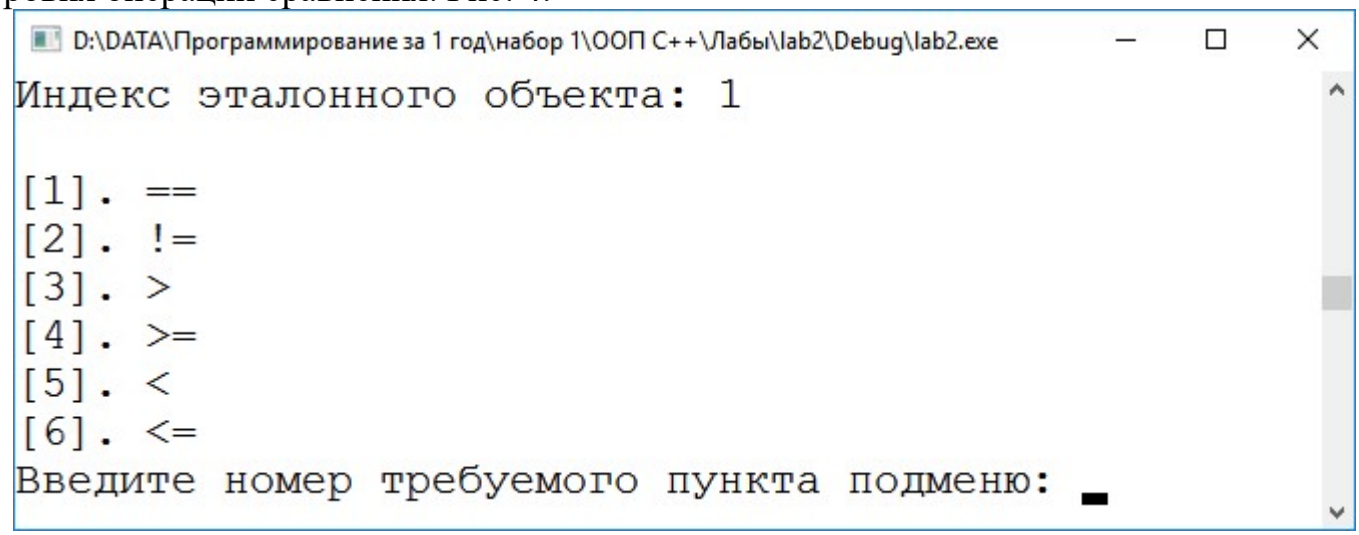

Рис. 4 Меню третьего уровня операций сравнения.

При выборе пункта 1 меню третьего уровня операций сравнения будут выведены результаты операции равно для выбранного эталонного объекта и всех объектов массива дробей и меню второго уровня перегрузки операторов. Рис. 5.

