Sample STL Program

```
#include <iostream>  // Include standard stream I/O classes
#include <algorithm> // Include STL generic algorithms
#include <vector>    // Include STL vector container
#include <functional> // Include STL function objects
using namespace std;

// Function Object
struct least_digit_less : public binary_function<int, int, bool> {
    bool operator()(const int &x, const int &y) {
      return ( (x % 10) < (y % 10) );
    }
};
```

```
int main() {
  vector<int> S;  // S is initially empty
  ostream_iterator<int> out(cout, " ");

  for (int i = 0; i < 25; ++i) S.push_back(i);

  vector<int>::iterator x = find(S.begin(), S.end(), 10);

  reverse(S.begin(), x);
  copy(S.begin(), S.end(), out); cout << endl;

  sort(S.begin(), S.end(), least_digit_less());
  copy(S.begin(), S.end(), out); cout << endl;
}
Output:
9 8 7 6 5 4 3 2 1 0 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
20 0 10 21 1 11 22 2 12 23 3 13 24 14 4 15 5 16 6 17 7 18 8 19 9
```

Family of Abstractions (Beyond Data Abstraction)

- Set of related data structures that share a common core of semantics
  - Three parts: a common set of operations, a set of data structures, and a set of generic algorithms
- Example: (Linear) sequences
  - Rich set of operations: inserters, permuters, partitions, searches
  - Several data structures: arrays, extensible arrays (STL `vector`), singly-linked lists, doubly-linked lists (STL `list`), doubly extensible arrays (STL `deque`)

- Dozens of useful sequence algorithms that can be implemented in generic form
- Other families
  - stacks, queues, priority queues,
  - trees, graphs
  - associative containers (sets, maps)
  - generic high level I/O library (generic windows)

Sample STL Program Revisited—Deque Version

```
#include <iostream>  // Include standard stream I/O classes
#include <algorithm> // Include STL generic algorithms
#include <deque>     // Include STL deque container
#include <functional> // Include STL function objects
using namespace std;

// Function Object
struct least_digit_less : public binary_function<int, int, bool> {
    bool operator()(const int &x, const int &y) {
      return ( (x % 10) < (y % 10) );
    }
};
```

```
int main() {
  deque<int> S;  // S is initially empty
  ostream_iterator<int> out(cout, " ");

  for (int i = 0; i < 25; ++i) S.push_back(i);

  deque<int>::iterator x = find(S.begin(), S.end(), 10);

  reverse(S.begin(), x);
  copy(S.begin(), S.end(), out); cout << endl;

  sort(S.begin(), S.end(), least_digit_less());
  copy(S.begin(), S.end(), out); cout << endl;
}
Output:
9 8 7 6 5 4 3 2 1 0 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
20 0 10 21 1 11 22 2 12 23 3 13 24 14 4 15 5 16 6 17 7 18 8 19 9
```

Sample STL Program Revisited—List Version

```
#include <iostream>  // Include standard stream I/O classes
#include <algorithm> // Include STL generic algorithms
#include <list>      // Include STL list container
#include <functional> // Include STL function objects
using namespace std;

// Function Object
struct least_digit_less : public binary_function<int, int, bool> {
    bool operator()(const int &x, const int &y) {
      return ( (x % 10) < (y % 10) );
    }
};
```

```
int main() {
  list<int> S;  // S is initially empty
  ostream_iterator<int> out(cout, " ");

  for (int i = 0; i < 25; ++i) S.push_back(i);

  list<int>::iterator x = find(S.begin(), S.end(), 10);

  reverse(S.begin(), x);       // or S.reverse();
  copy(S.begin(), S.end(), out); cout << endl;

  S.sort(least_digit_less());  // Can't use the generic sort
  copy(S.begin(), S.end(), out); cout << endl;
}
Output:
9 8 7 6 5 4 3 2 1 0 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
0 10 20 1 11 21 2 12 22 3 13 23 4 14 24 5 15 6 16 7 17 8 18 9 19
```

Organization into a Concept Hierarchy

- STL's vector, list, and deque containers can be viewed as *refinements* of a generic Sequence *concept*
- Analogy:

Concept	:	Refinement
C++ class	:	C++ inheritance

- What's the difference? Use concepts and refinements only for organizing *documentation*, not for implementation.
- Trying to use classes and inheritance for this kind of code organization $\Longrightarrow$ too much loss of efficiency
- But concepts and refinement can be extremely useful aids to *understanding* the relation between different generic components

Refinement of the Associative Container Concept

- A more interesting case of concept refinement: