Вступительная лекция

Введение объектно-ориентированное программирование

05.02.2025

Как зовут преподавателя?

- Соломин Михаил Геннадьевич
- Tg: @Min9fi1st
- Email: m.solomin@g.nsu.ru
- Основная коммуникация через telegram.

Что мы изучаем

Язык программирования C++
Курс длится 2 семестра.
1 семестр:

- Основы C++
- Основные принципы ООП (наследование, полиморфизм, инкапсуляция)
- Стандартная библиотека (контейнеры и алгоритмы)
- SOLID принципы

Что мы изучаем

2 семестр:

- Углубленное изучение шаблонов
- Многопоточное программирование на C++
- Паттерны проектирования

C++: inception

- Bjarne Stroustrup разработал C++ в 1983 году во время работы в AT&T Bell Labs для облегчения работы над крупномасштабными задачами.
- Изначально C++ компилировался в язык C и первый компилятор назывался Cfront.
- До сих пор общее подмножество C и C++ это почти весь C.

Объединение данных и методов

• Одной из главных особенностей C++ является объединение данных и методов их обработки.

```
struct Point  double x, y; ;
struct Triangle { struct Point pts[3]; }; // C way
double square(const struct Triangle *pt) {
    double sq = pt->pts[0].x * (pt->pts[1].y - pt->pts[2].y) +
    pt->pts[1].x * (pt->pts[2].y - pt->pts[0].y) +
    pt->pts[2].x * (pt->pts[0].y - pt->pts[1].y);
    return abs(sq) / 2.0;
}
```

Объединение данных и методов

• Одной из главных особенностей C++ является объединение данных и методов их обработки.

```
struct Point { double x, y; };
struct Triangle { // C++ way
    struct Point pts[3];
    double square() {
        double sq = pts[0].x * (pts[1].y - pts[2].y) +
        pts[1].x * (pts[2].y - pts[0].y) +
        pts[2].x * (pts[0].y - pts[1].y);
        return abs(sq) / 2.0;
    }
};
```

Объединение данных и методов

- Не обязательно определять функцию внутри.

```
struct Triangle {
    Point pts[3];
    double square() const; // const так как мы не меняем полей класса
};
double Triangle::square() const {
    double sq = pts[0].x * (pts[1].y - pts[2].y) +
    pts[1].x * (pts[2].y - pts[0].y) +
    pts[2].x * (pts[0].y - pts[1].y);
    return abs(sq) / 2.0;
}
```

Объединение данных и методов

- Использование получившегося кода.

```
Triangle t; // объект типа Triangle
t.pts[0] = Point {1.0, 1.0}; // запись поля
t.pts[1] = Point {3.0, 3.0};
t.pts[2] = Point {1.0, 2.0};
double a = t.square(); // вызов метода
```

Таким образом с методами занесёнными внутрь структуры, мы работаем так же как и с полями через точку либо через стрелочку.

Обсуждение: this

• Хорошо спроектированная структура данных на C часто также берёт “указатель на себя“ первым параметром.
• Делая его неявным, мы как бы говорим “сделай для себя“.

```
double square(const struct Triangle *pt) {
    double sq = pt->pts[0].x * (pt->pts[1].y - pt->pts[2].y) +
    pt->pts[1].x * (pt->pts[2].y - pt->pts[0].y) +
    pt->pts[2].x * (pt->pts[0].y - pt->pts[1].y);
    return abs(sq) / 2.0;
}
```

Обсуждение: this

• Хорошо спроектированная структура данных на C часто также берёт “указатель на себя“ первым параметром.
• Делая его неявным, мы как бы говорим “сделай для себя“.

```
double Triangle::square() const {
    double sq = this->pts[0].x * (this->pts[1].y - this->pts[2].y) +
    this->pts[1].x * (this->pts[2].y - this->pts[0].y) +
    this->pts[2].x * (this->pts[0].y - this->pts[1].y);
    return abs(sq) / 2.0; }
```

• Указывать явный this иногда необходимо. Но здесь это сделано без необходимости и это дурной тон.