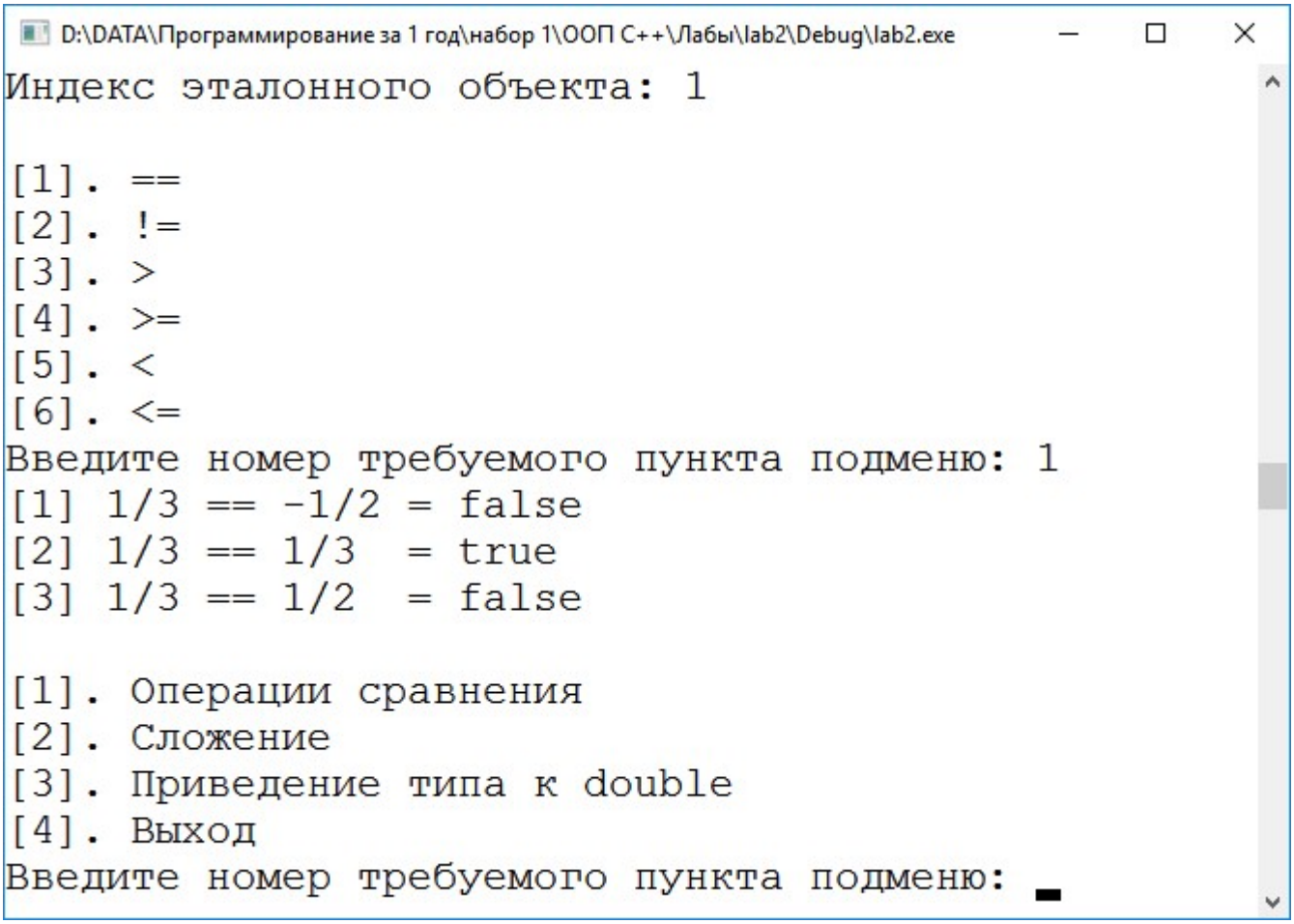

Рис. 5 Результат операции сравнения.

Для остальных пунктов меню третьего уровня операций сравнения работа программы будет аналогичной, в соответствии с выбранной операцией сравнения.

При выборе пункта 2 меню второго уровня перегрузки операторов будут выведены имеющиеся в массиве объекты-дроби и предложено ввести индекс эталонного объекта. Рис. 6.

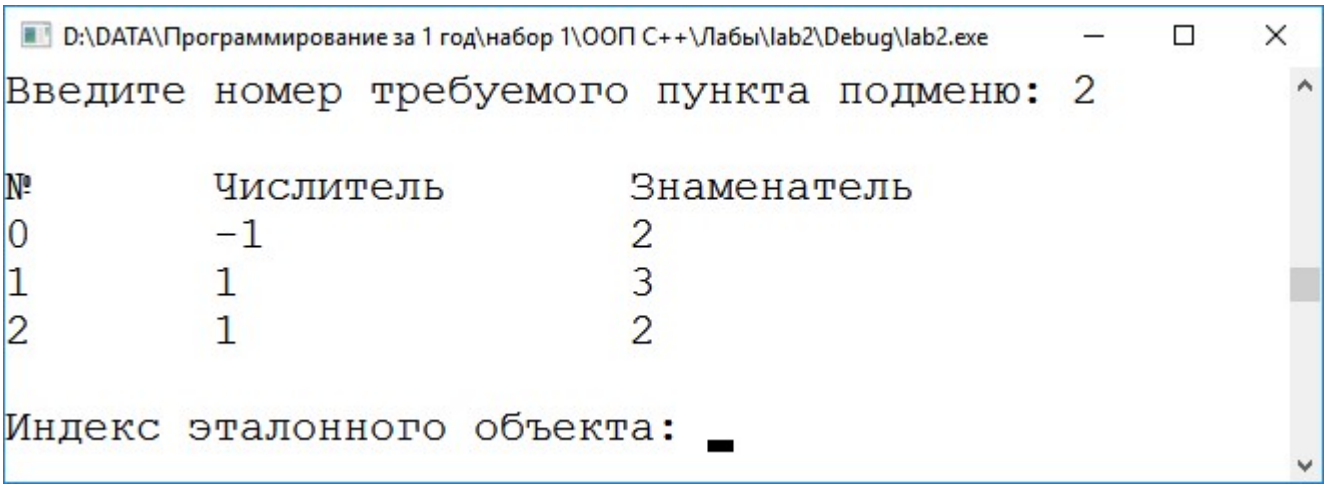

Рис. 6 Результат выбора пункта 2 меню второго уровня перегрузки операторов.

После ввода индекса эталонного объекта на экран будут выведены результаты операции сложения для выбранного эталонного объекта и всех объектов массива дробей и меню второго уровня перегрузки операторов. Рис. 7.

