© 2012

Использование ассоциативных контейнеров предполагает, что в хранимых данных выделяется ключ, который определяет конкретный элемент хранимых данных, и неключевая информация. Ассоциативные контейнеры обеспечивают быстрый доступ к данным по значениям ключа.

Основные операции для ассоциативных контейнеров:

*• вставка элемента (трудоёмкость O(log N));*

*• удаление элемента (трудоёмкость O(log N));*

• поиск элемента по ключу (трудоёмкость O(log N));

• последовательный просмотр (в порядке возрастания ключей, независимо от порядка вставок).

map обеспечивает хранение данных в формате

«ключ-значение» с уникальным значением ключа multimap обеспечивает хранение данных в формате

«ключ-значение» с повторяющимися значениями ключа set обеспечивает хранение только ключевых

данных с уникальным значением ключа multiset обеспечивает хранение только ключевых

данных с повторяющимися значениями ключа bitset предназначен для хранения битовых

последовательностей заданной длины

Ассоциативные контейнеры описаны в заголовочных файлах <mар> и <set>.

**#include <utility> template <typename T1, typename T2> class pair {**

T1 first; T2 second; ... }; Операции сравнения: <, == p1 < р2, если p1.first < р2.first или p1.first == р2.first && p1.second < р2.second

template <typename T1, typename T2> pair <T1, T2> make\_pair (T1 a, T2 b);

**#include <iostream> #include <utility> using namespace std; void main() {**

**pair <int, double> p1(10, 12.3), p2(p1); p2 = make\_pair(20, 12.3); // Эквивалентно p2 = pair <int, double >(20, 12.3)**

**cout << "p1: " << p1.first << " " << p1.second << endl; cout << "p2: " << p2.first << " " << p2.second << endl; p2.first -= 10; if (p1 == p2)**

**cout << "p1 == p2\n"; p1.second -= 1; if (p2 > p1)**

*cout << "p2 > pl\n"; } Результат: р1: 10 12.3 p2: 20 12.3 p1 == p2 p2 > p1*

Класс map является шаблоном, зависящим от двух или трех параметров:

►

типа ключа;

►

типа неключевых данных;

►

функционального класса, определяющего правила сравнения ключей (если для ключевого класса определена стандартная операция «меньше», третий параметр указывать не обязательно).

**#include <map> using namespace std;**

*map <int, string> m1; // ключ – целое число, неключевые данные – строки*

*map <int, string, greater <int> > m2; // использование стандартного функционального // класса greater <int> позволяет сортировать по // убыванию*

*map <int, pair <string, int> > m3; // неключевые данные – структура из двух полей*

**typedef map <int, string, greater <int>> m2; typedef map <int, string, greater <int>>::iterator it\_m2;**

**m2 M2;**

**void WriteMap(m2 L) {**

**cout << L.size() << ":\n"; for (it\_m2 i = L.begin(); i!=L.end(); i++) {**

**cout << i->first<< " "; cout.write((i->second).c\_str(),(i->second).size()); // cout << i->second.c\_str();**

**cout << "\n"; } cout << endl; }**

int main() {

M2.insert(make\_pair(15, "fifteen")); M2.insert(make\_pair(10, "ten")); M2.insert(make\_pair(50, "fifty")); WriteMap(M2); return 0; }

Результат: 3: 50 fifty 15 fifteen 10 ten

**T& operator[](const Key&)**

**typedef map <string, string> phonebook;**

***... phonebook Phb; Phb["kash"] = "123456789"; // вставляем новый элемент с ключом ”kash” // и номером телефона ”123456789” cout << Phb["kash"] << endl; // результат – строка ”123456789” cout << Phb["pupkin"] << endl; // результат – пустая строка, т.к. ключ не найден. // Однако в книгу будет вставлен новый элемент!***

**for (phonebook::iterator i=Phb.begin();**

**i!=Phb.end(); i++) cout << (\*i).first << ":" << (\*i).second << endl;**

***// прямой порядок***

**for (phonebook::reverse\_iterator i=Phb.rbegin();**

**i!=Phb.rend(); i++) cout << (\*i).first << ":" << (\*i).second << endl;**

***// обратный порядок***

*Поиск данных*

iterator find (const key\_type& ключ)

Возвращает элемент, ключ итератор которого на

равен заданному, или end() iterator upper\_bound (const key\_type& ключ)

Возвращает элемент, ключ итератор которого на

больше заданного, или end() iterator lower\_bound (const key\_type& ключ)

Возвращает элемент, меньше заданного, ключ итератор которого или на

end() не

pair <iterator, iterator> equal\_range (const key\_type& ключ)

Возвращает (lower\_bound, для интервал, элементы заданного с включающий заданным пару

upper\_bound) ключа (т. ключом)

все е.

