Lecture #17 | Standard Template Library: algorithm

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

Previously in Object-Oriented Programming

- STL container
  - `vector`
  - `list`
  - `map`
  - `unordered_map`
- STL iterators
  - `iterator`
  - `reverse_iterator`

Today’s topic

- Range-for
- STL algorithm
- More on STL containers/iterators/algorithms

Ranger for

- Syntax: `for (auto x: v)`
  - for each element `x` in `v`
- Range for provides simple mechanism to traverse the elements in `v` from the beginning to the end
- The scope of element variable `x` is the for-statement
- The range `v` should provide one of the following expression need to yield iterators
  - `v.begin(), v.end()`
  - `begin(v), end(v)`
- c.f., for built-in array `T v[n]`, `begin(v)` and `end(v)` means `v` and `v+N`

Example: range for

```
template<typename T> void println(T& c) {
    for (auto& i: c)
        cout << i << " ";
    cout << endl;
}
template<typename T> void println2(T& c) {
    for (auto it = c.begin(); it != c.end(); ++it)
        cout << *it << " ";
    cout << endl;
}
void foo() {
    vector<int> c1 {42, 23, 6, 1024};
    list<int> c2 {42, 23, 6, 1024};
    println(c1); //println<vector<int>>(c1);
    println(c2); //println<list<int>>(c2);
}
```

Algorithms

- STL provides most common algorithms (e.g., sort, search) on elements store in a container
  – An algorithm is a function template operating on sequences of elements
- Iterators are used to identify input and/or output
  – Two iterators are often used to specify range of input
  – Algorithm may return an iterator, a value, or modify elements in an output iterator (e.g., `copy()`)
- Some algorithms (e.g., `replace(), sort()`) modify elements in a container, but no algorithm add or remove elements of a container
- STL library provides *generic programming*, i.e., a style of computer programming in which algorithms are written in terms of types to-be-specified-later that are then instantiated when needed for specific types provided as parameters

Selected algorithms

`p=find(b,e,x)`	p is the first p in [b:e) so that *p==x
`p=find_if(b,e,f)`	p is the first p in [b:e) so that f(*p)==true
`n=count(b,e,x)`	n is the number of elements *q in [b:e) so that *q==x
`n=count_if(b,e,f)`	n is the number of elements *q in [b:e) so that f(*q,x)
`replace(b,e,v,v2)`	Replace elements *q in [b:e) so that *q==v by v2
`replace_if(b,e,f,v2)`	Replace elements *q in [b:e) so that f(*q) by v2
`p=copy(b,e,out)`	Copy [b:e) to [out:p)
`p=copy_if(b,e,out,f)`	Copy elements *q from [b:e) so that f(*q) to [out:p)
`p=move(b,e,out)`	Move [b:e) to [out:p)
`p=unique_copy(b,e,out)`	Copy [b:e) to [out:p); don't copy adjacent duplicates
`sort(b,e)`	Sort elements of [b:e) using < as the sorting criterion
`sort(b,e,f)`	Sort elements of [b:e) using f as the sorting criterion
`(p1,p2)=equal_range(b,e,v)`	[p1:p2) is the subsequence of the sorted sequence [b:e) with the value v; basically a binary search for v
`p=merge(b,e,b2,e2,out)`	Merge two sorted sequences [b:e) and [b2:e2) into [out:p)

