- 首先，为了初始化（或赋值或安插）元素，你必须传递 key/value pair，本例以嵌套式初值列（nested initializer list）完成，内层定义的是每个元素的 key 和 value，外层定义出所有元素。因此 {5,"tagged"} 具体指明被安插的第一个元素内容。
- 处理元素时你再一次需要和 key/value pair 打交道。每个元素的类型实际上是 pair<const *key*,*value*>（5.1.1 节第 60 页介绍过 pair 类型）。Key 之所以必须是常量，因为其内容如果被改动，会破坏元素的次序，而元素次序是由容器自动排序的。由于 pair 缺乏 output 操作符，你无法视它们为一个整体加以打印。因此你必须分别处理 pair 结构中的成员，它们分别名为 first 和 second。

  下面的式子取得 key/value pair 的第二成分，也就是 multimap 元素的 value：

```
elem.second
```

  下面的式子取得 key/value pair 的第一成分，也就是 multimap 元素的 key：

```
elem.first
```

最终，程序输出如下：

```
this is a multimap of tagged strings
```

C++11 之前并未明确规定等效元素（equivalent element，也就是 key 相同的元素）的排列次序。所以在 C++11 之前，"this" 和 "is" 的排列有可能与本书所显示的不同。但是 C++11 已经保证，新插入元素会被安插在 multiset 和 multimap 已有之等效元素的末尾。而且如果调用 insert()、emplace() 或 erase()，这些等效元素的次序也保证稳定不变。

#### 关联式容器的其他例子

6.2.4 节第 185 页有一个示例，使用 map 做成一个所谓的关联式数组（associative array）。

7.7 节详细讨论 set 和 multiset，附有更多例子。7.8 节详细讨论 map 和 multimap，也带有更多例子。

Multimap 也可以用作字典（dictionary）。 7.8.5 节第 348 页有一个例子。

### 6.2.3 无序容器（Unordered Container）

在无序（unordered）容器中，元素没有明确的排列次序。也就是如果安插 3 个元素，当你迭代容器内的所有元素时会发现，它们的次序有各种可能。如果安插第 4 个元素，先前 3 个元素的相对次序可能会被改变。我们唯一关心的是，某个特定元素是否位于容器内。甚至如果你有 2 个这种容器，其内有着完全相同的元素，元素的排列次序也可能不同。试着想象它是个袋子（bag）。

无序（unordered）容器常以 hash table 实现出来（如图6.3所示），内部结构是一个“由 linked list 组成”的 array。通过某个 *hash* 函数的运算，确定元素落于这个 array 的位置。Hash 函数运算的目标是：让每个元素的落点（位置）有助于用户快速访问

