МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная
математика»
Кафедра: 806 «Вычислительная математика и программирование»

Реферат
по курсу «Фундаментальная
информатика»
I семестр
Тема:
«Изучение основ ООП на языке C++»

Группа:	М8О-109Б-22
Студент:	Нгуен Н.Х.А.
Преподаватель:	Сысоев М.А.
Оценка:	
Дата:	

Москва, 2022

Содержание

1. История появления C++
2. Объектно-ориентированное программирование
3. Принципы ООП
4. Классы в ООП
5. Класс рациональных чисел
5.1. Конструкторы класса
5.2. Перегрузка операции присваивания
5.3. Перегрузка унарных операторов
5.4. Перегрузка бинарных операторов
5.5. Перегрузка операторов сравнения
5.6. Перегрузка операторов инкремента и декремента
5.7. Перезагрузка операторов ввода и вывода
6. Список литературы

1. История появления C++

В 1980 году Бьерн Страуструп в AT&T Bell Labs стал разрабатывать расширение языка C под условным названием C++. Стиль ведения разработки вполне соответствовал духу, в котором создавался и сам язык C, – в него вводились те или иные возможности с целью сделать более удобной работу конкретных людей и групп. Первый коммерческий транслятор нового языка, получившего название C++ появился в 1983 году. Он представлял собой препроцессор, транслировавший программу в код на C. Однако фактическим рождением языка можно считать выход в 1985 году книги Страуструпа. Именно с этого момента C++ начинает набирать всемирную популярность.

Главное нововведение C++ – механизм классов, дающий возможность определять и использовать новые типы данных. Программист описывает внутреннее представление объекта класса и набор функций-методов для доступа к этому представлению. Одной из заветных целей при создании C++ было стремление увеличить процент повторного использования уже написанного кода. Концепция классов предлагала для этого механизм наследования. Наследование позволяет создавать новые (производные) классы с расширенным представлением и модифицированными методами, не затрагивая при этом скомпилированный код исходных (базовых) классов. Вместе с тем наследование обеспечивает один из механизмов реализации полиморфизма базовой концепции объектно-ориентированного программирования, согласно которой, для выполнения однотипной обработки разных типов данных может использоваться один и тот же код. Собственно, полиморфизм тоже один из методов обеспечения повторного использования кода.

Введение классов не исчерпывает всех новаций языка C++. В нем реализованы полноценный механизм структурной обработки исключений, отсутствие которого в C значительно затрудняло написание надежных программ, механизм шаблонов – изощренный механизм макрогенерации, глубоко встроенный в язык, открывающий еще один путь к повторной используемости кода, и многое другое. Таким образом, генеральная линия развития языка была направлена на расширение его возможностей путем введения новых высокоуровневых конструкций при сохранении сколь возможно полной совместимости с ANSI C. Конечно, борьба за повышение уровня языка шла и на втором фронте: те же классы позволяют при грамотном подходе упрятывать низкоуровневые операции, так что программист фактически перестает непосредственно работать с памятью и системно-зависимыми сущностями.

2. Объектно-ориентированное программирование

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Объект – это нечто, имеющее четко определенные границы. Однако, этого недостаточно, чтобы отделить один объект от другого или дать оценку качества абстракции. Объект обладает состоянием, поведением и идентичностью; структура и поведение схожих объектов определяет общий для них класс; термины «экземпляр класса» и «объект» взаимозаменяемы.

Класс – это множество объектов, обладающих общей структурой, поведением и семантикой. Отдельный объект – это экземпляр класса. Класс представляет лишь абстракцию существенных свойств объекта.

3. Принципы ООП

Объектно-ориентированное программирование строится на трех основополагающих принципах: инкапсуляция, полиморфизм и наследование.

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя. Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

Наследование – это свойство системы, позволяющее описать новый класс на основе существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом. Необходимо отметить, что производный класс полностью удовлетворяет спецификации родительского, однако может иметь дополнительную функциональность. С точки зрения интерфейсов, каждый производный класс полностью реализует интерфейс родительского класса. Обратное не верно.

Полиморфизм – это свойство системы, дающее возможность обработки разных типов данных, т. е. принадлежащих к разным классам, с помощью "одной и той же" функции, или метода. В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий.

4. Классы в ООП

Класс является типом данных, определяемым пользователем. В классе задаются свойства и поведение какого-либо предмета или процесса в виде полей данных (аналогично структуре) и функций для работы с ними.

