

Технология ООП

Санкт-Петербургский государственный политехнический университет

11 октября 2011

```
#include <set>
template<class _Key,
class _Pr = less<_Key>,
class _Alloc = allocator<_Key> >
class set : public _Tree<_Tset_traits<_Kty, _Pr,
    _Alloc, false> >
{ // ordered red-black tree of key values, unique
  keys
```

Специфика set (множества)

- Данные всегда хранятся в упорядоченном виде, следовательно поиск происходит очень быстро (с логарифмической сложностью).
- Является самым простым из ассоциативных контейнеров, так как ключ совпадает со значением.
- Базовым классом является двоичное сбалансированное дерево.
- Значения в множестве уникальны, следовательно дубли игнорируются.
- Существует фиктивный элемент.
- Отсутствует операция `push_back()`, так как заполняется в соответствии с критерием упорядоченности, только методом `insert()`.

Пример задания множества

```
set <int> s; //???  
s.insert(10);  
s.insert(2);  
s.insert(30);  
s.insert(20);  
s.insert(1);  
s.insert(11);  
s.insert(10); //???  
//Распечатать значения элементов
```

- Сложный итератор, использующий ++ (—) умеет перемещаться на узел с бОльшим значением, выбирая наименьшее из поддеревьев
- Итератор предназначен только для чтения.
- Существует реверсивный итератор.

```
set<int> s;  
s.insert(10);  
...  
set<int>::iterator it=s.begin();  
*it = 33; //???
```

//Распечатать set в обратном порядке

Пример перестроения дерева

Если дерево «вырождается» в не сбалансированное, то оно перестраивается, изменяя корень (root).

```
set<int> s;  
s.insert(10);  
s.insert(9);  
s.insert(8); //дерево перестраивается  
s.insert(7);
```

Примеры создания set

```
int ar1[5] = {1,2,3,4,5};
set<int> s1(ar1, ar1+5);
//печать элементов
int ar2[5] = {5,4,3,2,1};
set<int> s2(ar2, ar2+5);
//печать элементов

vector<int> v;
//формирование значений v
set<int> s1(v.begin(), v.end());
//печать элементов s1
set<int> s2(v.rbegin(), v.rend());
//печать элементов s2
int ar1[10] = {1,2,3,4,5,1,3,5};
set<int> s(ar1, ar1+10); //???
//печать элементов s
```

Изменение критерия упорядоченности в set

```
int ar1[10] = {1,2,3,4,5};  
set<int, greater<int>> s(ar1, ar1+10); //печать  
      элементов  
set<int, greater<int>> ::iterator it = s.begin();  
  
struct Abs { //Пользовательский критерий  
      bool operator() (int x, int y) {  
              return abs(x)<abs(y);  
      }  
};  
int ar1[10] = {5, -1, -3, 2, 10};  
set<int, Abs> s(ar1, ar1+10); //печать элементов  
set<int, Abs>::iterator it = s.begin();
```

Пример упорядоченного чтения строк

```
ifstream f("my.txt");
string word;
set<string> words;
while(f >> word, !f.eof())
{
    words.insert(word);
}
f.close();
```

Специфические методы set

```
iterator lower_bound(const Key& _Key); //возвращает  
    итератор на элемент, значение которого >=_Key  
iterator upper_bound(const Key& _Key); //возвращает  
    итератор на элемент, значение которого >_Key  
pair <iterator, iterator> equal_range (const Key&  
    _Key); //возвращает пару итераторов
```

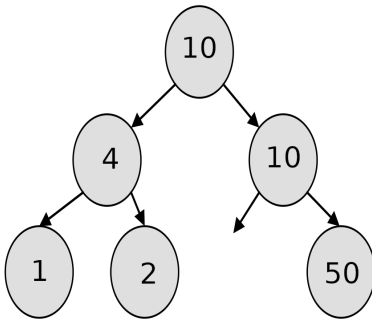
```
set<int> s; //10, 20, 25, 30  
it1 = s.lower_bound(15); //???  
it2 = s.lower_bound(30); //???  
it3 = s.upper_bound(15); //???  
it4 = s.upper_bound(30); //???  
//Распечатать от it1 до it2  
//Стереть от it3 до it4
```

```
#include <set>
template<class _Key,
class _Pr = less<_Key>,
class _Alloc = allocator<_Key> >
class multiset
: public _Tree<_Tset_traits<_Key, _Pr, _Alloc, true
> >
{// ordered red-black tree of key values, non-
unique keys
```

Специфика multiset (множество с повторениями)

- Допускает дублирование значений.
- Метод `find` возвращает итератор на первый элемент с искомым значением.

```
multiset <int> s;  
s.insert(10); s.insert(2);  
s.insert(10); s.insert(4);  
s.insert(1); s.insert(50);
```



```
#include <map>
template<class _Key,
class _Val,
class _Pr = less<_Key>,
class _Alloc = allocator<pair<const _Key, _Val> > >
class map
: public _Tree<_Tmap_traits<_Key, _Val, _Pr, _Alloc
, false> >
{// ordered red-black tree of {key, mapped} values,
    unique keys
```

Специфика map (словарь)

- Ассоциативный контейнер, в котором значение не совпадает с ключом.
- Ключи уникальны, следовательно для пары с существующим ключом модифицируется значение.
- Словарь отсортирован по ключам и критерий сравнения задается для ключей.
- Для хранения пары ключ/значение используется шаблон структуры pair.

Структура pair определена в <utility>.

```
template<typename T1, typename T2> struct pair {
    T1 first;
    T2 second;
    pair(const T1& t1, const T2& t2): first(t1),
        second(t2) { }
};
int main()
{
    pair<char, double> p('A', 1.1); //ключ/
        значение
    p.make_pair('B', 2.2);
    p.first = 'C';
    p.second = 3.3;
}
```

Псевдоним value_type в контейнере map сопоставляется pair<const Key, T>.

Методы для работы со значениями map

- `operator []` — оператор записи или доступа к значению по ключу. Если элемент отсутствовал, то он будет добавлен.
- Метод `insert()` — принимает пару ключ/значение, а возвращает пару (итератор, `bool`), где `bool` содержит `true`, если пара занесена.
- Методы `find()` и `erase()` осуществляют операции по указанному ключу, выполняют поиск и удаление значения с заданным ключом.

Пример использования map

```
map<string , int> book;  
book[string("Иванов")] = 1111111;  
pair< map<string , int>::iterator , bool> r = book.  
    insert(pair<string , int>("Петров", 2222222));  
//Печать записной книжки
```

Пример использования find в map

```
map<string, int>::iterator it = book.find(string("
    Петров"));
//проверка – если существует?

*it.second = 3333333; //???
*it.first = string("Сидоров"); //???
```

```
#include <map>
template<class _Key,
class _Val,
class _Pr = less<_Key>,
class _Alloc = allocator<pair<const _Key, _Val> > >
class multimap
: public _Tree<_Tmap_traits<_Key, _Val, _Pr, _Alloc
, true> >
{// ordered red-black tree of {key, mapped} values,
  non-unique keys
```

Специфика multimap (словарь с повторениями)

- Позволяет дублировать значения ключей, следовательно не определен operator[]. Элементы с одинаковыми ключами в контейнере хранятся в порядке занесения.
- Добавление элементов осуществляется только через insert().
- При удалении по ключу через erase() — удаляются все элементы, удовлетворяющие условию.
- Метод count() — возвращает количество пар, в которых ключ совпадает с указанным значением.