Стандартная библиотека шаблонов (STL)

19 июня 2017 г.

Основные компоненты

- **Контейнеры** (хранят совокупность объектов)
- **Итераторы** (обеспечивают доступ к содержимому контейнера)
- **Алгоритмы** (обрабатывают данные)
- **Адаптеры** (адаптируют компоненты к разным интерфейсам)
- **Функторы** (скрывают функцию в объекте)

Контейнеры STL

Контейнер – набор однотипных элементов, которые хранятся определенным образом

- vector
- list
- map
- set
- ...

```
#include <vector>
#include <list>
#include <map>
#include <set>

std::vector<int> myVector;
std::list<double> myList;
std::map<std::string, int> myMap;
std::set<std::string> mySet;
```

http://en.cppreference.com/w/cpp/container

Контейнеры

последовательные	• std::array (C++11), std::vector, std::deque, std::list, std::forward_list (C++11)
ассоциативные	• std::map, std::set std::multimap, std::multiset
неупорядоченные ассоциативные	• std::unordered_map (C++11), std::unordered_set (C++11), std::unordered_multimap (C++11), std::unordered_multiset (C++11)
адаптеры	• std::stack, std::queue, std::priority_queue

std::vector

- элементы хранятся последовательно
- быстрый доступ к элементу по индексу
- добавление в начало и конец – O(1)

Добавление в середину, поиск элемента – O(n)

std::vector: основные методы

operator[]	обращение по индексу
at()	обращение по индексу (с проверкой)
front() / back()	ссылка на первый / последний элемент
push_back()	добавление элемента в конец
pop_back()	удаление последнего элемента
clear()	удаление всех элементов
empty()	проверка на пустоту
size()	получение размера вектора
capacity()	получение емкости вектора
shrink_to_fit	освобожение неиспользуемой памяти

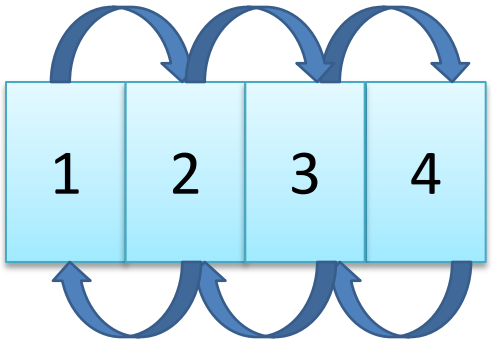
```
#include <iostream>
#include <vector>

void f(){
    std::vector<int> myVec;
    for (int i = 0; i <= 10; ++i){
        myVec.push_back(i);
    }

    myVec.pop_back();
    std::cout  << myVec.size() << std::endl;
    std::cout  << myVec.capacity() << std::endl;
    myVec.clear();
    std::cout  << myVec.size() << std::endl;
    std::cout  << myVec.capacity() << std::endl;
}
```

std::list

- двусвязный список
- элементы хранятся в разных областях памяти
- добавление, удаление в любой позиции – O(1)



Поиск элемента – O(n), нет обращения по индексу

std::list: основные методы

front()	доступ к первому элементу
back()	доступ к последнему элементу
push_back() и **pop_back()**	работа с концом списка
push_front() и **pop_front()**	работа с началом списка
insert()	вставка элемента
remove()	удаление элемента по значению
erase()	удаление элемента по позиции
clear()	удаление всех элементов
empty()	проверка на пустоту
size()	получение размера списка

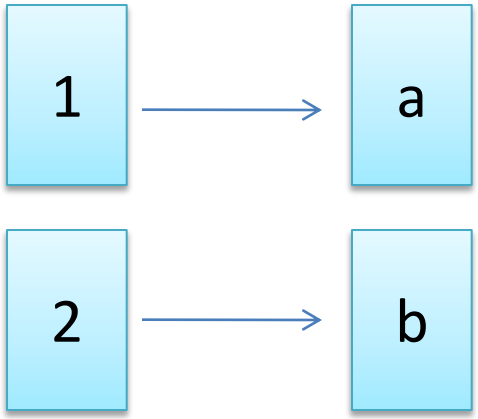
```
#include <iostream>
#include <list>

void f(){
    std::list<int> myList;
    for (int i = 10; i >= 0; --i){
        myList.push_back(i);
    }

    myList.sort();
    std::list<int>::iterator it = myList.begin();
    ++it;
    myList.insert(it, 6);
    myList.remove(1);
    myList.sort();
    myList.unique();
}
```

std::map

- отсортированный контейнер
- хранит пары «ключ – значение»
- все ключи уникальны и упорядочены
- поиск по ключу, вставка, удаление – O(log n)



std::map: основные методы

operator[ ]	ссылка на элемент (если ключа не было, то вставляет его)
count()	проверка на вхождение ключа (1, 0)
find()	получение итератора для заданного ключа
insert()	вставка пары ключ / значение
clear()	удаление всех элементов
erase()	удаление элемента по итератору или по ключу
size()	получение количества элементов
empty()	проверка на пустоту

