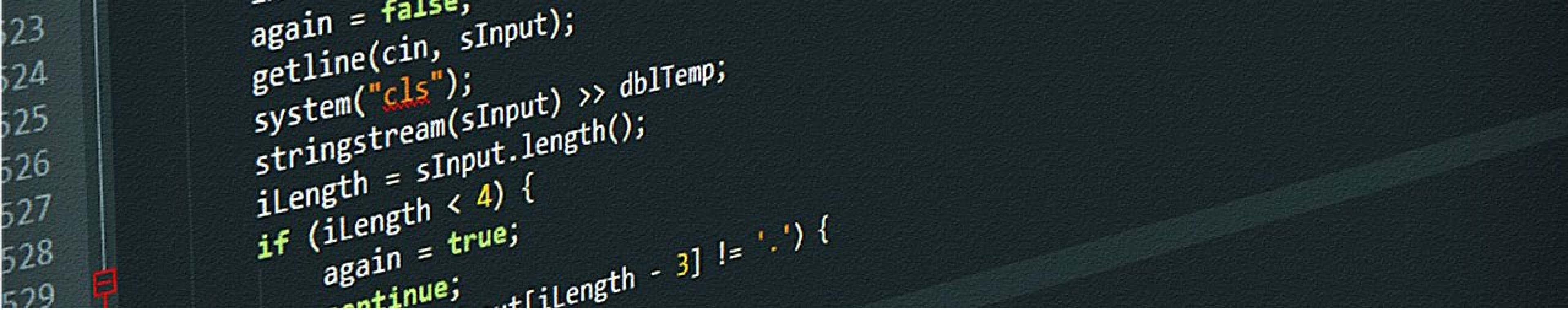


```
again = false;
getline(cin, sInput);
system("cls");
stringstream(sInput) >> dblTemp;
iLength = sInput.length();
if (iLength < 4) {
    again = true;
    ntinue;
            t[iLength - 3] != '.') {
```

Thomas

C23-09.1 Std library containers

Advanced algorithms and programming

v5

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Slides created using material from R. Peters and P. Puschmann / HTW Berlin

Standard library

Containers

Container classes are data structures for storing and effectivly operating on collections of data

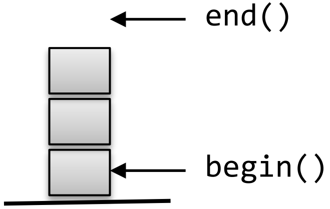
- Containers regulate the lifetime of the contained elements
- Containers provide functions for inserting and removing elements
- Containers provide functions for accessing contained elements, either directly or via iterators

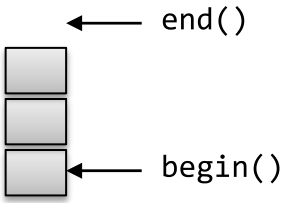
- See also http://www.cplusplus.com/reference/stl/

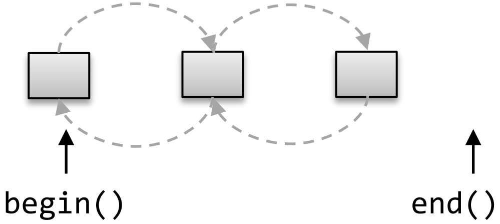
Standard library

Containers

Sequential containers	
`vector`	• Very fast access to the elements in constant time • Very little memory overhead for the container • Optimized to insert / remove data at the end
`deque`	• Relatively fast access to the elements in constant time • Optimized to insert / remove data at the beginning or end • Slightly more optimized than `vector` for frequent insertion / removal of data
`list`	• Optimized to insert / remove data at any position • Access to the elements only via iterators (max. access time increases linearly with the number of elements) • More memory overhead per element than with `vector` or `deque`



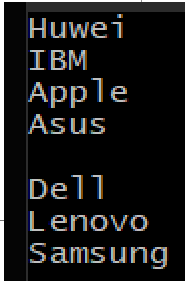




Standard library

Containers – simple vector example

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

int main() {
    std::vector<std::string> companies{ "Huwei", "IBM" };
    companies.push_back("Apple");
    companies.push_back("Asus");
    for (auto e : companies)
        std::cout << e << std::endl;

    companies.clear(); // removes all elements
    std::cout << std::endl;

    companies.push_back("Dell");
    companies.push_back("Lenovo");
    companies.push_back("Samsung");

