Перегрузка операторов

Тема 4

Дружественность (friend)

- Классы могут дружить с другими классами, отдельными функциями других классов, с обычными функциями
- Друзья класса получают доступ к private части класса
- Отношения дружественности требуют явного объявления
- Отношения дружественности несимметричны
- Отношения дружественности не транзитивны
- Друзья не наследуются
- Отношения дружественности нарушают инкапсуляцию

Перегрузка операторов

Позволяет использовать встроенные операторы для пользовательских типов

Правила перегрузки

- Оператор реализуется как функция, в имени которой есть слово operator и знак операции
- Оператор может быть перегружен как член класса, либо как внешняя функция
- Операторы `->` `[]` `()` `=` должны быть перегружены как члены класса

Правила перегрузки. Нельзя:

- создавать новые лексемы операторов;
- изменять арность операции;
- изменять приоритет операции;
- изменять ассоциативность операции;
- перегружать следующие операторы:

  `::` `.` `.*` `?:` `sizeof` `typedef`

- изменить смысл операции для встроенных типов.

Перегрузка операторов для класса рациональная дробь

Заголовочный файл Rational.h

```
class Rational
{
public:
…
   Rational operator+ (const Rational& right)const;
   Rational operator- (const Rational& right)const;
   Rational operator* (const Rational& right)const;
   Rational operator/(const Rational& right)const;
   operator double()const;
  …
   };
std::ostream& operator<<(std::ostream& out,
const  Rational& ref);
std::istream& operator>>(std::istream& in,
Rational& ref);
```

Описание оператора +

```
Rational Rational::operator +(const Rational& right)const
{
    Rational res;
    res.numerator = numerator*right.denominator+
                    denominator*right.numerator;
    res.denominator = denominator*right.denominator;
    res.Shorten();
    return res;
}
```

Файл исходного кода Rational.cpp

Описание оператора -

```
Rational Rational::operator -(const Rational &right) const
{
    Rational res;
    res.numerator = numerator*right.denominator-
                    denominator*right.numerator;
    res.denominator = denominator*right.denominator;
    res.Shorten();
    return res;
}
```

Файл исходного кода Rational.cpp

Описание оператора *

```
Rational Rational::operator *(const Rational &right) const
{
   Rational res;
   res.numerator   = numerator*right.numerator;
   res.denominator = denominator*right.denominator;
   res.Shorten();
   return res;
}
```

Файл исходного кода Rational.cpp

Описание оператора /

```
Rational Rational::operator /(const Rational &right) const {
    Rational res;
    res.numerator = numerator*right.denominator;
    res.denominator = denominator*right.numerator;
    if(0 == res.denominator)  {
        res.denominator = 1;
        res.numerator = 0;
        std::cout << "Error divide by zero" <<std::endl;
        return res;
    }
    res.Shorten();
    return res;
}
```

Файл исходного кода Rational.cpp

Описание оператора приведения к типу double

```
Rational::operator double()const
{
   return (double)numerator / denominator;
}
```

Файл исходного кода Rational.cpp

Описание операторов ввода/вывода

```
std::ostream& operator<<(std::ostream& out,
                         const Rational& ref){
        ref.Print();
        return out;
}
```

```
std::istream& operator>>(std::istream& in,
                         Rational& ref) {
        int n,d;
        in >> n >> d;
        ref.SetRational(n,d);
        return in;
}
```

Файл исходного кода Rational.cpp

Использование операторов

Файл исходного кода test.cpp

```
int main(){
    Rational r1(4,5), r2(3,9);
    std::cout << r1 << r2;
    std::cout << (double)r1 << std::endl
              << (double)r2 << std::endl;
    Rational res;
    res = r1 + r2;
    std::cout << res << (double)res << std::endl;
    res = r1 - r2;
    std::cout << res << (double)res << std::endl;
    res = r1 * r2;
    std::cout << res << (double)res << std::endl;
    res = r1 / r2;
    std::cout << res << (double)res << std::endl;
    std::cin >> r1;
    std::cout << r1 << (double)r1 << std::endl;
    return 0;
}
```

Особенности перегрузки инкремента/декремента

- Операции существуют в двух формах:
  - префиксная;
  - постфиксная.
- У постфиксной формы появляется фиктивный параметр `int`

Перегрузка операторов для класса вектор

Заголовочный файл vector.h

```
class Vector{
public:
…
    Vector& operator=(const Vector& right);      //copy =
    Vector& operator=(Vector&& right);           //move =
    Vector operator+(const Vector& right)const;
    Vector& operator+(double a);
    double & operator[](int index);              //differs by const
    double operator[](int index)const;           //differs by const
    Vector& operator++();                        //pre-fix
    Vector operator++(int);                      //post-fix
    Vector& operator--();                        //pre-fix
    Vector operator--(int);                      //post-fix
…  };
double operator +(double a, const Vector& right);
std::istream & operator>>(std::istream& in, Vector& ref);
std::ostream& operator<<(std::ostream& out, const Vector& ref);
```

