13. Templates II

20. The Standard Template Library

Container (1)

- A container is a class, a data structure, or an abstract data type (ADT) whose instances are collections of other objects

- Sequence Containers: implement data structures which can be accessed in a sequential manner.
  - vector, list, deque, arrays
- Container Adaptors : provide a different interface for sequential containers.
  - queue, priority_queue, stack
- Associative Containers : implement sorted data structures that can be quickly searched
  - set, multiset, map, multimap
- Unordered Associative Containers : implement unordered data structures that can be quickly searched
  - unordered_set, unordered_multiset, unordered_map, unordered_multimap

List (1)

```
// std::list is a container that supports constant time insertion and
removal of elements from anywhere in the container.

template<
    class T,
    class Allocator = std::allocator<T>
> class list;

template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```


List (2)

Member: begin, end()
Non-member: std::begin(…), std::end(…)

```
#include <iostream>
#include <list>
#include <string>
int main() {
   std::list<std::string> lst{ "zero", "one", "two", "three", "four"};

   for (auto i : lst)
      std::cout << i << ", ";

   std::cout << std::endl;
   for (auto iter = std::begin(lst); iter != std::end(lst); iter++)
      std::cout << *iter << ", ";

   //lst[0];
}
```

Set (1)

```
// std::set is an associative container that contains a sorted set of
// unique objects of type Key
// Multiset: multiple keys with equivalent values are allowed

template<
    class Key,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<Key>
> class set;
```


Set (2)

```
#include <iostream>
#include <set>

int main() {
    std::set<int> x;

    x.insert(10);
    x.insert(10);
    x.insert(11);
    x.insert(9);

    for (auto i : x)
        std::cout << i << ", ";
    std::cout << std::endl;

    for (auto it = x.begin() ; it != x.end() ;++it)
        std::cout << *it << ", ";
    std::cout << std::endl;
}
```

Map (1)

```
// std::map is a sorted associative container that contains key-value
// pairs with unique keys

template<
    class Key,
    class T,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<std::pair<const Key, T>>
> class map;
```


Map (2)

```
#include <iostream>
#include <string>
#include <map>

int main() {
    std::map<std::string, int> y;
    y.insert(std::pair<std::string, int>("+", 10));
    y.insert(std::pair<std::string, int>("+", 20));
    y.insert(std::pair<std::string, int>("<<", 30));
    y["*"] = 0;

    for (auto i : y) {
        std::cout << i.first << ", " << i.second << ", ";
        //std::cout << y[i.first] << ", ";
    }
    std::cout << std::endl;

    for (auto it = y.begin(); it != y.end(); ++it)
        std::cout << it->first << ", " << it->second << ", ";
}
```

std::pair is a class template that provides a way to store two heterogeneous objects as a single unit, <utility>

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

priority_queue (1)

```
// template<class T, class Container = std::vector<T>,
//    class Compare = std::less<typename Container::value_type>>
// class priority_queue;

#include <iostream>
#include <queue>
int main() {
   std::priority_queue<int> queue;// Container: vector, Compare: less
   queue.push(11);     queue.push(2);         queue.push(4);
   queue.push(15);     queue.push(15);        queue.push(4);
   queue.push(7);

   while (!queue.empty()) {
      std::cout << queue.top() << ' ';
      queue.pop();
   }
} // 15 15 11 7 4 4 2
```

Compare parameter is defined such that it returns true if its first argument comes before its second argument in a weak ordering. But because the priority queue outputs largest elements first, the elements that "come before" are actually output last.

priority_queue (2)

```
#include <iostream>
struct Cmp {
   bool operator()(int i1, int i2) {  // function call operator
      return i2 < i1;
   }
};
int main() {
   Cmp op;

   std::cout << std::boolalpha << op(2, 3) << '\n';
   std::cout << std::boolalpha << op(3, 2) << '\n';
   std::cout << std::boolalpha << op(3, 3) << '\n';
}
```

priority_queue (3)

```
#include <iostream>
#include <queue>
#include <vector>
struct Cmp {
   bool operator()(int i1, int i2) {  return i2 < i1;   }
};
int main() {  // std::greater<int>, <functional>
   std::priority_queue<int, std::vector<int>, Cmp> queue;
   queue.push(11);     queue.push(2);         queue.push(4);
   queue.push(15);     queue.push(15);        queue.push(4);
   queue.push(7);
   while (!queue.empty()) {
      std::cout << queue.top() << ' ';        queue.pop();
   }
} // 2 4 4 7 11 15 15
```


Iterators (1)

- An iterator is an object that allows a client to traverse and access elements of a data structure in an implementation independent way. C++ defines two global functions, std::begin and std::end, that produce iterators to the front and back, respectively, of a data structure like a vector or static array.

- Methods
  - operator*: used to access the element at the itertator's current position. The syntax is exactly like pointer dereferencing.
  - operator++: used to move the iterator to the next element within the data structure. The syntax is exactly like pointer arithmetic.
  - operator!=: used to determine whether two iterator objects currently refer to different elements within the data structure.

- Generality: one function template for different containers

Iterators (2)

