Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

24 марта 2015

Оператор []

```
int main()
{
MyArray a(10);
a[0] = 5;
a.operator[](1) = 65;
}
```

Перегрузка оператора []

```
class MyArray
{
int m_size;
int *arr;
};
double &MyArray::operator[](int n)
{
        if(n>=0 && n < m_size)
return arr[n];
else
{ //генерация исключений
  //или
  //return arr[0];
  //или
  //изменить размер, если оператор находится слева
      от знака равенства.
}
}
```

Перегрузка оператора []

- перегруженный оператор индексирования должен возвращать не значение элемента, а его адрес, для того чтобы можно было использовать выражение слева от знака равенства;
- для перегруженного оператора индексирования, можно предусмотреть проверку значений;
- для массивов из стандартных типов индекс может быть только целый, а для перегруженного оператора — любым.
- обычно в класс вводят еще один перегруженный константный оператор индексирования.

константный оператор индексирования

```
class A {
public:
double &operator[](int i);
const double &operator[](int i) const ;
};

A a1(10);
const A a2(10);

int tmp = a1[0]; //???
tmp = a2[0]; //???
a1[0] = 5; //???
a2[0] = 5; //???
```

Перегрузка оператора ++

Как отличить префиксный от постфиксного?

```
//A.h
class A {
int m_a;
public:
A(int = 0);
/* FIXME */ operator++(); //префиксной
/*FIXME */ operator++(int unused); //постфиксный
};
//A.cpp
... A::operator++()
{ ++m_a; return ... }

... A::operator++(int) //FIXME
//main.cpp
int main() {
 A a1(1), a2;
++a1; //a1.operator++();
a2 = a1++; //a1.operator++(0); }
```

Перегрузка оператора ++

```
//A.cpp
A& A::operator++()
{
++m_a;
return *this;
}

A A::operator++(int)
{
A a(*this);
m_a++;
return a;
}
```

Перегрузка бинарного оператора +

```
//A.h
class A {
int m_a;
public:
A(int a=0) { m_a = a; }
A operator+(const A&r);
};
//A.cpp
A A::operator+(const A& r)
{
return A(m_a + r.m_a);
}
```

Перегрузка бинарного оператора +. Глобальная функция

```
//A.h
class A {
int m_a;
public:
A(int a=0) { m_a = a; }
};
A operator+(const A& l, const A& r); //глобальная
    функция вне класса

//A.cpp
A operator+(const A& l, const A& r)
{
return A(l.m_a + r.m_a); //???
}
```

Перегрузка бинарного оператора +. Глобальная функция

```
//A.h
class A {
int m_a;
public:
A(int a=0) { m_a = a; }
friend A operator+(const A& l, const A& r); //
    функция - друг
};
A operator+(const A& l, const A& r); //глобальная
    функция вне класса

//A.cpp
A operator+(const A& l, const A& r)
{
return A(l.m_a + r.m_a); //???
}
```

В чем отличие от оператора += ?

Как будет выглядеть operator+ для MyString?

Перегрузка операторов >> и <<

```
//C.h
class MyClass
{
int x; int y;
friend ostream& operator << (ostream& out, MyClass&
    C);
friend istream& operator >> (istream& in, MyClass&
   C);
};
//C.cpp
ostream& operator << (ostream& out, MyClass& C)
{ return out << " x = " << C.x << " y " << C.y; }

istream& operator >> (istream& in, MyClass& C)
{ cout << "Enter x: "; in >> C.x;
  cout << "Enter y: "; in >> C.y;
  return in;
}
```

Внедряемые объекты

Отножение между классами «содержит» (has a).

```
//A.h
class A { int m_a;
public:
A(int a=0);
};
class B { int m_b;
public:
B(int a=0);
};
class C : public B
{ int m_c;
  A m_A;
public:
C(int a, int b, int c);
};

C c(1, 2, 3);
```

Отножение между классами «содержит» (has a).

- компилятор выделяет память для объекта C (с учетом внедренного объекта A);
- вызывает конструктор базового класса B;
- вызывает конструктор внедряемого объекта A;
- вызывает конструктор C.

Разрушение объекта происходит в обратном порядке: деструкторы C -> A -> B.

Внедряемые объекты

Отножение между классами «содержит» (has a).

```
C::C(int a, int b, int c)
{
//как проинициализировать:
m_c
m_b
m_A
???
}
```

Рекомендация: в конструктора всегда предпочитайте инициализацию вместо присваивания.
Переменные класса инициализируются в порядке объявления в классе, поэтому порядок их следования в списке инициализаторов значения не имеет.

Ассоциативный массив

- В ассоциативных массивах хранятся пары — ключ/значение.
- Операции поиска, сортировки, добавления, удаления — осуществляются по ключу.
- Ключ в большинстве ассоциативных массивов должен быть уникальный.

Записная книжка

```
class Pair {
MyString name; //КЛЮЧ – ФИО
int phone;
Pair(): name("NoName"), phone(0) {}
Pair(const char *key, int num) : name(key), phone(
    num) {}

bool operator== (const char *key)
{
return name == key; //???
}

friend class Book;
};
```

Записная книжка

```
class Book {
Pair **ar;
int size;
int capacity;
public:
Book() { size = capacity = 0; ar = nullptr; }
int &operator[](const char *);
};
```