

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Перегрузка операторов в языке C++

- ⊙ Что означают и чем отличаются:
 - ⊙ `int const* a;`
 - ⊙ `const int* b;`
 - ⊙ `static int* c = &a;`
 - ⊙ `const int& plus (int a, int b);`
 - ⊙ `int& plus (int a, int b) const;`
- ⊙ Какого типа указатель `this` передается в методы класса T?
 - ⊙ `T* const this`

ПЕРЕГРУЗКА ОПЕРАТОРОВ В С++



ПЕРЕГРУЗКА ОПЕРАТОРОВ

- ◎ С++ позволяет переопределить действие большинства операторов так, чтобы при использовании с объектами конкретного класса они выполняли заданные функции.
- ◎ Перегрузка операторов - это возможность одновременного существования нескольких различных реализаций одного оператора, различающихся типами параметров, к которым они применяются.
- ◎ Можно перегружать любые операторы, существующие в С++, за исключением:

`. * ? : :: # ## sizeof`

ФУНКЦИИ-ОПЕРАЦИИ

- ⊙ Перегрузка операций осуществляется с помощью функций специального вида (*функций-операций*) и подчиняется следующим правилам:
 - ⊙ сохраняются количество аргументов, приоритеты операций и правила ассоциации (справа налево или слева направо) по сравнению с использованием в стандартных типах данных
 - ⊙ нельзя переопределить операцию по отношению к стандартным типам данных
 - ⊙ функция-операция не может иметь аргументов по умолчанию
 - ⊙ функции-операции наследуются (за исключением =)

ФУНКЦИИ-ОПЕРАЦИИ

- ⦿ Перегрузка операций осуществляется с помощью функций специального вида (*функций-операций*) и подчиняется следующим правилам:
 - ⦿ сохраняются количество аргументов, приоритеты операций и правила ассоциации (справа налево или слева направо) по сравнению с использованием в стандартных типах данных
 - ⦿ нельзя переопределить операцию по отношению к стандартным типам данных
 - ⦿ функция-операция не может иметь аргументов по умолчанию
 - ⦿ функции-операции наследуются (за исключением =)
- ⦿ *Функцию-операцию можно определить тремя способами:*
 - ⦿ методом класса,
 - ⦿ дружественной функцией класса,
 - ⦿ обычной функцией.

тип `operator` операция (список параметров) { тело функции }

ПЕРЕГРУЗКА УНАРНЫХ ОПЕРАЦИЙ

ПЕРЕГРУЗКА УНАРНЫХ ОПЕРАЦИЙ

- ⊙ Унарная функция-операция, определяемая *внутри класса*, должна быть представлена с помощью нестатического метода без параметров, при этом операндом является вызвавший ее объект, например:

```
class monster
{
    ...
    monster & operator ++()
    {++health;
     return *this;}
    ...
}

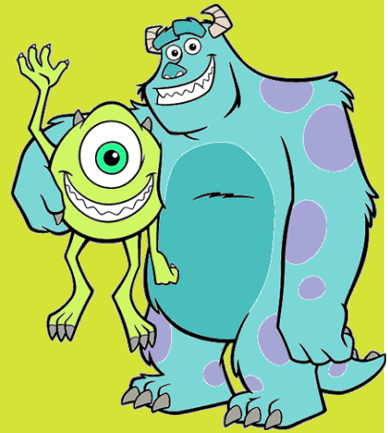
monster Vasia;
cout << (++Vasia).get_health();
```


ПЕРЕГРУЗКА УНАРНЫХ ОПЕРАЦИЙ

- ⊙ Если функция определяется *вне класса*, она должна иметь один параметр типа класса :

```
class monster
{...
    friend monster& operator ++( monster &M);
};

monster& operator ++(monster &M) {++M.health; return M;}
```



ИНФИКСНЫЙ / ПОСТФИКСНЫЙ ИНКРЕМЕНТ

- ⊙ Операции постфиксного инкремента и декремента должны иметь первый параметр типа `int`:

```
class monster
{...
    monster operator ++(int)
    {monster M(*this);
     health++;
     return M;}
};

monster Vasia;
cout << (Vasia++).get_health();
```

ПЕРЕГРУЗКА БИНАРНЫХ ОПЕРАЦИЙ

ПЕРЕГРУЗКА БИНАРНЫХ ОПЕРАЦИЙ

- ◎ Бинарная функция-операция, определяемая *внутри класса*, должна быть представлена с помощью нестатического метода с параметрами, при этом вызвавший ее объект считается первым операндом:

```
class monster
{...
    bool operator >(const monster &M)
    {
        if( health > M.get_health()) return true;
        return false;
    }
};
```

ПЕРЕГРУЗКА БИНАРНЫХ ОПЕРАЦИЙ

- ⊙ Если функция определяется *вне класса*, она должна иметь два параметра типа класса:

```
bool operator >(const monster &M1, const monster &M2)
{
    if( M1.get_health() > M2.get_health())
        return true;
    return false;
}
```

ПЕРЕГРУЗКА ОПЕРАЦИИ ПРИСВАИВАНИЯ

ОПЕРАЦИЯ ПРИСВАИВАНИЯ

- ⊙ Операция присваивания определена в любом классе по умолчанию как поэлементное копирование.
- ⊙ Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого.
- ⊙ Если класс содержит поля ссылок на динамически выделяемую память, необходимо определить собственную операцию присваивания.

