NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC1008 C and C++ Programming

Assistant Professor WANG Yong

yong-wang@ntu.edu.sg

CCDS, Nanyang Technological University

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Week 12
Templates and Standard Template Library

Outline

- Templates
  - Class templates

- Standard Template Library (STL)
  - Container, iterator, algorithm
  - STL `vector`
  - STL `list`
  - STL `map`

Templates

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- A template is a simple and yet very powerful tool in C++. The simple idea is to **pass data type as a parameter** so that we **don't need to write the same code for different data types**
- Two types of templates in C++:

  - Function Templates

  - Class Templates

How Do Templates Work?

- Templates are expanded at compiler time, which is like macros
- But different from macros, the compiler does type-checking before expanding templates
- The source code contains only function/class, but **compiled code may contain multiple copies of the same function/class**

The keyword "**typename**" can also be replaced by "**class**"

```
template <typename T>
T myMax(T x, T y)
{
    return (x > y)? x: y;
}

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```

https://www.geeksforgeeks.org/templates-cpp/

Class Templates

- Like function templates, class templates are useful when a class defines something that is independent of the data type.

- The example code implements a template Array class

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

template <typename T>
class Array {
private:
  T* ptr;
  int size;
public:
  Array(T arr[], int s);
  ~Array(); // Destructor to free memory
  void print();
};

template <typename T>
Array<T>::Array(T arr[], int s) {
  ptr = new T[s];
  size = s;
  for (int i = 0; i < size; i++)
  ptr[i] = arr[i];
}

template <typename T>
Array<T>::~Array() { delete[] ptr; }

template <typename T>
void Array<T>::print() {
  for (int i = 0; i < size; i++)
    cout << " " << *(ptr + i);
  cout << endl;
}

int main() {
  int arr[5] = { 1, 2, 3, 4, 5 };
  Array<int> a(arr, 5);
  a.print();
  return 0;
}
```

https://www.geeksforgeeks.org/templates-cpp/

Class Templates

- Like function templates, class templates are useful when a class defines something that is independent of the data type.

- The example code implements a template Array class

1. Key points of defining member functions

2. Key point of using class templates

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

template <typename T>
class Array {
private:
  T* ptr;
  int size;
public:
  Array(T arr[], int s);
  ~Array(); // Destructor to free memory
  void print();
};

template <typename T>
Array<T>::Array(T arr[], int s) {
  ptr = new T[s];
  size = s;
  for (int i = 0; i < size; i++)
  ptr[i] = arr[i];
}
```

Remember to free allocated memory

```
template <typename T>
Array<T>::~Array() { delete[] ptr; }

template <typename T>
void Array<T>::print() {
  for (int i = 0; i < size; i++)
    cout << " " << *(ptr + i);
  cout << endl;
}

int main() {
  int arr[5] = { 1, 2, 3, 4, 5 };
  Array<int> a(arr, 5);
  a.print();
  return 0;
}
```

https://www.geeksforgeeks.org/templates-cpp/

Class Templates

- Like function templates, class templates are useful when a class defines something that is independent of the data type.

- The example code implements a template Array class

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

template <typename T>
class Array {
private:
  T* ptr;
  int size;
public:
  Array(T arr[], int s);
  ~Array(); // Destructor to free memory
  void print();
};

template <typename T>
Array<T>::Array(T arr[], int s) {
  ptr = new T[s];
  size = s;
  for (int i = 0; i < size; i++)
  ptr[i] = arr[i];
}

template <typename T>
Array<T>::~Array() { delete[] ptr; }

template <typename T>
void Array<T>::print() {
  for (int i = 0; i < size; i++)
    cout << " " << *(ptr + i);
  cout << endl;
}

int main() {
  int arr[5] = { 1, 2, 3, 4, 5 };
  Array<int> a(arr, 5);
  a.print();
  return 0;
}
```

Program output:

```
1 2 3 4 5
```

https://www.geeksforgeeks.org/templates-cpp/

Class Templates with Multiple Parameters

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- We can pass more than one data type as arguments to class templates
- Syntax

```
template<class T1, class T2, ...>
class classname
{
    ...
    ...
};
```

