Из определения следует, что любое рациональное число в

смешанном виде определяется четырьмя составляющими:

знаком числа (число положительное или отрицательное);

целой частью;

числителем;

знаменателем.

Все составляющие дроби являются целыми числами. Знак дроби

тоже будем представлять в виде целого числа (1 – положительная

дробь, -1 – отрицательная дробь), поскольку это удобно при

реализации арифметических операций:

7

// класс "Рациональное число"

class Fraction

{

int sign; // знак дроби (+ или -)

int intPart; // целая часть дроби

int numerator; // числитель дроби

int denominator; // знаменатель дроби

. . .

}

При описании операций с дробями предполагаем, что объекты

класса Fraction находятся в смешанном виде. Результатом операции

над дробями может быть неправильная дробь, которую, согласно

предположению, необходимо перевести в смешанный вид. Для этого

необходимы методы «преобразования в смешанный вид»,

«сокращения дроби» и «выделения целой части». Данные методы

будут применяться при выполнении арифметических операций над

дробями или при создании дроби, гарантируя, что дробь после

завершения операции будет иметь смешанный вид. Таким образом,

пользователю класса нет необходимости выполнять операции

приведения дроби к смешанному виду, поскольку эта операция

выполняется автоматически. Поэтому методы преобразования в

смешанный вид, сокращения дроби и выделения целой части можно

описать как закрытые элементы класса.

// класс "Рациональное число"

class Fraction

{

int sign; // знак дроби (+ или -)

int intPart; // целая часть дроби

int numerator; // числитель дроби

int denominator; // знаменатель дроби

// метод преобразование дроби в смешанный вид

void GetMixedView()

{

. . .

}

// метод сокращения дроби

void Cancellation()

{

. . .

}

8

// метод выделения целой части дроби

void GetIntPart()

{

. . .

}

. . .

}

Деструктор также должен относиться к закрытым элементам

класса – это определено правилами языка С#.

К доступным элементам класса Fraction относятся

конструкторы, методы, реализующие арифметические операции,

методы сравнения, метод преобразования в вещественное число. Для

ввода дроби необходимо разработать статический метод Parse для

выделения дроби из символьной строки. Для вывода дроби удобно

переопределить неявную операцию преобразования в символьную

строку, которая будет осуществлять получение символьного

представления дроби.

Таким образом, полный состав класса Fraction может

выглядеть так:

class Fraction

{

int sign; // знак дроби (+ или -)

int intPart; // целая часть дроби

int numerator; // числитель дроби

int denominator; // знаменатель дроби

// метод преобразования дроби в смешанный вид

void GetMixedView()

{

. . .

}

// метод сокращения дроби

void Cancellation()

{

. . .

}

// метод выделения целой части дроби

void GetIntPart()

{

. . .

}

9

// конструктор без параметров

public Fraction()

{

. . .

}

// конструктор c параметрами

public Fraction(int n, int d, int i = 0, int s = 1)

{

. . .

}

// деструктор

~Fraction()

{

. . .

}

// метод сложения двух дробей

static public Fraction operator + (Fraction ob1, Fraction ob2)

{

. . .

}

// метод сложения дроби с целым числом

static public Fraction operator + (Fraction ob1, int a)

{

. . .

}

// метод сложения целого числа и дроби

static public Fraction operator + (int a, Fraction ob1)

{

. . .

}

// метод изменение знака дроби на противоположный

static public Fraction operator - (Fraction ob)

{

. . .

}

// метод вычитания двух дробей

static public Fraction operator - (Fraction ob1, Fraction ob2)

{

. . .

}

// метод вычитания из дроби целого числа

static public Fraction operator - (Fraction ob1, int a)

10

{

. . .

}

// метод вычитания дроби из целого числа

static public Fraction operator - (int a, Fraction ob1)

{

. . .

}

// метод умножения двух дробей

static public Fraction operator \* (Fraction ob1, Fraction ob2)

{

. . .

}

// метод умножения дроби на целое число

static public Fraction operator \* (Fraction ob1, int a)

{

. . .

}

// метод умножения целого числа и дроби

static public Fraction operator \* (int a, Fraction ob1)

{

. . .

}

// метод деления двух дробей

static public Fraction operator / (Fraction ob1, Fraction ob2)

{

. . .

