9-11 классы

Программирование на C++

Презентация занятия

Шаблоны в C++.

11 занятие

КОД БУДУЩЕГО

Минцифры России

инжиниринум
МГТУ им. Н.Э. Баумана

НАЦИОНАЛЬНЫЕ ПРОЕКТЫ РОССИИ
ЦИФРОВАЯ ЭКОНОМИКА

20.35 УНИВЕРСИТЕТ

2020

Программирование
на C++

Теоретическая часть

Шаблоны в C++.

11 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2020

Тема: Наследование в С++.

Шаблоны функций

Функции и классы помогают писать программы легко, безопасно и делать их более удобными для управления.
Однако пока у функций и классов есть все эти преимущества, в некоторых случаях они могут быть отчасти лимитированы требованиями к языку С++.

Например, возможно вы захотите написать функцию, которая подсчитывает сумму двух чисел, подобно следующей:

```
1   int sum(int a, int b)
2   {
3     return (a + b);
4   }
```

Становится необходимым писать новую функцию для каждого типа, например для double.

Тема: Наследование в С++.

Не будет ли лучше написать одну версию функции sum() для работы с параметрами любого типа?
Шаблоны функций дают нам возможность сделать это!
Основная идея использования шаблонов функций заключается в отсутствии необходимости определения точного типа для каждой переменной. Кроме того, C++ нас обеспечивает способностью определения функций, с помощью типов, которые называются параметрами типа шаблона.

Для определения шаблона функции, используйте ключевое слово template, после которого следует определение типа шаблона:

template <class T>

Тема: Наследование в С++.

```
1   #include <iostream>
2   using namespace std;
3
4   template <class T>
5   T sum(T a, T b)
6   {
7       return (a + b);
8   }
9
10  int main ()
11  {
12      cout << "Output:" << endl;
13      double x = 7.15, y = 15.54;
14      int a = 10, b = 20;
15      cout << sum(x, y) << endl;
16      cout << sum(a, b) << endl;
17      return (0);
18  }
```

```
Output:
22.69
30
```

Компилятор автоматически вызывает функцию для соответствующего типа.

Тема: Наследование в С++.

Шаблоны функций могут сохранить много времени, потому что они создаются один раз и работают с разными типами. Шаблоны функций улучшают управление кодом, потому что сокращается количество дублированного кода.

Повышение безопасности является другим преимуществом в использовании шаблонов функций, в связи с тем, что нет необходимости вручную копировать функции и менять типы.

Шаблоны функций также позволяют работать с множеством типов. Определение происходит с помощью разделения запятыми.

Тема: Наследование в С++.

```
1   #include <iostream>
2   using namespace std;
3
4   template <class T, class U>
5   T smaller(T a, U b)
6   {
7       return (a < b ? a : b);
8   }
9
10  int main ()
11  {
12      cout << "Output:" << endl;
13      int x = 72;
14      double y = 15.34;
15      cout << smaller(x, y) << endl;
16      return (0);
17  }
```

```
Output:
15
```

Тема: Наследование в С++.

Шаблоны классов

Так же как и с шаблонами функций, мы можем определить шаблоны классов, позволяя классам иметь элементы, которые используют параметры шаблонов в качестве типов.
Такой же синтаксис используется для определения шаблона класса:

```
template <class T>
class MyClass
{
        //code
};
```

Тема: Наследование в С++.

Шаблоны Классов

При определении ваших элементов функций вне класса, например, в другом исходном файле, будет использован специфический синтаксис.
Вам необходимо указать общий тип в угловых скобках после имени класса.

```
Output:
22
100.25
```

```
1   #include <iostream>
2   using namespace std;
3
4   template <class T>
5   class Pair {
6       private:
7           T first, second;
8       public:
9           Pair (T a, T b):
10          first(a), second(b) { }
11          T bigger();
12  };
13
14  template <class T>
15  T Pair<T>::bigger() {
16      return (first>second ? first : second);
17  }
18
19  int main()
20  {
21      cout << "Output:" << endl;
22      Pair <int> obj_int(11, 22);
23      Pair <double> obj_double(1.15, 100.25);
24      cout << obj_int.bigger() << endl;
25      cout << obj_double.bigger() << endl;
26      return (0);
27  }
```

Тема: Наследование в С++.

Специализация Шаблона

В случае регулярных шаблонов классов, способ хранения различных типов данных идентичен; для всех типов данных запускается один и тот же код.

Специализация шаблона позволяет определять различные исполнения шаблона, при передаче специфического типа в качестве аргумента.

Например, вам необходимо хранить символьный тип данных другим образом, не так, как мы делали с числовыми типами данных.

Тема: Наследование в С++.

```
#include <iostream>
using namespace std;

template <class T>
class MyClass
{
    public:
        MyClass (T x)
        {
            cout << x <<" -  isn't a char"<<endl;
        }
};

template < >
class MyClass<char>
{
    public:
        MyClass (char x)
        {
            cout << x <<" is a char!"<<endl;
        }
};
```

```
int main ()
{
    cout << "Output:" << endl;
    MyClass<int> ob1(42);
    MyClass<double> ob2(5.47);
    MyClass<char> ob3('s');
    return (0);
}
```

```
Output:
42 -  isn't a char
5.47 -  isn't a char
s is a char!
```

Программирование
на C++

Практическая часть

Шаблоны в C++.

11 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2020

Тема: Объектно-ориентированное программирование. Классы в С++.

Задание 1

Необходимо написать функцию, которая примет два числа, определит максимальное из них и вернет его в программу. Будем иметь ввиду, что в функцию мы можем передать числа разных типов. Возможен и случай, что одно число будет целым, а второе – вещественным.

*

Проверить работоспособность программы с типом данных char и string