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

// Class template with two parameters
template <class T1, class T2>
class Test {
  T1 a;
  T2 b;
public:
  Test(T1 x, T2 y) {
    a = x;
    b = y;
  }
  void show() {
    cout << a << " and " << b << endl;
  }
};

int main() {
  Test<float, int> test1(1.23, 123);
  Test<int, char> test2(100, 'W');
  test1.show();
  test2.show();
  return 0;
}
```

https://www.geeksforgeeks.org/templates-cpp/

Class Templates with Multiple Parameters

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- We can pass more than one data type as arguments to class templates
- Syntax

```
template<class T1, class T2, ...>
class classname
{
    ...
    ...
};
```

This is how we define and use a class template with multiple data types

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

// Class template with two parameters
template <class T1, class T2>
class Test {
  T1 a;
  T2 b;
public:
  Test(T1 x, T2 y) {
    a = x;
    b = y;
  }
  void show() {
    cout << a << " and " << b << endl;
  }
};

int main() {
  Test<float, int> test1(1.23, 123);
  Test<int, char> test2(100, 'W');
  test1.show();
  test2.show();
  return 0;
}
```

https://www.geeksforgeeks.org/templates-cpp/

C++'s Standard Library

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

C++'s Standard Library consists of four major pieces:

1) The entire C standard library
2) C++'s input/output stream library
   - std::cin, std::cout, stringstreams, fstreams, etc.
3) C++'s Standard Template Library (**STL**) ☜
   - **Containers, iterators, algorithms** (sort, find, etc.)
4) C++'s miscellaneous library
   - Strings, exceptions, memory allocation, localization

C++ Standard Template Library (STL)

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- STL: a library containing many **templates** for frequently used data structures and algorithms

- STL had three basic components:
  - **Containers**

    Generic class templates for storing collection of data
  - **Algorithms**

    Generic function templates for operating on containers
  - **Iterators**

    Generalized ‘smart’ pointers that facilitate use of containers.

    They provide an interface that is needed for STL algorithms to operate on STL containers

Containers

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- Containers: Data structures that hold **anything** (other objects)
- Two types of container classes in STL:
  - Sequence containers: organize and access data sequentially, as in an array. These include **`vector`**, `dequeue`, and **`list`**
  - Associative containers: use keys to allow data elements to be quickly accessed. These include `set`, `multiset`, **`map`**, and `multimap`

Containers

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- This class will mainly focus on **`vector`**, **`list`** and **`map`**

Container Name	Description
`vector`	An expandable array. Values may be added to or removed from the end or middle of a `vector`.
`deque`	Like a `vector`, but allows values to be added to or removed from the front.
`list`	A doubly linked list of data elements. Values may be inserted to or removed from any position. (You will learn more about linked lists in Chapter 17.)

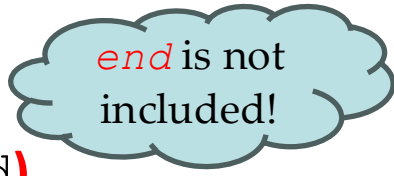
Container Name	Description
`set`	Stores a set of keys. No duplicate values are allowed.
`multiset`	Stores a set of keys. Duplicates are allowed.
`map`	Maps a set of keys to data elements. Only one key per data element is allowed. Duplicates are not allowed.
`multimap`	Maps a set of keys to data elements. Many keys per data element are allowed. Duplicates are allowed.

Note: The above tables are not an exhaustive list of all the containers in STL. More can be found here: https://cplusplus.com/reference/stl/

Iterators

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- Each container class has an associated `iterator` class (*e.g.* `vector<int>::iterator`) used to iterate through elements of the container
  - https://cplusplus.com/reference/iterator/iterator/?kw=iterator
  - Iterator range is from `begin` up to `end` i.e., [`begin`, `end`)
    - `end` is the one past the last container element!
  - Some container iterators support more operations than others
    - All can be incremented (++), copied, copy-constructed
    - Some can be dereferenced on right-hand side (*e.g.* `x = *it;`)
    - Some can be dereferenced on left-hand side (*e.g.* `*it = x;`)
    - Some can be decremented (--)
    - Some support random access ([], +, −, +=, −=, <, > operators)



Algorithms

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- STL contains algorithms that are implemented as **function templates** to perform common tasks on containers
- Requires `algorithm` header file
- The common tasks include searching, sorting, comparing, and editing, etc. Thus, `algorithm` includes

```
binary_search     count
for_each          find
find_if           max_element
min_element       random_shuffle
sort              and others
```