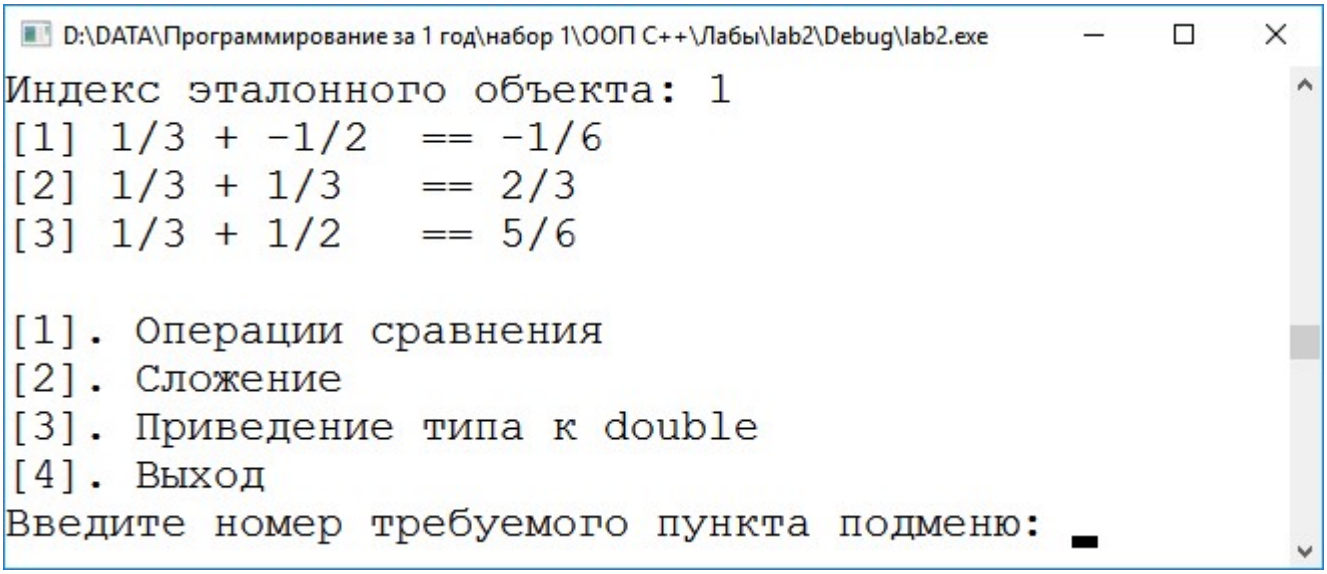

Рис. 7 Результат операции сложения.

При выборе пункта 3 меню второго уровня перегрузки операторов будут выведены имеющиеся в массиве объекты-дроби и результат применения к ним операции приведения типа. Рис. 8.

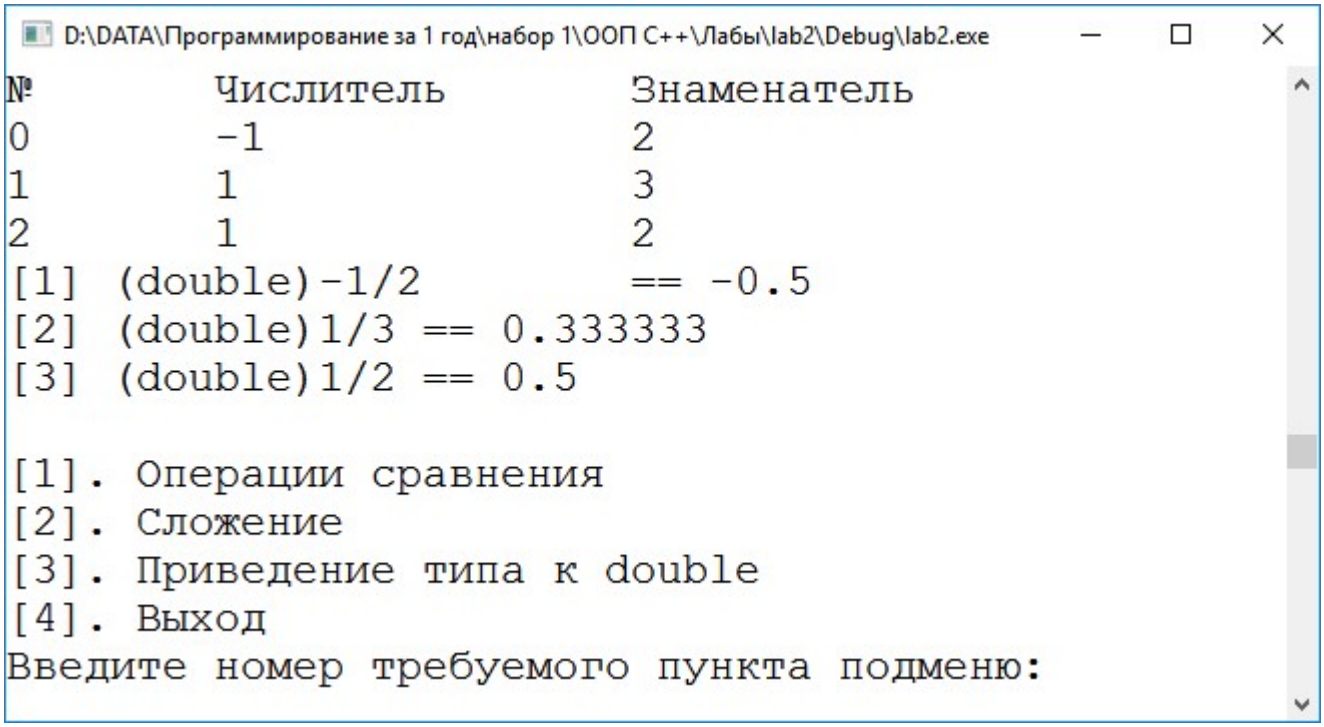

Рис. 8 Результат операции приведения типа.

При выборе пункта 4 меню второго уровня перегрузки операторов на экран будут выведены имеющиеся в массиве объекты-дроби и меню первого уровня. Рис. 9.

