Math 4997-3

Lecture 3: Iterators, Lists, and using library algorithms

Patrick Diehl

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

Notes

Notes

Reminder

Notes

Lecture 2

What you should know from last lecture

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

Notes

Iterators #include<iterator>

Notes

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

Notes

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

Notes

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

Notes

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.

Notes

Lists `#include<list>`

Notes

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]

Notes

Example lists[2]

```
#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

Notes

Library algorithms `#include<algorithm>`

Notes

Sorting[5]

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 expression[4]

```
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

Notes

Accumulate

Sum[6]

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

Multiplication[7]

```
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

Notes

Removing elements

Remove[8]

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

Remove_if[9]

```
//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/

Notes

Searching for existence of elements

Find[10]

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

Find_if[11]

```
//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/

Notes

Search[13]

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 position[12]

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

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

Notes

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 algorithm[14]

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

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

Notes

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());
```

Notes

Filling vectors

Fill vector with one value[15]

```
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 elements[16]

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

Appending 5 elements[17]

```
//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

Notes

Transform[18]

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

Notes

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;
}
```

Notes

Numeric limits #include<limits>

Notes

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]

Notes

Limits[19]

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

int main()
{
  std::cout << "type\tround()\teps\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

19 https://en.cppreference.com/w/cpp/types/numeric_limits

Notes

IO

Notes

Writing files[21]

```
// 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;
}
```

Mode[20]

- `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

Notes

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

Notes

Notes

Summary

Notes

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 Library[24]

[24] https://www.youtube.com/watch?v=asGZTCR53KY&list=PL7vEgTL3FalY2eBxud1wsfz8OKvE9sd_z

Notes

References

Notes

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.

[4] Niklaus Wirth.
Algorithms + Data Structures = Programs.
Prentice Hall PTR, Upper Saddle River, NJ, USA, 1978.

Notes

Notes

Notes

Notes