Example: algorithms

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>

using namespace std;
template<typename T>

int main(int argc, char** argv) {
    vector<int> values {42, 0, 23, 0, 6, 1024, 0, 0};
    vector<int>::iterator it;
    cout << "values: ";
    println(values); // 42 0 23 0 6 1024 0 0
    ...
```

Example: algorithms (cont.)

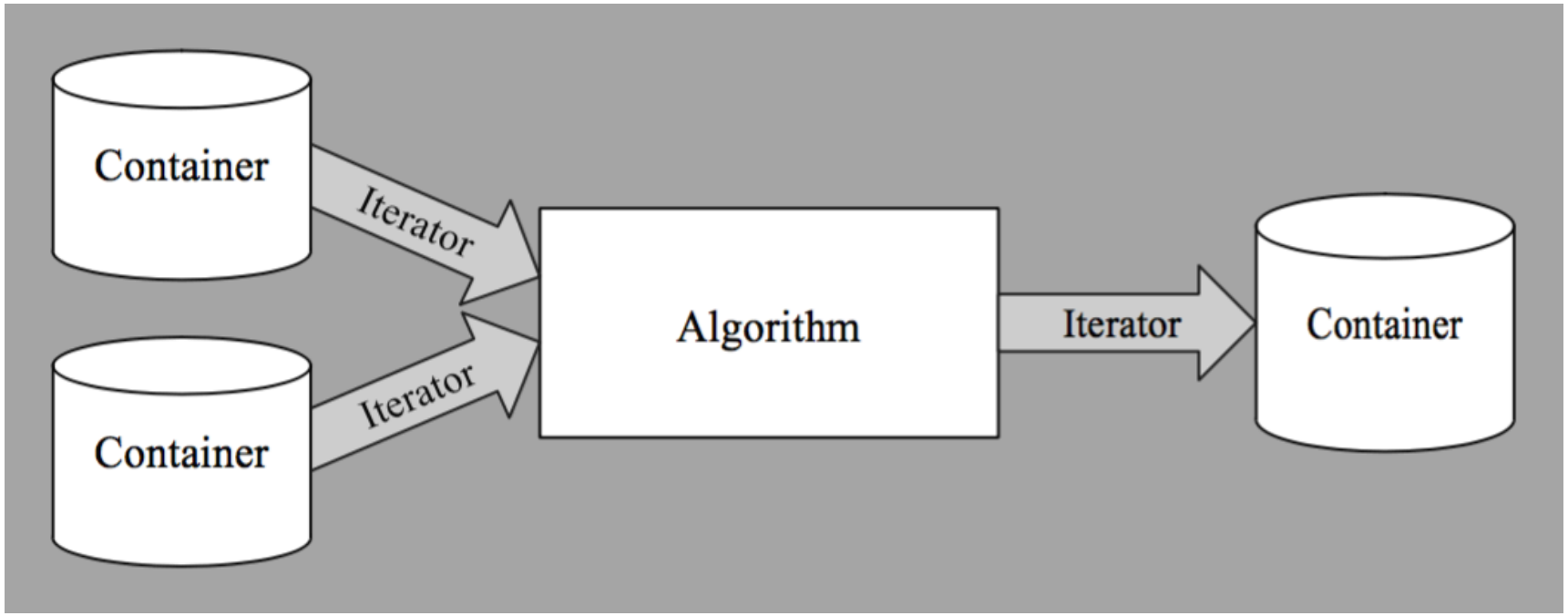
```
cout << "zero count: " << count(values.begin(), values.end(), 0) << endl; // 4
cout << "indices of 0: ";
it = find(values.begin(), values.end(), 0);
for (; it != values.end(); it = find(++it, values.end(), 0))
    cout << it - values.begin() << " ";
cout << endl; // 1 3 6 7

it = values.begin();
replace(it, it + 3, 0, 1);
cout << "after replace: ";
println(values); // 42 1 23 0 6 1024 0 0

vector<int> values2(8);
copy(it, it + 3, values2.begin());
cout << "values2: ";
println(values2); // 42 1 23 0 0 0 0 0
```

Example: algorithms (cont.)

