Семинар 7
Перегрузка операций, классы ресурсоёмких объектов

1. Перегрузка операций

Примеры перегрузки операций:

- Операция поразрядного сдвига << является перегруженной в классе выходных потоков (для cout).
- Операция сложения + является перегруженной для класса string, представляя собой операцию конкатенации строк (в конец первой строки дописывается вторая строка).

1. Перегрузка операций

Язык Си++ позволяет распространить действие стандартной операции на новые типы данных. Для этого существует *механизм перегрузки стандартных операций*, схожий для программиста с механизмом определения функций. Сама операция при применении этого механизма называется *перегруженной*.

1. Перегрузка операций

Формат определения операции-функции (operator function):

тип_возвр_знач operator знак операции
(спецификация_параметров)
{операторы тела операции-функции}

При необходимости может применяться прототип операции-функции.

1. Перегрузка операций

Например, для распространения действия бинарной операции * на объекты класса T:

```
T operator * (T x, T y)
```

После введения перегруженной операции выражение A*B интерпретируется как вызов функции operator *(A, B)

1. Перегрузка операций

Например, для распространения действия бинарной операции * на объекты класса T может быть введена внешняя функция:

```
T operator * (T x, T y)
```

После введения перегруженной операции выражение A*B интерпретируется как вызов функции operator *(A, B)

1. Перегрузка операций

Если операция-функция определяется как метод класса T, заголовок у неё будет таким:

T operator * (T y)

После введения перегруженной операции выражение A*B интерпретируется как обращение к методу A.operator *(B)

1. Перегрузка операций

Пример. Перегрузка операций для класса «комплексное число».

```
class myComplex{
        double re, im;
        public: myComplex(double r = 0.0, double i = 0.0) :re(r), im(i) {}
        double real(){return re;}
        double imag(){return im;}
        myComplex operator - (){return myComplex(-re, -im);}
        friend ostream & operator << (ostream &, const myComplex &);
        friend istream & operator >> (istream &, myComplex &);
};
ostream & operator << (ostream & output, const myComplex & c){
output << "real = " << c.re << ",\t image = " << c.im << endl;
return output;
}
istream & operator >> (istream & input, myComplex & c){
cout << "real = "; input >> c.re; cout << "image = "; input >> c.im; return input;
}
myComplex operator + (myComplex A, myComplex B){
return myComplex(A.real()+ B.real(), A.imag() + B.imag());
}
```

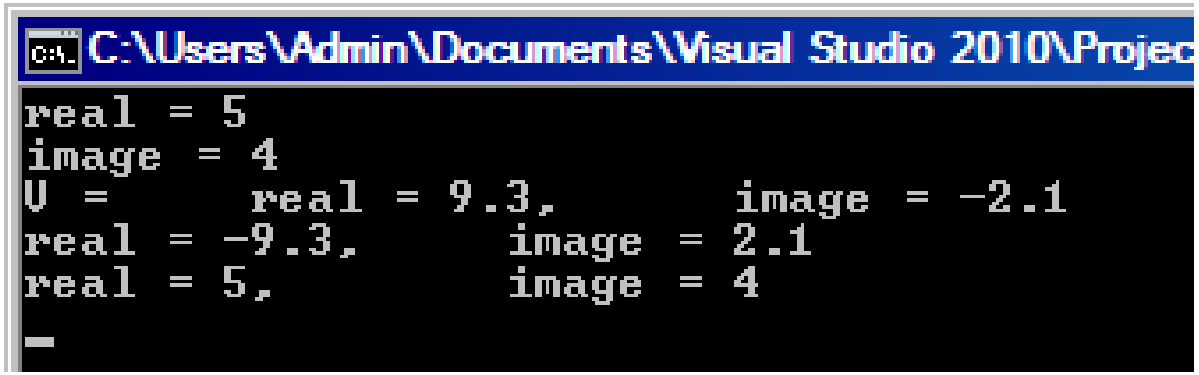
Дружественные операции-функции

Внешняя операция-функция

1. Перегрузка операций

Пример. Перегрузка операций для класса «комплексное число».

```
int _tmain(int argc, _TCHAR* argv[])
{
	myComplex V(4.3, -6.1);
	myComplex W;
	cin >> W; // operator>>(cin, W);
	V = V + W; // operator+(V, W)
	cout << "V =\t" << V; // operator<<(cout, V)
	V = -V; // V.operator-();
	cout << V << W; // Так как оператор-функция operator<< возвращает ссылку на
	// объект класса ostream, допустимо использование «цепочек» при выводе
	return 0;
}
```



