# STL:
# функторы и предикаты

26 июня 2017 г.

# Некоторые базовые алгоритмы

| | |
|---|---|
| find() | • находит первое вхождение значения в последовательность |
| count() | • подсчитывает количество вхождений указанного значения в последовательность |
| replace() | • заменяет элементы с указанным значением на новое |
| remove() | • удаляет элементы с указанным значением |

# Особенности алгоритмов

```
#include <algorithm>
```

- обрабатывают данные в контейнерах
- используют итераторы, по которым получают диапазон элементов
- алгоритмы, возвращающие итератор, обычно возвращают end()

# Пример

```
#include <list>
#include <algorithm>
#include <iostream>

void f(){
   std::list<int> data = {1, 8, -9, 4, 7, -18, 11, 0, 4, -7, 4};

   int res1 = std::count(data.begin(), data.end(), 4);

   auto it1 = std::find(data.begin(), data.end(), 11);

   *(++it1) = 12;

   std::replace(data.begin(), data.end(), -18, 6);

   auto it2 = std::remove(data.begin(), data.end(), -9);
   data.erase(it2, data.end());
}
```

# Задание 1

1. Создайте вектор чисел из 20 элементов со значениями от -10 до 10 и напечатайте его.

2. Сосчитайте, сколько раз в вашем списке содержится число 0.

3. Замените все вхождения числа -1 на 1.

4. Измените список так, чтобы после числа 5 шло число 6.

5. Удалите все вхождения числа -5.

6. Выведите получившийся список в консоль.

# Задание 2

Создайте список строк (std::list) и заполните его какими-нибудь данными. Напишите функцию, которая удаляет из списка все слова, начинающиеся на букву 'a'. У класса std::list есть следующий метод:

```
template<class UnaryPredicate>
void remove_if( UnaryPredicate p);
```

# Предикаты

- функциональные объекты (функции), которые возвращают значение типа bool
- бывают унарные (принимают один аргумент) и бинарные (принимают два аргумента)
- используются в качестве параметра в стандартных алгоритмах, например:
  - задают критерий поиска (унарные)
  - задают критерий сортировки (бинарные)
  - модифицируют элементы
- передаются без ()

# Предикаты (пример)

```
#include <list>
#include <algorithm>

bool deleteA(const std::string& s1)
{
   return s1[0] == 'a'; // нельзя передать символ в функцию
}

void f(){
   std::list<std::string> data = {"ab", "c", "cab", "ac", "bb"};
   data.remove_if(deleteA);
}
```

# Функторы

- вид классов, которые включают в себя перегруженный оператор вызова функции – operator()
- позволяют использовать объект как функцию
- могут применяться там, где нужна функция
- при первом вызове функтора происходит инициализация объекта (нужный параметр передается через конструктор объекта)

# Функторы (пример 1)

```
#include <string>
#include <iostream>

struct Comp {
   bool operator()(std::string& s1,  std::string& s2) const
        {  return s1.length() < s2.length(); }
};

void f(){
        Comp c;
        std::string str1 = "abcdef";
        std::string str2 = "abacaba";
        std::cout << c(str1, str2) << std::endl;
}
```

# Функторы (пример 2)

```
#include <string>
#include <list>

struct DeleteA {
    bool operator()(const std::string& s1) const
        {  return s1[0] == 'a'; }
};

void f(){
    std::list<std::string> data = {"ab", "c", "cab", "ac", "bb"};
    DeleteA obj;
    data.remove_if(obj);
    // или data.remove_if(DeleteA());
}
```

# Функторы (пример 3)

```
#include <string>
#include <list>

class DeleteSymb {
    char symbol;
 public:
    DeleteSymb(char s) : symbol(s) { };   // передаем аргумент
    bool operator()(const std::string& s1) const
        {  return s1[0] == symbol; }
};

void f(){
   std::list<std::string> data = {"ab", "c", "cab", "ac", "bb"};
   data.remove_if(DeleteSymb('a'));
}
```

# Алгоритмы с проверкой условия

| | |
|---|---|
| find_if() | • находит первое вхождение значения, соответствующее условию |
| count_if() | • подсчитывает количество выполнений условия |
| remove_if() | • удаляет элементы, если выполняется предикат |
| replace_if() | • заменяет элементы, если выполняется предикат |

# Пример

```
class Pred {
        int value;
public:
        Pred(const int v) : value(v) { }
        bool operator()(const int elem) { return elem > value; }
};

void f(){
   std::list<int> data = {1, 8, -90, 4, 7, -18, 11, 0, 4, -7, 4};
   int res = std::count_if(data.begin(), data.end(), Pred(1));
   std::list<int>::iterator it = std::find_if(data.begin(),
                                 data.end(), Pred(2));
   std::replace_if(data.begin(), data.end(), Pred(3), 0);
}
```

# Задание 3

1. Сосчитайте, сколько отрицательных чисел содержится в списке из задания 1.

2. Замените все отрицательные нечетные числа на 0.

3. Найдите значение первого нечетного числа.

4. Удалите все четные числа, которые больше, чем переданное значение.

5. Выведите получившийся список в консоль.

# Предопределенные функторы

| Функтор | Операция | Функтор | Операция |
|---|---|---|---|
| plus | + | equal_to | == |
| minus | - (бинарный) | not_equal_to | != |
| multiplies | * | greater | > |
| divides | / | less | < |
| modulus | % | greater_equal | >= |
| negate | - (унарный) | less_equal | <= |
| logical_and | && | logical_or | \|\| |
| logical_not | != | | |

- нужно включить <functional>
- не принимают аргументы

# std::bind (C++11)

- универсальный «связыватель»
- создает новый объект-функцию с заданным набором аргументов
- использует плейсхолдеры (_1, _2 и т. п.) – показывают, какой по счету переданный аргумент нужно подставлять в функцию

```
std::bind(функция, плейсхолдеры, аргументы)
```

# std::bind (пример)

```
void f(){
    using namespace std::placeholders;
    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>(), _1, 5)), 0);
    for (auto value : vec) { std::cout  << value << " "; }
}
```

```
void f(){
    using namespace std::placeholders;
    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>(), 5, _1)), 0);
    for (auto value : vec) { std::cout  << value << " "; }
}
```

# Предопределенные функторы (пример 1)

```
#include <string>
#include <list>
#include <functional>
#include <iostream>

void f(){
   std::list<std::string> data = {"ab", "c", "cat", "ac", "b", "cat"};

   data.remove_if(std::bind (std::equal_to<std::string>(),
                             std::placeholders::_1, "cat"));
   for (auto it = data.begin(); it != data.end(); ++it){
         std::cout << *it << " ";
   }
}
```

# Предопределенные функторы (пример 2)

```
#include <string>
#include <list>
#include <functional>
#include <iostream>

void f(){
    std::list<std::string> data = {"ab", "c", "cab", "ac", "bb"};
    data.sort();
    data.sort(std::greater<std::string>());

    for (auto it = data.begin(); it != data.end(); ++it){
        std::cout << *it << " ";
    }
}
```

# Вопросы?