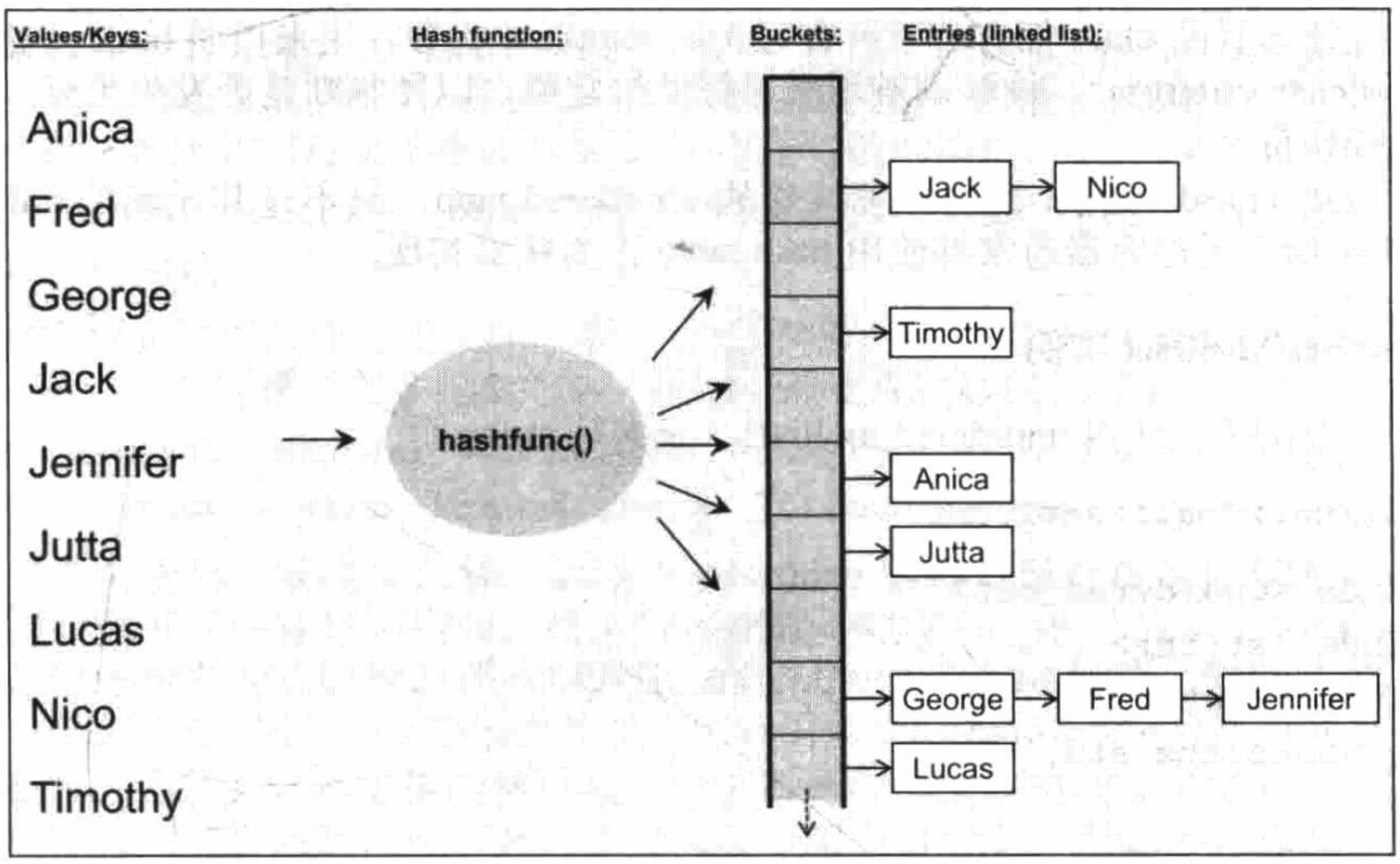


图 6.3 无序（Unordered）容器就是 Hash Table

任何一个元素，前提则是 hash 函数本身也必须够快。由于这样一个快速而完美的 hash 函数不一定存在（或不一定被你找到），抑或由于它造成 array 耗费巨额内存而显得不切实际，因此，退而求其次的 hash 函数有可能让多个元素落于同一位置上。所以设计上就让 array 的元素再被放进一个 linked list 中，如此一来 array 的每个位置（落点）就得以存放一个以上的元素。

无序（unordered）容器的主要优点是，当你打算查找一个带某特定值的元素，其速度甚至可能快过关联式容器。事实上无序容器提供的是摊提的常量复杂度（amortized constant complexity），前提是你有一个良好的 hash 函数。然而提供一个良好的 hash 函数并非易事（见 7.9.2 节第 363 页），你可能需要提供许多内存作为 bucket。

根据关联式容器的分类法，STL 定义出下面这些无序容器：

- **Unordered set** 是无序元素的集合，其中每个元素只可出现一次。也就是不允许元素重复。
- **Unordered multiset** 和 unordered set 的唯一差别是它允许元素重复。也就是 unordered multiset 可能内含多个有着相同 value 的元素。
- **Unordered map** 的元素都是 key/value pair。每个 key 只可出现一次，不允许重复。它也可以用作关联式数组（associative array），那是“索引可为任意类型”的 array（详见 6.2.4 节第 185 页）。
- **Unordered multimap** 和 unordered map 的唯一差别是允许重复。也就是 unordered multimap 可能内含多个“拥有相同 key”的元素。它可以用作字典（dictionary）（7.9.7 节第 383 页有一个例子）。

所有这些无序容器的 class 都有若干可有可无的 template 实参，用来指明 hash 函数和等效准则（equivalence criterion），该准则被用来寻找某给定值，以便判断是否发生重复。默认的等效准则是操作符 ==。

你可以把 unordered set 视为一种特殊的 unordered map，只不过其元素的 value 等同于 key。现实中所有无序容器通常都使用 hash table 作为底层实现。

### Unordered Set/Multiset 实例

下面是第一个例子，使用 unordered multiset，元素是 string：

```
// stl/unordmultiset1.cpp

#include <unordered_set>
#include <string>
#include <iostream>
using namespace std;

int main()
{
    unordered_multiset<string> cities {
        "Braunschweig", "Hanover", "Frankfurt", "New York",
        "Chicago", "Toronto", "Paris", "Frankfurt"
    };

    // print each element:
    for (const auto& elem : cities) {
        cout << elem << "  ";
    }
    cout << endl;

    // insert additional values:
    cities.insert( {"London", "Munich", "Hanover", "Braunschweig"} );

    // print each element:
    for (const auto& elem : cities) {
        cout << elem << "  ";
    }
    cout << endl;
}
```

包含必要的头文件

```
#include <unordered_set>
```

后，我们可以声明一个“元素为 string”的 unordered set 并给予初值：

```
unordered_multiset<string> cities { ... };
```

现在，如果打印所有元素，出现的次序可能不同于程序中所给的次序，因为其次序是不明确的。唯一保证的是重复元素——这的确有可能因为我们用的是 *multi*set——以其安插次序被组合在一起（因此其相对次序永远不变）。下面是可能的输出：

```
Paris Toronto Chicago New York Frankfurt Frankfurt Hanover
  Braunschweig
```

任何安插动作都有可能改变上述次序。事实上任何操作只要可能引发 rehashing 就可能改变上述次序。所以，在安插了更多元素之后，输出可能变成这样：

```
London Hanover Hanover Frankfurt Frankfurt New York Chicago
  Munich Braunschweig Braunschweig Toronto Paris
```