```
random_shuffle(values.begin(), values.end());
cout << "random order: ";
println(values); // 6 0 0 1 23 0 42 1024

sort(values.begin(), values.end());
cout << "sorted order: ";
println (values); // 0 0 0 1 6 23 42 1024

//auto range = equal_range(values.begin(), values.end(), 0);
pair<vector<int>::iterator, vector<int>::iterator> range = equal_range(values.begin(),
                                                                       values.end(),
                                                                       0);

cout << range.first - values.begin() << " "
     << range.second - values.begin() << endl;  // 0 3
}
```

Recap: STL components and interactions

- *Containers* manage collection of objects (e.g., `vector`, `list`, `map`)
- *Algorithms* process the elements in containers (e.g., `sort`, `search`)
- *Iterators* step through the elements in containers



Revisit: initialization of vector

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

int main()
{
    vector<int> v1 {1, 2, 3, 4}; // size is 4
    vector<string> v2;           // size is 0
    vector<Shape*> v3(23);       // size is 23;
                                 // initial element value: nullptr
    vector<double> v4(32,9.9);   // size is 32;
                                 // initial element value: 9.9
}
```

* From “Tour of C++”

initializer_list

- Declaration

```
template<typename T> class initializer_list;
```

- A lightweight proxy object that provides access to an array of objects of `type const T`, used for
  - Initialization of a variable or an object (often for containers)
  - Passing several elements (i.e., a list of values) as a function argument
- A constructor that takes a single argument of type `std::initializer_list` is called an *initializer-list constructor*
- Standard-library containers (e.g., vector and map) have initializer_list constructors, assignments, etc.

Example: initializer_list

```
#include <iostream>
#include <vector>
#include <initializer_list>
using namespace std;
template <typename T>
struct S {
    vector<T> v;
    S(initializer_list<T> l) : v(l) {
         cout << "constructed with a " << l.size() << "-element list\n";
    }
    void append(initializer_list<T> l) { v.insert(v.end(), l.begin(), l.end()); }
    pair<const T*, size_t> c_arr() const {
        return {&v[0], v.size()};  // copy list-initialization in return statement
                                   // this is NOT a use of initializer_list
    }
};
```

* http://en.cppreference.com/w/cpp/utility/initializer_list

Example: initializer_list (cont.)

```
int main() {
    S<int> s = {1, 2, 3, 4, 5}; // copy list-initialization
    s.append({6, 7, 8});      // list-initialization in function call
    cout << "The vector size is now " << s.c_arr().second << " ints:\n";

    for (auto n : s.v)
        cout << n << ' ';
    cout << '\n';

    cout << "Range-for over brace-init-list: \n";
    for (int x : {-1, -2, -3}) // the rule for auto makes this ranged-for work
        cout << x << ' ';
    cout << '\n';

    auto al = {10, 11, 12};   // special rule for auto
    cout << "The list bound to auto has size() = " << al.size() << '\n';
}
```

Reference: `vector::vector`

Constructor		Version
`explicit vector(const Allocator& alloc = Allocator());`	(1)	until C++14
`vector() : vector(Allocator()) {}` `explicit vector(const Allocator& alloc);`	(1)	since C++14

`explicit vector(size_type count,`
`const T& value = T(),`
`const Allocator& alloc = Allocator());`
	(2)	until C++11

`vector(size_type count,`
`const T& value,`
`const Allocator& alloc = Allocator());`
	(2)	since C++11

`explicit vector(size_type count);`	(3)	since C++11 until C++14
`explicit vector(size_type count,` `const Allocator& alloc = Allocator());`	(3)	since C++14

* http://en.cppreference.com/w/cpp/container/vector/vector

Reference: `vector::vector` (cont.)

