Лекция 5. Перегрузка операторов.

Артур Садеков

НИЖЕГОРОДСКИЙ ИНСТИТУТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
НИИТ

Операторные функции

- **Можно объявить функции для операторов:**
  +, -, *, /, &, |, =, >, <, +=, -=, ≈, !=, &&, ||, ++, --, [], (),
  new, delete, и т.д.
- **Нельзя определить:**
  :: . .* ?: sizeof typedef
- **Невозможно определить новую лексему оператора**
- **Имя операторной функции - operator@, например operator<< .**

Бинарные операторы

Можно определить в виде:

- **нестатической функции-члена с одним аргументом,**
- **функции-не-члена с двумя аргументами.**

Выражение aa@bb интерпретируется:

- **aa.operator@(bb)**
- **operator@(aa, bb)**

Бинарные операторы. Примеры.

```
class Coord {
 int x, y, z;
public:
 Coord  operator+  (Coord);
 Coord& operator=  (Coord);
 Coord& operator*= (int);
};
Coord operator*(Coord, int);
```

```
Coord operator*(Coord c,
                int m) {
  Coord temp = c;
  return temp*=m;
}
```

```
Coord Coord::operator+(Coord t) {
  Coord temp;
  temp.x = x + t.x;
  temp.y = y + t.y;
  temp.z = z + t.z;
  return temp;
}
```

```
Coord& Coord::operator=(Coord t) {
  x = t.x;
  y = t.y;
  z = t.z;
  return *this;
}
```

Операторы-члены и не-члены.

```
class Coord {
public:
 Coord&  operator+=(const Coord&);
 //...
};
Coord operator+(const Coord&, const Coord&);
```

```
Coord operator+(const Coord& c1, const Coord& c2)
{
  Coord temp = c1;
  return temp+=c2;
}
```

```
Coord&
operator+=(const
Coord& c)
{
  x += c.x;
  y += y.x;
  z += z.x;
  return *this;
}
```

Операторы-члены и не-члены.

```
class Coord {
public:
 bool operator==(const Coord& c) const;
 //...
};

bool operator!=(const Coord& c1, const Coord& c2);
```

```
bool operator!=(const Coord& c1, const Coord& c2)
{
  return !(c1==c2);
}
```

Унарные операторы

Можно определить в виде:

- **нестатической функции-члена без аргументов,**
- **функции-не-члена с одним аргументом.**

Выражение @aa интерпретируется как:

- **aa.operator@ ()**
- **operator@ (aa)**

Унарные постфиксные операторы

Для любого постфиксного оператора выражение aa@ интерпретируется как:

- **aa.operator@ (int)**
- **operator@ (aa, int)**

- **Аргумент int используется для указания на постфиксную форму**
- **Аргумент является фиктивным**

Унарные операторы. Примеры.

```
class Coord {
 int x, y, z;
public:
 Coord& operator-  ();
 Coord& operator++ ();
}
```

```
Coord& Coord::operator- () {
  x = -x;
  y = -y;
  z = -z;
  return *this;
}
```

```
Coord& Coord::operator++() {
  ++x;
  ++y;
  ++z;
  return *this;
}
```

```
Coord operator++(Coord& c, int i)
{
  Coord temp = c;
  ++c;
  return temp;
}
```

Предопределенный смысл операторов

- Оператор может быть объявлен только с синтаксисом, существующем для него в грамматике
- operator=, operator[ ], operator-> должны быть нестатическими функциями-членами
- Некоторые операторы определены так, что они равны комбинации других операторов, например ++a означает a+=1 или a=a+1. Компилятор сам об этом не заботится.
- Операторы = и & имеют предопределенный смысл для объектов класса, если они закрытые, этот смысл может стать недоступным.

Операторы и типы, определяемые пользователем

- Операторная функция может быть либо членом, либо иметь аргумент типа, определяемого пользователем.
- Операторная функция, у которой первый операнд принадлежит к встроенному типу, не может являться членом.

```
aa+2

aa.operator+(2)
```

```
2+aa

2.operator+(aa)
```

Конец