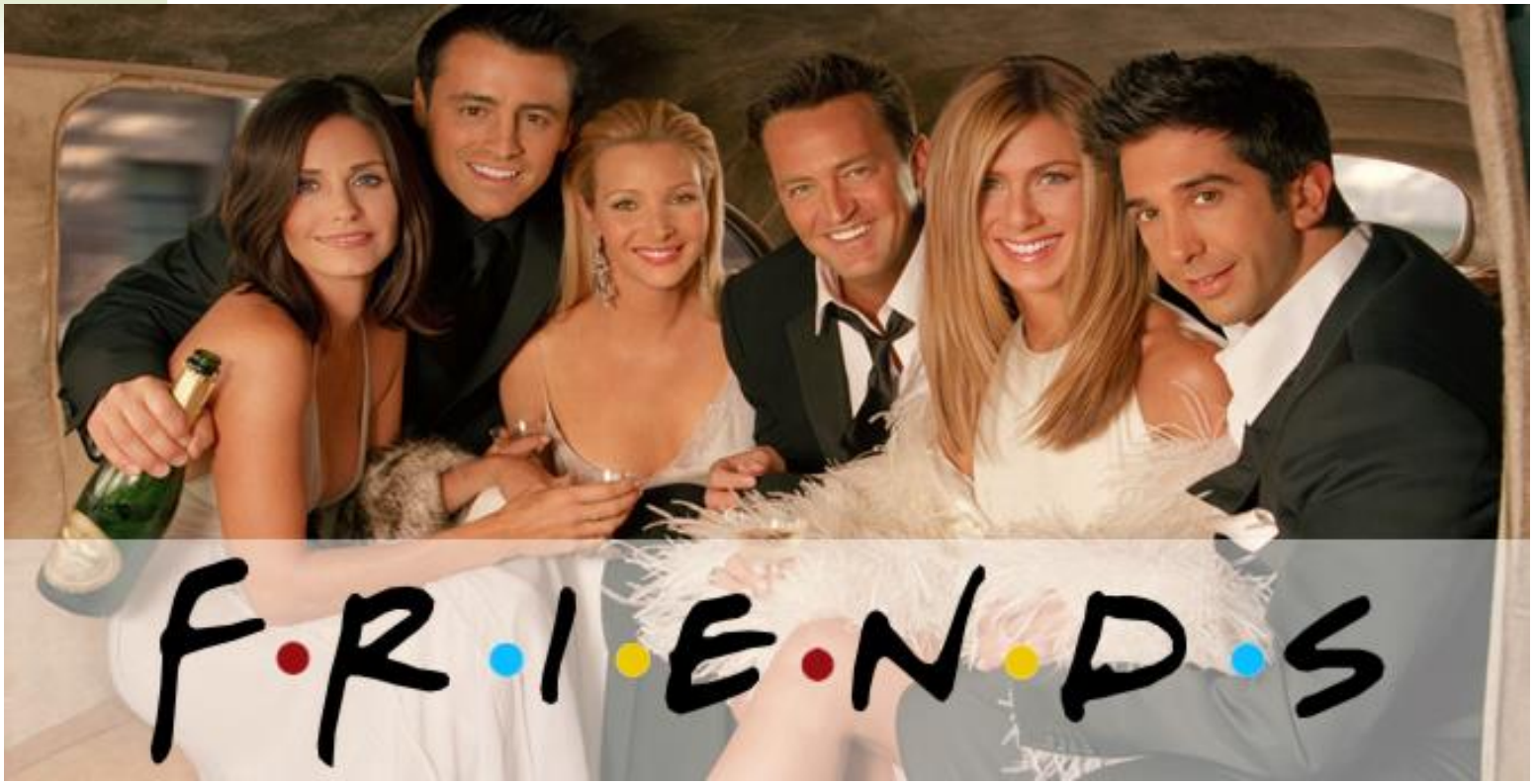
Объектно-ориентированное программирование

Дружественные функции и классы. Деструкторы. Перегрузка.

Дружественные функции и классы


F·R·I·E·N·D·S

Дружественная функция

- Дружественная функция объявляется *внутри класса*, к элементам которого ей нужен доступ, с ключевым словом `friend`.
- В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель `this` ей не передается.
- Одна функция может "дружить" сразу с несколькими классами

Пример дружественной функции

```
class monster;// Предварительное объявление класса
class hero
{
   ...
   void kill(monster &);
};

class monster
{
   ...
   friend int steal_ammo(monster &);
   /* Класс hero должен быть определен ранее */
   friend void hero::kill(monster &);
};

int steal_ammo(monster &M){return --M.ammo;}
void hero::kill(monster &M){M.health = 0; M.ammo = 0;}
```

Дружественный класс

- Если все методы какого-либо класса должны иметь доступ к скрытым полям другого, весь класс объявляется дружественным с помощью ключевого слова `friend`.
- Объявление `friend` не является спецификатором доступа и не наследуется. Обратите внимание на то, что класс сам определяет, какие функции и классы являются дружественными, а какие нет.

Деструктор

Деструкторы

- Деструктор - это особый вид метода, применяющийся для освобождения памяти, занимаемой объектом. Деструктор вызывается автоматически, когда объект выходит из области видимости:
  - для *локальных* переменных - при выходе из блока, в котором они объявлены;
  - для *глобальных* - как часть процедуры выхода из main;
  - для *объектов, заданных через указатели*, деструктор вызывается неявно при использовании операции delete (автоматический вызов деструктора при выходе указателя из области действия не производится).