Algorithms

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Algorithm	Description
`binary_search`	Performs a binary search for an object and returns true if the object is found. *Example:* `binary_search(iter1, iter2, value);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement performs a binary search on the range of elements, searching for `value`. The `binary_search` function returns `true` if the element was found and `false` if the element was not found.
`count`	Returns the number of times a value appears in a range. *Example:* `iter3 = count(iter1, iter2, value);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement returns the number of times `value` appears in the range of elements.
`find`	Finds the first object in a container that matches a value and returns an iterator to it. *Example:* `iter3 = find(iter1, iter2, value);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement searches the range, of elements for `value`. If `value` is found, the function returns an iterator to the element containing it.

Algorithms

Algorithm	Description
`for_each`	Executes a function for each element in a container. *Example:* `for_each(iter1, iter2, func);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The third argument, `func`, is the name of a function. The statement calls the function `func` for each element in the range, passing the element as an argument.
`max_element`	Returns an iterator to the largest object in a range. *Example:* `iter3 = max_element(iter1, iter2);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement returns an iterator to the element containing the largest value in the range.
`min_element`	Returns an iterator to the smallest object in a range. *Example:* `iter3 = min_element(iter1, iter2);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement returns an iterator to the element containing the smallest value in the range.
`random_shuffle`	Randomly shuffles the elements of a container. *Example:* `random_shuffle(iter1, iter2);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement randomly reorders the elements in the range.
`sort`	Sorts a range of elements. *Example:* `sort(iter1, iter2);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The statement sorts the elements in the range in ascending order.

STL `vector`

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- **A generic, dynamically resizable array**
  - http://www.cplusplus.com/reference/stl/vector/vector/
  - Elements are stored in ***contiguous*** memory locations
    - Elements can be accessed using pointer arithmetic if you'd like
    - Random access is O(1) time
  - Adding/removing from the end is cheap (amortized constant time)
  - Inserting/deleting from the middle or start is expensive (linear time)

You are recommended to use STL **`vector`**!

STL `vector`

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Defining a new `vector`

- Syntax: vector<of what>

- Remember to include the source file

```
#include <vector>
```

- Examples:

`vector<int>` - vector of integers.
`vector<string>` - vector of strings.
`vector<int *>` - vector of pointers to integers.
`vector<Shape>` - vector of `Shape` objects. `Shape` is a user defined class.

STL `vector`

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Using `vector`

- There are two ways to use the vector type:
  - Array style
  - **STL style** (*the recommended one!*)

STL vector

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- *Array style*: we mimic the use of the built-in array

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

int main() {
  const int N = 5;
  vector<int> ivec(N);

  // Input values into vector
  cout << "Enter 5 integers: ";
  for (int i = 0; i < N; ++i)
     cin >> ivec[i];

  // Display copied array values
  cout << "Array contents: ";
  for (int j = 0; j < N; ++j)
     cout << ivec[j] << " ";
  cout << endl;

  return 0;
}
```

STL vector

- *STL style*: we can use the iterator and algorithm provided in STL

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

int main() {
  int input;
  vector<int> ivec;

  // Input values into vector using iterator
  cout << "Enter 5 integers: ";
  while (cin >> input )
    ivec.push_back(input);

  // Display values using iterator
  cout << "Array contents: ";
  vector<int>::iterator it;
  for ( it = ivec.begin(); it != ivec.end(); ++it ) {
    cout << *it << " ";
  }

  cout << endl;
  return 0;
}
```

While-loop stops until an invalid input (e.g., letters) is given, which results in an input failure.

STL vector

- *STL style*: we can use the iterator and algorithm provided in STL

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

int main() {
  int input;
  vector<int> ivec;

  // Input values into vector using iterator
  cout << "Enter 5 integers: ";
  while (cin >> input )
    ivec.push_back(input);

  // Display values using iterator
  cout << "Array contents: ";
  vector<int>::iterator it;
  for ( it = ivec.begin(); it != ivec.end(); ++it ) {
    cout << *it << " ";
  }

  cout << endl;
  return 0;
}
```

push_back(): Insert a value to the back of the vector

An iterator for vector of integer

iterator range

STL vector + algorithm

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Sort a vector of integers

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

int main() {
  int input;
  vector<int> ivec;

  // Input values into vector using iterator
  cout << "Enter 5 integers: ";
  while (cin >> input )
    ivec.push_back(input);

  // sorting
  sort(ivec.begin(), ivec.end());

  // Display values using iterator
  cout << "Array contents: ";
  vector<int>::iterator it;
  for ( it = ivec.begin(); it != ivec.end(); ++it ) {
    cout << *it << " ";
  }

  cout << endl;
  return 0;
}
```

Sort the whole vector of integers.

