AN INTRODUCTION TO THE STL

The Standard Template Library

A library of well-known, frequently-used data structures.

Delroy A. Brinkerhoff

PRACTICAL DATA STRUCTURE IMPLEMENTATIONS

FUNDAMENTAL STRUCTURES

- Processed by the compiler
  - Simple variables
  - Structures
  - Arrays

LIBRARY STRUCTURES

- Too complex for the compiler (would make the compiler too big)
  - Lists
  - Sequences: vector, stack, queue, deque, etc.
  - Trees
  - Hash Tables

CATEGORIZING DATA STRUCTURES

SEQUENTIAL

- Elements accessed by position
  - Like an array index
  - insert(element, index)
  - remove(index)
  - at(index)
  - front
  - back

ASSOCIATIVE

- Elements are an aggregate type
  - Structure object
  - Class object
- One field is a *key*
- Programs search for the key, and if found, retrieve all data (fields) associated with it.

DATA STRUCTURE OPERATIONS

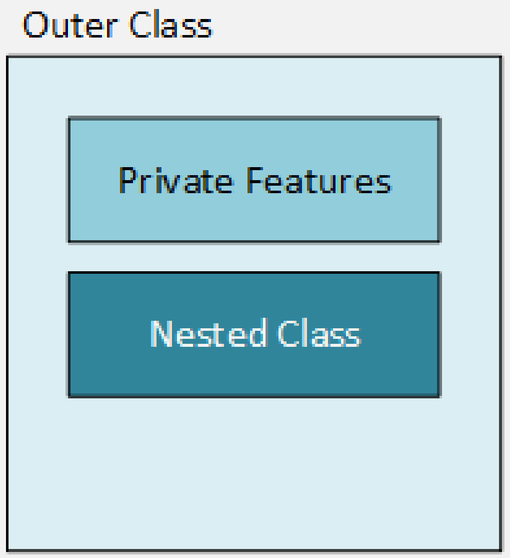
GENERAL

- Create
- Destroy
- Insert
- Access (search or find)
- Remove

STL `vector`

- Constructor
- Destructor
- `push_back(e), insert(pos, e)`
- `operator[i], front(), back()`
- `pop_back(), erase(), clear()`

STL ITERATORS

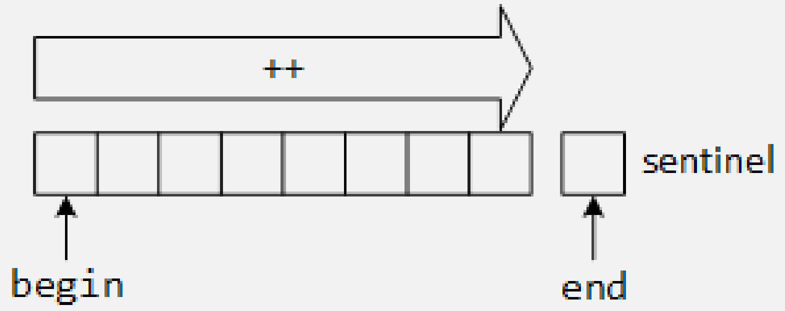


- STL containers implement iterators as nested classes
- `Outer::Nested`
- Each container has its own iterator
- Nested classes can access the outer class's private features (data and functions)
- Iterators "remember where they are" in the data structure

FORWARD ITERATORS

- `vector<int>::iterator`
  - `begin()`
  - `end()`

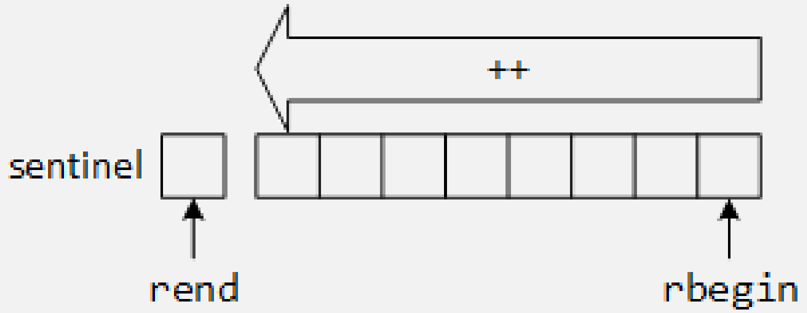
-
  ```
  for (auto i = v.begin(); i != v.end(); i++)
         cout << *i << " ";
  ```



REVERSE ITERATORS

- `vector<int>::iterator`
  - `rbegin()`
  - `rend()`

-
  ```
  for (auto i = v.rbegin(); i != v.rend(); i++)
          cout << *i << " ";
  ```


++
sentinel
rend
rbegin