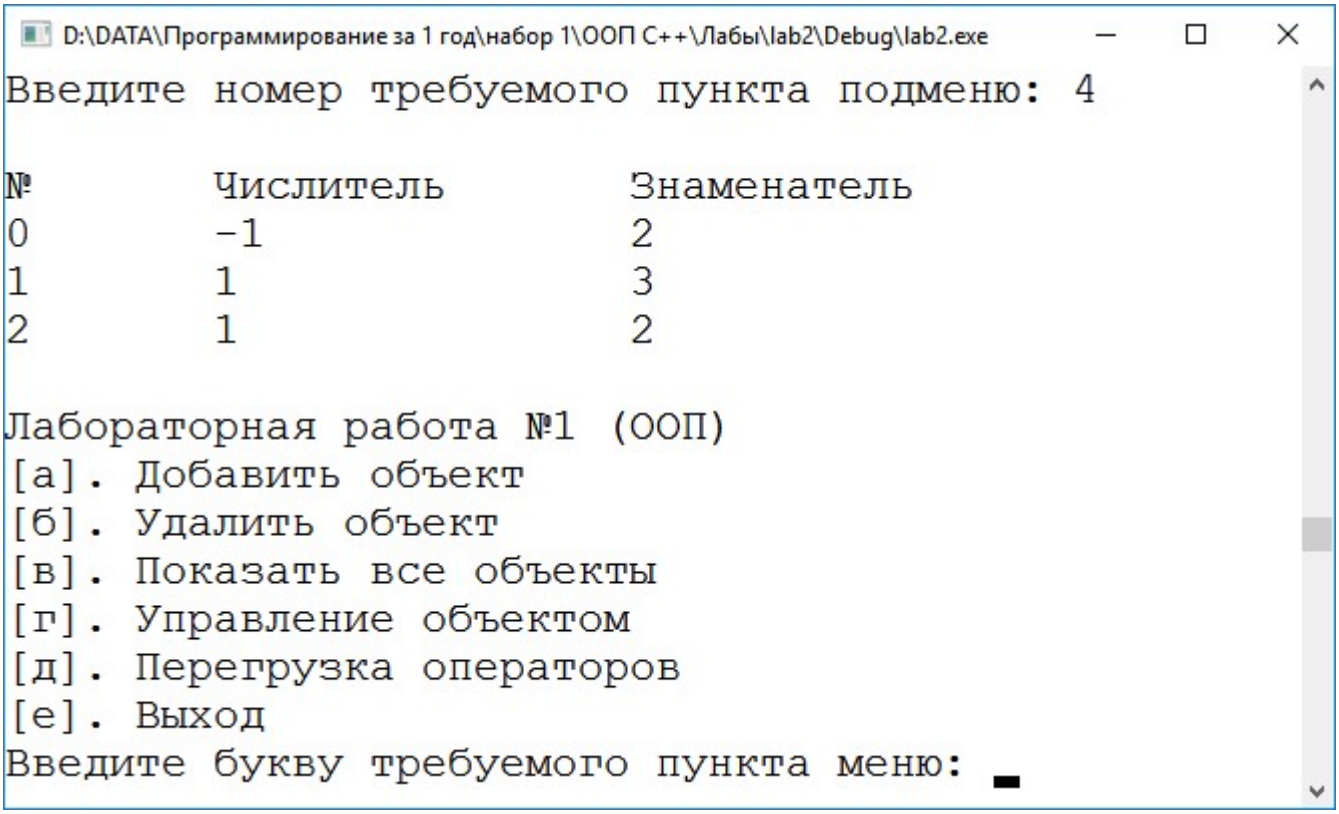

Рис. 9 Результат выбора пункта 4 меню второго уровня перегрузки операторов.

При выборе пункта е меню первого уровня происходит выход из программы.

5. Выводы

В результате выполнения данной лабораторной работы, получен практический опыт в проектировании и реализации методов перегрузки операторов. Целью данной работы является знакомство с перегрузкой операторов.

В данной лабораторной работе в реализованном в лабораторной работе №1 классе «Дроби» добавлены перегруженные операторы. В программе добавлена возможность применять к объектам класса следующие действия:

- Операция сложения «+»;
- Операция сравнения равно «==»;
- Операция сравнения не равно «!=»;
- Операция сравнения больше «>»;
- Операция сравнения больше или равно «>=»;
- Операция сравнения меньше «<»;
- Операция сравнения меньше или равно «<=»;
- Операция присваивания «=»;
- Операция приведение к типу double;
- Операция вывода в поток «<<»;
- Операция ввода из потока «>>».

Файлы программы находятся в репозитории по адресу: https://github.com/broitman-eugeny/lab2.

6. Код программы с комментариями

“ Fraction.h ”

```
#pragma once
#include <iostream>
#include <Windows.h>
const int FractionDimSize = 10;
//Класс дроби
class Fraction
{
	int Numerator;			//Числитель
	int Denominator;	//Знаменатель
	static int Count;	//Счетчик объектов
	static int Evklid(int N, int D);  //Определение НОД по алгоритму Евклида
public:
	Fraction();                                             //Конструктор по умолчанию
	Fraction(int N, int D);                         //Конструктор с параметрами
	Fraction(Fraction &F);                          //Конструктор копирования
	~Fraction();                                    //Деструктор
	void Reduction();                               //Сокращение дроби
	void Mul(Fraction &F);                          //Умножение дроби
	void Div(Fraction &F);                          //Деление дроби
	void SetNumerator(int N);               //Установка значения числителя
	void SetDenominator(int D);             //Установка значения знаменателя
	int GetNumerator();                             //Извлечение значения числителя
	int GetDenominator();                           //Извлечение значения знаменателя
	static int GetCount();                          //Извлечение количества созданных объектов
	void Print();                                   //Вывод на экран
	static void PrintAll(Fraction **PFraction);//Вывод на экран все объекты, указатели на которые находятся в PFraction
	int Sign()const;                                //Возвращает 1, если числитель и знаменатель имеют одинаковые знаки, и -1, если разные
	Fraction Abs()const;                    //Возвращает дробь с числителем и знаменателем без знака
	//Перегрузка операторов
	Fraction operator+(Fraction &Addendum);                 //Сложение 2-х объектов
	bool operator==(Fraction &FractionRight)const;  //Равенство 2-х объектов
	bool operator!=(Fraction &FractionRight)const;  //Неравенство 2-х объектов
	bool operator>(Fraction &FractionRight)const;   //Первый объект больше второго
	bool operator>=(Fraction &FractionRight)const;  //Первый объект больше или равен второму
	bool operator<(Fraction &FractionRight)const;   //Первый объект меньше второго
	bool operator<=(Fraction &FractionRight)const;  //Первый объект меньше или равен второму
	Fraction & operator=(Fraction &FractionRight);  //Присваивание
	operator double();                                      //Преобразование к типу double
	friend std::ostream& operator<<(std::ostream &out, Fraction &FractionOut);//Вывод в поток
	friend std::istream& operator >> (std::istream &in, Fraction &FractionIn);//Ввод из потока
};
//Функция меню
//PFraction - массив указателей на объекты типа Fraction, состоящий из FractionDimSize элементов
void Menu(Fraction **PFraction);
```