}

// метод деления дроби на целое число

static public Fraction operator / (Fraction ob1, int a)

{

. . .

}

// метод деления целого числа на дробь

static public Fraction operator / (int a, Fraction ob1)

{

. . .

}

// метод преобразования дроби в тип double

static public explicit operator double(Fraction ob)

{

. . .

11

}

// методы сравнения двух дробей

public static bool operator > (Fraction ob1, Fraction ob2)

{

. . .

}

public static bool operator < (Fraction ob1, Fraction ob2)

{

. . .

}

public static bool operator >= (Fraction ob1, Fraction ob2)

{

. . .

}

public static bool operator <= (Fraction ob1, Fraction ob2)

{

. . .

}

public static bool operator != (Fraction ob1, Fraction ob2)

{

. . .

}

public static bool operator == (Fraction ob1, Fraction ob2)

{

. . .

}

// статический метод преобразования строки в дробь

public static Fraction Parse(string str)

{

. . .

}

// метод получения строкового представления дроби – оператор

// преобразования в символьную строку

public static implicit operator string(Fraction ob)

{

. . .

}

}

12

1.2. Конструкторы и деструктор класса

«Рациональное число»

Для создания объекта определим конструктор с четырьмя

параметрами, соответствующими четырем структурным элементам

класса:

значение числителя;

значение знаменателя;

значение целой части;

знак числа.

Прототип конструктора имеет следующий вид:

// конструктор с параметрами

public Fraction(int n, int d, int i = 0, int s = 1)

Если при создании объекта не указываются значения целой части

и знака, то по умолчанию считается, что целая часть числа равна нулю

и число является положительным. Это определяется заданием

значений по умолчанию соответствующих параметров конструктора в

его объявлении (прототипе). Параметры, для которых указываются

значения по умолчанию, должны располагаться в конце списка

формальных параметров.

//конструктор класса «Рациональное число»

public Fraction(int n, int d, int i = 0, int s = 1)

{

intPart = i;

numerator = n;

denominator = d;

sign = s;

GetMixedView();

}

При создании объекта конструктору могут быть переданы

значения числителя и знаменателя, образующие неправильную или

сократимую дробь. В этом случае в теле конструктора после

инициализации свойств нужно преобразовать дробь в смешанный вид.

Это можно сделать путем вызова метода преобразования

GetMixedView().

Также определим в классе конструктор без параметров, который

может использоваться при создании дроби, равной нулю. В

13

конструкторе без параметров структурным свойствам присваиваются

конкретные значения:

// конструктор без параметров класса «Рациональное число»

public Fraction()

{

intPart = 0;

numerator = 0;

denominator = 1;

sign = 1;

}

Отдельно рассмотрим метод преобразования дроби в смешанную

и несократимую форму. В случаях, если значения числителя и

знаменателя задают неправильную или сократимую дробь, в методе

происходит выделение целой части, а затем осуществляется

сокращение.

// метод преобразования дроби в смешанный вид

void GetMixedView()

{

GetIntPart(); // выделение целой части числа

Cancellation(); // сокращение дроби

}

Если числитель дроби больше знаменателя, то выделяется целая

часть:

// метод выделения целой части рационального числа

void GetIntPart()

{

if(numerator >= denominator)

{

intPart += (numerator / denominator);

numerator %= denominator;

}

}

Сокращение дроби осуществляется путем деления числителя и

знаменателя дроби на их наибольший общий делитель, который

вычисляется с помощью алгоритма Евклида.

// метод сокращения рациональной дроби

void Cancellation()

{

if(numerator != 0)

{

14

int m = denominator,

n = numerator,

ost = m%n;

// вычисление НОД(числителя, знаменателя)

// алгоритмом Евклида

while(ost != 0)

{

m = n;

n = ost;

ost = m % n;

}

int nod = n;

if(nod != 1)

{

numerator /= nod;

denominator /= nod;

}

}

}

Деструктор класса выводит сообщение о том, что уничтожен

объект класса Fraction.

