


Math 4997-3

Lecture 3: Iterators, Lists, and using library algorithms

Patrick Diehl 

<https://www.cct.lsu.edu/~pdiehl/teaching/2021/4997/>

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.



Reminder

Lecture 2

What you should know from last lecture

- ▶ Monte Carlo Methods
- ▶ Random numbers
- ▶ Containers like `std::vector`
- ▶ Functions

Iterators `#include<iterator>`

Iterators: `#include<iterator>`

When we know that we access the elements of the vector sequentially, we can let the compiler know that we are doing this by using iterators.

Iterators are values that

- ▶ identifies a container and an element in the container
- ▶ let us access the value stored in that element
- ▶ provides operations for moving between elements
- ▶ are needed for the algorithms of the standard library

Iterating over vectors

Easiest

```
std::vector<int> values;  
for(size_t i = 0 ; i < values.size(); i++)  
    std::cout << values[i] << std::endl;
```

Using the `size_type`¹

```
std::vector<int> values;  
std::vector<int>::size_type i = 0;  
for(; i < values.size(); i++)  
    std::cout << values[i] << std::endl;
```

¹ https://en.cppreference.com/w/cpp/types/size_t

Advanced iterating over vectors

Example

```
for(
    std::vector<int>::const_iterator iter =
    values.begin();
    iter != values.end();
    ++iter
)
{
    std::cout << *iter << std::endl;
}
```

Features

- ▶ `const_iterator` allows read-only access
- ▶ `++iter` increments the iterator to the next element
- ▶ Dereference the iterator `*iter` to access the value

Erasing elements with iterators gets easier

Using the basic way

```
std::vector<int> values = {1,2,3};  
values.erase(values.begin()+i)
```

Using iterators

```
values.erase(iter)
```

Note that with an iterator there is no need to compute the position anymore!

Useful feature

```
iter = values.erase(iter)
```

Returns the iterator pointing to the element after the erasure.

Lists `#include<list>`

Lists vs Vectors

Vectors `#include<vector>`

- ▶ Are sufficient for small amount of elements (around 7000)
- ▶ Is optimized to access elements arbitrary
- ▶ Performs well adding one element by time to its end

Lists `#include<list>`

- ▶ Are slower for small amount of elements
- ▶ Are optimized to insert and delete elements anywhere

Complexity

- ▶ Inserting/Removing: Vector $\mathcal{O}(n^2)$ vs List $\mathcal{O}(n)$ [4, 3]

Example lists²

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

int main()
{
    std::list<double> values;
    double x;
    while (std::cin >> x)
    {
        values.push_back(x);
    }
    double sum =
        std::accumulate(values.begin(), values.end(), 0.0f);
    std::cout << "Average: "
        << sum / values.size() << std::endl;
}
```

²<https://en.cppreference.com/w/cpp/container/list>

Library algorithms `#include<algorithm>`

Sorting⁵

Sorting using <

```
std::sort(s.begin(), s.end());
```

Sorting using >³

```
std::sort(s.begin(), s.end(), std::greater<int>());
```

Advanced sorting using a lambda expression⁴

```
std::sort(s.begin(), s.end(), [](int a, int b) {  
    return a > b; } );
```

We will look into lambda expression later in more detail

³ <https://en.cppreference.com/w/cpp/utility/functional/greater>

⁴ <https://en.cppreference.com/w/cpp/language/lambda>

⁵ <https://en.cppreference.com/w/cpp/algorithm/sort>

Accumulate

Sum⁶

```
int sum = std::accumulate(v.begin(), v.end(), 0);
```

Multiplication⁷

```
int product =  
    std::accumulate(v.begin(), v.end(), 1,  
        std::multiplies<int>());
```

Note that zero is the initial value of the accumulate

Various

- ▶ `std::inner_product`
- ▶ `std::partial_sum`

⁶<https://en.cppreference.com/w/cpp/algorithm/accumulate>

⁷<https://en.cppreference.com/w/cpp/utility/functional/multiplies>

Removing elements

Remove⁸

```
std::list<int> l = { 1,100,2,3,10,1,11,-1,12 };  
l.remove(1); // remove both elements equal to 1
```

Remove_if⁹

```
//Define function  
bool IsOdd (int i) {  
    return ((i%2)==1);  
}  
//Check for the first odd number  
l.remove_if(IsOdd); // remove all odd numbers
```

⁸ <https://en.cppreference.com/w/cpp/algorithm/remove>

⁹ http://www.cplusplus.com/reference/algorithm/remove_if/

Searching for existence of elements

Find¹⁰

```
int n1 = 3;
std::vector<int> v{0, 1, 2, 3, 4};
auto result1 =
    std::find(std::begin(v), std::end(v), n1);
```

Find_if¹¹

```
//Define function
bool IsOdd (int i) {
    return ((i%2)==1);
}
//Check for the first odd number
std::vector<int>::iterator it =
    std::find(std::begin(v), std::end(v), IsOdd);
```

¹⁰ <https://en.cppreference.com/w/cpp/algorithm/find>

¹¹ http://www.cplusplus.com/reference/algorithm/find_if/

Search¹³

Find a substring within a string

Check for the substring

```
std::string name = "Math 4997-3";  
std::string exp = "4997";  
std::search(name.begin(), name.end(),  
            exp.begin(), exp.end()) != name.end();
```

Get the leading position¹²

```
auto it = std::search(name.begin(), name.end(),  
                    std::boyer_moore_searcher(  
                    exp.begin(), exp.end()));  
std::cout << it - name.begin() << std::endl;
```

¹²https://en.cppreference.com/w/cpp/utility/functional/boyer_moore_searcher

¹³<https://en.cppreference.com/w/cpp/algorithm/search>

Copy

Copy the content of vector a to vector b

Without library algorithm

