Прохорова Юлия

Задание

К.в. 48
Упр. 39
Общ. 87

Контрольные вопросы

1. 5. Многовато. Геттеры, сеттеры и операторы (кроме =) к специальным функциям-членам не относятся.
2. 5
3. 5
4. 5
5. 5
6. 5
7. 5
8. 3 -2. (ПОЯСНИТЬ!) Не только один лишь перемещающий оператор присваивания занят в семантике перемещения.
9. 5
10. 5

Упражнения

Упражнение 1

10

На будущее - разделяйте ваш проект на main.cpp, MyClass.h, MyClass.cpp.
Стоит объявлять функциии, не изменяющие переменные класса, как const. У вас такими являются, например, void get_arr()

1.1. -5. `Myclass(const Myclass &myclass) :n(myclass.n), v(move(myclass.v)){`
Использование move() к константному объекту - очень плохая практика: вызов move() подразумевает, что её аргумент будет изменен. Вам же его нужно не менять, а глубоко скопировать.

1.2. -5. Перед вызовом delete:

```
Myclass& operator=(Myclass&& myclass) {
    if (&myclass == this)
        return *this;

    delete arr;
```

стоит убедится, что его аргумент не nullptr, в противном случае это может привести к неопределенному поведению. У вас массив arr может быть равен nullptr, например, если объект создавался конструктором по умолчанию. Тоже самое относится к перемещаемому оператору присваивания.

1.3. -5. Такую функцию

```
void get_arr() {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
    cout << endl;
```

Лучше назвать print_arr(). “Get” подразумевает, что функция-член является геттером, т.е. возвращает значение переменной-члена arr, а не печатает её.

Упражнение 2

29

2.1. -5. Разделите ваше программу на main.cpp, rational.cpp и rational.h

2.2. -5 Поместите вашу библиотеку rational в собственное пространство имён.

2.3. -5. Вынесите код сокращения дроби из конструктора в отдельную функцию-член. И вызывайте её всякий раз, когда дробь была изменена.

2.4. -1. Юлия, в вашем классе нет указателей, так что нет необходимости писать руками свою семантику копирования и перемещения. Достаточно всё оставить по умолчанию:

```
Rational(const Rational &) = default;
Rational(Rational&&) = default;
Rational& operator=(const Rational&) = default;
Rational & operator=(Rational&&) = default;
```

2.5. -5. Вот это очень плохой метод сравнения дробей:

```
bool operator > (Rational a, Rational b) {
    return (a.Numerator() / (double)a.Denominator()) >
double(b.Numerator() / (double)b.Denominator());
}
bool operator < (Rational a, Rational b) {
    return (a.Numerator() / (double)a.Denominator()) < (b.Numerator() /
(double)b.Denominator());
```

```
}
```

Во-первых, функции-члены `Numerator и Denominator` возвращают int, поэтому их отношение будет трактоваться компилятором как целочисленное деление, а уже его результат будет приведён к double. У вас будет ⅓ == ½, потому что double(1/3) == 0.0 и double(1/2) == 0.0.

Во-вторых, при делении может возникнуть ошибка округления или не хватит машинной точность double, поэтому две неравные дроби могут оказаться равными.

Лучше для проверки равенства дробей p/q и m/n использовать перемножение: p*n == q*m. Это ещё и быстрее, потому что операция целочисленная, а не с плавающей точкой.