

Технологии индустриального программирования

Лекция 4

Шаблоны

Р.В. Шамин

профессор кафедры индустриального программирования

Что такое шаблоны?

Механизмы шаблонов или парадигма обобщенного программирования основаны на том, что код функций и классов может быть одинаковым для различных типов данных, поэтому можно определить и реализовать функцию для обобщенного типа данных. А в момент компиляции при вызове функции обобщенный тип будет заменен на реальный.

```
1  #include <iostream>
2  using namespace std;
3
4  int add(int a, int b) {
5      int c;
6      c = a + b;
7      return c;
8  }
9
10 string add(string a, string b) {
11     string c;
12     c = a + b;
13     return c;
14 }
15
16 int main() {
17     cout << "\n\n" << add(7, 8);
18     cout << "\n\n" << add("mama", "papa") << "\n\n";
19     return 0;
20 }
```

Различие только в типе данных

Код функции одинаковый

Что такое шаблоны?


Используем шаблон:

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T> T add(T a, T b) {
5      T c;
6      c = a + b;
7      return c;
8  }
9
10 int main() {
11     cout << "\n\n" << add(7, 8);
12     cout << "\n\n" << add(string("mama"), string("papa")) << "\n\n";
13     return 0;
14 }
```

Определяем не функцию, а шаблон



Определяем обобщенный тип данных



Использование указателей

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T> void calc(T *a, int N) {
5      *a = 0;
6      for (int n = 0; n < N; n++) {
7          *a += (n + 1);
8      }
9  }
10
11 int main() {
12     int a;
13     double x;
14     calc(&a, 10);
15     calc(&x, 100);
16
17     cout << "\n\n" << a << "\t" << x << "\n\n";
18
19     return 0;
20 }
```



55 5050

Явное указание типа данных

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T> T calc(T a, T b) {
5      return a / b;
6  }
7
8  int main() {
9
10     int c = calc<int>(10, 3);
11
12     cout << "\n\n c = " << c << "\n\n";
13
14     return 0;
15 }
```

Явно указываем какой тип данных используем



c = 3

Перегрузка функций

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T> T calc(T a, T b) {
5      return a * b;
6  }
7
8  template<typename T> T calc(T a) {
9      return a * a;
10 }
11
12 int main() {
13
14     cout << "\n\n" << calc(7, 8) << "\t" << calc(6.0) << "\n\n";
15
16     return 0;
17 }
```



56 36

Использование нескольких типов параметров

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T, typename S> void print(T a, S b) {
5      cout << "\n\n" << a << "\n\n";
6      cout << b << "\n\n";
7  }
8
9  int main() {
10
11      print(1024, string("Mama"));
12
13      return 0;
14  }
```

Два разных типа данных

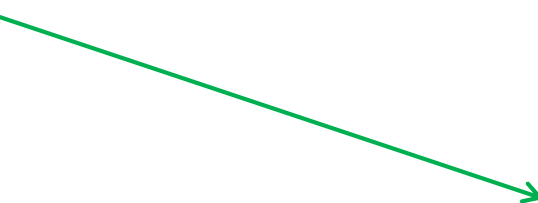
1024

Mama

Автоматическое выведение типов

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T, typename S> decltype(auto) div(T a, S b) {
5      |   return a / b;
6  }
7
8  int main() {
9
10     auto c = div(1024, 3.0);
11
12     cout << "\n\n c = " << c << "\n\n";
13
14     return 0;
15 }
```

Не знаем какой тип будет возвращен



c = 341.333

Механизмы шаблонов можно использовать и для создания шаблонов классов.

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T>
5  class TItem {
6  |   T name;
7  public:
8      TItem(T name) {
9          |   this->name = name;
10         }
11         T get_name() {
12             |   return name;
13         }
14 };
15
16 int main() {
17     TItem<int>* item1 = new TItem<int>(23);
18     TItem<string> item2(string("Box"));
19
20     cout << "\n\n" << item1->get_name();
21     cout << "\n\n" << item2.get_name() << "\n\n";
22
23     delete item1;
24
25     return 0;
26 }
```



23

Box

Специализация шаблона

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  template<typename T>
6  class TItem {
7  |   T name;
8  public:
9  |   TItem(T name) {
10 |       this->name = name;
11 |   }
12 |   T get_name() {
13 |       return name;
14 |   }
15 };
16
17 template<>
18 class TItem<int> {
19 |   int name;
20 public:
21 |   TItem(int name) {
22 |       this->name = name;
23 |   }
24 |   string get_name() {
25 |       return "number: " + to_string(name);
26 |   }
27 };
28
29 int main() {
30 |   TItem<int> item1(23);
31 |   TItem<string> item2(string("Box"));
32 |   cout << "\n\n" << item1.get_name();
33 |   cout << "\n\n" << item2.get_name() << "\n\n";
34 |   return 0;
35 }
```


Специализированный шаблон

Специализированное определение метода

number: 23
Box

Наследование шаблонов

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T>
5  class TItem {
6  |   T name;
7  public:
8  |   TItem(T name) {
9  |       this->name = name;
10 |   }
11 |   T get_name() {
12 |       return name;
13 |   }
14 };
15
16 template<typename T>
17 class TBox: public TItem<T> {
18 public:
19 |   TBox(T name): TItem<T>(name) { }
20 |   T get_name_box() {
21 |       cout << "\n\nI am Box!\n\n";
22 |       return TItem<T>::get_name();
23 |   }
24 };
25
26 int main() {
27 |   TBox<int> Box(123);
28 |   cout << "\n\n" << Box.get_name_box() << "\n\n";
29
30 |   return 0;
31 }
```



I am Box!

123