STL: алгоритмы

27 июня 2017 г.

Повторение

- Что такое стандартная библиотека шаблонов?
- Какие компоненты есть в STL?
- Какие последовательные контейнеры существуют в STL?
- Что такое ассоциативный массив?
  Какие контейнеры этого вида есть в STL?
- Какие виды итераторов выделяют в STL?
- Что такое функторы?
- Что такое предикаты?

Алгоритмы

```
#include <algorithm>
```

- библиотечные функции для стандартных операций (поиск, замена, сортировка и т. п.)
- обрабатывают данные в контейнерах
- используют итераторы, по которым получают диапазон элементов

Виды алгоритмов

немодифицирующие	• не изменяют данные в контейнере
модифицирующие	• изменяют данные в контейнере
алгоритмы сортировки	• сортируют элементы контейнера

и др.

Немодифицирующие алгоритмы (1)

Алгоритм	Назначение
count()	подсчитывает количество вхождений заданного значения в последовательность
count_if()	подсчитывает количество выполнений условия
find()	находит первое вхождение значения
find_if()	находит первое вхождение, соответствующее условию
find_if_not()	находит первое вхождение, которое не соответствует условию
adjacent_find()	находит первый элемент, который совпадает с соседним
all_of()	проверяет, выполняется ли предикат для всех элементов (возвращает true или false)
any_of()	проверяет, выполняется ли предикат хотя бы для одного из элементов (возвращает true или false)
none_of()	проверяет, верно ли, что предикат не выполняется ни для одного из элементов (возвращает true или false)

Немодифицирующие алгоритмы (пример 1)

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

void f(){
    std::list<int> myList = {3, 4, 7, 8, 1, 0, 2, 7};
    std::cout << std::all_of(myList.begin(), myList.end(),
          std::bind(std::greater<int>(), std::placeholders::_1, 0)) <<
                                                    std::endl;
    std::cout << std::none_of(myList.begin(), myList.end(),
          std::bind(std::less<int>(), std::placeholders::_1, 0)) <<
                                                    std::endl;
}
```

Немодифицирующие алгоритмы (2)

Алгоритм	Назначение
equal()	проверяет, совпадают ли элементы из первой и второй последовательностей с учетом порядка следования (принимает 3 или 4 аргумента)
is_permutation()	проверяет, является ли вторая последовательность некоторой перестановкой первой последовательности (принимает 3 аргумента)
mismatch()	находит первое значение, в котором две последовательности различаются (принимает 3 или 4 аргумента, возвращает пару итераторов)
search()	находит первое вхождение последовательности (принимает 4 аргумента)
search_n()	находит переданное значение, которое повторяется n раз (принимает 4 аргумента)
for_each()	выполняет немодифицирующие операции для каждого элемента последовательности

Немодифицирующие алгоритмы (пример 2)

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

void f(){
   std::string str = "Человек человеку волк! Волк – это волк!!!";
   std::string pattern = "–";
   std::cout << std::equal(str.begin(), str.end(), pattern.begin())
                                                    << std::endl;
   std::string::iterator it = std::search(str.begin(), str.end(),
                  pattern.begin(), pattern.end());
   std::string::iterator it1 = std::search_n(str.begin(), str.end(), 3,
                  '!');
}
```

Модифицирующие алгоритмы

Алгоритм	Назначение
copy()	копирует последовательность
replace()	заменяет элементы с указанным значением
replace_if()	заменяет элементы, если выполняется предикат
replace_copy()	копирует последовательность, заменяя элементы с указанным значением
remove()	удаляет элементы с указанным значением
remove_if()	удаляет элементы , если выполняется предикат
remove_copy()	копирует последовательность, удаляя элементы с заданным значением
fill()	заменяет все значения на заданное
reverse()	меняет порядок элементов на обратный

Модифицирующие алгоритмы

Алгоритм	Назначение
transform()	выполняет заданную операцию над каждым элементом
unique()	удаляет одинаковые соседние элементы
unique_copy()	копирует последовательность, удаляя одинаковые соседние элементы
swap()	меняет местами два элемента по ссылке
iter_swap()	меняет местами два элемента по итераторам
rotate()	циклически сдвигает элементы в последовательности

Модифицирующие алгоритмы (пример 1)

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

void f(){
  std::vector<int> data = {1, 8, -90, 4, 7, -18, 11, 0, 4, -7, 4};
  std::vector<int> copy_data(data.size());

  std::copy(data.begin(), data.end(), copy_data.begin());

  std::vector<int> new_v;
  std::back_insert_iterator< std::vector<int>> back_it(new_v);
  std::copy_if(data.begin(), data.end(), back_it,
       std::bind (std::less<int>(), std::placeholders::_1, 0));
}
```

Модифицирующие алгоритмы (пример 2)

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

void f(){
   std::vector<int> data = {1, 8, -90, 4, 7, -18, 11, 0, 4, -7, 4};
   std::vector<int> new_v;
    std::back_insert_iterator< std::vector<int>> back_it(new_v);
    std::unique_copy(data.begin(), data.end(), back_it);

   std::vector<int> new_v1;
    std::back_insert_iterator< std::vector<int>> back_it(new_v1);
    std::remove_copy(data.begin(), data.end(), back_it, -18);
}
```

Минимумы и максимумы

Алгоритм	Назначение
min()	меньшее из двух
max()	большее из двух
minmax()	возвращает 2 значения в порядке от меньшего к большему
min_element()	минимальное значение в последовательности
max_element()	максимальное значение в последовательности
minmax_element()	пара из минимального и максимального значений

```
void f(){
  std::vector<int> data = {1, 8, -90, 4, 7, -18, 11, 0, 4, -7, 4};
  std::cout << *std::max_element(data.begin(), data.end()) << "\n";
  std::cout << *std::min_element(data.begin(), data.end());
}
```

Алгоритмы сортировки

Алгоритм	Назначение
sort()	быстрая сортировка (QuickSort)
partial_sort()	сортирует элементы в части последовательности
stable_sort()	при сортировке сохраняет порядок повторяющихся элементов
is_sorted()	проверяет, отсортирован ли диапазон

```
void f(){
  std::vector<int> vec = {1, 8, -90, 4, 7, -18, 11, 0, 4, -7, 4};
   std::sort(vec.begin(), vec.end());
   std::stable_sort(vec.begin(), vec.end());
   std::partial_sort(vec.begin(), vec.begin() + 3, vec.end());
}
```

Другие алгоритмы

- алгоритмы над множествами
- алгоритмы разделения
- алгоритмы двоичного поиска
- операции над множествами
- операции над кучей
- числовые операции

Вопросы?