STL vector + algorithm

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

sort()

- Formal Definition:

```
void sort(Iterator begin, Iterator end);
```

- *Example:*

```
vector<int> ivec;
// Fill ivec with integers …
sort(ivec.begin(), ivec.end())
```

STL vector – Member Functions

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- More member functions can be found in the official documents
  https://cplusplus.com/reference/vector/vector/?kw=vector

Member functions

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin	Return const_iterator to beginning (public member function)
cend	Return const_iterator to end (public member function)
crbegin	Return const_reverse_iterator to reverse beginning (public member function)
crend	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)
shrink_to_fit	Shrink to fit (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace	Construct and insert element (public member function)
emplace_back	Construct and insert element at the end (public member function)

Allocator:

get_allocator	Get allocator (public member function)

Non-member function overloads

relational operators	Relational operators for vector (function template)
swap	Exchange contents of vectors (function template)

Template specializations

vector<bool>	Vector of bool (class template specialization)

STL `list`

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- **A generic doubly-linked list**
  - http://www.cplusplus.com/reference/stl/list/
  - Elements are ***not*** stored in contiguous memory locations
    - Does not support random access (*e.g.* cannot do list[5])
  - Some operations are much more efficient than vectors
    - `lists` is much quicker than `vectors` to insert or add elements to their front or their back, since `lists` do not need to shift the other elements and have pointers to the first and last element (no traversal required)
    - Can iterate forward or backwards
  - Has a built-in **sort** member function
    - Doesn't copy! Manipulates list structure instead of element values

STL `list` – Member Functions

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Member Function	Examples and Description
`back`	`cout << list.back() << endl;` The `back` member function returns a reference to the last element in the list.
`empty`	`if (list.empty())` The `empty` member function returns `true` if the `list` is empty. If the `list` has elements, it returns `false`.
`end`	`iter = list.end();` `end` returns a bidirectional iterator to the end of the list.

`erase`	`list.erase(iter);`
	`list.erase(firstIter, lastIter)`
	The first example causes the `list` element pointed to by the iterator `iter` to be removed. The second example causes all of the `list` elements from `firstIter` to `lastIter` to be removed.

`front`	`cout << list.front() << endl;` `front` returns a reference to the first element of the `list`.
`insert`	`list.insert(iter, x)` The `insert` member function inserts an element into the `list`. This example inserts an element with the value `x`, just before the element pointed to by `iter`.
`merge`	`list1.merge(list2);` `merge` inserts all the items in `list2` into `list1`. `list1` is expanded to accommodate the new elements plus any elements already stored in `list1`. `merge` expects both lists to be sorted. When `list2` is inserted into `list1`, the elements are inserted into their correct position, so the resulting list is also sorted.
`pop_back`	`list.pop_back();` `pop_back` removes the last element of the list.
`pop_front`	`list.pop_front();` `pop_front` removes the first element of the list.
`push_back`	`list.push_back(x);` `push_back` inserts an element with value `x` at the end of the list.
`push_front`	`list.push_front(x);` `push_front` inserts an element with value `x` at the beginning of the list.

STL `list` – Member Functions

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Member Function	Examples and Description
`reverse`	`list.reverse();` `reverse` reverses the order in which the elements appear in the list.
`size`	Returns the number of elements in the `list`.
`swap`	`list1.swap(list2)` The swap member function swaps the elements stored in two `lists`. For example, assuming `list1` and `list2` are `lists`, this statement will exchange the values in the two lists.
`unique`	`list.unique();` `unique` removes any element that has the same value as the element before it.

