Шаблоны

А. Г. Фенстер, fenster@fenster.name

4 ноября 2009 г.

Конспект семинара №3 по объектно-ориентированному программированию. Пожалуйста, не относитесь к этому как к руководству по C++ или ООП: здесь изложен только самый минимум информации по теме; тем не менее, этого минимума должно хватить для выполнения второго задания.

О найденных ошибках и неточностях, пожалуйста, пишите мне по адресу fenster@fenster.name. Также буду рад получить любые комментарии по поводу этого текста.

В первой и второй частях было введено понятие класса и показано, как определяются конструктор, деструктор и основные операторы.

В этой части описано только создание шаблонов в языке C++. Для выполнения второго задания вам нужно вспомнить, как устроены динамические списки в языке C (материал прошлого семестра). Пожалуйста, ознакомьтесь с текстом на эту тему, чтобы вспомнить, о чём идёт речь.

Напоминаю, что выделение памяти в языке C++ производится при помощи оператора `new` (вместо `malloc`), а освобождение — при помощи оператора `delete` (вместо `free`).

1 Шаблоны функций

При программировании нередко возникают ситуации, когда хочется иметь одну и ту же функцию, определённую для аргументов разных типов. Пусть, например, мы хотим определить функцию поиска минимального из двух чисел. Язык C++ (в отличие от C) позволяет определить

несколько функций с одним и тем же именем, но различными типами параметров, поэтому мы можем написать:

```
int min(int a, int b)
{
    return (a < b) ? a : b;
           /* условное выражение: если a < b, то a, иначе b */
}
double min(double a, double b)
{
    return (a < b) ? a : b;
}
```

и, наконец, для определённого ранее класса `Rational` (если дополнительно определить для него `bool operator<`:

```
Rational min(Rational a, Rational b)
{
    return (a < b) ? a : b;
}
```

Легко заметить, что содержимое этих трёх функций не отличается.

Чтобы не писать одно и то же много раз, программист на языке C написал бы *макрос* при помощи `#define`:

```
#define min(a, b) ((a) < (b) ? (a) : (b))
```

Этот вариант работает так: увидев идентификатор `min` в программе, препроцессор просто заменит его на указанное справа выражение, подставив вместо `a` и `b` соответствующие выражения. Так как такая текстовая замена не учитывает, что на самом деле представляют собой `a` и `b`, при написании таких макросов необходимо использовать скобки вокруг аргументов и вокруг всего результирующего выражения, в противном случае результат подстановки может отличаться от того, что мы рассчитываем получить.

Для решения этой проблемы в языке C++ используются *шаблоны* (templates). В данном примере нам нужен шаблон функции: функция, которая, помимо своих обычных параметров, зависит от ещё одного параметра: вместо конкретных типов `int`, `double`, `Rational` мы будем использовать некоторый тип `T`. То, что этот тип — параметр, указывается так:

```
template <typename T>
T min(T a, T b)
{
    return (a < b) ? a : b;
}
```

Теперь мы имеем возможность вызвать функцию `min` для любого типа, для которого определена операция сравнения. Одно такое определение функции вводит сразу несколько функций, отличающихся только типом параметров и результата:

```
int a = 5, b = 7, c;
c = min<int>(a, b);

double d = 1.4, e = 0.42, f;
f = min<double>(d, e);

Rational g(3, 5), h(1, 2), i;
i = min<Rational>(g, h);
```

На самом деле, в данном случае явное указание типа (`<int>`, `<double>`, `<Rational>`) можно опустить, т. к. оно автоматически будет определено из контекста.

Заметьте, что в примере выше вызвать `min(a, d)` нельзя: компилятор не сможет определить, какой из вариантов — `min<int>` или `min<double>` — использовать. Указав это явно, можем получить два различных результата: в случае `min<int>(a, d)` получим 1 (`d = 1.4` будет приведено к 1), в случае `min<double>(a, d)` получим 1.4.

2 Шаблоны классов

Аналогичная идея используется при создании шаблонов классов: указывается тип-параметр, который используется в описании полей и методов класса.

Рассмотрим для примера шаблон класса, хранящего вектор длины 2 из элементов некоторого типа.

```
template <typename T>
class Vector
{
private:
    T arr[2];
public:
    Vector(T a, T b)
    {
        arr[0] = a;
        arr[1] = b;
    }
    T operator[](int index) const
    {
        if (index < 0 || index >= 2)
            throw "error";
        return arr[index];
    }
    /* T& operator[](int index) пишется аналогично, */
    /* прочие операторы опустим для краткости */
};
```

Описав такой класс, мы можем создать, например, вектор целых чисел:

```
Vector<int> v1(1, 2);
```

вектор вещественных чисел:

```
Vector<double> v1(1.5, 2.2);
```

и, вообще говоря, вектор элементов совершенно любого типа: главное, чтобы для этого типа были определены все необходимые операции.

В задании на «пятёрку» необходимо реализовать класс-итератор, перебирающий элементы списка. Для этого класса необходимо определить операторы * и ->. Подробная информация об этом будет в следующей части.