```
typedef std::vector<int>::const_iterator it vit;
for(vit it = a.begin(); it != a.end(); ++it)
{
    b.push_back(*it);
}
```

With library algorithm¹⁴

```
std::copy(a.begin(), a.end(), b.begin());
```

¹⁴<https://en.cppreference.com/w/cpp/algorithm/copy>

Insert

Append the content of vector a to vector b

Without library algorithm

```
typedef std::vector<int>::const_iterator it vit;  
for(vit it = a.begin(); it != a.end(); ++it)  
{  
    b.push_back(*it);  
}
```

With library algorithm

```
b.insert(b.end(), a.begin(), a.end());
```

Filling vectors

Fill vector with one value¹⁵

```
std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};  
//Replace all elements by -1  
std::fill(v.begin(), v.end(), -1);
```

Replacing the first 5 elements¹⁶

```
//Replace the first 5 elements by -1  
std::fill_n(v1.begin(), 5, -1);
```

Appending 5 elements¹⁷

```
//Appending 5 elements  
std::fill_n(std::back_inserter(v), 5, -1);
```

¹⁵ <https://en.cppreference.com/w/cpp/algorithm/fill>

¹⁶ https://en.cppreference.com/w/cpp/algorithm/fill_n

¹⁷ https://en.cppreference.com/w/cpp/iterator/back_inserter

Transform¹⁸

Convert to upper case letters

```
std::string s("hello");
std::transform(s.begin(), s.end(), s.begin(),
[] (unsigned char c) ->
unsigned char { return std::toupper(c); });
```

¹⁸<https://en.cppreference.com/w/cpp/algorithm/transform>

Partition

```
#include<algorithm>
#include<iterator>
int main(){
    std::vector<int> v = {0,1,2,3,4,5,6,7,8,9};
    std::cout << "Original vector:\n    ";
    for(int elem : v) std::cout << elem << ' ';

    auto it = std::partition(v.begin(), v.end(),
        [](int i){return i % 2 == 0;});

    std::cout << "\nPartitioned vector:\n    ";
    std::copy(std::begin(v), it,
        std::ostream_iterator<int>(std::cout, " "));
    std::cout << " * ";
    std::copy(it, std::end(v),
        std::ostream_iterator<int>(std::cout, " "));
    std::cout << std::endl;
}
```

Numeric limits `#include<limits>`

Limits

```
#include <limits>
#include <iostream>

int main()
{
    std::cout << "type\tmin()\t\tmax()\n";
    std::cout << "int\t"
        << std::numeric_limits<unsigned int>::min() << '\t'
        << std::numeric_limits<unsigned int>::max() << '\n';
    std::cout << "int\t"
        << std::numeric_limits<int>::min() << '\t'
        << std::numeric_limits<int>::max() << '\n';
}
```

- ▶ `::min` returns the smallest finite value of the given type
- ▶ `::max` returns the largest finite value of the given type

More details about IEEE 754 [2, 1]

Limits¹⁹

```
#include <limits>
#include <iostream>

int main()
{
    std::cout << "type\tround()\tsteps\tmin()\t\tmax()\n";
    std::cout << "double\t"
        << std::numeric_limits<double>::round_error() << '\t'
        << std::numeric_limits<double>::epsilon() << '\t'
        << std::numeric_limits<double>::min() << '\t'
        << std::numeric_limits<double>::max() << '\n';
}
```

- ▶ `::round_error` returns the maximum rounding error of the given floating-point type
- ▶ `::epsilon` returns the difference between 1.0 and the next representable value of the given type

10

Writing files²¹

```
// basic file operations
#include <iostream>
#include <fstream>

int main () {
    std::ofstream myfile;
    myfile.open ("example.txt", std::ios::out);
    myfile << "Writing this to a file.\n";
    myfile.close();
    return 0;
}
```

Mode²⁰

- ▶ out Open for writing (Default)
- ▶ app Append to the end

²⁰ https://en.cppreference.com/w/cpp/io/ios_base/openmode

²¹ https://en.cppreference.com/w/cpp/io/basic_ofstream

Reading files^{22 23}

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

int main () {
    std::string line;
    std::ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while ( getline (myfile,line) )
        {
            std::cout << line << '\n';
        }
        myfile.close();
    }
    return 0;
}
```

²² https://en.cppreference.com/w/cpp/io/basic_ifstream

²³ https://en.cppreference.com/w/cpp/io/basic_fstream

Summary

Summary

After this lecture, you should know

- ▶ Iterators
- ▶ Lists
- ▶ Library algorithms
- ▶ Numerical limits
- ▶ Reading and Writing files

Further reading:

C++ Lecture 1 - The Standard Template Library²⁴

²⁴https://www.youtube.com/watch?v=asGZTCR53KY&list=PL7vEgTL3Faly2eBxud1wsfz80KvE9sd_z

References

References I

- [1] IEEE Standard for Floating-Point Arithmetic.
IEEE Std 754-2008, pages 1–70, Aug 2008.
- [2] David Goldberg.
What every computer scientist should know about
floating-point arithmetic.
ACM Computing Surveys (CSUR), 23(1):5–48, 1991.
- [3] Donald Ervin Knuth.
*The art of computer programming: Fundamental
Algorithms*, volume 1.
Pearson Education, 1968.

References II

[4] Niklaus Wirth.

Algorithms + Data Structures = Programs.

Prentice Hall PTR, Upper Saddle River, NJ, USA,
1978.