Объектно-ориентированное программирование

Наследование в языке 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.
- Если такая операция не определена, то при вызове операции, например Type b = a + 3 будет неявно вызван конструктор a (int), если такой определен в классе.

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);
}
```

Вместо функций malloc и free можно использовать глобальные операторы ::new и ::delete