// деструктор

~Fraction()

{

Console.WriteLine("Дробь " + this + " уничтожена.");

}

Далее в функции Main() приведены различные способы создания

объектов класса Fraction с помощью конструкторов.

static void Main(string[] args)

{

// создание дроби 2/3

Fraction d1 = new Fraction(2, 3, 0, 1);

// создание дроби -2 4/5

Fraction d2 = new Fraction(4, 5, 2, -1);

// создание дроби 2 1/3

Fraction d3 = new Fraction(4, 3, 1, 1);

// создание дроби 1 2/3

Fraction d4 = new Fraction(10, 6);

// создание дроби 3/7

Fraction d5 = new Fraction(3, 7);

// создание дроби 2 3/8

Fraction d6 = new Fraction(3, 8, 2);

// создание рационального числа 0

Fraction d7 = new Fraction();

. . .

}

15

1.3. Перегрузка операций для класса

«Рациональное число»

Для использования знаков арифметических операций и операций

сравнения перегрузим соответствующие операторы.

Поскольку любая дробь является вещественным числом,

переопределим оператор явного преобразования объекта класса

Fraction к вещественному типу данных double:

// операция преобразования дроби в тип double

public static explicit operator double(Fraction ob)

{

double res = (double)ob.sign\*(ob.intPart \* ob.denominator +

ob.numerator) / ob.denominator;

return res;

}

Данное преобразование удобно будет использовать при сравнении

дробей.

Перегрузку операций сравнения двух дробей (больше, больше или

равно, меньше, меньше или равно, равно, не равно) осуществим с

помощью статических методов класса Fraction. Эти операторы

должны возвращать значение типа bool. Заметим, что эти операторы

следует перегружать «парами». Например, если класс содержит

перегруженную операцию ”==”, то обязательно в нем должна быть

перегружена и операция ”!=”.

Операторы ”==” и ”!=” осуществляют поэлементное сравнение

двух дробей, которые представлены в смешанном виде. Остальные

операторы сравнения используют преобразование дроби к

вещественному числу и сравнивают полученные значения.

// операции сравнения двух дробей

public static bool operator == (Fraction ob1, Fraction ob2)

{

if (ob1.sign != ob2.sign || ob1.intPart != ob2.intPart ||

ob1.numerator \* ob2.denominator !=

ob1.denominator \* ob2.numerator)

return false;

return true;

}

16

public static bool operator != (Fraction ob1, Fraction ob2)

{

if (ob1.sign == ob2.sign && ob1.intPart == ob2.intPart &&

ob1.numerator \* ob2.denominator ==

ob1.denominator \* ob2.numerator)

return false;

return true;

}

public static bool operator > (Fraction ob1, Fraction ob2)

{

if ((double)ob1 <= (double)ob2)

return false;

return true;

}

public static bool operator < (Fraction ob1, Fraction ob2)

{

if ((double)ob1 >= (double)ob2)

return false;

return true;

}

public static bool operator >= (Fraction ob1, Fraction ob2)

{

if ((double)ob1 < (double)ob2)

return false;

return true;

}

public static bool operator <= (Fraction ob1, Fraction ob2)

{

if ((double)ob1 > (double)ob2)

return false;

return true;

}

Каждая арифметическая операция (“+”, “–“, “\*”, ”/”) перегружена

тремя методами-операторами для случаев, когда:

операндами операции являются объекты класса Fraction;

первый операнд – дробь, второй – целое число;

первый операнд – целое число, второй – объект-дробь.

Результатом выполнения этих операторов является новая дробь.

Рассмотрим подробнее реализацию операторов сложения.

Оператор сложения двух дробей после формирования результата

осуществляет преобразование к смешанному виду.

17

// операция сложения двух дробей

static public Fraction operator + (Fraction ob1, Fraction ob2)

{

Fraction res=new Fraction();

res.numerator = ob1.sign \* (ob1.intPart \* ob1.denominator +

ob1.numerator) \* ob2.denominator + ob2.sign \*(ob2.intPart

\*

ob2.denominator + ob2.numerator) \* ob1.denominator;

res.denominator = ob1.denominator \* ob2.denominator;

if (res.numerator < 0)

{

res.numerator \*= -1;

res.sign = -1;

}

res.GetMixedView();

return res;

}

В определении функции сложения дроби с целым числом

осуществляется преобразование этого целого числа в дробь (создается

новый объект класса Fraction, значение которого равно целому

числу) и вызов оператора сложения двух дробей.