Constructor		Version

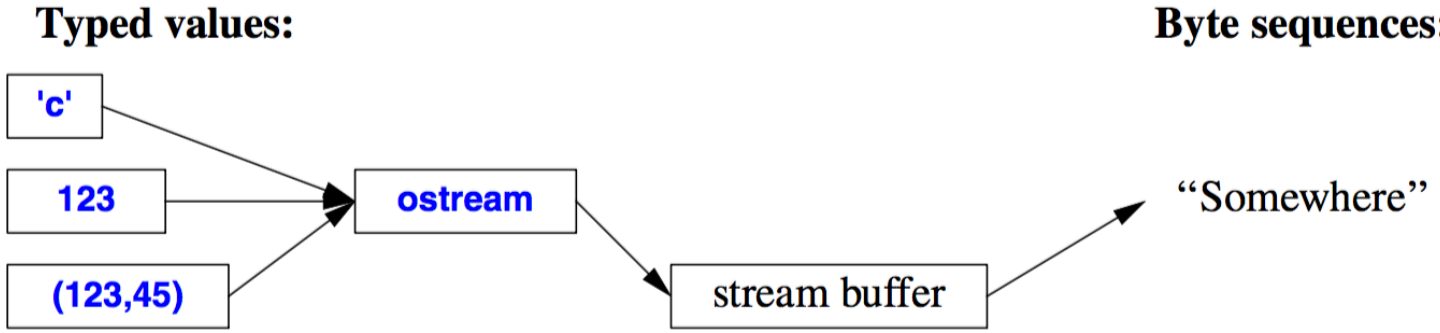
`template<typename InputIt>`
`vector(InputIt first, InputIt last,`
`  const Allocator& alloc = Allocator());`
	(4)	

`vector(const vector& other);`	(5)	
`vector(const vector& other, const Allocator& alloc);`	(5)	since C++11
`vector(vector&& other);`	(6)	since C++11
`vector(vector&& other, const Allocator& alloc);`	(6)	since C++11
`vector(std::initializer_list<T> init,` `  const Allocator& alloc = Allocator());`	(7)	since C++11

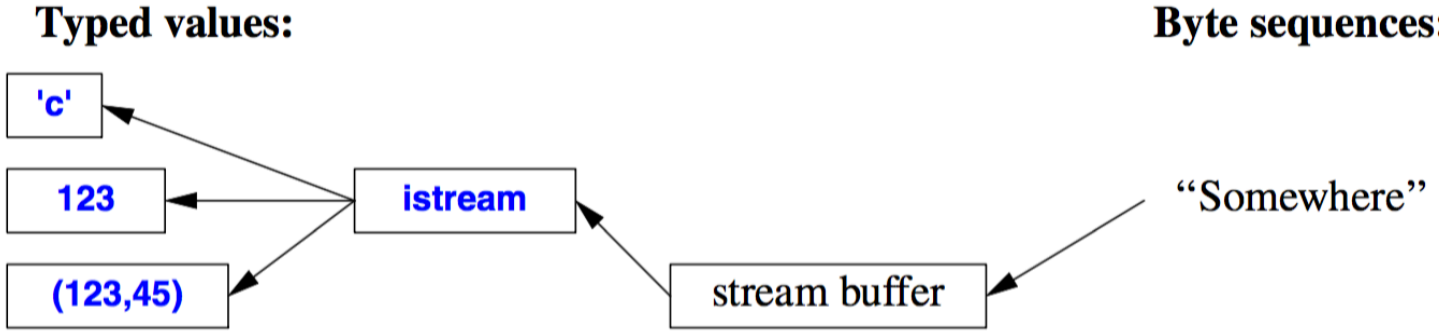
* http://en.cppreference.com/w/cpp/container/vector/vector

I/O Streams

- The I/O stream library provides formatted and unformatted buffered I/O of text and numeric values
- `ostream` converts typed objects to a stream of characters (bytes)



- `istream` converts a stream of characters (bytes) to typed objects



* From "A tour of C++"

Example: I/O stream

```
void readword(istream& is) {
    string buffer;
    int line = 0;
    while (is) {
        is >> buffer;
        cout << ++line << ": " << buffer << endl;
    }
}
void readline(istream& is)
{
    string buffer;
    int word = 0;
    while (is) {
        getline(is, buffer);
        cout << ++word << ": " << buffer << endl;
    }
}
```

```
void main()
{
    readword(cin);
    // or
    // readline(cin);
}
```