    // size_t is an unsigned int
    for (size_t i = 0; i < companies.size(); i++)
    {
        std::cout << companies[i] << std::endl;
    }
}
```

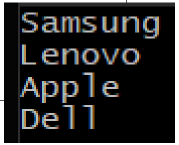


```
Huwei
IBM
Apple
Asus

Dell
Lenovo
Samsung
```

Standard library

Containers – simple deque example (with pointers)

```
#include <iostream>
#include <string>
#include <deque>

int main() {
    std::deque<std::string*> companies;
    companies.push_back(new std::string("Apple"));
    companies.push_back(new std::string("Dell"));
    companies.push_front(new std::string("Lenovo"));
    companies.push_front(new std::string("Samsung"));
    for (auto p : companies)
        std::cout << *p << std::endl;

    // clear() does not free the memory of the pointers
    // so we must call delete for each of the pointers
    for (size_t i = 0; i < companies.size(); i++) {
        delete companies[i];
    }
}
```

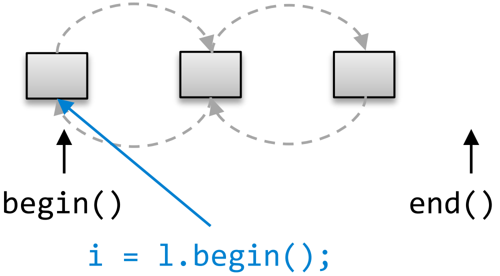


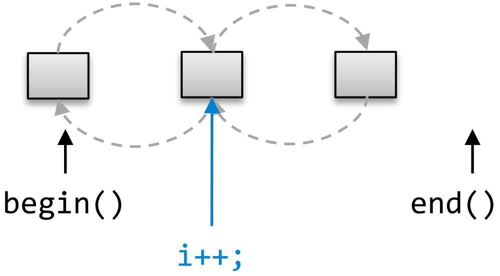
```
Samsung
Lenovo
Apple
Dell
```

Standard library
Containers – iterators

- For all containers you can also access the elements using "iterators"
- For some containers (e.g., `list`) access via iterators is even the only possibility

- Iterators treat container elements as list of connected elements and allow access to elements by dereferencing the iterator, like a pointer
- If an iterator is incremented, it refers to the next element





Standard library

Containers – iterators

- Iterators are set via member functions of the containers

`begin()`	Points to the first element in the container
`end()`	Points to the end of the container (after the last element)
`rbegin()`	Last container element goes through the elements in reverse order
`rend()`	End of the container in reverse order (before the first element)
`cbegin()`, `cend()`, `crbegin()`, `crend()`	Same as above, but as `const_iterator`

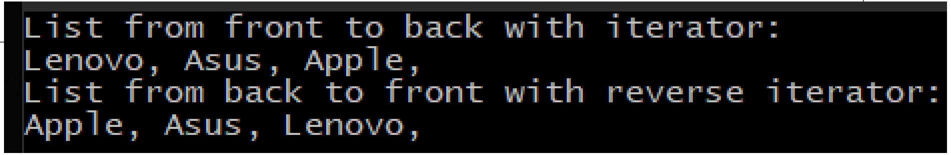
- Many member functions of the containers require iterators instead of concrete elements or position indices for inserting, removing or accessing elements
- Algorithms for containers in the standard library (e.g., `sort`) work only with iterators and are therefore generally applicable to all containers

Standard library

Containers – List example with iterators

(1/2)

```cpp
#include <iostream>
#include <string>
#include <list>

int main() {
    std::list<std::string> companies;
    companies.push_back("Apple");
    companies.push_front("Asus"); // this will be added to the front
    companies.push_front("Lenovo"); // and this too

    std::cout << "List from front to back with iterator: " << std::endl;
    for (std::list<std::string>::iterator it = companies.begin(); it != companies.end(); it++)
        std::cout << *it << ", "; // Access element by dereferencing the iterator

    std::cout << "\nList from back to front with reverse iterator: " << std::endl;
    for (std::list<std::string>::reverse_iterator it = companies.rbegin(); it != companies.rend(); it++)
        std::cout << *it << ", ";