ОПЕРАЦИЯ ПРИСВАИВАНИЯ



Чтобы сохранить семантику операции, операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент - ссылку на присваиваемый объект:

```
monster& operator = (const monster &M)
{
    if (&M == this) return *this; // Проверка на самоприсваивание
    if (name) delete [] name;
    if (M.name)
    {
        name = new char [strlen(M.name) + 1];
        strcpy(name, M.name);
    }
    else name = 0;
    health = M.health; ammo = M.ammo; skin = M.skin;
    return *this;
}
```


ОСОБЕННОСТИ ОПЕРАЦИИ ПРИСВАИВАНИЯ

- ⊙ Возвращаемое значение – ссылка, как и передаваемое значение для возможности формирования цепочек присваивания

```
int a, b, c, d, e;  
a = b = c = d = e = 42;
```



```
a = (b = (c = (d = (e = 42)))));
```

- ⊙ Возвращаемое значение не `const` – просто запомните...

```
Monster a, b, c;  
...  
(a = b) = c; // !!??
```



ПЕРЕГРУЗКА СОСТАВНЫХ И БИНАРНЫХ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

СОСТАВНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

- ⊙ Операции: += -= *=
- ⊙ Основная особенность – это деструктивные операции, которые заменяют значение существующего объекта.

```
MyClass & MyClass::operator+=(const MyClass &rhs) {  
    ... // Составное присваивание.  
  
    return *this;  
}
```

```
MyClass mc;  
...  
(mc += 5) += 3;
```

БИНАРНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

- ◎ Операции: + - *
- ◎ Основной принцип: лучше всего реализовать бинарные арифметические операции посредством составных арифметических операций

```
// Добавить значение текущего объекта к другому объекту
// вернуть новую сущность с результатом.
const MyClass MyClass::operator+(const MyClass &other) const {
    MyClass result = *this;    // Делаем копию самого себя
    result += other;          // Прибавляем значение другого объекта
    return result;           // Возвращаем результат
}
```

СИГНАТУРА БИНАРНОЙ ОПЕРАЦИИ

- ⊙ Мы возвращаем
 - ⊙ `const MyClass` – зачем?

```
MyClass a, b, c;  
...  
(a + b) = c; // Что это?!
```

- ⊙ Оно ничего не сделает, но будет компилироваться. Но зачем?

ПЕРЕГРУЗКА ИНЫХ ОПЕРАТОРОВ

ПЕРЕГРУЗКА ПРИВЕДЕНИЯ ТИПА

- ⊙ Можно определить функции-операции, которые будут осуществлять преобразование класса к другому типу.

```
monster::operator int(){return health;}  
...  
monster Vasia;  
cout << int(Vasia);
```

- ⊙ Тип возвращаемого значения и параметры указывать не требуется. Можно определять виртуальные функции преобразования типа.

ПЕРЕГРУЗКА NEW И DELETE

- ⊙ Переменная объектного типа в динамической памяти создаётся в два этапа:
 - ⊙ Выделяется память с помощью оператора `new`.
 - ⊙ Вызывается конструктор класса.
- ⊙ Удаляется такая переменная тоже в два этапа:
 - ⊙ Вызывается деструктор класса.
 - ⊙ Освобождается память с помощью оператора `delete`.

ПЕРЕГРУЗКА NEW И DELETE

- ⊙ Операторы new и delete можно перегрузить. Для этого есть несколько причин:
 - ⊙ Можно увеличить производительность за счёт кеширования: при удалении объекта не освобождать память, а сохранять указатели на свободные блоки, используя их для вновь конструируемых объектов.
 - ⊙ Можно выделять память сразу под несколько объектов.
 - ⊙ Можно реализовать собственный "сборщик мусора" (garbage collector).
 - ⊙ Можно вести лог выделения/освобождения памяти.

СИГНАТУРА NEW И DELETE

- ⊙ Оператор `new` принимает размер памяти, которую необходимо выделить, и возвращает указатель на выделенную память.
- ⊙ Оператор `delete` принимает указатель на память, которую нужно освободить.

```
void *operator new (size_t size);  
void operator delete (void *p);
```

РЕАЛИЗАЦИЯ NEW И DELETE

```
class A {  
  
public:  
    void *operator new(size_t size);  
    void operator delete(void *p);  
};  
  
void *A::operator new(size_t size) {  
    printf("Allocated %d bytes\n", size);  
    return malloc(size);  
}  
  
void A::operator delete(void *p) {  
    free(p);  
}
```

- ⊙ Перегрузка операций производится посредством методов, начинающихся с ключевого слова «operator»
- ⊙ Могут быть определены как внутри класса, так и вне его.
- ⊙ Могут быть унарные и бинарные
- ⊙ Можно переопределять такие операции как
 - ⊙ ->
 - ⊙ []
 - ⊙ ()
 - ⊙ new и delete
 - ⊙ Арифметические операции и многое другое