Существенным свойством класса является то, что детали его реализации скрыты от пользователей класса за интерфейсом. Интерфейсом класса являются заголовки его открытых методов. Таким образом, класс как модель объекта реального мира является черным ящиком, замкнутым по отношению к внешнему миру.

Идея классов является основой объектно-ориентированного программирования (ООП). Основные принципы ООП были разработаны еще в языках Simula-67 и Smalltalk, но в то время не получили широкого применения из-за трудностей освоения и низкой эффективности реализации. В C++ эти концепции реализованы эффективно и непротиворечиво, что и явилось основой успешного распространения этого языка и внедрения подобных средств в другие языки программирования.

Класс – это описание определяемого типа. Любой тип данных представляет собой множество значений и набор действий, которые разрешается выполнять с этими значениями. Например, сами по себе числа не представляют интереса – нужно иметь возможность ими оперировать: складывать, вычитать, вычислять квадратный корень и т. д. В C++ множество значений нового типа определяется задаваемой в классе структурой данных, а действия с объектами нового типа реализуются в виде функций и перегруженных операций C++.

Данные класса называются полями (по аналогии с полями структуры), а функции класса – методами. Поля и методы называются элементами класса.

5. Класс рациональных чисел

В качестве примера создадим класс рациональных чисел. Для этого требуется задать его свойства и поведение.

- В этом классе 4 скрытых поля: numerator (числитель), denominator (знаменатель), gcd (функция вычисления НОД), reduce (функция сокращения дроби).

```
class RationalNumber {
private:
    int numerator;
    int denominator;

    int gcd(int a, int b) {
        return (!b) ? a : gcd(b, a % b);
    }

    void reduce() {
        int d = gcd(numerator, denominator);
        numerator /= d;
        denominator /= d;
        if (denominator < 0) {
            numerator *= -1;
            denominator *= -1;
        }
    }
...
}
```

- Получить значения numerator и denominator извне можно с помощью методов Numerator() и Denominator().

```
int Numerator() const {
    return numerator;
}

int Denominator() const {
    return denominator;
}
```

5.1. Конструкторы класса

- Конструктор предназначен для инициализации объекта и вызывается автоматически при его создании. По умолчанию, числитель равен 0, знаменатель 1. При инициализации производятся проверка знаменателя на равенство нулю и сокращение дроби.

```
RationalNumber(int numerator_ = 0, int denominator_ = 1) :
        numerator(numerator_), denominator(denominator_) {
    assert(denominator_ != 0);
    reduce();
}
```

- Конструктор копирования – это специальный вид конструктора, получающий в качестве единственного параметра указатель на объект этого же класса. Этот конструктор вызывается в тех случаях, когда новый объект создается путем копирования существующего.

```
RationalNumber(const RationalNumber &other) {
    numerator = other.numerator;
    denominator = other.denominator;
}
```

- Конструктор перемещения позволяет переместить объект вместо его копирования для увеличения производительности. Он получает в качестве единственного параметра rvalue ссылку (на временный объект).

```
RationalNumber(RationalNumber &&other) noexcept {
    numerator = other.numerator;
    denominator = other.denominator;
    other.numerator = 0;
    other.denominator = 1;
}
```

5.2. Перегрузка операции присваивания

- Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого. В качестве параметра принимается единственный аргумент – ссылку на присваиваемый объект.

```
RationalNumber &operator=(const RationalNumber &other) = default;
```

- Операция присваивания move аналогична конструктору перемещения.

```
RationalNumber &operator=(RationalNumber &&other) noexcept {
    numerator = other.numerator;
    denominator = other.denominator;
    other.numerator = 0;
    other.denominator = 1;
    return *this;
}
```

5.3. Перегрузка унарных операторов

- Унарный плюс (формально ничего не меняет).

```
RationalNumber operator+() const {
    return RationalNumber(numerator, denominator);
}
```

- Унарный минус возвращает объект с противоположным знаком.

```
RationalNumber operator-() const {
    return RationalNumber(-numerator, denominator);
}
```

5.4. Перегрузка бинарных операторов

- Функции операторов +=, -=, *=, /= определены как члены классов. Левый операнд доступен через указатель this и представляет текущий объект, а правый операнд передается в функцию в качестве единственного параметра.

```
RationalNumber &operator+=(const RationalNumber &rhs) {
    numerator = numerator * rhs.denominator + rhs.numerator * denominator;
    denominator *= rhs.denominator;
    reduce();
    return *this;
}

RationalNumber &operator-=(const RationalNumber &rhs) {
    numerator = numerator * rhs.denominator - rhs.numerator * denominator;
    denominator *= rhs.denominator;
    reduce();
    return *this;
}

RationalNumber &operator*=(const RationalNumber &rhs) {
    numerator *= rhs.numerator;
    denominator *= rhs.denominator;
    reduce();
    return *this;
}

RationalNumber &operator/=(const RationalNumber &rhs) {
    int temp = rhs.numerator;
    numerator *= rhs.denominator;
    denominator *= temp;
    reduce();
    return *this;
}
```