“Fraction.cpp”

```
#include "Fraction.h"
int Fraction::Count = 0;
int Fraction::Evklid(int N, int D)//Определение НОД по алгоритму Евклида
{
	int K;

	while (0 != (K = N%D))
	{
		N = D;
		D = K;
	}
	return (D>0) ? D : -D;
}
//Конструктор по умолчанию
Fraction::Fraction():Numerator(1), Denominator(1)
{
	Count++;
}
```

```
//Конструктор с параметрами
Fraction::Fraction(int N, int D) : Numerator(N), Denominator(D)
{
	Count++;
}
//Конструктор копирования
Fraction::Fraction(Fraction & F) : Numerator(F.Numerator), Denominator(F.Denominator)
{
	Count++;
}
//Деструктор
Fraction::~Fraction()
{
	Count--;
}
//Сокращение дроби
void Fraction::Reduction()
{
	int CommonDevider= Evklid(Numerator, Denominator);//наибольший общий делитель по алгоритму
Евклида
	Numerator /= CommonDevider;
	Denominator /= CommonDevider;
}
//Умножение дроби
void Fraction::Mul(Fraction & F)
{
	Numerator *= F.Numerator;
	Denominator *= F.Denominator;
}
//Деление дроби
void Fraction::Div(Fraction & F)
{
	Numerator *= F.Denominator;
	Denominator *= F.Numerator;
}
//Установка значения числителя
void Fraction::SetNumerator(int N)
{
	Numerator = N;
}
//Установка значения знаменателя
void Fraction::SetDenominator(int D)
{
	Denominator = D;
}
//Извлечение значения числителя
int Fraction::GetNumerator()
{
	return Numerator;
}
//Извлечение значения знаменателя
int Fraction::GetDenominator()
{
	return Denominator;
}
//Извлечение количества созданных объектов
int Fraction::GetCount()
{
	return Count;
}
//Вывод на экран
void Fraction::Print()
{
	std::cout << Numerator << "\t\t" << Denominator << std::endl;
}
void Fraction::PrintAll(Fraction ** PFraction)
{
	std::cout << std::endl << "№" << "\t" << "Числитель" << "\t" << "Знаменатель" << std::endl;
	for (int i = 0, c = Fraction::GetCount(); i < c; i++)
```

```
	{
		std::cout << i << "\t";
		PFraction[i]->Print();
	}
}
//Возвращает 1, если числитель и знаменатель имеют одинаковые знаки, и -1, если разные
int Fraction::Sign()const
{
	if (Numerator < 0 && Denominator < 0 || Numerator >= 0 && Denominator>0)
	{
		return 1;
	}
	else
	{
		return -1;
	}
}
Fraction Fraction::Abs()const
{
	Fraction Temp;
	Temp.Numerator = (Numerator < 0) ? -Numerator : Numerator;
	Temp.Denominator = (Denominator < 0) ? -Denominator : Denominator;
	return Temp;
}
//Сложение 2-х объектов
Fraction Fraction::operator+(Fraction & Addendum)
{
	Fraction Temp;
	Temp.Numerator = Numerator*Addendum.Denominator + Addendum.Numerator*Denominator;
	Temp.Denominator = Denominator * Addendum.Denominator;
	Temp.Reduction();
	return Temp;
}
//Равенство 2-х объектов
bool Fraction::operator==(Fraction & FractionRight)const
{
		return
Sign()*(Abs().Numerator)*(FractionRight.Abs().Denominator)==(FractionRight.Sign())*(FractionRight.Ab
s().Numerator)*(Abs().Denominator);
}
//Неравенство 2-х объектов
bool Fraction::operator!=(Fraction & FractionRight)const
{
	return !(*this == FractionRight);
}
//Первый объект больше второго
bool Fraction::operator>(Fraction & FractionRight)const
{
	return
Sign()*(Abs().Numerator)*(FractionRight.Abs().Denominator)>(FractionRight.Sign())*(FractionRight.Abs
().Numerator)*(Abs().Denominator);
}
//Первый объект больше или равен второму
bool Fraction::operator>=(Fraction & FractionRight)const
{
	return
Sign()*(Abs().Numerator)*(FractionRight.Abs().Denominator)>=(FractionRight.Sign())*(FractionRight.Ab
s().Numerator)*(Abs().Denominator);
}
//Первый объект меньше второго
bool Fraction::operator<(Fraction & FractionRight)const
{
	return !(operator>=(FractionRight));
}
//Первый объект меньше или равен второму
bool Fraction::operator<=(Fraction & FractionRight)const
{
	return !(operator>(FractionRight));
}
```

```
//Присваивание
Fraction & Fraction::operator=(Fraction & FractionRight)
{
	Numerator = FractionRight.Numerator;
	Denominator = FractionRight.Denominator;
	return *this;
}
//Преобразование к типу double
Fraction::operator double()
{
	return (double)Numerator / (double)Denominator;
}
```

