

CME 211: Lecture 20

Topics:

- C++ containers
- map
- set
- and more

Container iteration

Container iteration example 1

src/iter1.cpp:

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<double> vec;

    vec.push_back(7);
    vec.push_back(11);
    vec.push_back(42);

    for (auto v : vec)
        v++;

    for (auto v : vec)
        std::cout << v << std::endl;

    return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/iter1.cpp -o src/iter1
$ ./src/iter1
7
11
42
```

Container iteration example 2

src/iter2.cpp:

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<double> vec;
```

```

vec.push_back(7);
vec.push_back(11);
vec.push_back(42);

for (auto &v : vec)
    v++;

for (auto v : vec)
    std::cout << v << std::endl;

return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/iter2.cpp -o src/iter2
$ ./src/iter2
8
12
43

```

Map

- A C++ map is analogous to a dictionary in Python
- Need to specify data type for both the key and the value when instance is declared

Our first map

src/map1.cpp:

```

#include <iostream>
#include <map>

int main()
{
    std::map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    std::cout << "dir[2] = " << dir[2] << std::endl;
    std::cout << "dir[0] = " << dir[0] << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map1.cpp -o src/map1
$ ./src/map1
dir[2] = south
dir[0] = north

```

Map iteration

src/map2.cpp:

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

int main()
{
    std::map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    for (auto d : dir)
        std::cout << "d[" << d.first << "] = " << d.second << std::endl;

    for (auto &d : dir)
        std::cout << "d[" << d.first << "] = " << d.second << std::endl;

    return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map2.cpp -o src/map2
$ ./src/map2
d[0] = north
d[1] = east
d[2] = south
d[3] = west
d[0] = north
d[1] = east
d[2] = south
d[3] = west
```

Older style iteration

src/map3.cpp:

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

int main()
{
    std::map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    for (std::map<int, std::string>::iterator i = dir.begin(); i != dir.end(); i++)
```

```

        std::cout << "d[" << i->first << "] = " << i->second << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map3.cpp -o src/map3
$ ./src/map3
d[0] = north
d[1] = east
d[2] = south
d[3] = west

```

Nonexistent keys

src/map4.cpp:

```

#include <iostream>
#include <map>

int main()
{
    std::map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    std::cout << "dir.size() = " << dir.size() << std::endl;
    std::cout << "dir[5] = " << dir[5] << std::endl;
    std::cout << "dir.size() = " << dir.size() << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map4.cpp -o src/map4
$ ./src/map4
dir.size() = 4
dir[5] =
dir.size() = 5

```

Nonexistent keys

src/map5.cpp:

```

#include <iostream>
#include <map>

int main()
{
    std::map<int, std::string> dir;

```

```

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    std::cout << "dir.size() = " << dir.size() << std::endl;
    std::cout << "dir.at(5) = " << dir.at(5) << std::endl;
    std::cout << "dir.size() = " << dir.size() << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map5.cpp -o src/map5
$ ./src/map5
dir.size() = 4
dir.at(5) =
libc++abi.dylib: terminating with uncaught exception of type std::out_of_range: map::at: key not found

```

Testing for a key

src/map6.cpp:

```

#include <iostream>
#include <map>

int main()
{
    std::map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    std::cout << "dir.count(2) = " << dir.count(2) << std::endl;
    std::cout << "dir.count(5) = " << dir.count(5) << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map6.cpp -o src/map6
$ ./src/map6
dir.count(2) = 1
dir.count(5) = 0

```

Testing for a key

src/map7.cpp:

```

#include <iostream>
#include <map>

```

```

int main() {
    std::map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    int key = 2;
    auto iter = dir.find(key);
    if (iter == dir.end()) {
        std::cout << "key " << key << " is not present" << std::endl;
    }
    else {
        std::cout << "key " << key << " is present" << std::endl;
        std::cout << "value is " << iter->second << std::endl;
    }

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map7.cpp -o src/map7
$ ./src/map7
key 2 is present
value is south

