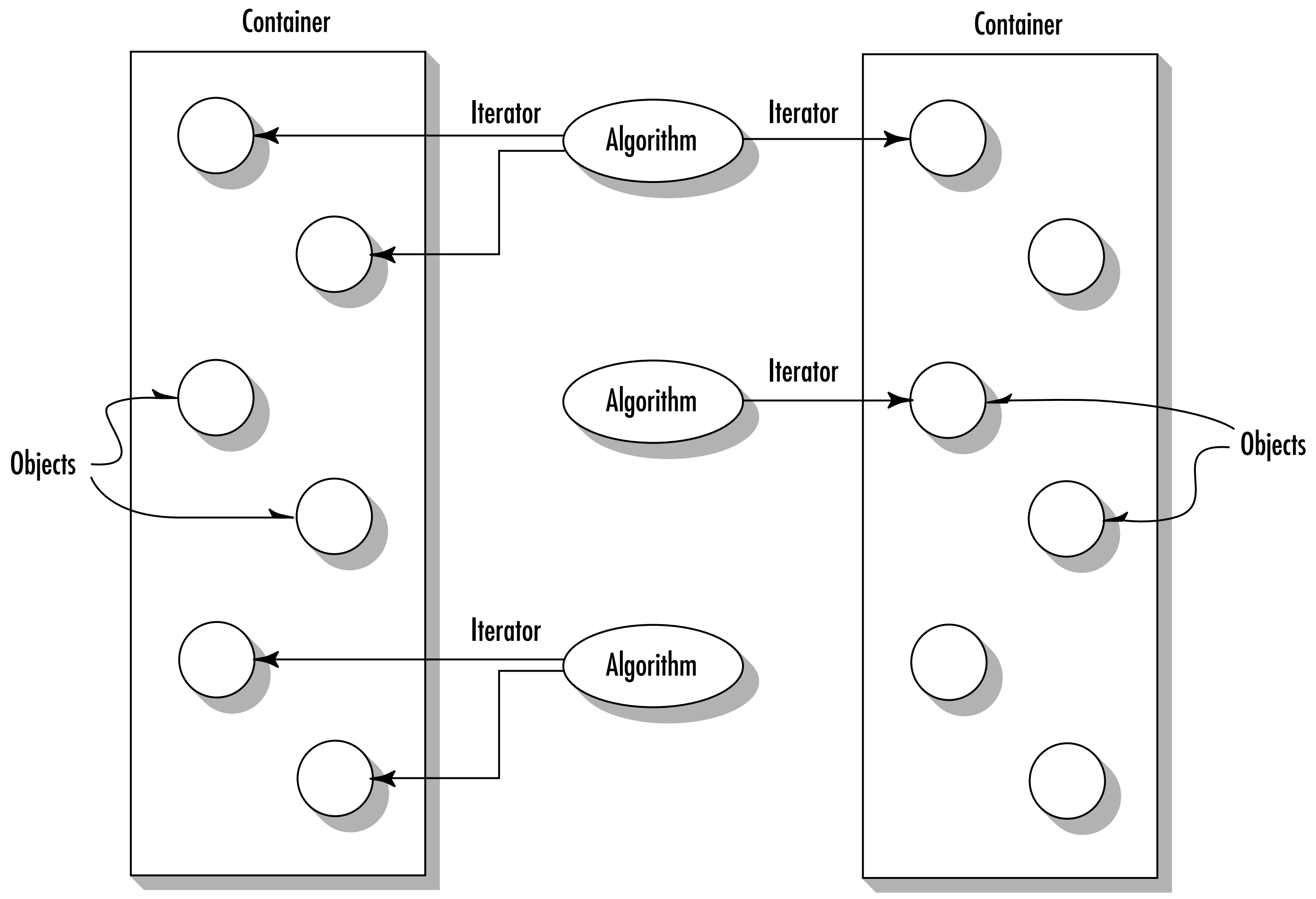
STL

Introduction

- The C++ Standard Template Library (STL) provides standard ways for storing and processing data.
- Three main STL components:
  - Containers
  - Algorithms
  - Iterators

Containers, Algorithms and Iterators



Algorithms use iterators to act on objects in containers

Containers

- Containers store data, which can be either built-in types or user-defined class objects.
  - Each container class is a template.
- Two main types of containers
  - Sequence containers: array, vector, list, deque
  - Associative containers: set, multiset, map, multimap
- Container adapters: special purpose containers, derived from base containers.
  - Stack, queue, priority queue.

Base sequence containers

Container	Characteristic	Advantages and Disadvantages
ordinary C++ array	Fixed size	Quick random access Slow to insert or erase in middle Size cannot be changed at runtime
vector	Expandable array	Quick random access Slow to insert or erase in middle Quick to insert or erase at end
list	Doubly linked list	Quick to insert or delete at any location Quick to access at both ends Slow random access
deque	Like vector	Quick random access Slow to insert or erase in middle Quick to insert or erase at end

Member functions common to all containers

Name	*Purpose*
`size()`	Returns the number of items in the container
`empty()`	Returns true if container is empty
`max_size()`	Returns size of the largest possible container
`begin()`	Returns an iterator to the start of the container, for iterating forwards through the container
`end()`	Returns an iterator to the past-the-end location in the container, used to end forward iteration
`rbegin()`	Returns a reverse iterator to the end of the container, for iterating back-ward through the container
`rend()`	Returns a reverse iterator to the beginning of the container; used to end backward iteration

list container

- A templated doubly linked list
- Includes functions such as push_front, push_back, insert, erase, etc.
  - Same interface as the List implemented in Lab Assignment 4.
- Also includes functions such as reverse, merge, unique.
  - `reverse()` reverses the list.
  - `merge()` merges two sorted list, ensuring the output is also sorted.
  - `unique()` removes adjacent elements with the same value.