“Main.cpp”

```
//Лабораторная работа №2.
//Вариант №5.
//1) Программу, разработанную в лабораторной работе № 1, модифицировать таким образом,
//чтобы появилась возможность применять к объектам класса следующие операторы:
//•	Инкремент, декремент(можно только одну из префиксной и постфиксной форм).
//•	Сложение объектов класса.
//•	Операторы сравнения.
//•	Оператор присваивания.
//•	Оператор приведения к типу int(или другому численному).
//•	Операторы ввода и вывода в поток.
//ПРИМЕЧАНИЕ: логика использования оператора выбирается самостоятельно.
//Неуместные для данного класса операторы можно не реализовывать, заменив их какими - то другими.
//Общее количество перегруженных в классе операторов – около 8 - 10.
//2) Интерфейс пользователя должен обеспечивать :
//•	В начале работы программы создавать константный эталонный объект,
//и предоставлять пользователю возможность выбора операции сравнения.
//Эта операция применяется к эталонному объекту и всем остальным.Результаты выводить в виде таблицы.
//Таблица может выглядеть так(для класса время) :
//[1] 00 : 00 : 00  >   12:00 : 00    false
//[2] 17 : 10 : 00  >   12:00 : 00    true
//[3] 11 : 11 : 00  >   12:00 : 00    false           …..
//Но предпочтительней предусмотреть в этом цикле применение нескольких операторов сравнения
//с целью вывода корректного результата :
//[1] 00 : 00 : 00  <   12 : 00 : 00
//[2] 17 : 10 : 00  >   12 : 00 : 00
//[3] 12 : 00 : 00 = 12 : 00 : 00
//•	Предусмотреть возможность создания нового объекта,
//как результата выполнения  математической операции над одним из имеющихся.
//Операндами являются уже существующие в массиве объекты,
//либо с фиксированными номерами(0 - й для унарных операторов, 0 - й и 1 - й для бинарных),
//либо номера могут запрашиваться с клавиатуры.
#include "Fraction.h"
void main()
{
	Fraction **PFraction=new Fraction *[FractionDimSize];//массив указателей из FractionDimSize
элементов
	SetConsoleCP(1251);//Ввод русских букв
	SetConsoleOutputCP(1251);//Вывод русских букв
	Menu(PFraction);
}
```