STL list

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Example 1 of list

Program output:

```
0 10 20 30 40 50 60 70 80 90
90 80 70 60 50 40 30 20 10 0
```

```
// This program demonstrates the STL list container.
#include <iostream>
#include <list> // Include the list header.
using namespace std;

int main()
{
   // Define a list object.
   list<int> myList;

   // Define an iterator for the list.
   list<int>::iterator iter;

   // Add values to the list.
   for (int x = 0; x < 100; x += 10)
      myList.push_back(x);

   // Display the values.
   for (iter = myList.begin(); iter != myList.end(); iter++)
      cout << *iter << " ";
   cout << endl;

   // Now reverse the order of the elements.
   myList.reverse();

   // Display the values again.
   for (iter = myList.begin(); iter != myList.end(); iter++)
      cout << *iter << " ";
   cout << endl;

   return 0;
}
```

STL list + algorithm

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Example 2 of list

```
#include <iostream>
#include <list>
#include <algorithm> // for std::for_each

using namespace std;

// Function to print an element
void printElement(int x) {
   cout << x << " ";
}

int main() {
   // Create a list of integers
   list<int> myList;

   // Add values to the list.
   for (int x = 0; x < 100; x += 10)
      myList.push_back(x);

  // Use for_each to apply printElement function to each element
   cout << "List elements: ";
   for_each(myList.begin(), myList.end(), printElement);
   cout << endl;

   return 0;
}
```

Program output:

```
List elements: 0 10 20 30 40 50 60 70 80 90
```

STL list + algorithm

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Algorithm	Description
`for_each`	Executes a function for each element in a container. *Example:* `for_each(iter1, iter2, func);` In this statement, `iter1` and `iter2` point to elements in a container. (`iter1` points to the first element in the range, and `iter2` points to the last element in the range.) The third argument, `func`, is the name of a function. The statement calls the function `func` for each element in the range, passing the element as an argument.

STL map

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- One of C++'s *associative* containers: a key/value table, implemented as a search tree
  - http://www.cplusplus.com/reference/stl/map/
  - General form: `map<key_type, value_type> name;`
  - **Keys must be *unique***
    - `multimap` allows duplicate keys
  - Efficient lookup (O(log n)) and insertion (O(log n))
    - Access `value` via `name[key]`
  - Elements are type `pair<key_type, value_type>` and are stored in *sorted* order (key is field `first`, value is field `second`)
    - Key type must support less-than operator (<)

STL map

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- `std:pair<key_type, value_type>`
  - This class couples together a pair of values, which may be of different types

class template

std::pair <utility>

```
template <class T1, class T2> struct pair;
```

Pair of values

This class couples together a pair of values, which may be of different types (T1 and T2). The individual values can be accessed through its public members first and second.

Pairs are a particular case of tuple.

Template parameters

T1

Type of member first, aliased as first_type.

T2

Type of member second, aliased as second_type.

Member types

member type	definition	notes
first_type	The first template parameter (T1)	Type of member first.
second_type	The second template parameter (T2)	Type of member second.

Member variables

member variable	definition
first	The first value in the pair
second	The second value in the pair

Member functions

(constructor)	Construct pair (public member function)
pair::operator=	Assign contents (public member function)
pair::swap	Swap contents (public member function)

https://cplusplus.com/reference/utility/pair/?kw=pair

STL map – Member Functions

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Member functions

(constructor)	Construct map (public member function)
(destructor)	Map destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin	Return const_iterator to beginning (public member function)
cend	Return const_iterator to end (public member function)
crbegin	Return const_reverse_iterator to reverse beginning (public member function)
crend	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace	Construct and insert element (public member function)
emplace_hint	Construct and insert element with hint (public member function)

Observers:

key_comp	Return key comparison object (public member function)
value_comp	Return value comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

Allocator:

get_allocator	Get allocator (public member function)

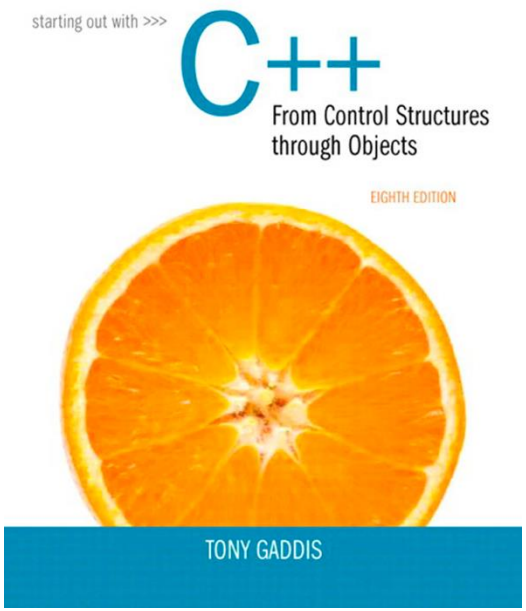
https://cplusplus.com/reference/map/map/?kw=map