File stream

- The C++ standard library provides I/O streams for file input/output (defined in `<fstream>`)
  - `ifstream`: for reading from a file
  - `ofstream`: for writing to a file
  - `fstream`: for reading from and writing to a file
- Example

```
string str = "Hello";
ofstream ofs("testfile.txt");
if (!ofs)
    cout << "cannot open a file\n";
ofs << str;
ofs.close(); // not necessary if ofs is automatically destroyed
ifstream ifs("testfile.txt");
if (!ifs)
    cout << "cannot open a file\n";
ifs >> str;
```

Example: file stream

```
int main() {
    string str = "Hello";
    int d = 42;
    double pi = 3.14;
    ofstream ofs("testfile.txt");
    ofs << str << " " << d << " " << pi << endl;
    ofs.close(); // manual file close
                 // c.f., file is automatically closed when ofs is destroyed
    ifstream ifs("testfile.txt");
    ifs >> str >> d >> pi;
    cout << "str: " << str << endl << "d: " << d << endl << "pi: " << pi << endl;
    ifs.close();

    ifs.open("fstream.cpp");
    readline(ifs);
}
```

String stream

- The C++ standard library provides I/O streams based on a string (defined in `<sstream>`)
  - `istringstream`: for reading from a string
  - `ostringstream`: for writing to a string
  - `stringstream`: for reading from and writing to a string

Example: string stream

```
using namespace std;
int main() {
    // default constructor (input/output stream)
    stringstream buf1;
    buf1 << 7;
    int n = 0;
    buf1 >> n;
    cout << "buf1 = " << buf1.str() << " n = " << n << '\n'; // buf1 = 7 n = 7
    // input stream
    istringstream inbuf("-10");
    inbuf >> n;
    cout << "n = " << n << '\n'; // n = -10
    // output stream in append mode (C++11)
    ostringstream buf2("test", ios_base::ate);
    buf2 << '1';
    cout << buf2.str() << '\n'; // test1
}
```

istream_iterator

- Declaration

```
template<class T, class CharT = char, class Traits = std::char_traits<CharT>,
         class Distance = std::ptrdiff_t >
class istream_iterator:
    public std::iterator<std::input_iterator_tag, T, Distance, const T*, const T&>
```

- A single-pass input iterator that reads object of type T from basic_istream object
- Constructors

Constructor	Note
`istream_iterator();`	Construct an end-of-stream iterator
`istream_iterator(istream_type& stream);`	Associate an iterator with the given stream
`istream_iterator(const istream_iterator& other) = default;`	Copy constructor

ostream_iterator

- Declaration

```
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>
```

- A single-pass input iterator that reads object of type T from `basic_istream` object
- Constructors

Constructor	Note
`ostream_iterator();`	Construct an end-of-stream iterator
`ostream_iterator(istream_type& stream);`	Associate an iterator with the given stream, with delim as a delimiter

Putting it all together; separate words from inputs

```
void parser(istream& is) {
    for (string line; getline(is, line); ) {
        if (line == "qq")
            break;

        istringstream iss {line};
        vector<string> words {istream_iterator<string> {iss},
                              istream_iterator<string> {}};

        cout << "tokenized words\n---------------\n";
        for (string& w: words)
            cout << w << endl;
    }
}
int main() {
    parser(cin);
}
```

Reading list

- Learn C++
  - C++ standard library: Ch. 16.3-4

What we will cover next time

- Additional topics on C++
  - Function object
  - Exception handling
  - RAII
  - RTTI

Reading list

- Learn C++
  - C++ standard library: Ch. 9.8
- STL containers
  - List of member functions: http://en.cppreference.com/w/cpp/container
- (Advanced)
  - LLVM C++ standard library implementation: http://libcxx.llvm.org
    - Follow ‘building libc++’ link for download

ANY QUESTIONS?