Iterators in STL

Iterators in STL

- Iterator categories
- Main functionality
- Combining containers and iterators

2

What is an Iterator? (1/2)

- An entity that is used to traverse the elements of a collection
- Collections may be STL containers, regular C arrays, C++ iostream
- Positioned at exactly one place in a collection at any point in time

3

What is an Iterator? (2/2)

- Remains positioned there until instructed to move
- In fact, a pointer-type object
- Acts as interface between algorithms and data structures

4

Iterator Value Types

- Dereferenceable
  - Iterator points to element in the datastructure
- Past the end
  - Iterator points after the last element in the datastructure
- Singular
  - Points to nothing (Like NULL pointer)

5

Obtaining an Iterator

- For arrays: a pointer in an array is an iterator
- STL containers provide functions that return iterators:
  - `begin()` // Iterator at the first element
  - `end()` // Iterator after last element

6

Iterator Categories (1/2)

- Trivial iterator
  - May be dereferenced to refer to some type
- Input
  - Read one item at a time, forward direction
- Output
  - Write one item at a time, forward direction
- Forward
  - Combination of input + output

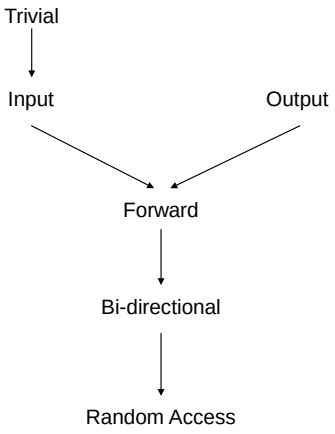
7

Iterator Categories (2/2)

- BI-directional
  - Forward + ability to travel backwards
- Random access
  - BI-directional + ability to jump by an arbitrary distance

8

Hierarchical Iterator Categories



9

Trivial Iterator

- Simple iterator that does not iterate
- Refinement of Assignable, Default Contructible and Equality Comparable
- It can be dereferenced
- Trivial iterator can be
  - Mutable
  - Constant
- Trivial Iterator example: Pointer to a variable

10

Trivial Iterator Functions

- X x; // Default constructor
- *x // Dereference
- *x=t // Dereference assignment
  // (only for mutable iterators)
- x->m // Member access

11

Iterator functions Common to other Iterators

- ++i // Advance one element and
  // return i's new value
- i++ // Advance one element and
  // return i's previous value

12

Input Iterators

- Read only iterator
- Refinement of Trivial Iterator
- Iterates in forward direction only
- No guarantee for possibility to pass through the same range twice
- Example: class istream_iterator

13

Input Iterator Main Functions

```
*i        // Return a read-only reference to
          // element at i's current position
i == j    // True if both i and j are both
          // positioned at same element
i != j    // Negation of i == j
```

14

Output Iterators

- Write values into a sequence
- No possibility to read
- Is refinement of Assignable and Default Constructible
- Iterates in forward direction only
- Different subtypes
- No guarantee for possibility to pass through the same range twice

15

Output Iterator Main Functions

- *i // Returns a writeable reference to
  // element at i's current position
- i = j // Set i's position to the same as j's
- Note: a == operator may not be available

16

Forward Iterators

- Both an input iterator and an output iterator
- Reading and writing in one direction
- Possible to save a forward iterator and use to start traversing from same position
- Useful for multipass algorithms
  - As opposed to single pass algorithms

17

BI-directional Iterators

- Refinement of Forward Iterator
- Allows traversal in both directions
- Needed for some some algorithms (e.g. reverse)
- Efficient traversal
- Example: list<TYPE>::iterator

18

BI-directional Iterator Functions

- --i // Retreat one element and return i's
  // new value
- i-- // Retreat one element and return i's
  // previous value

19

Random Access Iterators

- Refinement of Bi-directional Iterator
- Four previous iterators are not sufficient for all algorithms
- Stronger requirements on iterators demanded by some algorithms
- Demand that any position in sequence is reachable not just previous and next
- Example: Vector<TYPE>::iterator

20

Random Access Iterator Fuctions (1/2)

- i+=n // Advance by n locations and return i's
  // new value
- i-=n // Retreat by n locations and return i's
  // new value
- i+n // Return an iterator that is positioned n
  // elements ahead of i's current position
- i-n // Return an iterator that is positioned n
  // elements behind i's current position

21

Random Access Iterator Fuctions (2/2)

- i[n] // Return a reference to the n-th element
  // of i's associated collection
- i<j // Compare if I's position is lower than j's

22

Example
Iterators

```
std::list<int> l;           // Empty list
std::vector<int> v(10);     // Vector with 10 elements

// Fill list
for (int i=0; i<10; i++) l.push_back(i);

// Copy list elements to vector
std::list<int>::iterator il=l.begin();
std::vector<int>::iterator iv=v.begin();
while (il!=l.end())
{
  *(iv++)=*(il++);
}

// Print vector
for (iv=v.begin(); iv!=v.end(); iv++)
{
  std::cout<<(*iv)<<", ";
}
std::cout<<std::endl;
```

23