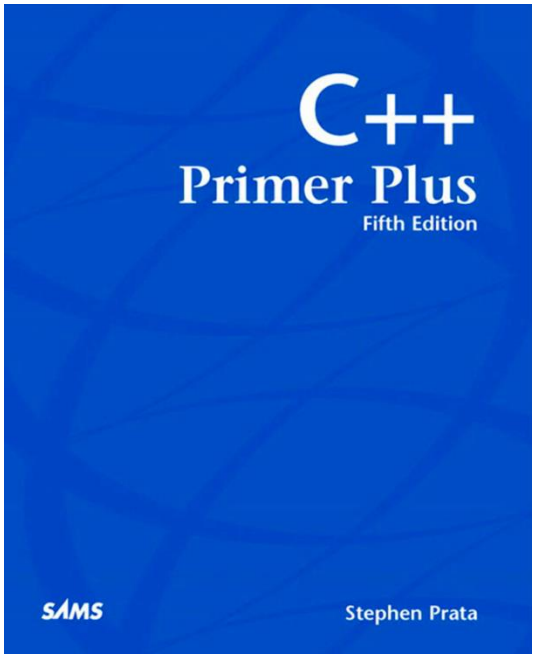
STL map

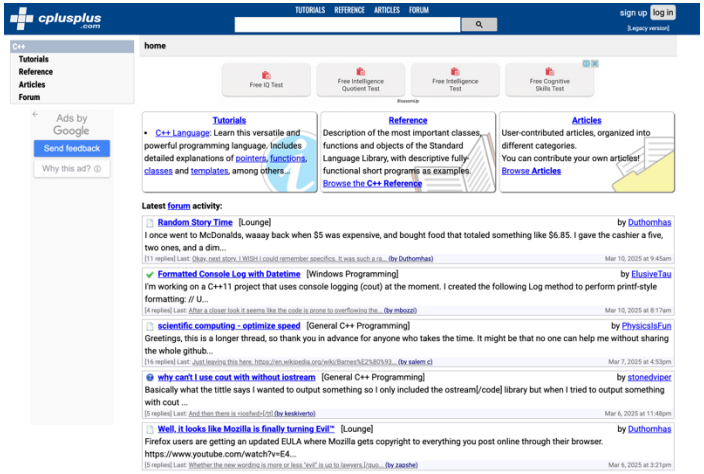
```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
  map<string, int> studentGrades;

  // Insert elements into the map
  studentGrades["Alice"] = 90;
  studentGrades["Bob"] = 85;
  studentGrades["Charlie"] = 78;
  studentGrades["David"] = 92;

  // Display all students and their grades
  cout << "Student Grades:\n";
  map<string, int>::iterator it;
  for (it = studentGrades.begin(); it != studentGrades.end();++it) {
     cout << it->first << " -> " << it->second << endl;
  }

  // Searching for a student
  string name = "Charlie";
  map<string, int>::iterator search = studentGrades.find(name);
  if (search != studentGrades.end()) {
    cout << "\n" << name << "'s Grade: " << search->second << endl;
  } else {
    cout << "\nStudent not found!" << endl;
  }

  // Removing a student
  studentGrades.erase("Bob");

  // Displaying updated map
 cout << "\nUpdated Student Grades:\n";
 for (it = studentGrades.begin(); it != studentGrades.end(); ++it) {
    cout << it->first << " -> " << it->second << endl;
 }
 return 0;
}
```

Program output:

```
Student Grades:
Alice -> 90
Bob -> 85
Charlie -> 78
David -> 92

Charlie's Grade: 78

Updated Student Grades:
Alice -> 90
Charlie -> 78
David -> 92
```







References:

[1] Tony Gaddis. Starting out with C++ from control structures through objects, 8th edition. Chapter 16.3-16.5, Chapter 17.5.

[2] Prata, Stephen. C++ primer plus. Sams Publishing, 2002, 5th edition. Chapters 16.

[3] https://cplusplus.com/

Questions?

Thank You!