“Menu.cpp”

```
#include "Fraction.h"
//Функция отображения меню
void ShowMenu()
{
	std::cout << std::endl << "Лабораторная работа №1 (ООП)";
	std::cout << std::endl << "[а]. Добавить объект";
	std::cout << std::endl << "[б]. Удалить объект";
	std::cout << std::endl << "[в]. Показать все объекты";
	std::cout << std::endl << "[г]. Управление объектом";
	std::cout << std::endl << "[д]. Перегрузка операторов";
	std::cout << std::endl << "[е]. Выход";
	std::cout << std::endl << "Введите букву требуемого пункта меню: ";
}
//Функция отображения подменю добавления объекта
void ShowMenu2()
```

```
{
	std::cout << std::endl << "[1]. Значения по умолчанию";
	std::cout << std::endl << "[2]. Копия уже существующего в массиве объекта";
	std::cout << std::endl << "[3]. Ввод значений числителя и знаменателя";
	std::cout << std::endl << "[4]. Выход";
	std::cout << std::endl << "Введите номер требуемого пункта подменю: ";
}
//Функция отображения подменю управления объектом
void ShowMenu3()
{
	std::cout << std::endl << "[1]. Изменение содержимого объекта с заданным номером";
	std::cout << std::endl << "[2]. Сокращение дроби объекта с заданным номером";
	std::cout << std::endl << "[3]. Умножение дроби объекта с заданным номером";
	std::cout << std::endl << "[4]. Деление дроби объекта с заданным номером";
	std::cout << std::endl << "[5]. Выход";
	std::cout << std::endl << "Введите номер требуемого пункта подменю: ";
}
//Функция отображения подменю перегрузки операторов
void ShowMenu4()
{
	std::cout << std::endl << "[1]. Операции сравнения";
	std::cout << std::endl << "[2]. Сложение";
	std::cout << std::endl << "[3]. Приведение типа к double";
	std::cout << std::endl << "[4]. Выход";
	std::cout << std::endl << "Введите номер требуемого пункта подменю: ";
}
//Функция отображения подподменю операций сравнения
void ShowMenu5()
{
	std::cout << std::endl << "[1]. ==";
	std::cout << std::endl << "[2]. !=";
	std::cout << std::endl << "[3]. >";
	std::cout << std::endl << "[4]. >=";
	std::cout << std::endl << "[5]. <";
	std::cout << std::endl << "[6]. <=";
	std::cout << std::endl << "Введите номер требуемого пункта подменю: ";
}
//Функция меню
//PFraction - массив указателей на объекты типа Fraction, состоящий из FractionDimSize элементов
void Menu(Fraction **PFraction)
{
	char C = -1;
	int C3 = -1;
	int i, c;
	int Count;
	while (C != 'e')//д - выход
	{
		ShowMenu();
		std::cin >> C;
		while (C<'a' || C>'e' || std::cin.fail())
		{
			std::cin.clear();
			std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка буфера
			std::cout << std::endl << "Неверно введен символ, введите еще раз: ";
			std::cin >> C;
		}
		switch (C)
		{
		case 'a'://Добавить объект
			if (Fraction::GetCount() == FractionDimSize)//Если массив заполнен
			{
				std::cout << "Извините, массив заполнен" << std::endl;
			}
			else
			{
				int C2 = -1;
				while (C2 != 4)//4 - выход
				{
					ShowMenu2();
```

```
                                        std::cin >> C2;
                                        while (C2<1 || C2>4 || std::cin.fail())
                                        {
                                                std::cin.clear();
                                                std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка бу-
фера
                                                std::cout << std::endl << "Неверно введен номер, введите
еще раз: ";
                                                std::cin >> C2;
                                        }
                                        int Index, IndexOfCopy;
                                        switch (C2)
                                        {
                                        case 1://Значения по умолчанию
                                                Index = Fraction::GetCount();
                                                PFraction[Index] = new Fraction();
                                                std::cout << "Количество объектов: " << Frac-
tion::GetCount() << std::endl;
                                                break;
                                        case 2://Копия уже существующего в массиве объекта
                                                if (Fraction::GetCount() == 0)
                                                {
                                                        std::cout << "Сначала добвьте объекты!";
                                                        break;
                                                }
                                                std::cout << "Введите индекс копируемого объекта: ";
                                                std::cin >> IndexOfCopy;
                                                Count = Fraction::GetCount();
                                                while (IndexOfCopy<0 || IndexOfCopy >= Count ||
std::cin.fail())
                                                {
                                                        std::cin.clear();
                                                        std::cin.ignore(std::cin.rdbuf()-
>in_avail());//Очистка буфера
                                                        std::cout << std::endl << "Индекс введен неверно,
введите еще раз: ";
                                                        std::cin >> IndexOfCopy;
                                                }
                                                Index = Fraction::GetCount();
                                                PFraction[Index] = new Fraction(*PFraction[IndexOfCopy]);
                                                std::cout << "Количество объектов: " << Frac-
tion::GetCount() << std::endl;
                                                break;
                                        case 3://Ввод значений числителя и знаменателя
                                                Index = Fraction::GetCount();
                                                PFraction[Index] = new Fraction();
                                                std::cin >> *(PFraction[Index]);
                                                std::cout << "Количество объектов: " << Frac-
tion::GetCount() << std::endl;
                                                break;
                                        }//switch (C)
                                }//while (C2 != 4)//4 - выход
                        }//case 'a'://Добавить объект
                        break;
                case '6'://Удалить объект
                        if (Fraction::GetCount() == 0)
                        {
                                std::cout << "Сначала добвьте объекты!";
                                break;
                        }
                        std::cout << "Введите индекс удаляемого объекта: ";
                        int Index;
                        std::cin >> Index;
                        Count = Fraction::GetCount();
                        while (Index<0 || Index>= Count || std::cin.fail())
                        {
                                std::cin.clear();
                                std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка буфера
                                std::cout << std::endl << "Индекс введен неверно, введите еще раз: ";
```

```
                        std::cin >> Index;
                }
                c = Fraction::GetCount() - 1;
                for (i = Index; i < c; i++)
                {
                        PFraction[i]->SetNumerator(PFraction[i+1]->GetNumerator());
                        PFraction[i]->SetDenominator(PFraction[i + 1]->GetDenominator());
                }
                delete PFraction[i];
                std::cout << "Количество объектов: " << Fraction::GetCount() << std::endl;
                break;
        case 'в'://Показать все объекты
                Fraction::PrintAll(PFraction);
                break;
        case 'г'://Управление объектом
                if (Fraction::GetCount() == 0)
                {
                        std::cout << "Сначала добвьте объекты!";
                        break;
                }
                Fraction::PrintAll(PFraction);
                std::cout << "Введите индекс управляемого объекта: ";
                std::cin >> Index;
                Count = Fraction::GetCount();
                while (Index<0 || Index >= Count || std::cin.fail())
                {
                        std::cin.clear();
                        std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка буфера
                        std::cout << std::endl << "Индекс введен неверно, введите еще раз: ";
                        std::cin >> Index;
                }
                while (C3 != 5)//5- выход
                {
                        Fraction::PrintAll(PFraction);
                        ShowMenu3();
                        std::cin >> C3;
                        while (C3 < 1 || C3>5 || std::cin.fail())
                        {
                                std::cin.clear();
                                std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка буфера
                                std::cout << std::endl << "Неверно введен номер, введите еще раз:
";
                                std::cin >> C3;
