Лямбда-функции

11 июля 2017 г.

Повторение

- Какие группы стандартных алгоритмов есть в STL?
- В чем достоинства стандартных алгоритмов?
- Какие стандартные алгоритмы в STL работают с условиями?
- Что такое предикаты?
- Что такое функторы?
- Зачем нужен связыватель std::bind?

Скомпилируется ли код?

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

bool del(const std::string& str) const
   { return str.size() == 3; }

void f(){
   std::vector<std::string> data {"abc", "c", "cba", "ac", "b"};
   size_t res = std::count_if(data.begin(), data.end(), del);
   std::cout << res << std::endl;
}
```

Скомпилируется ли код?

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

bool del(const std::string& str)
   { return str.size() == 3; }

void f(){
   std::vector<std::string> data = {"abc", "c", "cba", "ac", "b"};
   std::vector<std::string> res;
   std::copy_if(data.begin(), data.end(), res.begin(), del);
   for (auto& s : res) { std::cout << s << " "; }
}
```

Скомпилируется ли код?

```
#include <vector>
#include <algorithm>

class Pred {
   int div;
public:
   Pred(const int a) : div(a) { }
   bool operator()(const int value, const int x) const
         { return value % div == 0; }
};

void f(){
   std::vector<int> data = {2, 4, 6, 3, 7, 10, 18, 4, 2, 2};
   std::replace_if(data.begin(), data.end(), Pred(2), 0);
}
```

Скомпилируется ли код?

```
#include <vector>
#include <algorithm>

class Pred {
        int div;
public:
        Pred(int x) : div(x) { }
        bool operator()(const int a, const int b) const {
                if (a % div == 0 && b % div == 0){ return a < b; }
                return a % div == 0;
        }
};
void f(){
   std::vector<int> data = {2, 4, 6, 3, 7, 10, 18, 4, 2, 2};
   std::sort(data.begin(), data.end(), Pred(2));
}
```

Скомпилируется ли код?

```
class Pred {
    char del_symb;
public:
    Pred() { }
    Pred(char s) : del_symb(s) { }
    bool operator()(const std::string str) const {
        if ( std::find(str.begin(), str.end(), del_symb) != str.end() )
            { return true; }
        return false;
    }
};

void f(){
    std::vector<std::string> d = {"dog", "cat", "parrot", "mouse"};
    d.erase(std::remove_if(d.begin(), d.end(), Pred), d.end());
}
```

Что выведется в консоль?

```
void f(){
    std::vector<double> vec = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    std::replace_if(vec.begin(), vec.end(),
                    std::bind(std::less<double>(),
                                        std::placeholders::_1, 4), 0);
    for (auto value : vec) { std::cout  << value << " "; }
}
```

```
void f(){
    std::vector<double> vec = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    std::replace_if(vec.begin(), vec.end(),
                    std::bind(std::greater_equal<double>(),
                                        std::placeholders::_1, 4), 0);
   for (auto value : vec) { std::cout  << value << " "; }
}
```

Задание

Создайте вектор из 15 целых чисел. Замените в этом векторе на -1 все числа, кратные 5. Воспользуйтесь стандартным алгоритмом из библиотеки STL, написав для него:

а) предикат;

б) функтор.

Лямбда-функции (C++11)

```
[список_захвата](параметры) -> возвр._тип { тело_функции };
```

- анонимные (безымянные) функции
- используются вместо функторов
- могут делать код более компактным

```
[ ](int x) { return x % 2 == 0; };

[ ](const std::string& str) { return str[0] == 'a'; };

[ ](const std::string& str) -> bool
   { if (str[0] == 'a' && str.size() < 4 ) { return true; }
   else { return false; }
};
```

Возвращаемый тип

- если в функции один return, то возвращаемый тип выводится компилятором
- если в функции не один return, то возвращаемый тип следует указать за списком параметров после -> (до C++14)
- если return'а нет, то выводится тип void

```
[ ](int x) { return x % 2 == 0; };

[ ](const std::string& str) -> bool
   {  if (str[0] == 'a' && str.size() < 4 ) { return true; }
   else { return false; }
};
```

Сохранение под именем

- лямбда-функции можно присваивать имя
- лямбду можно вызывать по имени так же, как и обычную функцию
- подходит тогда, когда одну и ту же лямбду нужно использовать несколько раз

```
auto f1 = [ ](const int x) { return x % 2 == 0; };
bool res = f1(10);

size_t res1 = std::count_if(vec1.begin(), vec1.end(), f1);
size_t res2 = std::count_if(vec2.begin(), vec2.end(), f1);
```

Захват переменных

- захват переменных – передача переменных из области видимости в лямбду, так что к этим переменным можно обращаться по их именам
- имена захватываемых переменных указываются в []

```
const int div = 2;
auto f1 = [ div ](const int x) { return x % div == 0; };
bool res = f1(10);
```

Способы захвата

[ ]	• ничего не захватывается
[имя]	• переменная захватывается по значению
[&имя]	• переменная захватывается по ссылке
[&]	• все переменные из области видимости захватываются по ссылке
[=]	• все переменные из области видимости захватываются по значению
[&имя1, имя2]	• переменная 1 захватывается по ссылке, а переменная 2 – по значению

Переменные, захваченные по значению, – const

Захват переменных (пример 1)

```
#include <vector>
#include <algorithm>

void f(){
    const int max = 50;
    int min = 10;
    int count_value = 0;
    std::vector<int> vec = {1, 2, 4, 15, 6, 7, 28, 9, 10, 15, 89, 20, 9};
    std::replace_if(vec.begin(), vec.end(),
            [max, min, &count_value](const int x) -> bool {
                if (min < x && x < max) { count_value++; return true; }
                else { --min; return false; }   // Error
            },
     0);
}
```

Захват переменных (пример 2)

```
#include <vector>
#include <algorithm>

void f(){
    const int max = 50;
    int min = 10;
    int count_value = 0;
    std::vector<int> vec = {1, 2, 4, 15, 6, 7, 28, 9, 10, 15, 89, 20, 9};
    std::replace_if(vec.begin(), vec.end(),
            [max, min, &count_value](const int x) mutable -> bool {
                if (min < x && x < max) { count_value++; return true; }
                else { --min; return false; }   // OK
            },
     0);
}
```

Лямбда вместо предиката (пример)

```
#include <vector>
#include <algorithm>
#include <iostream>

void f(){
   std::vector<int> vec = {1, 2, 4, 5, 6, 7, 8, 9, 10, 15, 89, 20, 9};
   std::replace_if(vec.begin(), vec.end(),
         [ ](const int x) {
            return x % 5 == 0;
         },
         -1);
   for (auto value : vec) { std::cout  << value << " "; }
}
```

Вопросы?