Семестр 2. Лекция 1. Шаблоны.

Евгений Линский

17 Февраля 2017

Проблема

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

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

- Классы расчитаны только на один тип данных
- Новый тип потребует создание нового класса “вручную”
- Как можно это исправить?

Решение в стиле C. Препроцессор.

Пусть классы для каждого нового типа генерирует 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]; \
    } \
};
```

- \— перенос строки в макросе
- ## — конкатенация строки и параметра макроса

Решение в стиле C. main.

```
//main.c
#include "MyArray.h"

MyArray(int);
MyArray(double);

int main() {
    MyArray_int a;
    MyArray_double b;
    ...
}
```

- вместо MyArray(int) препроцессор подставит полный текст макроса, заменив TYPE на int.

Решение в стиле C. main.

```
//main.c
#include "MyArray.h"

MyArray(int);
MyArray(double);

int main() {
    MyArray_int a;
    MyArray_double b;
    ...
}
```

- вместо MyArray(int) препроцессор подставит полный текст макроса, заменив TYPE на int.
- Препроцессор
  - программист и компилятор видят разный исходный текст (сообщения об ошибках компиляции, отладка)
  - препроцессор не выясняет “смысл” токена перед его заменой (int TYPE)

Решение в стиле C. main.

```
//main.c
#include "MyArray.h"

MyArray(int);
MyArray(double);

int main() {
    MyArray_int a;
    MyArray_double b;
    ...
}
```

- вместо MyArray(int) препроцессор подставит полный текст макроса, заменив TYPE на int.
- Препроцессор
  - программист и компилятор видят разный исходный текст (сообщения об ошибках компиляции, отладка)
  - препроцессор не выясняет “смысл” токена перед его заменой (int TYPE)
- Q: a.cpp и b.cpp включатют “MyArray.h”. double definition?

Решение в стиле C++. Шаблонный класс.

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

Шаблоны. main.

```
//main.cpp
#include "MyArray.h"
int main() {
    MyArray< int > a;
    MyArray< double > b;
    MyArray< <MyArray<int> > c;
    ...
}
```

- Подстановку делает компилятор, а не препроцессор

Шаблоны. main.

```
//main.cpp
#include "MyArray.h"
int main() {
    MyArray< int > a;
    MyArray< double > b;
    MyArray< <MyArray<int> > c;
    ...
}
```

- Подстановку делает компилятор, а не препроцессор
- Код шаблонного класса (объявление и определение) всегда помещается в заголовочный файл. Q: А почему нельзя в cpp?

Шаблоны. main.

```
//main.cpp
#include "MyArray.h"
int main() {
    MyArray< int > a;
    MyArray< double > b;
    MyArray< <MyArray<int> > c;
    ...
}
```

- Подстановку делает компилятор, а не препроцессор
- Код шаблонного класса (объявление и определение) всегда помещается в заголовочный файл. Q: А почему нельзя в cpp?
- Иногда их эстетических соображений объявление помещают в “MyArray.h”, а определение в “MyArray_impl.h”

Шаблоны.

- *template<typename T>* и *template<class T>* синонимы
- До c++11 лучше не писать *MyArray«MyArray<int»*: компилятор путает со сдвигом вправо ».
- Длинное объявление типа можно укоротить *typedef MyArray< <MyArray<int> > MyArray2d;*
- Методы шаблонного класса всегда *inline* (Q: почему?)

Жаргон:

- *T* — шаблонный параметр
- Иногда говорят, что шаблоны это “полиморфизм времени компиляции”

Еще примеры методов.

```
//MyArray.h
template <typename T>
MyArray<T>::MyArray(size_t s){
    size = s;
    array = new T [size];
}
```

```
//MyArray.h
template <typename T>
MyArray<T>& MyArray<T>::operator=(const MyArray<T> &a){
    ...
    return *this;
}
```

Несколько шаблонных параметров.

```
template <typename K, typename V>
class TreeItem {
private:
    K key;
    V value;
    ...
};
```

Шаблонная функция.

```
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);
```

Шаблонная функция.

```
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);
```

Несколько шаблонных параметров.

```
template <typename T, typename V>
void copy(MyArray<T> &a, MyArray<V> &b){
  if (a.size() != b.size()){
    return;
  }
  for (size_t i = 0;i != a.size(),++i){
    a.set(i,b.get(i))
  }
```

Вызов:

```
copy<int, double>(a,b);
```

Вывод шаблонных параметров.

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

- Компилятор может понять, какие аргументы у шаблона функции, если список параметров функции однозначно идентифицирует набор аргументов шаблона.
- ДЗ, Q: а когда компилятор не может вывести (deduce) шаблонные параметры? примеры?

ООП vs Шаблоны

ООП:

```
class Comparable{
    virtual int compare(comparable* b)=0;
};
void nsort(Comparable** m, size_t size){}
```

Для примитивных типов надо написать “обертку” (наследник *Comparable*)

Шаблоны:

```
template < typename T >
void nsort(T* t,size_t size){}
```

Для каждого типа будет сгенерирована отдельная функция.