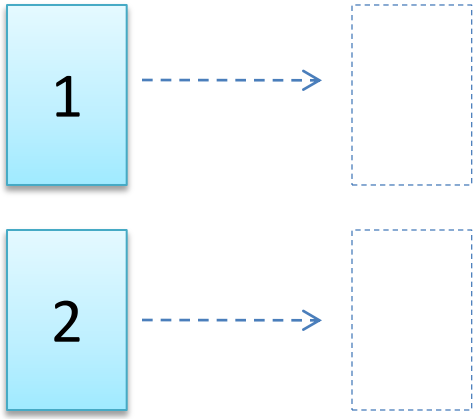
```cpp
#include <iostream>
#include <map>

void f(){
    std::map<std::string, int> myMap;
    myMap["int"] = 4;
    myMap["float"] = 4;
    myMap["double"] = 8;

    if (myMap.count("char") == 0){
        myMap["char"] = 1;
    }
    for (auto it = myMap.begin(); it != myMap.end(); ++it){
        std::cout << it->first << " " << it->second << std::endl;
    }
}
```

std::set

- хранит только ключи
- все ключи уникальны и упорядочены
- поиск ключа, вставка, удаление – O(log n)



std::set: основные методы

empty()	проверка на пустоту
size()	получение количества элементов
count()	проверка на вхождение ключа (1, 0)
find()	получение итератора для заданного ключа
insert()	вставка элемента
clear()	удаление всех элементов
erase()	удаление элементов по итератору или по ключу
swap()	обменивает содержимое

```
#include <iostream>
#include <set>

void f(){
    std::set<std::string> mySet;
    mySet.insert("int");
    mySet.insert("double");
    mySet.insert("float");

    if (mySet.count("char") == 0){
        mySet.insert("char");
    }
    for(auto it = mySet.begin(); it != mySet.end(); ++it){
        std::cout << *it << std::endl;
    }
}
```

Quiz

Какой контейнер выбрать?

- список посещенных городов
- студенты с их средними баллами
- плейлист проигрывателя
- телефонная книга
- список книг в домашней библиотеке
  (вы носите его с собой, чтобы не купить книгу, которая у вас уже есть)
- список друзей, которых вы пригласите на ДР

Итераторы

- обеспечивают доступ к элементам контейнера и перемещение от одного элемента к другому
- похожи на указатели (все указатели являются по своей сути итераторами)
- синтаксис похож на синтаксис указателей (доступны операции *, ->, ++ и др.)

Виды итераторов

- **входные** (доступ для чтения данных из контейнера в одном направлении)
- **выходные** (доступ для записи данных в контейнер в одном направлении)
- **однонаправленные** (доступ для записи и чтения в одном напр.)
- **двунаправленные** (доступ к следующему и предыдущему элементам)
- **произвольного доступа** (доступ к любому элементу по индексу)

Операции на итераторах

Входные	Выходные	Однонаправленные	Двунаправленные	Произвольного доступа
operator++	operator++	operator++	operator++	operator++
			operator--	operator--
operator*	operator*	operator*	operator*	operator*
operator->		operator->	operator->	operator->
operator==		operator==	operator==	все операторы
operator!=		operator!=	operator!=	сравнения
operator=	operator=	operator=	operator=	operator=
				operator+
				operator-
				operator+=
				operator-=
				operator[]

Итераторы (пример)

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

void f(){
    std::vector<int> vec = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    std::vector<int>::iterator it;
    for (it = vec.begin(); it != vec.end(); ++it){
            std::cout << *it << std::endl;
    }
    std::vector<int>::reverse_iterator r_it;
    for (r_it = vec.rbegin(); r_it != vec.rend(); ++r_it){
            std::cout << *r_it << std::endl;
    }
}
```

Специальные функции

```
std::advance(итератор, сдвиг);
std::distance(один_итератор, другой_итератор);
```

- advance() – увеличивает итератор на заданную величину
- distance() – возвращает количество элементов между двумя итераторами

Специальные функции (пример)

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

void f(){
    std::vector<int> vec = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    std::vector<int>::iterator it = vec.begin();
    std::vector<int>::iterator it_end = vec.end();

    int size = std::distance(it, it_end);
    std::cout << size << std::endl;          // 10

    std::advance(it, 3);
    std::cout << *it << std::endl;           // 4
}
```

Вопросы?