1. Перегрузка операций

Важные особенности перегрузки операций:

- Си++ не позволяет вводить операции с совершенно новыми обозначениями
- Существуют операции, не допускающие перегрузки:

.(выбор метода или поля данных объекта)

.*(обращение к методу или полю данных через указатель)

?: :: sizeof # ##

1. Перегрузка операций

Важные особенности перегрузки операций:

- При перегрузке операций нет возможности изменять приоритеты операций
- Нельзя изменять синтаксис выражений, т.е. нет возможности ввести бинарную операцию ++ или унарную -=
- Операции-функции operator =, operator [], operator -> должны быть нестатическими методами того класса, для которого они определены (а не внешними функциями)

1. Перегрузка операций

Важные особенности перегрузки операций:

• Если в выражение с бинарной операцией объект класса должен входить только как правый операнд, то операция-функция не может быть методом класса (пример – операции-функции operator<< и operator>>, показанные выше)

1. Перегрузка операций

Важные особенности перегрузки операций:

• Перегрузка префиксных и постфиксных унарных операторов имеет отличие:

Для префиксной формы (например, ++x)	**++x означает: x.operator ++() или operator ++(x)**
Для постфиксной формы (например, x++)	++x означает: x.operator ++(int) или operator ++(x, int)

1. Перегрузка операций

Пример перегрузки префиксной операции:

```
myType & operator ++(){
    x+=2; // Здесь может быть что угодно
    return *this;
}
```

1. Перегрузка операций

Важные особенности перегрузки операций:

- Если операторы * и = являются перегруженными, то это не значит, что к объектам класса можно применять оператор *=
- Нельзя изменить смысл выражения, если в него не входит объект класса, введённого пользователем. Например, нельзя определить операцию-функцию для операндов-указателей

1. Перегрузка операций

Распространение действия операций на новые классы служит для встраивания (агрегации) класса в систему типов, уже существующих в языке. Для бинарной операции во многих случаях достаточно определить только три варианта сочетаний операндов:

- стандартный_тип, класс
- класс, стандартный_тип
- класс, класс

1. Перегрузка операций

В классе myComplex была перегружена операция +:

```
myComplex operator + (myComplex A, myComplex B){
return myComplex(A.real()+ B.real(), A.imag() + B.imag());
}
```

Но выражения в следующих операторах будут допустимы из-за существования конструктора myComplex(double r = 0.0, double i = 0.0) :

```
myComplex C(1.0, 2.0);
myComplex E;
E = 4.0 + C; // operator +(myComplex(4.0), C)
E = E + 2.0;
E = C + E;
E = C + 20; // operator +(C, myComplex(double(20)))
C = C + 'x'; // то же, но здесь тип char приводится к double
```

1. Перегрузка операций

Операторы преобразования:

```
operator double() const {
        return re;
}

int main() {
    myComplex mc1(5, 6);
    double d = (double)mc1;
    cout << d; // 5
}
```

- const после имени метода означает, что метод не будет изменять значения полей класса (метод не меняет состояние объекта)
- Не указывается возвращаемое значение

2. Классы ресурсоёмких объектов

Поддержка присваивания обеспечивается в каждом классе за счёт присутствия в его определении соответствующей операции-функции operator =. Она либо явно определяется в классе программистом, либо неявно добавляется компилятором. То же самое справедливо по отношению к конструктору копирования.

2. Классы ресурсоёмких объектов

Если объект использует динамическую память, то чаще всего необходимо использование «глубокого копирования» при выполнении присваивания и вызове конструктора копирования - т.е. копирования не только полей данных объекта, но и динамической памяти.

2. Классы ресурсоёмких объектов

Пример. Глубокое копирование для класса «точка в многомерном пространстве».

