Junior C++ developer

Лекция 7

- Перегрузка операторов

- В философии C++ пользовательские типы данных должны иметь такие же возможности как и встроенные.
- То есть можно использовать *операторы* (такие как +, -, [ ], и т. д.) для собственных типов данных!
- Для этого необходимо рассказать компилятору что мы хотим получить от оператора.
  - Данный механизм называется ***перегрузкой операторов***.

Перегрузка операторов

Пользовательские типы данных могут использовать операторы, точно так же как и базовые типы

```
int i = j + 3;
Order newOrder = oldOrder +
newOrder;
```

STL содержит множество полезных перегрузок операторов

- Оператор **+** (например для строк)
- Оператор **>>** и **<<** для ввода и вывода данных
- Оператор **++**

Возможности и ограничения

В C++ мы можем перегрузить практически ***любой*** известный оператор.

Определить новый оператор над встроенными типами или переписать существующий ***нельзя***.

Перегружайте операторы тогда и только тогда, когда это имеет ***смысл***.

Смысл перегрузки должен быть ***очевиден***.

Ограничения

- Нельзя определить новый оператор, например, `operator**`.
- Следующие операторы перегружать нельзя:
  1. `?:` (тернарный оператор);
  2. `::` (доступ к вложенным именам);
  3. `.` (доступ к полям);
  4. `.*` (доступ к полям по указателю);
  5. `sizeof`, `typeid` и операторы каста.

Ограничения

- Следующие операторы можно перегрузить только в качестве методов:
  a. `=` (присваивание);
  b. `->` (доступ к полям по указателю);
  c. `()` (вызов функции);
  d. `[]` (доступ по индексу);
  e. `->*` (доступ к указателю-на-поле по указателю);
  f. операторы конверсии и управления памятью.

Ограничения

- Количество операндов, порядок выполнения и ассоциативность операторов определяется стандартной версией.
- Как минимум один операнд должен быть пользовательского типа. Typedef не считается.

Способы перегрузки

- Через дружественные функции
- Через обычные функции
- Через методы класса

MyObject < Something

bool MyClass::operator<(OtherType something)

operator< метод класса MyClass.

Принимает один аргумент. Результат сравнения ***this*** и ***something.***

this < something

Something < MyObject

bool operator<(OtherType something,

MyClass mc)

operator< свободная функция.

Принимает два аргумента. Результат сравнения ***something*** и ***mc***

something < ***mc***

```
class Vector3D
{

public:
    Vector3D():
    x(0), y(0), z(0) {}

    Vector3D(int x, int y, int z):
    x(x), y(y), z(z) {}


private:
    int x, y, z;
};
```

Класс для которого будет реализовывать перегрузку операторов.

Operator=

Ключевое слово

Символ оператора

Защита от случайных изменений

```
Vector3D& operator= (Vector3D const &rhs)
{
    cout << "operator = \n";
    if (this != &rhs)
    {
        this->x = rhs.x, this->y = rhs.y, this->z = rhs.z;
    }
    return *this;
}
```

Проверка что не пытаемся присвоить самого себя

Возвращаем указатель на себя

Operator+

Является методом класса. Возвращает ***this*** + ***vector***

```
class Vector3D
{
public:
    ...
    Vector3D operator+(Vector3D const &vector)
{
    return Vector3D(this->x + vector.x, this->y + vector.y, this->z + vector.z);
}

private:
    int x, y, z;
};
```

Operator+

```
class Vector3D
{
public:
    ...
    int getZ() const
    { return z; }
    int getY() const
    { return y; }
    int getX() const
    { return x; }
private:
    int x, y, z;};
```

Является свободной функцией. Принимает два аргумента. Результат ***vector1*** + ***vector2***

```
Vector3D operator+(Vector3D const &vector1, Vector3D const
&vector2)
{
   return Vector3D(vector1.getX() + vector2.getX(),
                   vector1.getY() + vector2.getY(),
                   vector1.getZ() + vector2.getZ());
}
```

Operator<<

Объявляем дружественную функцию

```
class Vector3D
{
public:
    ...
    friend ostream& operator<< (ostream& os, Vector3D const &vector);

private:
    int x, y, z;
};

ostream& operator<< (ostream& os, Vector3D const &vector)
{
    os << vector.x << "; " <<  vector.y << "; " <<  vector.z << endl;
    return os;
}
```

Реализуем как свободную функцию

Правило

Используйте перегрузку операторов через обычные функции, вместо дружественных, если для этого не требуется добавление дополнительных функций в класс.

Не все может быть перегружено через дружественные функции

Через методы класса перегружаются операторы:

- ***operator=*** присваивания
- ***operator[]*** индекса
- ***operator()*** вызова функции
- ***operator->*** выбора члена

Это требования языка.

Не все может быть перегружено через методы класса

Перегрузка операторов через методы класса не используется, если левый операнд не является классом (например, int), или это класс, который мы не можем изменить (например, std::ostream).

Какой способ использовать?

- Для операторов присваивания (=), индекса ([]), вызова функции (()) или выбора члена (->) используйте перегрузку через методы класса.
- Для унарных операторов используйте перегрузку через методы класса.
- Для перегрузки бинарных операторов, которые изменяют левый операнд (например, operator+=) используйте перегрузку через методы класса, если это возможно.
- Для перегрузки бинарных операторов, которые не изменяют левый операнд (например, operator+) используйте перегрузку через обычные/дружественные функции.