```
#include <iostream>
#include <vector>
int main() {
   std::vector<int> vec {10, 20, 30, 40, 50};

   std::vector<int>::iterator iter = std::begin(vec);
   // auto iter = std::begin(vec);
   std::cout << *iter << '\n'; // 10

   iter++;
   std::cout << *iter << '\n'; // 20

   *iter = 100;
   for(auto x : vec)
      std::cout << x << ' ';
}
```


Iterators (3)

```
#include <iostream>
#include <vector>
int main() {
   std::vector<int> vec {10, 20, 30, 40, 50};

   for (auto iter = std::begin(vec); iter != std::end(vec); iter++)
   // for (auto iter = vec.begin(); iter != vec.end(); iter++)
      std::cout << *iter << ' ';

   std::cout << '\n';
}

// int arr[] = {10, 20, 30, 40, 50};
// for (auto iter = std::begin(arr); iter != std::end(arr); iter++)
//    std::cout << *iter << ' ';
```

Iterators (4)

```
#include <iostream>
#include <vector>
void print_reverse(const std::vector<int>& v) {
   auto p = std::end(v);
   while (p != std::begin(v)) {
      p--;
      std::cout << *p << ' ';
   }
   std::cout << '\n';
}
int main() {
   std::vector<int> vec(20);
   for (int i = 0; i < vec.size(); i++)
      vec[i] = i;

   print_reverse(vec);
}
```

std::reverse_iterator
std::ranges::views::reverse(…) , <ranges>, c++20

Iterator Ranges

```
#include <iostream>
#include <vector>
int main() {
   std::vector<int> vec{ 10, 20, 30, 40, 50 };

   for (int i = 1; i <= vec.size(); ++i) {
      for (auto iter = std::begin(vec); iter != std::begin(vec) + i
         ; iter++)
         std::cout << *iter << ' ';
      std::cout << '\n';
   }
}

//10
//10 20
//10 20 30
//10 20 30 40
//10 20 30 40 50
```

Algorithms in the Standard Library (1)

```
// template<class ForwardIt, class T>
// void iota(ForwardIt first, ForwardIt last, T value);//until  C++20
// <numeric>, "eye-oh-tuh"
// Fills the range [first, last) with sequentially increasing values,
// starting with value and repetitively evaluating ++value.

std::vector<int> seq(1000);
// Populate the vector with 0, 1, 2, 3, ..., 999
std::iota(std::begin(seq), std::end(seq), 0);
```


Algorithms in the Standard Library (2)

```
// template<class InputIt, class T>
// InputIt find(InputIt first, InputIt last,
//    const T& value); // until  C++20
// <algorithm>

std::vector<int> seq(1000, 0);
std::iota(std::begin(seq), std::end(seq), 0);
auto iter = std::find(std::begin(seq), std::end(seq), 678);
if (iter != std::end(seq))
   std::cout << *iter << " is present" << '\n';
else
   std::cout << "678 is NOT present" << '\n';
```

Algorithms in the Standard Library (3)

```
// template<class InputIt, class OutputIt>
// OutputIt copy( InputIt first, InputIt last,
//    OutputIt d_first ); // until  C++20
// <algorithm>

std::vector<int> seq2(seq.size() - 2);
std::copy(std::begin(seq) + 1, std::end(seq) - 1,
  std::begin(seq2));

// template<class InputIt, class OutputIt, class UnaryOperation>
// OutputIt transform( InputIt first1, InputIt last1,
// OutputIt d_first, UnaryOperation unary_op ); // until  C++20
// <algorithm>

std::string name = "Fred";
std::transform(std::begin(name), std::end(name),
   std::begin(name), std::toupper);
```


Algorithms in the Standard Library (4)

```
// template<class T, class CharT = char,
//   class Traits = std::char_traits<CharT>>
// class ostream_iterator : public std::iterator
//   <std::output_iterator_tag, void, void, void, void> // until  C++17
// <iterator>
// constructor
// ostream_iterator(ostream_type& stream, const CharT* delim);


std::vector<int> vec { 10, 20, 30, 35, 40, 45, 50, 55 };
auto strm = std::ostream_iterator<int>(std::cout, " ");
std::copy(std::begin(vec), std::end(vec), strm);
std::cout << '\n';
```

Algorithms in the Standard Library (5)

```
// template<class InputIt, class T>
// typename iterator_traits<InputIt>::difference_type
//   count( InputIt first, InputIt last,
//      const T &value ); // until  C++20
// <algorithm>

// template<class InputIt, class UnaryPredicate>
// typename iterator_traits<InputIt>::difference_type
//   count_if( InputIt first, InputIt last,
//      UnaryPredicate p ); // until  C++20
// <algorithm>

template <class T>
bool is_even(T n) {
   return n % 2 == 0;
}

std::vector<int> seq { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
int even_count
   = count_if(std::begin(seq), std::end(seq), is_even<int>);
```


Algorithms in the Standard Library (6)

```
// template<class RandomIt, class URBG>
// void shuffle( RandomIt first, RandomIt last, URBG&& g );
// <algorithm>
// g – UniformRandomBitGenerator
// Reorders the elements in the given range [first, last) such that
// each possible permutation of those elements has equal probability
// of appearance.

