The Standard Template Library

Kafi Rahman

Assistant Professor @CS
Truman State University

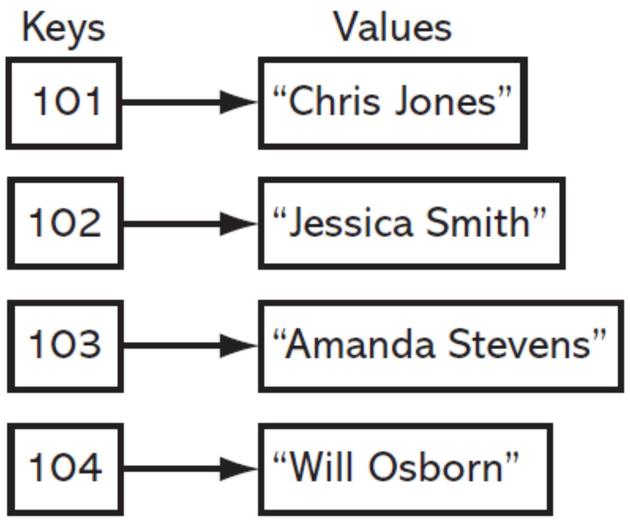
17.4

The map, multimap, and unordered_map Classes

Maps – General Concepts

- A map is an associative container.
- Each element that is stored in a map has two parts: a key and a value.
- To retrieve a specific value from a map, you use the key that is associated with that value.
- This is similar to the process of looking up a word in the dictionary, where the words are keys and the definitions are values.
  - Another example would be looking up student information by using the student id.

Maps



- Example: a map in which employee IDs are the keys and employee names are the values.
- You use an employee's ID to look up that employee's name.

The map Class

- You can use the STL map class to store key-value pairs.
- The keys that are stored in a map container are unique – no duplicates.
- The map class is declared in the <map> header file.

map Class Constructors

Default Constructor	map<keyDataType, valueDataType> name; Creates an empty map.
Range Constructor	map<keyDataType, valueDataType> name(iterator1, iterator2); Creates a map that is initialized with a range of values from another map. iterator1 marks the beginning of the range and iterator2 marks the end.
Copy Constructor	map<keyDataType, valueDataType> name(map2); Creates a map that is a copy of map2.

The map Class

- Example: defining a map container to hold employee ID numbers (as ints) and their corresponding employee names (as strings):

```
map<int, string> employees;
```

Key data type   Value data type

Initializing a map

```
map<int, string> employees =
{
    {101, "Chris Jones"}, {102, "Jessica Smith"},
    {103, "Amanda Stevens"}, {104, "Will Osborn"}
};
```

```
In the first element, the key is 101 and the value is "Chris Jones".
In the second element, the key is 102 and the value is "Jessica Smith".
In the third element, the key is 103 and the value is "Amanda Stevens".
In the fourth element, the key is 104 and the value is "Will Osborn".
```

The Overloaded [] Operator

- You can use the [] operator to add new elements to a map.
- General format:

  ```
  mapName[key] = value;
  ```

- This adds the key-value pair to the map.
- If the key already exists in the map, it's associated value will be changed to value.

The Overloaded [] Operator

```
map<int, string> employees;
employees[110] = "Beth Young";
employees[111] = "Jake Brown";
employees[112] = "Emily Davis";


//After this code executes, the employees map
will contain the following elements:
Key = 110, Value = "Beth Young"
Key = 111, Value = "Jake Brown"
Key = 112, Value = "Emily Davis"
```

The pair Type

- Internally, the elements of a map are stored as instances of the pair type.
- pair is a struct that has two member variables: first and second.
- The element's key is stored in first, and the element's value is stored in second.
- The pair struct is declared in the <utility> header file.
    - When you #include the <map> header file, <utility> is automatically included as well.

```
template <class TF, class TS>
class pair
{
    public:
        TF first;
        TS second;
    ...
};
```

Inserting Elements with the insert() Member Function

- The map class provides an insert() member function that adds a pair object as an element to the map.
- You can use the STL function template make_pair to construct a pair object.
- The make_pair function template is declared in the <utility> header file.

Inserting Elements with the insert() Member Function

```
map<int, string> employees;
employees.insert(make_pair(110, "Beth Young"));
employees.insert(make_pair(111, "Jake Brown"));
employees.insert(make_pair(112, "Emily Davis"));

// After this code executes, the employees map
will contain the following elements:
Key = 110, Value = "Beth Young"
Key = 111, Value = "Jake Brown"
Key = 112, Value = "Emily Davis"
```

Note: If the element that you are inserting with the `insert()` member function has the same key as an existing element, the function will NOT insert the new element.

Inserting Elements with the emplace() Member Function

```
map<int, string> employees;
employees.emplace(110, "Beth Young");
employees.emplace(111, "Jake Brown");
employees.emplace(112, "Emily Davis");

// After this code executes, the employees map will
contain the following elements:
Key = 110, Value = "Beth Young"
Key = 111, Value = "Jake Brown"
Key = 112, Value = "Emily Davis"
```

`Note: If the element that you are inserting with the emplace() member function has the same key as an existing element, the function will NOT insert the new element.`

- The map class also provides an emplace() member function that adds an element to the map.

