Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

17 февраля 2015

На Си Вы могли просто делать ошибки, на C++ Вы сможете их также наследовать

Включает в себя:

- Процедурные возможности — акцент делается в основном на реализацию программы посредством эффективны функций. Применятеся для небольших программ, где требуется оптимизировать время и объем занимаемой памяти.
- Объектно-ориентированные средства (классы) — время выполнения и объем программы увеличиваются, но программист получает выигрыш при создании больши программных продуктов, который выражается в уменьшении трудоемкости.

Объектно ориентированные средства C++ позволяют разработчику ПО: Создавать собственные пользовательские типы данных (классы) для описания объектов реального мира и использовать их также как и объекты базового типа.

Базовые понятия ООП

- Инкапсуляция — каждый класс представляет уникальный набор данных и операций (методов) над этими данными. Объект обычно представляется как «Черный ящик» в котором скрыты детали реализации.
- Наследование — возможность добавления в класс новых свойств на основе базовых классов.
- Полиморфизм — получение разного поведения объектов во время выполнения посредством одного и того же кода.

Переход от процедурного программирования

```
struct Date {
int year;
int month;
int day;
};

int DayOfYear(const Date *date);

//main.cpp
int main()
{
Date d = {10, 10, 2015};
std::cout << DayOfYear(&d);
}
```

Объектно-ориентированное решение

```
class Date {
int year;
int month;
int day;
public:
int DayOfYear(const Date *date);
};


//main.cpp
int main()
{
Date d; //создание объекта
std::cout << d.DayOfYear(); //вызов метода класса
}
```

Объектно-ориентированное решение

Данные содержвается в самом объекте, как в капсуле. Главный принцип ООП — не получайте посредством объекта данные, необходимые для совершения операции. Вместо этого «попросите» объект содержащий данные сделать эту операцию для Вас. Этот принцип называется делегированием.

Объявление класса

```
class имя_класса {
список_членов_класса: переменные (member variables)
    и методы (member functions)
};
```

Объявление класса

Компилятор извлекает из класса следующую информацию:

- новый пользовательский тип данных;
- имена и типы членов класса, включая типы переменных и прототипы функций;
- спецификаторы доступа (ограничения по использованию функций и переменных класса).

Абстракция

Основное назначение классов — описывать объекты реального мира, а следовательно проектирование класса == моделированию.

Абстрагируем животное

```
enum SEX {MALE, FEMALE};

class Animal {
unsigned int m_age;
bool m_isPet;
SEX m_sex;
};
```

Нарушение инкапсуляции

```
enum SEX {MALE, FEMALE};

struct Animal {
unsigned int m_age;
bool m_isPet;
SEX m_sex;
};

int main()
{
Animal an;
an.m_age = 100;
an.m_isPet = true;
}
```

Защита данных

```
enum SEX {MALE, FEMALE};

class Animal {
private: //защита данных от изменения
unsigned int m_age;
bool m_isPet;
SEX m_sex;
};

int main()
{
Animal an;
an.m_age = 100; //ошибка компилятора
an.m_isPet = true; //ошибка компилятора
}
```

Изменение защищенных данных

```
enum SEX {MALE, FEMALE};

class Animal {
private: //защита данных от изменения
unsigned int m_age;
bool m_isPet;
SEX m_sex;
public:
void setAge(int age) { m_age = age; }
};

int main()
{
Animal an;
an.setAge(100); //???
an.m_age = 50; //???
}
```

Спецификаторы доступа

- Каждый уровень доступа к данным и функциям определяется ключевыми словами public, private, protected.
- Не обязательно для каждого члена класса указывать спецификатор доступа.
- По умолчанию объявлены как private.
- Секции с одним и тем же ключевым словом может быть сколько угодно и идти они могут в любом порядке.
- private и protected члены другого объекта такого же класса будут внутри всех методов класса.

Создание объекта

Создание экземпляра класса осуществляется также как создание переменной.

```
#include "animal.h"

int main()
{
Animal a;
}
```

Объявление и определение inline методов

Создание экземпляра класса осуществляется также как создание переменной.

```
class Animal {
public:
void setPet() { isPet = true; } //неявно
};

class Animal {
public:
inline void setPet(); //явно
};

inline void Animal::setPet() { isPet = true; }
```

Объявление и определение не-inline методов

Создание экземпляра класса осуществляется также как создание переменной.

```
//animal.h
class Animal {
public:
void setPet();
};

//animal.cpp
void Animal::setPet() { isPet = true; }
```

Указатель this

- указатель this формируется компилятором внутри нестатических методов, следовательно использовать его можно только внутри класса.
- для класса A тип указателя this — A* const.