Lecture 13

C++ Programming

Arne Kutzner

Hanyang University / Seoul Korea

C++ Standard Template Library (STL)

Basic Contents of C++ STL

- Set of classes and implementations of algorithms for:
  - Strings and Character string manipulation
  - Collections of objects (e.g. lists, sets, maps)
  - Input / Output
  - Standard algorithms (Sorting, Searching, Merging etc.)
  - Mathematical types and algorithms (complex, min/max, permutations)

The **`string`** class

- Encapsulates a **`char*`** and the associated string-handling functionality
- Conversion to and from **`char*`**
- Concatenation, append, insert, replace, substring extraction
- Copy constructor
- I/O
- Length and capacity
- Comparison
- Assignment
- Access to individual chars
- Search for substrings

► Examples in file **`STL-string.cpp`**

Sequence Container Class Templates

- Variations on a linear sequence of values
- Templated on the type of element they store
- Operations (Selection)
  - Pushing and popping of elements
  - Insertion and deletion
  - Random access (not to all container types)
  - Reverse
  - Iterative Access

Sequence Container Overview

Library Name	Description	Example
`<vector>`	A dynamic array	STL-vector.cpp
`<list>`	randomly changing sequence of items	STL-list.cpp
`<stack>`	A sequence of items with pop and push at one end only (LIFO)	-
`<queue>`	A Sequence of items with pop and push at opposite ends (FIFO)	-
`<deque>`	Double Ended Queue with pop and push at both ends	STL-deque.cpp

Iterators

- Act like pointers which can only step through a container.
- Most containers allow you to step through values by returning an iterator
- "Find" operations on a container usually return an iterator, which steps through all matching values
- Many generic algorithms provided which make use of iterators

Iterators / Example

```
deque<string> names;

names.push_back("Kim");
names.push_back("Lee");
names.push_back("Gang");
names.push_back("Park");

deque<string>::iterator next = names.begin();
deque<string>::iterator last = names.end();

for (; next != last; next++)
    cout << next->data() << endl;
```

Definition of the iterator-type for deque<string> is in deque<string>

► Example in file **`STL-iterator.cpp`**

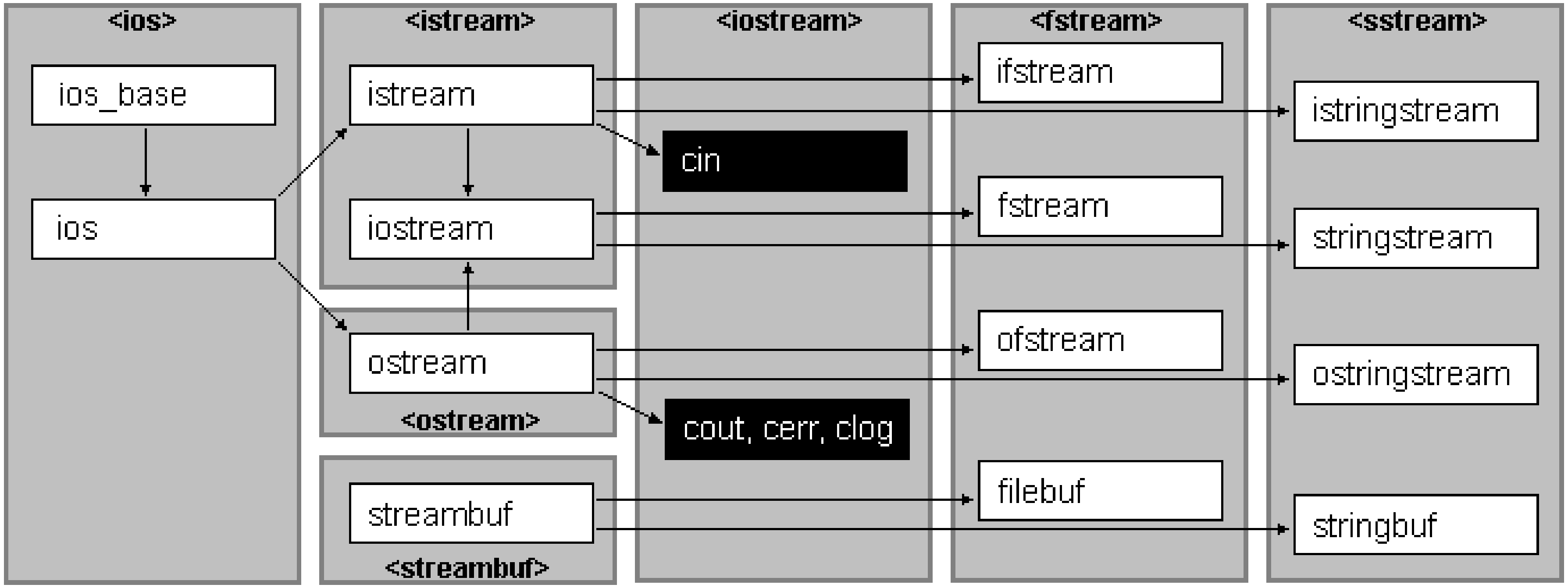
Associative Container Classes

- Sequences that allow us to use keys other than integers
- Several different forms, e.g.:
  - maps (give a key, get/set a value)
  - sets (check whether inside some set or not)
- Operations (Selection)
  - Insertion and deletion
  - Iterative Access
  - Check for existence of some element

Associative Container Overview

Library Name	Description	Example
`<set>`	An unordered collection of items	-
`<map>`	An collection of pairs of items indexed by the first one	`STL-map.cpp`
`<bitset>`	A subset of a fixed and small set of items (The class provides something like an array of bits / optimizing for space allocation)	-

`iostream` Library



- The **`iostream`** library is an object-oriented library that provides input and output functionality using streams.

`iostream` Classes

- **`istream/ostream`**
  for terminal I/O
- **`ifstream/ofstream`**
  for file I/O
- **`istringstream/ostringstream`**
  for I/O operations into/out of strings

`ofstream` Example:

These 2 flags indicate that we want to open the file "test.txt" for writing, where we append at the end of the file

```
#include <fstream>
…
ofstream outfile;
outfile.open ("test.txt",
 ofstream::out | ofstream::app);
outfile << "This sentence is
 appended to the file content\n";
outfile.close();
…
```

Algorithms (include **`<algorithm>`**)

- Non-modifying sequence operations Examples:
  - **`find`**:Find value in range
  - **`count`**: Count appearances of value in range
  - **`equal`**: Test whether the elements in two ranges are equal
  - **`search`**: Find subsequence in range
  - **`for_each`** : Apply function to range
    - allows some form of “higher order programming” in C++
    - Takes some function as argument

► Example in file **`STL-for-each.cpp`**

Algorithms (include **`<algorithm>`**)

- Modifying sequence operations Examples:
  - **`copy`**: Copy range of elements
  - **`swap`**: Exchange values of two objects
  - **`replace`**: Replace value in range
  - **`rotate`**: Rotate elements in range
  - **`transform`** : Apply function to range
    - In contrast to **`for_each`** you can change the elements of the sequence
    - Takes some function as argument

► Example in file **`STL-transform.cpp`**

Algorithms (include **`<algorithm>`**)

- Sorting / Examples:
  - **`sort`** :sort elements in range
    ► Example in file **`STL-sort.cpp`**
  - **`stable_sort`**: Sort elements preserving order of equivalents
- Binary search / Examples:
  - **`merge`** :Merge sorted ranges
  - **`inplace_merge`**: Merge consecutive sorted ranges without need for external space
    ► Example in file **`STL-merge.cpp`**
- Priority Heaps + Heap Operations
- Min / Max Computation