Деструкторы

- При уничтожении массива деструктор вызывается для каждого элемента удаляемого массива. Для динамических объектов деструктор вызывается при уничтожении объекта операцией `delete`. При выполнении операции `delete`[] деструктор вызывается для каждого элемента удаляемого массива.

Деструкторы

- Имя деструктора начинается с тильды (~), непосредственно за которой следует имя класса. Деструктор:
  - не имеет аргументов и возвращаемого значения;
  - не может быть объявлен как `const` или `static`;
  - не наследуется;
  - может быть виртуальным;
  - может вызываться явным образом путем указания полностью уточненного имени; это необходимо для объектов, которым с помощью `new` выделялся конкретный адрес.

Пример деструктора

```
#include <string.h>

class String {
public:
   String( char *ch );  // Declare constructor
   ~String();           //  and destructor.
private:
   char    *_text;
   size_t  sizeOfText;
};

// Define the constructor.
String::String( char *ch ) {
   sizeOfText = strlen( ch ) + 1;
   _text = new char[ sizeOfText ];
   if( _text )
      strcpy_s( _text, sizeOfText, ch );
}
```

```
// Define the destructor.
String::~String() {
   // Deallocate the memory
   // that was previously reserved
   //  for this string.
   if (_text)
      delete[] _text;
}

int main() {
   String str("The piper in the glen...");
}
```

Перегрузка операторов в C++

Перегрузка операторов

- C++ 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 C++, за исключением:

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

Перегрузка операторов

- Перегрузка операторов - это особый вид полиморфизма, представляющий «синтаксический сахар» языка программирования и позволяющий использовать стандартные операторы языка для взаимодействия с собственными типами данных.
- Выражение

```
add (a, multiply (b,c))
```

с использованием перегрузки операторов можно описать как

```
a + b * c
```

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

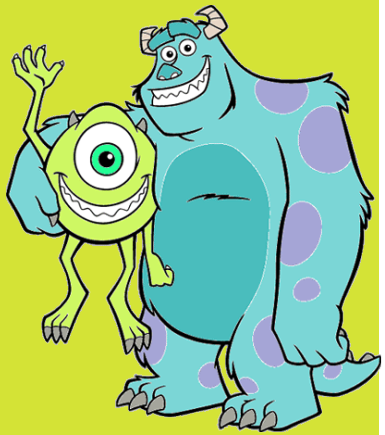
- Унарная функция-операция, определяемая *внутри класса*, должна быть представлена с помощью нестатического метода без параметров, при этом операндом является вызвавший ее объект, например:

```
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/постфиксыный 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) const
        {
          if( health < M.get_health()) return true;
          return false;
        }
};
```

- Операция “<“ часто переопределяется для структур данных для дальнейшей возможности их сортировки.

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;
  }
```

Пример - перегрузка операции «+»

вне класса

```
Time operator+(const Time& lhs,
               const Time& rhs)
{
  Time temp = lhs;
  temp.seconds += rhs.seconds;
  temp.minutes += temp.seconds / 60;
  temp.seconds %= 60;
  temp.minutes += rhs.minutes;
  temp.hours += temp.minutes / 60;
  temp.minutes %= 60;
  temp.hours += rhs.hours;
  return temp;
}
```

внутри класса

```
Time Time::operator+
     (const Time& rhs) const
{
  Time temp = *this;
  temp.seconds += rhs.seconds;
  temp.minutes += temp.seconds / 60;
  temp.seconds %= 60;
  temp.minutes += rhs.minutes;
  temp.hours += temp.minutes / 60;
  temp.minutes %= 60;
  temp.hours += rhs.hours;
  return temp;
}
```

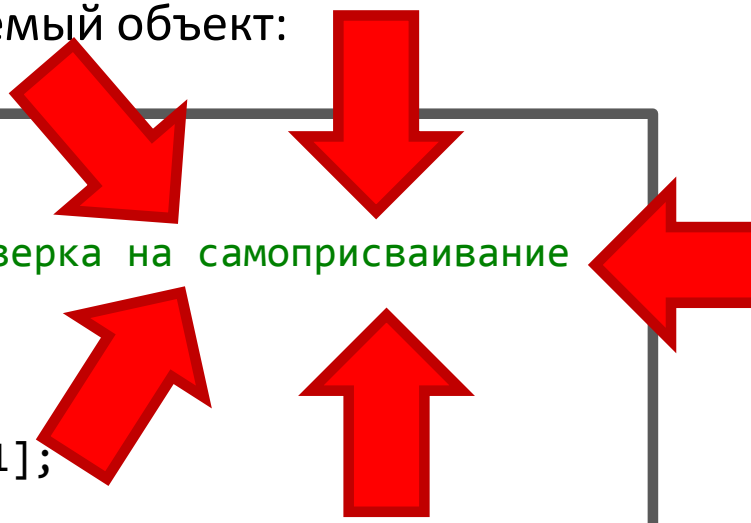
Перегрузка операции присваивания

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 присвания

- 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;  // !!??
```

Итого

- Друзья: методы, классы (нарушают инкапсуляцию)
- Деструкторы: если в классе есть объекты, выделенные в динамической памяти, необходимо явно уничтожить и освободить память деструкторе
- 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 переприсваивать такие операции как
  - ->
  - []
  - ()
  - new и delete
  - Арифметические операции и многое другое