- Функции операторов +,-,*,/ не являются частью класса RationalNumber и определены вне его. Они является бинарными, поэтому принимают два параметра, объекты класса RationalNumber. Возвращают функции так же объекты этого класса.

```
RationalNumber operator+(const RationalNumber &lhs, const RationalNumber
&rhs) {
    RationalNumber result = lhs;
    result += rhs;
    return result;
}

RationalNumber operator-(const RationalNumber &lhs, const RationalNumber
&rhs) {
    RationalNumber result = lhs;
    result -= rhs;
    return result;
}

RationalNumber operator*(const RationalNumber &lhs, const RationalNumber
&rhs) {
    RationalNumber result = lhs;
    result *= rhs;
    return result;
}

RationalNumber operator/(const RationalNumber &lhs, const RationalNumber
&rhs) {
    RationalNumber result = lhs;
    result /= rhs;
    return result;
}
```

5.5. Перегрузка операторов сравнения

- Оператор ==

```
bool operator==(const RationalNumber &rhs) const {
    return numerator == rhs.Numerator() && denominator == rhs.Denominator();
}
```

- Оператор !=

```
bool operator!=(const RationalNumber &rhs) const {
    return !(*this == rhs);
}
```

5.6. Перегрузка операторов инкремента и декремента

- Префиксные операторы должны возвращать ссылку на текущий объект после совершения операции.

```
RationalNumber &operator++() {
    numerator += denominator;
    // здесь нет необходимости вызывать reduce(), так как дробь остаётся
несократимой
    return *this;
}

RationalNumber &operator--() {
    numerator -= denominator;
    return *this;
}
```

- Постфиксные операторы должны возвращать значение объекта до совершения операции.

```
RationalNumber operator++(int) {
    RationalNumber prev = *this;
    ++*this;
    return prev;
}

RationalNumber operator--(int) {
    RationalNumber prev = *this;
    --*this;
    return prev;
}
```

5.7. Перезагрузка операторов ввода и вывода

- Перегрузка оператора чтения из потока >>. В качестве левого операнда оператор >> должен иметь неконстантную ссылку на поток istream, а значит данный оператор всегда определяется внешней функцией, а не методом класса. Кроме того, функция должна быть объявлена дружественной, чтобы иметь возможность модификации объекта.

```
class RationalNumber {
...
        friend std::istream &operator>>(std::istream &, RationalNumber &);
};
std::istream &operator>>(std::istream &in, RationalNumber &n) {
    in >> n.numerator >> n.denominator;
```

```
    return in;
}
```

- Перегрузка оператора вывода в поток <<. В качестве левого операнда оператор << должен иметь неконстантную ссылку на поток ostream, а значит данный оператор всегда определяется внешней функцией, а не методом класса. В качестве результата следует возвращать исходную ссылку на поток.

```
std::ostream &operator<<(std::ostream &os, const RationalNumber &n) {
    return os << "(" << n.Numerator() << "/" << n.Denominator() << ")";
}
```

6. Список литературы

1. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Г. Буч и др. // Пер. с англ. – М.: «И.Д. Вильямс», 2010. – 720 с.

2. Павловская, Т.А. C/C++. Программирование на языке высокого

уровня / Т.А. Павловская // СПб.: Питер, 2009. – 461 с.

3. Подбельский, В.В. Язык Си++ : Учеб. пособие для вузов по направлениям «Приклад. математика» и «Вычисл. машины, комплексы, системы и

сети» / В.В. Подбельский // М.: Финансы и статистика , 2001 г. – 559 с.

4. Страуструп, Б. Язык программирования C++. Специальное издание.

Пер. с англ. / Б. Страуструп // СПб., М.: «Невский диалект» – «Издательство

БИНОМ», 2008 г. – 1104 с.

5. Майерс С. Эффективное использование C++ / С. Майерс // СПб.: Питер, 2006 г. – 240 с.

6. Майерс С. Эффективное использование C++. 35 новых способов улучшить стиль программирования /С. Майерс // СПб.: Питер, 2006. –224 с