Computing III

COMP.2010

Vector in a Nutshell

By: Sirong Lin, PhD

University of Massachusetts Lowell

Learning with Purpose

Class vector Overview

(textbook 7.3)

Vector

Vectors:

- a **container** that can “grow and shrink” during program execution
  - a container is an object that can contain other objects

vector — “arrays that grow and shrink” when programming is running

Vector Class

declaration: formed from Standard Template Library (STL), using *template* class

Syntax: **vector<Base_Type> variableName**

any type can be "plugged in" to Base_Type, e.g., int, char, user-defined class

produces "new" class for vectors with that type

Vector Class

Syntax: **vector<Base_Type> variable**

Example: vector<int> v;

vector <int> is a class type

v is vector of type int

Vector Class (Cont'd)

add elements — call member function ***push_back()***

adds three elements and gives initial values to the first three elements of the vector

```
#include <vector>
using namespace std;
…
vector<int> v;
v.push_back(0);
v.push_back(1);
v.push_back(2);
```

Vector Class (Cont'd)

support random access, indexed like arrays

index starting from 0

v[i] can be used to *read* and *change* the i^{th} element, e.g.,

```
v[0] = 9;
```

yet can't initialize the i^{th} element using v[i], must **add elements to the vector first**

Vector — size()

the member function *size()* can be used to determine how many elements are in a vector, returns unsigned int, e.g.,

v.size( ) returns 3

```
for (int i = 0; i < v.size(); i++)
    cout << v[i] << endl;
```

Iterator

An *iterator* is an object that can be used with a container to gain access to elements in the container

- a generalization of the notion of a pointer
- hide detailed implementations
- each container in Standard Template Library (STL) has own iterator types

vector iterator

vector has a random access iterator

- **random access** means that you can go directly to any particular element in one step

vector iterator (Cont'd)

One way of iterating through the vector —

C approach (what we've learned so far)

use index (i.e., subscript) *i* from 0 to one less than vector size to access each element, v[i]

```
for (int i = 0; i < v.size(); i++)
{
    cout << v[i] << endl;
 }
```

vector iterator (Cont'd)

Alternatively, use **iterator** — **C++ approach**

p is an vector iterator type

```
vector <int>::iterator p;
for (p = v.begin(); p != v.end(); p++)
{
   cout << *p << endl;
}
```

vector iterator (Cont'd)

```
vector <int>::iterator p;
for (p = v.begin(); p != v.end(); p++)
{
    cout << *p << endl;
}
```

v.begin(): returns an iterator for the container *v,* which references to the 1st item in *v*

v.end(): returns a sentinel that indicates "one past the end" (doesn't have a value)

vector iterator (Cont'd)

```
vector <int>::iterator p;
for (p = v.begin(); p != v.end(); p++)
{
    cout << *p << endl;
}
```

p is an iterator variable — conceptually a pointer

starts from v.begin()

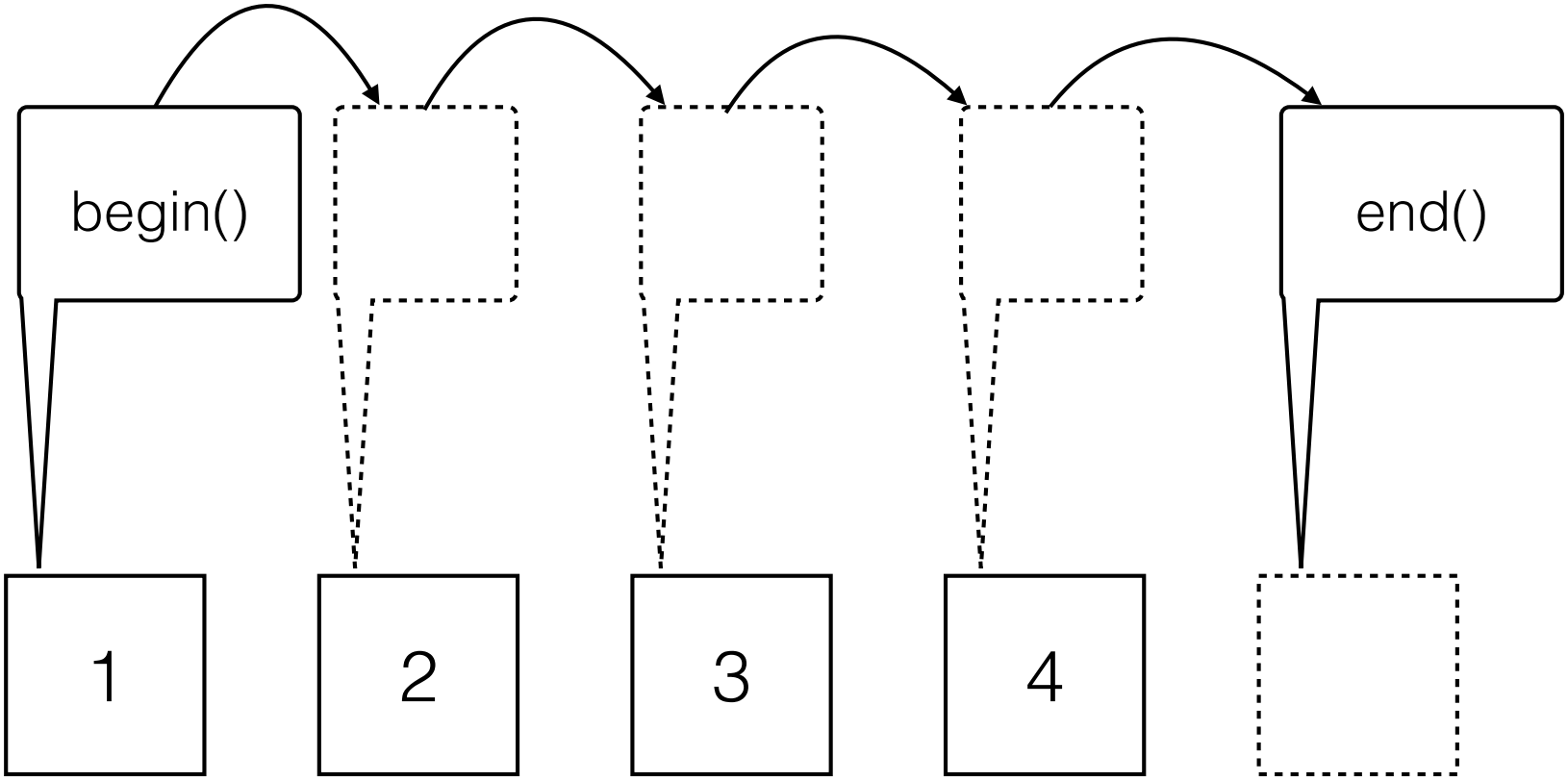
iterates through the elements in the vector one by one

tests if it equals to v.end(), when not, increment p by doing ++p

*p: gives access to the element referenced by p

++, --, ==, != operators are overloaded

vector iterator (Cont'd)



Incrementing Iterators