```

Key order

src/map8.cpp:

```

#include <iostream>
#include <map>

int main()
{
    std::map<int, std::string> dir;

    dir[2] = std::string("south");
    dir[3] = std::string("west");
    dir[1] = std::string("east");
    dir[0] = std::string("north");

    for (auto &d : dir)
        std::cout << d.first << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map8.cpp -o src/map8
$ ./src/map8
0
1

```

2
3

Map and tuples

src/map9.cpp:

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

int main() {
    std::ifstream f("dist.female.first");
    if (not f.good()) {
        std::cerr << "ERROR: Failed to open file" << std::endl;
        return 1;
    }

    std::map<std::string, std::tuple<double, double, int>> names;

    std::string name;
    double perc1, perc2;
    int rank;
    while(f >> name >> perc1 >> perc2 >> rank) {
        names[name] = std::make_tuple(perc1, perc2, rank);
    }

    for(auto &data : names) {
        std::cout << data.first << " " << std::get<0>(data.second) << std::endl;
    }

    return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/map9.cpp -o src/map9
$ ./src/map9
BARBARA 0.98
DOROTHY 0.727
ELIZABETH 0.937
JENNIFER 0.932
LINDA 1.035
MARGARET 0.768
MARIA 0.828
MARY 2.629
PATRICIA 1.073
SUSAN 0.794
```

Using functions

src/readnames.hpp:

```

#ifndef READNAMES_HPP
#define READNAMES_HPP

#include <map>
#include <string>
#include <tuple>

std::map<std::string, std::tuple<double, double, int>> ReadNames(std::string filename);

#endif /* READNAMES_HPP */

src/readnames.cpp:
#include <fstream>
#include <iostream>

#include "readnames.hpp"

std::map<std::string, std::tuple<double, double, int>> ReadNames(std::string filename)
{
    std::ifstream f(filename);

    std::map<std::string, std::tuple<double, double, int>> names;

    std::string name;
    double perc1, perc2;
    int rank;
    while(f >> name >> perc1 >> perc2 >> rank) {
        names[name] = std::make_tuple(perc1, perc2, rank);
    }

    return names;
}

#pragma once: only include this file once (not standard)

src/testname.hpp:
#pragma once

#include <map>
#include <string>
#include <tuple>

double TestName(std::map<std::string, std::tuple<double, double, int>> names,
                std::string name);

src/testname.cpp:
#include <iostream>

#include "testname.hpp"

double TestName(std::map<std::string, std::tuple<double, double, int>> names,
                std::string name)
{
    double percentage = 0.;

```



```

    auto match = names.find(name);
    if (match != names.end())
    {
        percentage = std::get<0>(match->second);
    }

    return percentage;
}

```

Using functions

src/main.cpp:

```

#include <iostream>
#include <string>
#include <vector>

#include "readnames.hpp"
#include "testname.hpp"

int main()
{
    auto names = ReadNames("dist.female.first");

    std::vector<std::string> tests;
    tests.push_back("LINDA");
    tests.push_back("PETER");
    tests.push_back("DOROTHY");

    for(auto test : tests)
    {
        std::cout << test << " " << TestName(names, test) << std::endl;
    }

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/main.cpp src/readnames.cpp src/testname.cpp -o src/main
$ ./src/main
LINDA 1.035
PETER 0
DOROTHY 0.727

```

Sets

src/set.cpp:

```

#include <algorithm>
#include <fstream>
#include <iostream>
#include <set>
#include <string>

```

```

std::set<std::string> ReadNames(std::string filename)
{
    std::set<std::string> names;

    std::ifstream f(filename);
    if (not f.is_open())
    {
        std::cerr << "ERROR: Could not read file " << filename << std::endl;
        return names;
    }

    std::string name;
    double perc1, perc2;
    int rank;
    while (f >> name >> perc1 >> perc2 >> rank)
    {
        names.insert(name);
    }
    f.close();

    return names;
}

int main()
{
    auto fnames = ReadNames("dist.female.first");
    auto mnames = ReadNames("dist.male.first");

    std::set<std::string> common;

    std::set_intersection(fnames.begin(), fnames.end(), mnames.begin(), mnames.end(),
                          std::inserter(common, common.begin()));

    std::cout << fnames.size() << " female names" << std::endl;
    std::cout << mnames.size() << " male names" << std::endl;
    std::cout << common.size() << " common names" << std::endl;

    return 0;
}

```

Output:

