

Тема 6

Ассоциативные контейнеры

Ассоциативные контейнеры

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 (*трудоемкость $O(\log N)$*);
- удаление элемента (*трудоемкость $O(\log N)$*);
- поиск элемента по ключу (*трудоемкость $O(\log N)$*);
- последовательный просмотр (в порядке возрастания ключей, независимо от порядка вставок).

Типы ассоциативных контейнеров

<code>map</code>	обеспечивает хранение данных в формате «ключ-значение» с уникальным значением ключа
<code>multimap</code>	обеспечивает хранение данных в формате «ключ-значение» с повторяющимися значениями ключа
<code>set</code>	обеспечивает хранение только ключевых данных с уникальным значением ключа
<code>multiset</code>	обеспечивает хранение только ключевых данных с повторяющимися значениями ключа
<code>bitset</code>	предназначен для хранения битовых последовательностей заданной длины

Ассоциативные контейнеры описаны в заголовочных файлах `<map>` и `<set>`.

Вспомогательный класс pair

```
#include <utility>

template <typename T1, typename T2> class pair {
    T1 first;
    T2 second;
    ... };
```

Операции сравнения: <, ==

p1 < p2, если

p1.first < p2.first или

p1.first == p2.first && p1.second < p2.second

```
template <typename T1, typename T2>
pair <T1, T2> make_pair (T1 a, T2 b);
```

Пример формирования пар


```
#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 > p1\n";
}
```

Результат:

```
p1: 10 12.3
p2: 20 12.3
p1 == p2
p2 > p1
```

Работа с контейнером *map*

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).
- 

Примеры описаний контейнера map

```
#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
```

Пример работы с контейнером map

```
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;  
}
```


Пример работы с контейнером map (продолжение)

```
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

Телефонная книга

```
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["rupkin"] << endl;
```

```
// результат - пустая строка, т.к. ключ не найден.
```

```
// Однако в книгу будет вставлен новый элемент!
```

Телефонная книга (продолжение)

```
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



Методы класса map

Поиск данных

<code>iterator find</code> <code>(const key_type& <i>КЛЮЧ</i>)</code>	Возвращает итератор на элемент, ключ которого равен заданному, или <code>end()</code>
<code>iterator upper_bound</code> <code>(const key_type& <i>КЛЮЧ</i>)</code>	Возвращает итератор на элемент, ключ которого больше заданного, или <code>end()</code>
<code>iterator lower_bound</code> <code>(const key_type& <i>КЛЮЧ</i>)</code>	Возвращает итератор на элемент, ключ которого не меньше заданного, или <code>end()</code>
<code>pair <iterator,</code> <code>iterator> equal_range</code> <code>(const key_type& <i>КЛЮЧ</i>)</code>	Возвращает пару <code>(lower_bound, upper_bound)</code> для заданного ключа (т. е. интервал, включающий все элементы с заданным ключом)

Методы класса map

Вставка данных

```
pair <iterator, bool>  
insert (value_type&  
элемент)
```

Вставляет новый элемент. Поле second результата содержит true при успешной вставке и false – в противном случае, а поле first – 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 новый элемент. Параметр «позиция» показывает, с какой позиции словаря предполагается искать место для вставки

Методы класса map

Удаление данных	
void erase(iterator позиция)	Удаляет элемент по значению итератора.
size_type erase(const key_type& ключ)	Удаляет элемент по значению ключа; возвращает количество удаленных элементов
void erase (iterator первый, iterator граница)	Удаляет интервал элементов

Примеры работы с телефонной книгой

1. Найти 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;
```

Примеры работы с телефонной книгой

2. Выдать 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;
```



Примеры работы с телефонной книгой

3. Вставить новый элемент по паре «ключ – неключевые данные».

```
Phb.insert(phonebook::value_type("kirill",  
                                   "2353555"));
```

ИЛИ

```
Phb.insert(make_pair("kirill", "2353555"));
```



Примеры работы с телефонной книгой

4. Удалить 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);
    }
}
```

Особенности работы с контейнером multimap

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;
```

Примеры работы с телефонной книгой

5. Найти 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;
```

Примеры работы с телефонной книгой

6. Исправить 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;  
    }
```

Пример работы с множеством (set)

```
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;
```

Пример работы с множеством (set) (продолжение)

```
// 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