// метод сложения дроби с целым числом

static public Fraction operator + (Fraction ob1, int a)

{

// если к дроби прибавляется число, равное 0,

// результат совпадает с операндом-дробью

if (a == 0)

return new Fraction(ob1.numerator, ob1.denominator,

ob1.intPart, ob1.sign);

// создание новой дроби ob2 = a

Fraction ob2=new Fraction(0, 1, Math.Abs(a), a/Math.Abs(a));

Fraction res = ob1 + ob2; //сложение двух дробей

return res;

}

// метод сложения целого числа и дроби

static public Fraction operator + (int a, Fraction ob1)

{

// если к дроби прибавляется число, равное 0,

// результат совпадает с операндом-дробью

if (a == 0)

return new Fraction(ob1.numerator, ob1.denominator,

ob1.intPart, ob1.sign);

// создание новой дроби ob2 = a

Fraction ob2=new Fraction(0, 1, Math.Abs(a), a/Math.Abs(a));

Fraction res = ob1 + ob2; //сложение двух дробей

18

return res;

}

Аналогичным образом определяют и другие арифметические

операции.

Обсудим особенности ввода и вывода рациональных дробей.

Поскольку при вводе пользователь задает символьную строку,

представляющую дробь, то задача ввода дроби сводится к созданию

метода преобразования символьной строки в дробь. При выводе

происходит противоположная операция. Поэтому для организации

вывода в класс Fraction введем операцию неявного преобразования

дроби в тип string, а для ввода – статический метод Parse()

формирования объекта-дроби из строки.

Для представления дроби пользователю удобно использовать ее

привычный математический вид с учетом существования целой и/или

дробной части. Именно такой вид формируется в операции

преобразования дроби в строку.

// операция получения строкового представления дроби

static public implicit operator string(Fraction ob)

{

string res = "";

// знак числа выводится, только если число отрицательно

if (ob.sign < 0)

res = res + "-";

// если целая часть не равна 0, выводим ее

if (ob.intPart != 0)

res = res + ob.intPart;

// дробная часть печатается, если числитель не равен 0

if (ob.numerator != 0)

res = res + " " + ob.numerator + "/" + ob.denominator;

// если и целая часть и дробная часть равны 0,

// то число равно 0

if (ob.intPart == 0 && ob.numerator == 0)

res = "0";

return res;

}

В том же формате будет осуществляться и ввод данных из

символьной строки. В методе Parse() производится выделение

составных частей числа – знака, целой части, числителя и знаменателя.

// метод получения дроби из строки

public static Fraction Parse(string str)

19

{

int intPart, numerator, denominator, sign;

// разделение строки на подстроки

// с помощью разделителя-пробела

string [] strs=str.Split(' ');

string[] strs1;

Fraction res;

if (strs.Length == 1)

{

// в строке не было найдено пробелов

// производим разделение строки по символу ‘/’

strs1 = str.Split('/');

if (strs1.Length == 1)

{

// число задано в виде только целой части

// выделяем целую часть

intPart = int.Parse(strs1[0]);

// в зависимости от значения целой части,

// формируем новую дробь

if (intPart!=0)

res = new Fraction(0, 1, Math.Abs(intPart),

intPart / Math.Abs(intPart));

else

res = new Fraction(0, 1, Math.Abs(intPart), 1);

return res;

}

else

{

// число задано в виде только дробной части

// выделяем отдельно числитель и знаменатель

numerator = int.Parse(strs1[0]);

denominator = int.Parse(strs1[1]);

sign = 1;

// определяем знак числа по знаку числителя

if (numerator < 0)

{

numerator = -numerator;

sign = -1;

}

// формируем новую дробь и приводим ее

// к несократимому виду

res = new Fraction(numerator, denominator, 0, sign);

res.GetMixedView();

return res;

}

}

// дробь задана в смешанном виде

// отделяем дробную часть по разделителю ‘/’

strs1 = strs[1].Split('/');

intPart = int.Parse(strs[0]);

// определяем знак числа по знаку целой части

20

if (intPart < 0)

{

intPart = -intPart;

sign = -1;

}

else

sign = 1;

numerator = int.Parse(strs1[0]);

denominator = int.Parse(strs1[1]);

// формируем новую дробь и приводим ее

// к несократимому виду

res = new Fraction(numerator, denominator, intPart, sign);

res.GetMixedView();

return res;

}

Приведем пример использования объектов класса Fraction и

операций работы с ними (Рис. 1.1).

static void Main(string[] args)