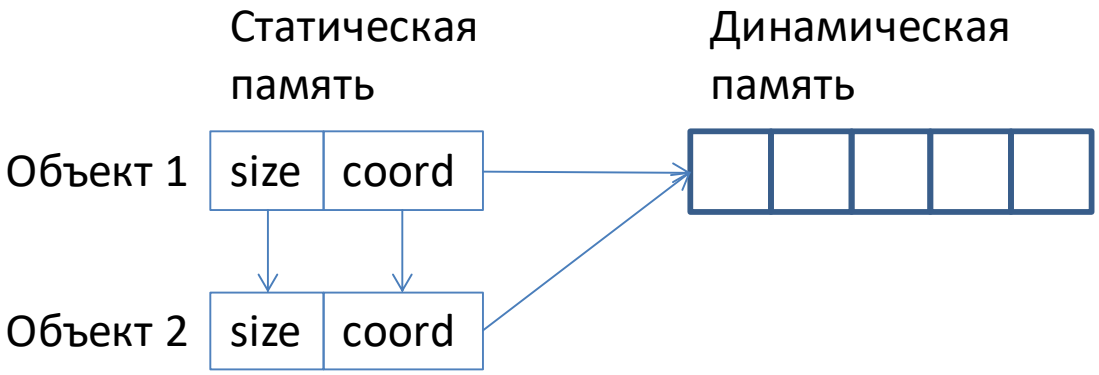
```
class point{
        int size;
        double * coord;
        public:
        point(int n=1, double z = 0.0){ // Конструктор общего вида
                size = n;
                coord = new double[size];
                for(int i=0; i<size; i++)
                coord[i] = z;
        }
        point(const point & p) :size(p.size){ // Конструктор копирования
                coord=new double[size];
                for(int i=0; i<size; i++)
                                coord[i] = p.coord[i];
        }
        …
```

2. Классы ресурсоёмких объектов

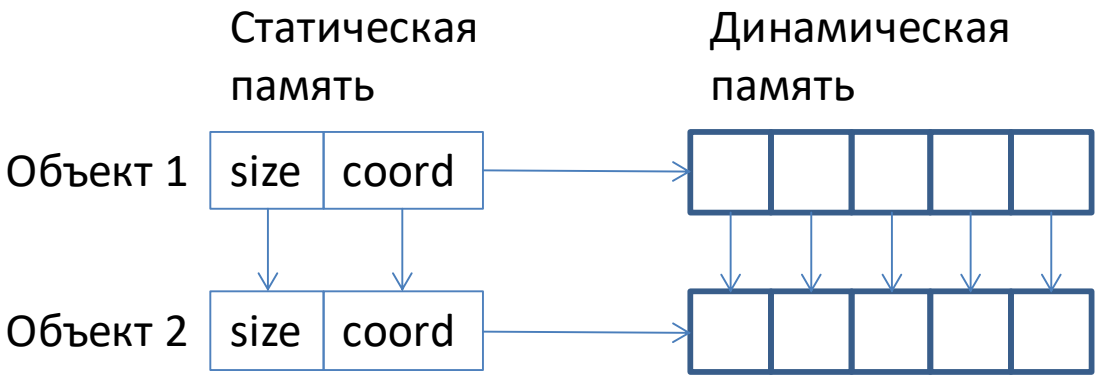

```
…
        point & operator=(const point &p){ // Перегрузка операции присваивания
                if(this!=&p){
                        delete [] coord;
                        coord = new double [size = p.size];
                        for(int i=0; i<size; i++)
                                coord[i] = p.coord[i];
                }
                return * this;
        }
        friend ostream & operator<<(ostream & out, point p);
        ~point(){delete [] coord;}
};
ostream &operator<<(ostream & out, point p){
        out<<"size="<<p.size;
        for(int i=0; i<p.size; i++)
                out << "\t["<<i<<"]="<<p.coord[i];
        out<<endl;
        return out;
}
```

2. Классы ресурсоёмких объектов

Было



Стало



2. Классы ресурсоёмких объектов

Задания.

1. Переопределить операцию [] для класса point таким образом, чтобы применение данной операции к экземпляру данного класса возвращало i-й элемент динамического массива coord.
2. Создать класс учеников, имеющий поля char * name (имя класса), int countScholar (количество учеников), int countStudy (количество предметов), int * ages (возраст учеников), double ** marks (двумерный динамический массив оценок). Добавить в класс переопределение оператора присваивания и конструктора копирования.