                        }
                        Fraction *F = new Fraction();//Создание объекта и инкремент Count;
                        switch (C3)
                        {
                        case 1://Изменение содержимого объекта с заданным номером
                                std::cin >> *(PFraction[Index]);
                                break;
                        case 2://Сокращение дроби объекта с заданным номером
                                PFraction[Index]->Reduction();
                                break;
                        case 3://Умножение дроби объекта с заданным номером
                                std::cout << "Множитель:";
                                std::cin >> *F;
                                PFraction[Index]->Mul(*F);
                                break;
                        case 4://Деление дроби объекта с заданным номером
                                std::cout << "Делитель:";
                                std::cin >> *F;
                                PFraction[Index]->Div(*F);
                                break;
                        }//switch (C3)
                        delete F;//Удаление объекта и декремент Count
                }//while (C3 != 5)//5- выход
                break;
        case 'д'://Перегрузка операторов
                if (Fraction::GetCount() == 0)
```

```
                        {
                                std::cout << "Сначала добвьте объекты!";
                                break;
                        }
                        Fraction::PrintAll(PFraction);
                        int C4 = -1;
                        int C5 = -1;
                        char *Compare = "", *CompareResult = "";
                        while (C4 != 4)//4- выход
                        {
                                ShowMenu4();
                                std::cin >> C4;
                                while (C4 < 1 || C4>4 || std::cin.fail())
                                {
                                        std::cin.clear();
                                        std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка буфера
                                        std::cout << std::endl << "Неверно введен номер, введите еще раз:
";
                                        std::cin >> C4;
                                }
                                Fraction::PrintAll(PFraction);
                                Count = Fraction::GetCount();
                                Fraction *F = new Fraction();//Создание объекта и инкремент Count;
                                switch (C4)
                                {
                                case 1://[1]. Операции сравнения
                                        std::cout << std::endl << "Индекс эталонного объекта: ";
                                        std::cin >> Index;
                                        while (Index<0 || Index >= Count || std::cin.fail())
                                        {
                                                std::cin.clear();
                                                std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка бу-
фера
                                                std::cout << std::endl << "Индекс введен неверно, введите
еще раз: ";
                                                std::cin >> Index;
                                        }
                                        ShowMenu5();
                                        std::cin >> C5;
                                        while (C5 < 1 || C5>6 || std::cin.fail())
                                        {
                                                std::cin.clear();
                                                std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка бу-
фера
                                                std::cout << std::endl << "Неверно введен номер, введите
еще раз: ";
                                                std::cin >> C5;
                                        }
                                        for (i = 0; i < Count; i++)
                                        {
                                                switch (C5)
                                                {
                                                case 1:
                                                        Compare = " == ";
                                                        CompareResult = (*(PFraction[Index])==
*(PFraction[i])) ?"\t= true" :"\t= false" ;
                                                        break;
                                                case 2:
                                                        Compare = " != ";
                                                        CompareResult = (*(PFraction[Index]) !=
*(PFraction[i])) ? "\t= true" : "\t= false";
                                                        break;
                                                case 3:
                                                        Compare = " > ";
                                                        CompareResult = (*(PFraction[Index]) >
*(PFraction[i])) ? "\t= true" : "\t= false";
                                                        break;
                                                case 4:
                                                        Compare = " >= ";
```

```
                                                CompareResult = (*(PFraction[Index]) >=
*(PFraction[i])) ? "\t= true" : "\t= false";
                                                break;
                                        case 5:
                                                Compare = " < ";
                                                CompareResult = (*(PFraction[Index]) <
*(PFraction[i])) ? "\t= true" : "\t= false";
                                                break;
                                        case 6:
                                                Compare = " <= ";
                                                CompareResult = (*(PFraction[Index]) <=
*(PFraction[i])) ? "\t= true" : "\t= false";
                                                break;
                                        }
                                        std::cout << "[" << i+1 << "] " << *(PFraction[Index]) <<
Compare << *(PFraction[i]) << CompareResult << std::endl;
                                }
                                break;
                        case 2://[2]. Сложение
                                std::cout << std::endl << "Индекс эталонного объекта: ";
                                std::cin >> Index;
                                while (Index<0 || Index >= Count || std::cin.fail())
                                {
                                        std::cin.clear();
                                        std::cin.ignore(std::cin.rdbuf()->in_avail());//Очистка бу-
фера
                                        std::cout << std::endl << "Индекс введен неверно, введите
еще раз: ";
                                        std::cin >> Index;
                                }
                                for (i = 0; i < Count; i++)
                                {
                                        *F = *(PFraction[Index]) + *(PFraction[i]);
                                        std::cout << "[" << i + 1 << "] " << *(PFraction[Index]) <<
" + " << *(PFraction[i]) << "\t== " << *F << std::endl;
                                }
                                break;
                        case 3://[3]. Приведение типа к double
                                for (i = 0; i < Count; i++)
                                {
                                        std::cout << "[" << i + 1 << "] " << "(double)" <<
*(PFraction[i]) << "\t== " << (double)(*(PFraction[i])) << std::endl;
                                }
                                break;
                        }//switch (C4)
                        delete F;//Удаление объекта и декремент Count
                }//while (C4 != 4)//4- выход
                break;
        }//switch (C)
    }//while (C != 'e')//e - выход
    //Очистка динамической памяти
    for (i = 0, c = Fraction::GetCount(); i < c; i++)
    {
        delete PFraction[i];
    }
}
```

“FractionFriends.cpp”

```
#include "Fraction.h"
//Вывод в поток
std::ostream & operator<<(std::ostream & out, Fraction & FractionOut)
{
    out << FractionOut.Numerator << "/" << FractionOut.Denominator;
    return out;
}
//Ввод из потока
std::istream & operator >> (std::istream & in, Fraction & FractionIn)
{
    std::cout << std::endl << "Числитель: ";
    in >> FractionIn.Numerator;
```

```
	while (in.fail())
	{
		in.clear();
		in.ignore(in.rdbuf()->in_avail());//Очистка буфера
		std::cout << std::endl << "Неверно введено число, введите еще раз: ";
		in >> FractionIn.Numerator;
	}
	std::cout << std::endl << "Знаменатель: ";
	in >> FractionIn.Denominator;
	while (FractionIn.Denominator == 0 || in.fail())
	{
		in.clear();
		in.ignore(in.rdbuf()->in_avail());//Очистка буфера
		std::cout << std::endl << "Неверно введено число, введите еще раз: ";
		in >> FractionIn.Denominator;
	}
	return in;
}
```