Prints: 10 15 20 25 30 35 40

```cpp
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> l1, l2;
  int arr1[] = {40, 30, 20, 10};
  int arr2[] = {15, 20, 25, 30, 35};
  for (auto e: arr1)
    l1.push_back(e);
  for (auto e: arr2)
    l2.push_back(e);

  l1.reverse();
  l1.merge(l2);
  l1.unique();

  for (auto e: l1)
    cout << e << " ";
  cout << endl;
}
```

Container adapter: priority_queue

- Special-purpose container implementing a max heap.
- Simpler interface
  - push, top, pop, empty, size
  - No iterator
- Internally uses a sequence container which can also be explicitly specified.
  - Sequence container must support random access.
  - Either array, vector or deque.

priority_queue: Example-1

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

using namespace std;

int main()
{
  auto data = {1, 8, 5, 6, 3, 4, 0, 9, 7, 2};
  priority_queue<int> q1;
  for (auto e : data)
    q1.push(e);

  cout << q1.top() << endl;

 }
```

Prints:
9

priority_queue: Example-2

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

using namespace std;

int main()
{
  auto data = {1, 8, 5, 6, 3, 4, 0, 9, 7, 2};
  priority_queue<int, vector<int>> q2(data.begin(), data.end());

  for(; !q2.empty(); q2.pop())
    cout << q2.top() << " ";
  cout << endl;

 }
```

Underlying sequence container list won’t work

Prints: 9 8 7 6 5 4 3 2 1 0

priority_queue: Example-3

```
class IntCell{
public:
    explicit IntCell(int initialValue=0)
        : storedValue(initialValue) {}
    int read() const {return storedValue;}
    void write(int x) {storedValue = x;}
private:
    int storedValue;
};
```

```
int main()
{
  auto data = {1, 8, 5, 6, 3, 4, 0, 9, 7, 2};

  priority_queue<IntCell, vector<IntCell>, CompareCells> q3;
  for (auto e : data)
    q3.push(IntCell(e));

  for(; !q3.empty(); q3.pop())
    cout << q3.top().read() << " ";
  cout << endl;
}
```

Functor or Function Object

```
class CompareCells
{
  public:
  bool operator() (const IntCell & c1, const IntCell & c2)
  {return c1.read() > c2.read(); }
};
```

Prints: 0 1 2 3 4 5 6 7 8 9

Associative Containers

- Associative containers are not sequential; they use keys to access data.
- Two kind of associative containers: sets and maps.
  - Both are internally implemented using trees.
  - Allows for efficient searching, insertion and deletion

Set

- Stores unique values
  - Cannot change elements once stored.
- Interface functions: insert, erase, clear, find, upper_bound, lower_bound.
  - Also provides iterators.

```cpp
#include <iostream>
#include <set>
using namespace std;

int main()
{
  set<int> midsemMarks;

  midsemMarks.insert(60);
  midsemMarks.insert(40);
  midsemMarks.insert(100);
  midsemMarks.insert(100);
  midsemMarks.insert(0);

  for(auto e : midsemMarks)
    cout << e << " ";
  cout << endl;
}
```

Prints: 0 40 60 100

Set: Example-2

```
#include <iostream>
#include <set>
using namespace std;

int main()
{
  set<int, greater<int>> midsemMarks;

  midsemMarks.insert(60);
  midsemMarks.insert(40);
  midsemMarks.insert(100);
  midsemMarks.insert(100);
  midsemMarks.insert(0);

  for(auto e : midsemMarks)
    cout << e << " ";
  cout << endl;
}
```

Prints: 100 60 40 0

Multiset

- Allows duplicate values
  - Stored values cannot be modified.

```
#include <iostream>
#include <set>
using namespace std;

int main()
{
  multiset<int, greater<int>> midsemMarks;

  midsemMarks.insert(60);
  midsemMarks.insert(40);
  midsemMarks.insert(100);
  midsemMarks.insert(100);
  midsemMarks.insert(0);

  for(auto e : midsemMarks)
    cout << e << " ";
  cout << endl;
}
```

Prints: 100 100 60 40 0

map

- Stores key-value pairs.
  - Key is unique
- Internally implemented using Red-Black Trees
- Interface contains functions such as find, count, clear, erase, etc.
  - Also supports array-like indexing with keys.

```cpp
#include <iostream>
#include <map>
#include <utility>
using namespace std;

int main()
{
  map<int, int> midsemMarks;

  midsemMarks.insert(pair<int,int> (1,60));
  midsemMarks.insert(pair<int,int> (2,40));
  midsemMarks.insert(pair<int,int> (3,100));
  midsemMarks.insert(pair<int,int> (4,100));
  midsemMarks.insert(pair<int,int> (5, 0));

  for(auto & e : midsemMarks)
    cout << e.first << " " << e.second << endl;
}
```