std::vector<int> vec{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
std::random_device dev;
std::mt19937 generator(dev());

std::shuffle(std::begin(vec), std::end(vec), generator);
```

Algorithms in the Standard Library (7)

```
// template< class ForwardIt, class Generator >
// void generate( ForwardIt first, ForwardIt last,
//    Generator g); // until  C++20
// <algorithm>

int Fn() { return 10; }
std::vector<int> vec(10);
std::generate(std::begin(vec), std::end(vec), Fn);

// template< class InputIt, class T >
// T accumulate(InputIt first, InputIt last, T init); // until  C++20
// template< class InputIt, class T, class BinaryOperation >
// T accumulate( InputIt first, InputIt last, T init,
//    BinaryOperation op ); // until  C++20
// <numeric>

std::vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = std::accumulate(v.begin(), v.end(), 0);
int product =
   std::accumulate(v.begin(), v.end(), 1, std::multiplies<int>());
// template< class T > struct multiplies; // until  C++14
```


Algorithms in the Standard Library (8)

```
// template< class ForwardIt, class UnaryPredicate >
// ForwardIt partition( ForwardIt first, ForwardIt last,
//    UnaryPredicate p ); // until  C++20
// <algorithm>

bool Fn(const std::string& w) { return w.length() > 3; }
std::vector<std::string> words{ "fred", "ella", "adam", "jo", "pat",
     "mel", "anna", "ed", "oscar", "will", "tom", "ingrid" };
auto it = std::partition(std::begin(words), std::end(words), Fn);

// [begin, it), [it, end)
```

Algorithms in the Standard Library (9)

```
// template< class InputIt1, class InputIt2, class OutputIt >
// OutputIt merge( InputIt1 first1, InputIt1 last1,
//    InputIt2 first2, InputIt2 last2,
//    OutputIt d_first); // until  C++20
// <algorithm>


std::vector<int> merged(nums1.size() + nums2.size());
std::merge(std::begin(nums1), std::end(nums1),
   std::begin(nums2), std::end(nums2), std::begin(merged));
```


Algorithms in the Standard Library (10)

```
std::remove(std::begin(nums), std::end(nums), 10);
std::remove_if(std::begin(nums), std::end(nums), is_even);
// Delete elements from index 3 to index 7
nums.erase(std::begin(nums) + 3, std::begin(nums) + 8);

auto it = std::remove(v.begin(), v.end(), 1);
v.erase(it, v.end());

// std::reverse reverses the elements of a container
// std::all_of returns true if all the elements in a container
//    satisfy a predicate,
// std::any_of returns true if at least one of the elements in
//    a container satisfy a predicate,
// std::none_of returns true if none of the elements in a container
//    satisfy a predicate,
// std::find_if returns an iterator to the first element in
//    a container that satisfies a predicate

// https://www.cplusplus.com/reference/algorithm/
```

wchar_t (1)

```
// Wide character: wchar_t

#include <iostream>                                  wcscpy, wcslen, …
#include <string>
int main() {
    wchar_t wch = L'A';
    wchar_t wstr1[100] = L"C++ Programming";
    std::wstring wstr2(L"C++ Programming");

    std::cout << wch << std::endl;
    std::cout << wstr1 << std::endl;

    std::wcout << wch << std::endl;
    std::wcout << wstr1 << std::endl;
    std::wcout << wstr2 << std::endl;

    std::cout << sizeof(wchar_t) << std::endl;
}
```


wchar_t (2)

- MBCS (Multi Byte Character Set): variable-width encoding

- Unicode Transformation Format (UTF)
  - UTF-8
  - UTF-16
  - UTF-32

```
wchar_t wch = L'A';
wchar_t wstr1[100] = L"C++ Programming";
std::wstring wstr2(L"C++ Programming");

std::wcout << wch << std::endl;
std::wcout << wstr1 << std::endl;
std::wcout << wstr2 << std::endl;

std::wcout << sizeof(wchar_t) << std::endl; // 2, 4
```

wchar_t (3)

```
#include <iostream>
#include <string>
#include <locale>
#include <codecvt>
#ifdef _MSC_VER
#include <io.h>
#include <fcntl.h>
#endif
int main() {
#ifdef _MSC_VER
    constexpr char cp_utf16le[] = ".1200"; // UTF-16 little-endian locale.
    setlocale(LC_ALL, cp_utf16le);    _setmode(_fileno(stdout), _O_WTEXT);
#else
    constexpr char locale_name[] = "";
    std::setlocale(LC_ALL, locale_name);
    std::locale::global(std::locale(locale_name));
    std::wcin.imbue(std::locale());    std::wcout.imbue(std::locale());
#endif
    std::string u8(u8"スタ ивт 한글");
    std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>, wchar_t> convert;
    std::wstring wide = convert.from_bytes(u8);
    std::wcout << wide << std::endl;
}
```

참고] https://copyprogramming.com/howto/c-can-t-get-wcout-to-print-unicode-and-leave-cout-working