Обсуждение: this

• Хорошо спроектированная структура данных на C часто также берёт “указатель на себя“ первым параметром.
• Делая его неявным, мы как бы говорим “сделай для себя“.

```
double Triangle::square() const {
    double sq = pts[0].x * (pts[1].y - pts[2].y) +
    pts[1].x * (pts[2].y - pts[0].y) +
    pts[2].x * (pts[0].y - pts[1].y);
    return abs(sq) / 2.0;
}
```

• Здесь мы не пишем this, но подразумеваем его.

```
Triangle t; t.square(); // this == &t
```

Обобщение данных и методов

• Ещё одна важная концепция это обобщение через механизм шаблонов.
• Конкретный тип: точка из двух целых координат.

```
struct Point_int { int x, y; };
Point_int p;
```

• Обобщённый тип: точка из двух любых координат.

```
template <typename T> struct Point { T x, y; };
Point<int> pi;
Point<double> pd;
```

Обобщение данных и методов

• Тот же треугольник можно обобщить на любые типы точек.

```
template <typename T> struct Point { T x, y; };
template <typename U> struct Triangle {
   Point<U> pts[3]; // U будет подставлено как T в Point
// почему я изменил интерфейс на double_square?
   U double_square() {
   U sq = pts[0].x * (pts[1].y - pts[2].y) +
    pts[1].x * (pts[2].y - pts[0].y) +
     pts[2].x * (pts[0].y - pts[1].y);
    return (sq > 0) ? sq : -sq; // почему больше не abs?
   }
};
```

Обсуждение

- С одной стороны обобщение создаёт возможности.

Triangle<double> t;

Triangle<float> tf;

- С другой стороны оно создаёт проблемы.
- Это очень часто ходит рука об руку.

Обобщение функций

- Пожалуй единственным способом написать на C максимум двух чисел является макрос.

define MAX(x, y) (((x) > (y)) ? (x) : (y))

- Перечислите все проблемы в этом макросе.

Обобщение функций

• Пожалуй едиснтвенным способом написать на C максимум двух чисел является макрос.

```
define MAX(x, y) (((x) > (y)) ? (x) : (y))
```

• Перечислите все проблемы в этом макросе.

• На C++ шаблон функции лишён этих проблем.

```
template <typename T> T max(T x, T y) {
    return (x > y) ? x : y;
}
```

Обобщение вместо void*

- Стандартная функция из библиотеки C.

void qsort (void* base, size_t num, size_t size, int (*compar)(const void*,const void*));

- Что можно с ней сделать используя шаблоны?

Обобщение вместо void*

- Первая итерация.

```
template <typename T, typename Comp>
void qsortpp (T* base, size_t num, Comp compare);
```

- Вместо передачи указателя и длины, можно передавать два указателя на начало и конец интервала.

Обобщение вместо void*

• Вторая итерация.

template <typename T, typename Comp>
void qsortpp (T* start, T* fin, Comp compare);

• Вместо указателей можно использовать указателе-подобные объекты, так называемые итераторы и получить.

template <typename It, typename Comp = std::less<> >
void sort (It start, It fin, Comp compare);

Обсуждение

- Что будет работать быстрее?

qsort(narr, nelts, sizeof(int), intless);

или

std::sort(narr, narr + nelts);

- Давайте поставим эксперимент и попробуем истрактовать результат.

Стандартная библиотека

- C++ имеет массу стандартных обобщённых контейнеров и обобщённых алгоритмов над ними.
- Например list это стандартный двусвязный список.
- Работа с ним не сложнее, чем с самописным треугольником.

```
Triangle<double> t; // создать треугольник
t.pts = {{1, 0}, {2, 1}, {3, 2}}; // задать точки
double sq = t.double_square() / 2; // вычислить площадь
```

Стандартная библиотека

• C++ имеет массу стандартных обобщённых контейнеров и обобщённых алгоритмов над ними.
• Например std::list это стандартный двусвязный список.
• Работа с ним не сложнее, чем с самописным треугольником.

```
std::list<int> lst; // создать список
lst.push_back(2); // добавить несколько узлов
lst.push_back(1);
lst.push_front(1); // 1, 2, 1
lst.remove_if(1); // удалить все единицы
```

Мы только потрогали воду лапкой

- Язык C++ богат возможностями.
- Если вы просто знаете C, то многим вы можете начинать пользоваться прямо сейчас.
- Но увы, язык C++ сложен и коварен.
- Использовать его, не обладая глубоким пониманием происходящего часто бывает неприятно.

Как преуспеть в домашних работах

- Организация кода.
  - Убедитесь, что вы умеете работать с системой контроля версий, что ваши программы разбиты на переиспользуемые модули, что модули логично названы.
- Система сборки.
  - Лучше всего если вы освоите cmake, но любая другая система тоже подойдёт. Проверьте сборку под Windows и под Linux.
- Тестирование и обработка ошибок ввода.
  - Не бойтесь писать и даже генерировать end-to-end тесты: вход, выход.
- Качество кода.
  - Убедитесь, что ваши методы достаточно короткие, что их имена логичны, что на не модифицирующих методах не забыт const.