次序究竟会不会变化，或变成怎样，取决于 rehashing 策略，而它在某种程度上可被程序员影响。例如你可以保留足够空间，使得直到出现一定量的元素才发生 rehashing。此外，为确保你在处理所有元素的同时还可以删除元素，C++ standard 保证，删除元素不会引发 rehashing。但是删除之后的一次安插动作就有可能引发 rehashing。详见 7.9 节第 355 页。

一般而言，关联式容器提供的接口和无序容器相同，只有声明式可能不同，而且无序容器还提供特殊的成员函数，可影响内部行为或检阅当前状态。因此本处的例子只有头文件和类型不同于“使用寻常 multiset”的例子，后者出现于 6.2.2 节第 178 页。

再次提醒，在 C++11 之前，你必须使用迭代器访问元素，6.3.1 节第 193 页有一个例子。

### Unordered Map 和 Multimap 实例

出现于第 179 页的例子是针对 multimap 设计的，却也适用于 unordered multimap，只要你在 include 指示符中以 `unordered_map` 替换 `map` 并在容器声明式中以 `unordered_multimap` 替换 `multimap`：

```
#include <unordered_map>
...
unordered_multimap<int,string> coll;
...
```

唯一的不同是，本例的元素次序不明确。然而在大多数平台上，元素仍会被排序，因为默认是以 modulo 操作符作为 hash 函数。把“排序后的次序”（sorted order）视为“不明确次序”（undefined order）当然合法。不过上述的“排序”现象并不保证一定会有，而且如果你添加更多元素，元素的次序也将可能不同。

下面是另一个 unordered map 例子。此例使用的 unordered map，其 key 是个 string 而其 value 是个 double：

```
// stl/unordmap1.cpp

#include <unordered_map>
#include <string>
```

```
#include <iostream>
using namespace std;

int main()
{
    unordered_map<string,double> coll { { "tim", 9.9 },
                                        { "struppi", 11.77 }
                                      };

    // square the value of each element:
    for (pair<const string,double>& elem : coll) {
        elem.second *= elem.second;
    }

    // print each element (key and value):
    for (const auto& elem : coll) {
        cout << elem.first << ": " << elem.second << endl;
    }
}
```

包含必要的 map、 string 和 iostream 头文件后，我们声明一个 unordered map，并以两个元素作为初始元素。这里运用了嵌套式初值列（nested initializer list），所以

```
{ "tim", 9.9 }
```

和

```
{ "struppi", 11.77 }
```

是被用来初始化 map 的两个元素。

接下来，对每个元素的 value 执行平方运算：

```
for (pair<const string,double>& elem : coll) {
    elem.second *= elem.second;
}
```

我要再一次提示你，看得出来元素类型是由 constant string 和 double 组成的 pair<>（见 5.1.1 节第 60 页）。因此我们无法改动元素的 key，也就是其 first 成员：

```
for (pair<const string,double>& elem : coll) {
    elem.first = ...;      // ERROR: keys of a map are constant
}
```

像此前很多例子一样，自 C++11 开始，我们不再需要明白指出元素类型，因为在一个 range-based for 循环内，类型可被推导出来（根据容器）。基于此，负责输出所有元素的第二循环中使用 auto。事实上，由于声明 elem 为 const auto&，我们得以避免产生很多拷贝（copy）：

```
for (const auto& elem : coll) {
    cout << elem.first << ": " << elem.second << endl;
}
```

这个程序的一个可能的输出是：

```
struppi: 138.533
tim: 98.01
```

但不保证如此，因为实际次序不明确（无序）。如果我们改用一个寻常的 map，就能保证“带着 key "struppi"” 的元素必定在“带着 key "tim"” 的元素之前，因为 map 会以 key 为根据对元素排序，而 "struppi" 小于 "tim"。7.8.5 节第 345 页有另一个例子，使用 map 并以 STL 算法和 lambda 取代 range-based for 循环。

### Unordered 容器的其他例子

所有无序（unordered）容器都提供若干可有可无的 template 实参，用来指明诸如 hash 函数和等效比较式（equivalence comparison）。标准库为基础类型和 string 准备了一个默认的 hash 函数，至于其他类型，我们必须提供自己的 hash 函数。这部分将在 7.9.2 节第 363 页讨论。

下一节有个例子使用 map 作为所谓的关联式数组（associative array）。7.9 节详细讨论 unordered 容器并附带其他例子。Unordered multimap 也可用作字典（dictionary）（7.9.7 节第 383 页有一个例子）。

## 6.2.4 关联式数组（Associative Array）

不论 map 或 unordered map，都是 key/value pair 形成的集合，每个元素带着独一无二的 key。如此的集合也可被视为一个关联式数组（associative array），也就是“索引并非整数”的 array。也因此，刚才说的那两个容器都提供了下标操作符 []。

考虑下面这个例子：

```
// stl/assoarray1.cpp

#include <unordered_map>
#include <string>
#include <iostream>
using namespace std;

int main()
{
    // type of the container:
    // - unordered_map: elements are key/value pairs
    // - string: keys have type string
    // - float: values have type float
    unordered_map<string,float> coll;
```