{

// создание дроби 2/3

Fraction r1 = new Fraction(2, 3, 0, 1);

Console.WriteLine("r1 = " + r1);

// создание дроби 5/7

Fraction r2 = new Fraction(5, 7, 0, 1);

Console.WriteLine("r2 = " + r2);

Console.WriteLine("-r2 = " + -r2);

Console.WriteLine("r2 = " + (double)r2);

// вызов оператора "==" для двух дробей

if (r1 == r2)

Console.WriteLine("r1 == r2");

else

Console.WriteLine("r1 != r2");

// вызов оператора ">" для двух дробей

if (r1 > r2)

Console.WriteLine("r1 > r2");

else

Console.WriteLine("r1 <= r2");

// вызов оператора "<=" для двух дробей

if (r1 > r2)

Console.WriteLine("r1 <= r2");

else

Console.WriteLine("r1 > r2");

// вызов оператора "+" для двух дробей

Fraction d = r1 + r2;

Console.WriteLine("r1 + r2 = " + d);

// вызов оператора "+" для дроби и числа

d = r1 + (-11);

Console.WriteLine("r1 + (-11) = " + d);

// вызов оператора "+" для числа и дроби

21

d = 5 + r1;

Console.WriteLine("5 + r1 = " + d);

. . .

}

Рис.1.1. Демонстрация операций с классом Fraction.

1.4. Быстрая сортировка массива дробей

Тип Fraction может использоваться в качестве подставляемого

вместо обобщенного типа данных при генерации функций и классов на

основе обобщений. Приведем пример использования обобщения

функции быстрой сортировки для массива дробей. В сгенерированном

методе с типом Fraction требуется организовать сравнение двух

дробей. Для этого класс Fraction должен раскрывать интерфейс

IComparable и иметь метод сравнения дробей CompareTo().

class Fraction : IComparable

{

int sign; // знак дроби (+ или -)

int intPart; // целая часть дроби

int numerator; // числитель дроби

int denominator; // знаменатель дроби

// метод сравнения двух дробей

public int CompareTo(object ob)

{

if (this < (ob as Fraction)) return -1;

if (this > (ob as Fraction)) return 1;

return 0;

}

22

. . .

}

Определим обобщенный метод быстрой сортировки, указав

ограничение на использование в качестве обобщенных типов только

тех, которые раскрывают интерфейс IComparable. Пусть этот метод

является статическим для главного класса приложения Program.

class Program

{

static void QuickSort<T>(T [] m, int l, int r)

where T: IComparable

{

if(l == r)

return;

int i = l, j = r;

// выбирается элемент, делящий массив на две части

T selected = m[l];

// осуществляется поиск и перестановка элементов, меньших

// выбранного, с конца массива и больших выбранного

// с начала массива.

while(i != j)

{

while(m[j].CompareTo(selected) >= 0 && j > i)

j--;

if(j > i)

{

m[i] = m[j];

while(m[i].CompareTo(selected) <= 0 && i < j)

i++;

m[j] = m[i];

}

}

// выбранный элемент устанавливается на надлежащее место

m[i] = selected;

// если существуют элементы слева от выбранного,

// сортируем эту часть массива

if(l <= i-1)

QuickSort(m, l, i-1);

// то же проводится с правой частью массива,

// если она существует

if( i+1 < r)

QuickSort(m , i+1, r);

}

static void Main(string[] args)

{

Fraction[] a = { new Fraction(4,2,3,1),

new Fraction(1,6,5,-1),

23

new Fraction(0,2,7,1),

new Fraction(10,2,3,-1),

new Fraction(2,13,20,1),

new Fraction(0,1),

new Fraction(1,1,3,1),

new Fraction(2,5,7,-1),

new Fraction(4,8,2,-1),

new Fraction(4,1,3,-1)};

for (int i = 0; i < a.Length; i++)

Console.Write(a[i]+" ");

Console.WriteLine();

// генерация и вызов функции быстрой сортировки

// для массива дробей

QuickSort(a, 0, 9);

for (int i = 0; i < a.Length; i++)

Console.Write(""+a[i]+" ");

Console.WriteLine();

. . .

}

Это же обобщение можно применять и для массива чисел типов

int, double, а также для массивов, элементами которых являются

объекты классов (пользовательских или библиотечных), которые

раскрывают интерфейс IComparable.