Retrieving Elements with the at() Member Function

```
// Create a map containing employee IDs and names.
map<int, string> employees =
{
    {101, "Chris Jones"}, {102, "Jessica Smith"},
    {103, "Amanda Stevens"}, {104, "Will Osborn"}
};

// Retrieve a value from the map.
cout << employees.at(103) << endl;
```

Displays "Amanda Stevens"

- You can use the at() member function to retrieve a map element by its key

Retrieving Elements with the at() Member Function

```
// Create a map containing employee IDs and names.
map<int, string> employees =
{
    {101, "Chris Jones"}, {102, "Jessica Smith"},
    {103, "Amanda Stevens"}, {104, "Will Osborn"}
};

// Retrieve a value from the map.
if (employees.count(103))
    cout << employees.at(103) << endl;
else
    cout << "Employee not found.\n";
```

The count() member function returns 1 if the specified key exists, or 0 otherwise.

- To prevent the at() member function from throwing an exception (if the specified key does not exist), use the count member function to determine whether it exists

Deleting Elements

- You can use the erase() member function to retrieve and delete a map element by its key:

```
// Create a map containing employee IDs and names.
map<int, string> employees =
{
    {101, "Chris Jones"}, {102, "Jessica Smith"},
    {103, "Amanda Stevens"}, {104, "Will Osborn"}
};

// Delete the employee with ID 102.
employees.erase(102);
```

Deletes Jessica Smith from the map

Stepping Through a map with for Loop

```
// Create a map containing employee IDs and names.
map<int, string> employees =
{
    {101, "Chris Jones"}, {102, "Jessica Smith"},
    {103, "Amanda Stevens"}, {104, "Will Osborn"}
};

// Display each element.
for (pair<int, string> element : employees)
{
    cout << "ID: " << element.first << "\tName: "
           << element.second << endl;
}
```

Remember, each element is a pair.

Stepping Through a map with for Loop: auto

```
// Create a map containing employee IDs and names.
map<int, string> employees =
{
    {101, "Chris Jones"}, {102, "Jessica Smith"},
    {103, "Amanda Stevens"}, {104, "Will Osborn"}
};

// Display each element.
for (auto element : employees)
{
    cout << "ID: " << element.first << "\tName: "
         << element.second << endl;
}
```

auto simplifies this

Using an Iterator With a map

- The begin() and end() member functions return a bidirectional iterator of the iterator type
- The cbegin() and cend() member functions return a bidirectional iterator of the const_iterator type
- The rbegin() and rend() member functions return a reverse bidirectional iterator of the reverse_iterator type
- The crbegin() and crend() member functions return a reverse bidirectional iterator of the const_reverse_iterator type

Using an Iterator With a map

- When an iterator points to a map element
  - it points to an instance of the pair type.
- The element has two member variables: first and second.
- The element's key is stored in first, and the element's value is stored in second.

Using an Iterator With a map

Program 17-11

```
// This program demonstrates an iterator with a map.
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main()
{
    // Create a map containing employee IDs and names.
    map<int, string> employees =
       { {101,"Chris Jones"}, {102,"Jessica Smith"},
         {103,"Amanda Stevens"},{104,"Will Osborn"} };

    // Create an iterator.
    map<int, string>::iterator iter;

    // Use the iterator to display each element in the map.
    for (iter = employees.begin(); iter != employees.end(); iter++)
    {
        cout << "ID: " << iter->first
             << "\tName: " << iter->second << endl;
    }

    return 0;
}
```

Program Output

```
ID: 101 Name: Chris Jones
ID: 102 Name: Jessica Smith
ID: 103 Name: Amanda Stevens
ID: 104 Name: Will Osborn
```

Storing Objects Of Your Own Classes as Values in a map

- If we want to store an object as a value in a map, there is one requirement for that object's class:
  - It must have a default constructor.
- Consider the following Contact class⋯

Storing Objects Of Your Own Classes as Values in a map

```
class Contact
{
private:
      string name;
      string email;
public:
      Contact()
      {   name = "";
          email = ""; }

      Contact(string n, string em)
      {   name = n;
          email = em; }

      void setName(string n)
      {   name = n; }

      void setEmail(string em)
      {   email = em; }

      string getName() const
      {   return name; }

      string getEmail() const
      {   return email; }
};
#endif
```

Default constructor

Storing Objects Of Your Own Classes as Values in a map

Program 17-14

```
#include <iostream>
#include <string>
#include <map>
#include "Contact.h"
using namespace std;

int main()
{
    string searchName;    // The name to search for

    // Create some Contact objects
    Contact contact1("Ashley Miller", "amiller@faber.edu");
    Contact contact2("Jacob Brown", "jbrown@gotham.edu");
    Contact contact3("Emily Ramirez", "eramirez@coolidge.edu");

    // Create a map to hold the Contact objects.
    map<string, Contact> contacts;

    // Create an iterator for the map.
    map<string, Contact>::iterator iter;

    // Add the contact objects to the map.
    contacts[contact1.getName()] = contact1;
    contacts[contact2.getName()] = contact2;
    contacts[contact3.getName()] = contact3;
```

In the map, the keys are the contact names, and the values are the Contact objects.

Storing Objects Of Your Own Classes as Values in a map

```

    // Get the name to search for.
    cout << "Enter a name: ";
    getline(cin, searchName);

    // Search for the name.
    iter = contacts.find(searchName);

    // Display the results.
    if (iter != contacts.end())
    {
        cout << "Name: " << iter->second.getName() << endl;
        cout << "Email: " << iter->second.getEmail() << endl;
    }
    else
    {
        cout << "Contact not found.\n";
    }

    return 0;
}
```

Program Output (with Example Input Shown in Bold)

```
Enter a name: Emily Ramirez [Enter]
Name: Emily Ramirez
Email: eramirez@coolidge.edu
```

Program Output (with Example Input Shown in Bold)

```
Enter a name: Billy Clark [Enter]
Contact not found.
```

Thank you