Перегрузка копирующего оператора =

Можно использовать без предварительной перегрузки

Алгоритм

1. Защита от самоприсваивания
2. Очистка ранее выделенной памяти
3. Выделение нового блока памяти и работа с ним
4. Возврат измененного объекта

Перегрузка копирующего оператора =

```
Vector& Vector::operator =(const Vector &right) {
    if(this != &right)//защита от самоприсваивания
    {
        delete[] arr;//очистка памяти
        //выделение нового блока и работа с ним
        arr = new double[size = right.size];
        for(int i = 0;i < size; i++)
            arr[i] = right[i];
    }
    return *this;//возврат объекта
}
```

Файл исходного кода vector.cpp

Перегрузка перемещающего оператора =

Файл исходного кода vector.cpp

```
Vector& Vector::operator =(Vector &&right) {
    if(this != &right)//защита от самоприсваивания
    {
        delete [] arr;//очистка памяти
        arr = std::move(right.arr);
        size = right.size;
        right.arr = nullptr;
        right.size = 0;
    }
    return *this;      //возврат объекта
}
```

Перегрузка оператора +

```
Vector Vector::operator +(const Vector &right) const
{
    Vector temp(size+ right.size);
    for(int i = 0; i < size; i++)
        temp[i] = arr[i];
    for(int i = 0; i < right.size; i++)
        temp[i+size] = right[i];
    return temp;
}
```

Файл исходного кода vector.cpp

Перегрузка оператора +

```
Vector& Vector::operator +(double a)
{
    for(int i = 0; i < size; i++)
        arr[i] += a;
    return *this;
}
```

Файл исходного кода
vector.cpp

Перегрузка оператора []

```
double & Vector::operator [](int index)
{
    if(index>=0 && index< size)
        return arr[index];
    if(index<0)
        return arr[0];
    return arr[size-1];
}
```

Файл исходного кода vector.cpp

```
double Vector::operator [](int index)const
{
    Vector temp = (Vector)(*this);
    return temp[index];
}
```

Вызов ранее определенного оператора (без const)

Перегрузка оператора ++

```
Vector& Vector::operator ++() { //префиксная форма
        for(int i = 0; i < size; i++)
                arr[i]++;
        return *this;
}
```

```
Vector Vector::operator ++(int) { //постфиксная форма
        Vector temp(*this);
        ++*this; //использование префиксной формы ++
        return temp;
}
```

Файл исходного кода vector.cpp

Перегрузка оператора --

```
Vector& Vector::operator --() { //префиксная форма
        for(int i = 0; i < size; i++)
                arr[i]--;
        return *this;
}
```

```
Vector Vector::operator --(int) { //постфиксная форма
        Vector temp(*this);
        --*this; //использование префиксной формы --
        return temp;
}
```

Файл исходного кода vector.cpp

Перегрузка внешнего оператора +

```
double operator +(double a, const Vector& right)
{
    double sum = a;
    for(int i = 0;i < right.GetSize(); i++)
        sum += right[i];
    return sum;
}
```

Файл исходного кода vector.cpp

Перегрузка операторов ввода/вывода

```
std::istream & operator>>(std::istream& in, Vector& ref)
{
        for(int i = 0;i < ref.GetSize(); i++)
                in >> ref[i];
        return in;
}
```

```
std::ostream& operator<<(std::ostream& out, const Vector& ref){
        ref.Print();
        return out;
}
```

Файл исходного кода vector.cpp

Использование операторов

```
const int SIZE = 10;

double v[SIZE];

for(int i = 0; i < SIZE; i++)

    v[i] = (double)
    (rand())/(rand()+1);

Vector v1(v, SIZE), v2(SIZE);

Vector v3;

std::cin >> v2;

std::cout << v1 << std::endl
          << v2 << std::endl;

v3 = v2 + v1;

std::cout << v3 <<std::endl;
```

```
double sum, pi = 3.14;

sum = pi + v3;

std::cout << sum  << std::endl;

std::cout << ++v3 << std::endl;

std::cout << v1-- << std::endl;

std::cout << v1   << std::endl;

v2 = v1;

std::cout << v2 << std::endl;

v2 = v3;

std::cout << v2 << std::endl;

std::cout << v3 << std::endl;
```

Файл исходного кода test.cpp

Конец