МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Курсовой проект

**Реализация класса**

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АТ-03  Студент:  Буханцев Максим Олегович | Преподаватель:  Воронов Виталий Владимирович |

Новосибирск

2021

**Оглавление:**

* Техническое задание
* Тип данных
* Описание класса
  + Типы элементы
  + Функции элементы
  + Арифметические операторы
  + Операторы сравнения
  + Операторы преобразования к базовому типу
  + Оператор вывода
* Приложение
  + Функция НОД (Алгоритм Евклида).
  + Перезагрузка операторов.

**Техническое задание**

Разработать класс, представляющий заданный тип данных (ТД). Класс должен включать в себя (если это необходимо и подходит по смыслу к ТД):

* Конструкторы (по умолчанию, с параметрами, копии);
* Деструктор;
* Функцию вывода на экран;
* Перегруженные операторы;
  + Оператор присваивания;
  + Бинарные арифметические (+, -, \*, /) с операндами ТД и ТД, ТД и базовый тип, базовый тип и ТД;
  + Арифметика с накоплением (+=, -=);
  + Унарные (++, --) в префиксной и постфиксной форме;
  + Логические (<, >, ==, !=);
  + Операторы взятия элемента ([] или ()) по номеру или ключу;
  + Операторы преобразования типа к любому базовому;

Написать головную программу, обеспечивающую проверку всех реализованных свойств и методов класса.

**Тип данных**

Вариант 7

Точные числа (дробь).

**Описание класса**

Класс определен в заголовочном файле “fract\_num.h”.

class fract\_num

Реализация методов класса находится в файле “fract\_num.cpp”.

Дробь может работать только с типом данных int.

Элементы дроби (числитель, знаменатель) хранятся в переменных типа данных

long int.

Для данного класса перегружены операторы: присваивания; бинарные (+, -, \*, /) ТД и базовый тип, базовый тип и ТД; арифметики с накоплением (+=, -=); унарные (++, --) в префиксной и постфиксной форме; логические (<, >, ==, !=); преобразования к базовому типу (int и double).

**Типы элементы**

|  |  |
| --- | --- |
| Тип элемента | Определение |
| numerator | Числитель. private |
| denominator | Знаменатель. private |

**Функции элементы**

|  |  |
| --- | --- |
| Функция | Описание |
| gcd | Алгоритм Евклида, который сокращает дробь на НОД.privare |
| (конструктор по умолчанию) | Создает fract\_num со значением 0/0. public |
| (конструктор с параметрами) | Создает fract\_num с заданными параметрами ( 2 параметра – числитель, знаменатель) . public |
| (конструктор копирования) | Создает новую fract\_num, копию переданной (1 параметр – fract\_num) . public |
| operator= | Оператор присваивания. public |

**Арифметические операторы**

|  |  |
| --- | --- |
| operator+(fract\_num, fract\_num) | Возвращает сумму двух fract\_num. public |
| operator+(fract\_num, int) | Возвращает сумму fract\_num и int. public |
| operator+(int, fract\_num) | Возвращает сумму int и fract\_num. public |
| operator-(fract\_num, fract\_num) | Возвращает разность двух fract\_num. public |
| operator-(fract\_num, int) | Возвращает разность fract\_num и int. public |
| operator-(int, fract\_num) | Возвращает разность int и fract\_num. public |
| operator\*(fract\_num, fract\_num) | Возвращает произведение двух fract\_num. public |
| operator\*(fract\_num, int) | Возвращает произведение fract\_num и int. public |
| operator\*(int, fract\_num) | Возвращает произведение int и fract\_num. public |
| operator/(fract\_num, fract\_num) | Возвращает частное двух fract\_num. public |
| operator/(fract\_num, int) | Возвращает частное int и fract\_num. public |
| operator/(int, fract\_num) | Возвращает частное int и fract\_num. public |
| operator+=(fract\_num) | Прибавляет fract\_num к fract\_num, для которого вызывается. public |
| operator-=(fract\_num) | Вычитает fract\_num из fract\_num, для которого вызывается. public |
| operator++() | Префиксный оператор, прибавляет к числителю знаменатель. public |
| operator--() | Префиксный оператор, отнимает от числителя знаменатель. public |
| operator++(int) | Постфиксный оператор, прибавляет к числителю знаменатель. public |
| operator--(int) | Постфиксный оператор, отнимает от числителя знаменатель. public |

**Операторы сравнения**

|  |  |
| --- | --- |
| operator== | Сравнение двух дробей. public |
| operator!= | Сравнение двух дробей. public |
| operator< | Сравнение двух дробей. public |
| operator> | Сравнение двух дробей. public |

**Операторы преобразования к базовому типу**

|  |  |
| --- | --- |
| operator int | Возвращает целую часть дроби. public |
| operator double | Возвращает дробь в виде дробного числа. public |