*Вставка данных*

pair <iterator, bool> insert (value\_type& элемент)

Вставляет Поле содержит вставке случае, second а и поле false true новый результата

при first – в элемент.

противном успешной

– итератор на добавленный элемент template <class Iter> void insert (Iter первый, Iter граница)

Вставляет элементы из другого контейнера

iterator insert (iterator позиция, value\_type& элемент)

Вставляет Параметр показывает, словаря предполагается

«позиция» новый с какой элемент.

позиции

искать место для вставки

*Удаление данных*

void erase(iterator позиция)

Удаляет элемент по значению итератора.

size\_type erase(const key\_type& ключ)

Удаляет значению возвращает элемент ключа;

количество по

удаленных элементов void erase (iterator первый, iterator граница)

Удаляет интервал элементов

Найти 1.

телефон по фамилии или выдать сообщение об отсутствии данных.

**string N1; cout << "Enter name: "; cin >> N1; phonebook::iterator i=Phb.find(N1); cout << (i==Phb.end() ? "Record not found" :**

**(\*i).second) << endl;**

Выдать 2.

содержимое телефонной книги с именами, начинающимися на ”k”.

**for (phonebook::iterator i=Phb.lower\_bound("k");**

**i!=Phb.lower\_bound("l"); i++) cout << i->first << ": " << (\*i).second << endl;**

Вставить 3.

новый элемент по паре «ключ – неключевые данные».

**Phb.insert(phonebook::value\_type("kirill",**

**"2353555")); или**

**Phb.insert(make\_pair("kirill", "2353555"));**

Удалить 4.

из телефонной книги все записи с пустыми номерами телефонов

**phonebook::iterator i, j; for (i=Phb.begin(); i != Phb.end();) {**

**j=i; i++; if ((\*j).second.length()==0) {**

**cout << "Deleting: " << (\*j).first << endl; Phb.erase(j); } }**

Операция доступа по индексу запрещена, и вместо нее надо пользоваться итераторами.

Пусть phonebook описан как multimap:

**typedef multimap <string, string> phonebook; typedef pair <phonebook::iterator,**

**phonebook::iterator> phinterval;**

Найти 5.

данные для всех телефонов, записанных с именем ”kash”.

**phonebook Phb; phinterval Ph\_int; phonebook::iterator i;**

**Ph\_int = Phb.equal\_range("kash");**

**for (i=Ph\_int.first; i != Ph\_int.second; i++)**

**cout << (\*i).first << ":" << (\*i).second << endl;**

Исправить 6.

данные для телефонов, записанных с именем ”kash”.

**Phint = Phb.equal\_range("kash"); for (i=Phint.first; i != Phint.second; i++) if ((\*i).second =="kash@telegraph.by"){**

**(\*i).second = "kash@telegraf.by"; break; }**

**typedef set<int, less<int> > set\_i; // greater<int> set\_i::iterator i;**

**void main() {**

**int a[4] = {4, 2, 1, 2}; set\_i s1; // Создается пустое множество // Множество создается копированием массива set\_i s2(a, a + 4); set\_i s3(s2); // Работает конструктор копирования s2.insert(10); // Вставка элементов s2.insert(6); for ( i = s2.begin(); i != s2.end(); ++i) // Вывод**

**cout << \*i << " "; cout << endl;**

**// Переменная для хранения результата equal\_range:**

**pair <set\_i::iterator, set\_i::iterator > p;**

**p = s2.equal\_range(2); cout << \*(p.first)<< " " << \*(p.second) << endl;**

**p = s2.equal\_range(5); cout << \*(p.first)<< " " << \*(p.second) << endl; }**

*Результат работы программы:*

*1 2 4 6 10 2 4 6 6*

Спасибо за внимание

24