```

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/set.cpp -o src/set
$ ./src/set
ERROR: Could not read file dist.male.first
10 female names
0 male names
0 common names

```

Additional data structures

- `std::array` (C++ 2011)
- `std::list`

- `std::forward_list` (C++ 2011)
- `std::unordered_map` (C++ 2011)
- `std::unordered_set` (C++ 2011)

Array example

src/array.cpp:

```
#include <array>
#include <iostream>

int main()
{
    std::array<double,4> a;

    a.fill(1.);
    a[2] = 3.;

    for (auto val : a)
        std::cout << val << std::endl;

    return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/array.cpp -o src/array
$ ./src/array
1
1
3
1
```

Linked lists

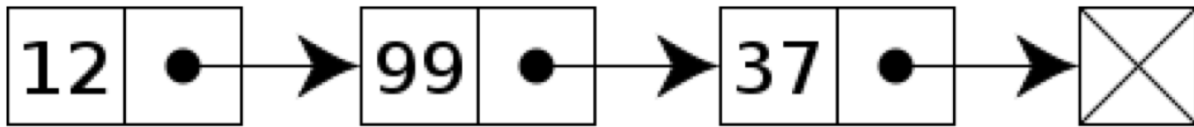
- Ordered data sequence similar to a C++ vector or Python list, but data is not stored contiguously
- Sense of order is maintained via links
- There is additional storage overhead for the links
- But this allows for insertion and removal operations in constant time

List example

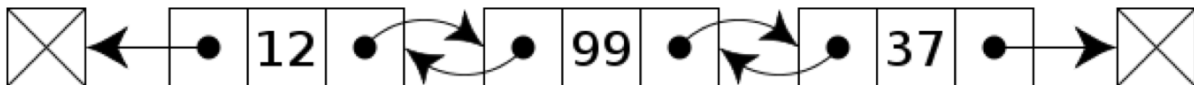
src/list.cpp:

```
#include <iostream>
#include <list>

int main()
{
    std::list<int> l;
    l.push_back(42);
}
```



Singly linked list



Doubly linked list

Figure 1: fig

```
l.push_back(17);
l.push_back(9);

auto it = l.begin();
advance(it, 1);
l.erase(it);

for (auto val : l)
    std::cout << val << std::endl;

return 0;
}

$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/list.cpp -o src/list
$ ./src/list
42
9
```

Maps and sets

- Python dictionaries and sets are internally implemented by using hashing
- For hashing implementation, time complexity for data access is (amortized) constant time
- Instances of C++ `std::map` and `std::set` are internally implemented using a tree data structure
- For a tree, time complexity for data access is $O(\log n)$
- Reference: <http://www.cplusplus.com/reference/map/map/operator%5B%5D/>

Unordered maps and sets

- In the C++ 2011 standard the `std::unordered_map` and `std::unordered_set` were added
- Like Python, internal implementation is based on hashing
- Faster access, but entries are no longer ordered (but that usually doesn't matter)

Unordered map example

src/unordered_map.cpp:

```
#include <iostream>
#include <unordered_map>

int main()
{
    std::unordered_map<int, std::string> dir;

    dir[0] = std::string("north");
    dir[1] = std::string("east");
    dir[2] = std::string("south");
    dir[3] = std::string("west");

    std::cout << "dir[2] = " << dir[2] << std::endl;
    std::cout << "dir[0] = " << dir[0] << std::endl;

    return 0;
}
```

Output:

```
$ clang++ -std=c++11 -Wall -Wextra -Wconversion src/unordered_map.cpp -o src/unordered_map
$ ./src/unordered_map
dir[2] = south
dir[0] = north
```

Reading

- **C++ Primer, Fifth Edition** by Lippman et al.
- Chapter 11: Associative Containers: Sections 11.1 - 11.3