**Оператор вывода**

|  |  |
| --- | --- |
| operator << | Выводит дробь в консольное окно. public |

**Приложение**

**Алгоритм НОД.**

Сокращаем числитель и знаменатель на наибольший общий делитель. Вызов функции происходит в конструкторе с параметрами и конструкторе копирования.

int fract\_num::gcd(long int a, long int b) {

int c;

while (b)

{

c = a % b;

a = b;

b = c;

}

return abs(a);

}

**Конструкторы.**

Конструктор по умолчанию просто обнуляет дробь (Получается что дробь равна 0/0).

fract\_num::fract\_num()

{

numerator = 0;

denominator = 0;

}

Конструктор с параметрами получает на вход два числа с типа long int. Первое это числитель, а второе знаменатель. Далее вызывается функция gcd, которая находит НОД и делит числитель и знаменатель на него, если он не равен 0.

fract\_num::fract\_num(long int n, long int d)

{

int k = gcd(n, d);

if (k > 0)

{

numerator = n / k;

denominator = d / k;

}

else

{

numerator = n;

denominator = d;

}

}

Конструктор копирования получает на вход fract\_num, потом вызываем функцию gcd, которая находит НОД, и если НОД не равен 0, то делит на него, а потом копирует полученные числитель и знаменатель.

fract\_num::fract\_num(const fract\_num& a)

{

int k = gcd(a.numerator, a.denominator);

if (k > 0)

{

numerator = a.numerator / k;

denominator = a.denominator / k;

}

else

{

numerator = a.numerator;

denominator = a.denominator;

}

}

**Вывод на экран.**

Перегруженный оператор потоков дает нам возможность выводить fract\_num в привычном виде.

std::ostream& operator<<(std::ostream& os, const fract\_num& a)

{

os << a.numerator << "/" << a.denominator << '\n';

return os;

}

**Бинарные арифметические операторы** (ТД и ТД, ТД и базовый тип, базовый тип и ТД) имеют схожую логику перезагрузки. Поэтому я опишу только работу перегрузки бинарного оператора + с параметрами ТД и ТД, базовый и ТД.

ТД и ТД для оператора +.

Если знаменатель одинаковый, то просто складываем числители, иначе приводим к общему знаменателю и складываем дроби.

fract\_num operator+ (const fract\_num& a, const fract\_num& b)

{

if (a.denominator == b.denominator)

return fract\_num(a.numerator + b.numerator, a.denominator);

else

return fract\_num(a.numerator \* b.denominator + b.numerator \* a.denominator, a.denominator \* b.denominator);

}

ТД и базовый для оператора +.

У базового типа числитель равен ему, а знаменатель единице. Приводим к обшему знаменателю и складываем.

fract\_num operator+ (const fract\_num& a, const int& b)

{

return fract\_num(a.numerator + b \* a.denominator, a.denominator);

}

Другие бинарные операторы перезагружаются аналогично, главное сохранять их арифметическую логику.

**Арифметика с накоплением.**

Если знаменатель одинаковый, то просто складываем числители, иначе приводим к общему знаменателю и складываем дроби. В конце возвращаем указатель this.

fract\_num fract\_num::operator+= (const fract\_num& b)

{

if (this->denominator == b.denominator)

{

this->numerator = this->numerator + b.numerator;

}

else

{

this->numerator = this->numerator \* b.denominator + b.numerator \* this->denominator;

this->denominator = this->denominator \* b.denominator;

}

return \*this;

}

Оператор -= имеют такую же логику, но он не складывает, а вычитает.

**Унарные арифметические операторы.**

Префиксная и постфиксная форма отличается только тем, что префиксная не получается на вход int, а постфиксная получает.

Оператор ++ прибавляет знаменатель к числителю.

fract\_num operator++ (const fract\_num& a)

{

return fract\_num(a.numerator + a.denominator, a.denominator);

}

Оператор – вычитает из числителя знаменатель.

fract\_num operator-- (const fract\_num& a, int)

{

return fract\_num(a.numerator - a.denominator, a.denominator);

}

**Логические операторы.**

Логические операторы получают на вход два fract\_num. Далее они в соответствии со своей логикой сравнивают их.

Приведу в пример перегрузку оператора !=.

Сначала мы приведем дроби к общему знаменателю, и если после этого у них отличаются числители, то они не равны.

ool operator!= (const fract\_num& a, const fract\_num& b)

{

return (a.numerator \* b.denominator != b.numerator \* a.denominator) ? true : false;

}

**Преобразование ТД к базовому.**

Я решил приводить дробь к двум базовым типам данных: int и double.

При приведении к int мы получаем целую часть от дроби.

fract\_num::operator int() { return numerator/denominator; }

При приведении к double мы получаем число к плавающей точкой, которое равно нашей дроби с точностью double.

fract\_num::operator double() { return double(numerator) / double(denominator); }