    std::cout << std::endl;
}
```



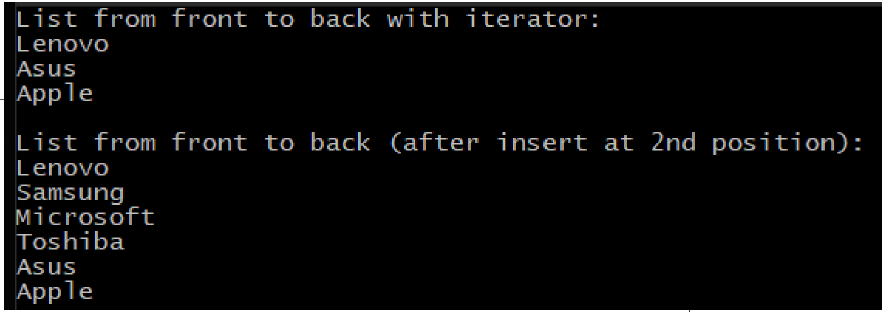
```
List from front to back with iterator:
Lenovo, Asus, Apple,
List from back to front with reverse iterator:
Apple, Asus, Lenovo,
```

Standard library

Containers

(2/2)

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

int main() {
    std::list<std::string> companies;
    companies.push_back("Apple");
    companies.push_front("Asus"); // note: this will be added to the front
    companies.push_front("Lenovo"); // and this too

    std::cout << "List from front to back with iterator: " << std::endl;
    for (std::list<std::string>::iterator it = companies.begin(); it != companies.end(); it++)
        std::cout << *it << std::endl; // Access element by dereferencing the iterator

    std::vector<std::string> vec{ "Samsung", "Microsoft", "Toshiba" };

    // Insert only works via iterators: insert(destination position, source start, source end)
    std::list<std::string>::iterator pos = companies.begin();
    pos++; // let iterator point to 2nd position in list
    companies.insert(pos, vec.begin(), vec.end());

    std::cout << "\nList from front to back (after insert at 2nd position): " << std::endl;

    for (std::list<std::string>::iterator it = companies.begin(); it != companies.end(); it++)
        std::cout << *it << std::endl;
}
```



```
List from front to back with iterator:
Lenovo
Asus
Apple

List from front to back (after insert at 2nd position):
Lenovo
Samsung
Microsoft
Toshiba
Asus
Apple
```

Standard library

Containers

Sequential containers	
`stack`	- LIFO (last in – first out) container, where elements are inserted and extracted only from one end of the container
`queue`	- Container implementing the functionality of a queue with FIFO (first in – first out) access - Elements are inserted on the back of the container and popped from the front

Standard library

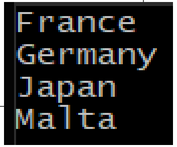
Containers

Associative containers	
`set`	• Access via a key, not via the position • Saves each element only once (key = value) • Elements are stored in sorted order
`map`	• Access via a key, not via the position • key, value - pairs (each value can be addressed by a key) • Each key may only occur once (keys are unique) • Elements are sorted by key

Standard library

Containers – set example

```cpp
#include <iostream>
#include <string>
#include <set>

int main() {
    std::set<std::string> countries;
    countries.insert("Malta");
    countries.insert("Germany");
    countries.insert("France");
    countries.insert("Japan");

    //  access (and demonstrate that set is sorted)
    for (auto e : countries)
        std::cout << e << std::endl;
}
```

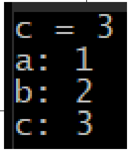


```
France
Germany
Japan
Malta
```

Standard library

Containers – map example

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

int main()
{
    std::map<char, int> numbers;
    numbers['a'] = 1;
    numbers[‘c'] = 2;
    numbers[‘b'] = 3;

    // access via key ...
    std::cout << "c = " << numbers['c'] << std::endl;

    // or access via iterators  (this also demonstrates that map is sorted)
    for (std::map<char, int>::iterator it = numbers.begin(); it != numbers.end(); it++)
    {
        // it points to an std::pair<KEY, VALUE> object!
        char key = it->first;
        int val = it->second;
        std::cout << key << ": " << val << std::endl;
    }
}
```



```
c = 3
a: 1
b: 2
c: 3
```

htw Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de