Билеты к экзамену C++
МКН, Современное программирование
семестр II

Тамарин Вячеслав

June 16, 2020

Contents

Вопрос 1 Шаблоны 2
i Решение в стиле C 2
ii Шаблонные классы 2
iii Шаблонные функции 3
iv Специализация 4
v Шаблонный параметр, не являющийся типом 4

Вопрос 1 Шаблоны

- решение в стиле C (`#define`)
- шаблонные классы
- шаблонные функции
- специализация шаблонов (частичные и полные; в т.ч. для функций)
- шаблонный параметр, не являющийся типом

i Решение в стиле *C*

Пусть есть класс массива для целых чисел или умный указатель

```
class MyArray {
private:
    int *array;
};

class scoped_ptr {
private:
    GaussNumber *ptr;
}
```

Эти классы рассчитаны только для одного типа данных и для каждого типа придется вручную создавать новый тип.

Решить проблему можно с помощью `#define`. Классы для каждого нового типа будет генерировать препроцессор с помощью макросов.

MyArray.h

```
#define MyArray(TYPE) class MyArray_##TYPE {\
private: \
    TYPE *array; \
    size_t size; \
public: \
    TYPE get(size_t index) { \
        return array[index]; \
    } \
};
```

main.c

```
#include "MyArray.h"
MyArray(int);
MyArray(double);

int main() {
    MyArray_int a; // вместо MyArray(int) будет полный текст макроса
    MyArray_double b;
}
```

Проблема: Программист и компилятор видят разный исходный текст, разные сообщения од ошибках, препрепроцессор заменит любое подходящее слово на данный код.

ii Шаблонные классы

MyArray.h

```
template <typename T>
class MyArray {
private:
    T *array;
    size_t size;
```

```
public:
    T& get(size_t index) {
        return array[i];
    }
};
```

Можно вынести определение методов за пределы объявления класса

MyArray.h

```
template<class T> // синоним template<typename T>
T& MyArray<T>::operator[] (size_t index) {
    return array[i];
}
```

main.cpp

```
#include "MyArray.h"
int main() {
    MyArray<int> a;
    MyArray<double> b;
    MyArray<MyArray<int>> c; // лучше не писать до c++11
}
```

Особенности:

1. Подстановку делает компилятор, а не препроцессор
2. Код шаблонного класса всегда в заголовочном файле
3. Иногда помещают в `MyArray_impl.h`
4. Увеличивается время компиляции
5. Методы шаблонного класса всегда `inline`

iii Шаблонные функции

```
template <class T>
void swap(T &a, T &b) {
    T t(a);
    a = b;
    b = t;
}
int i = 10, j = 20;
swap<int>(i, j);
```

```
template <typename V>
void reverse(MyArray<V> &a) {
    V t;
    for (size_t i = 0; i < a.size()/2; ++i) {
        t = a.get(i);
        a.set(i, a.get(a.size() - i - 1);
        a.set(a.size() - i - 1, t);
    }
}
// Вызов
reverse<int>(a);
```

Вывод шаблонных параметров

Компилятор может понять, какие аргументы у шаблона функции, если это однозначно определяется.

```
MyArray<int> a;
MyArray<double> b;
reverse(a);
reverse(b);
```

iv Специализация

Идея: оптимизация для конкретного класса.

Общая версия

```
template<typename T>
class Array {
private:
    T *a;
public:
    Array (size_t size) {
        a = new T [size];
        ...
    }
};
```

Полная специализация

Для bool

```
template<>
class Array<bool> {
private:
    char *a;
public:
    Array (size_t size) {
        a = new char [(size-1)/8 + 2];
        ...
    }
};
```

Частичная специализация

Для массивов

```
template<class T>
class Array <Array<t>> {
    T **a;
};
```

v Шаблонный параметр, не являющийся типом

```
template<size_t Size>
class Bitset {
private:
    char m[(Size-1)/8 + 1];
public:
    bool get(size_t index) { ... }
};

Bitset<128> b1;
```