CS 162
Intro to Programming II
Standard Template Library
Standard Template Library

• Contains libraries for common data structures
• Much of STL consists of container classes that hold collections of data
  – Eg. vector, stack, queue
• The classes and algorithms in STL are generic ie. they use templates
Iterators

- **Iterator**: generalization of a pointer
- Abstracts away the details of the implementation
- Provides common interface to iterate through different container classes
Iterators

• Common operators on iterators:
  
  ++ (goes to the next item)
  -- (goes to the previous item)
  == and != (to test if two iterators point to the same data location)
  * (dereference operator that returns the data pointed to by the iterator)
Iterators

• Each container class in STL has its own iterator type

• The vector class has
  std::vector<int>::iterator

• The list class has
  std::list<int>::iterator
Iterators

• Let’s see how we use an iterator to loop through a vector of ints.

• The old way of doing this:

```cpp
int zeroOutVector(std::vector<int> v)
{
    for( int i = 0; i < v.size(); i++ )
    {
        std::cout << v[i] << std::endl;
        v[i] = 0;
    }
}
```
Iterators

• The new way with iterators:

```cpp
int zeroOutVector(std::vector<int> v) {
    std::vector<int>::iterator p;
    for( p = v.begin(); p != v.end(); p++ ) {
        std::cout << *p << std::endl;
        *p = 0;
    }
}

ffff
```
Iterators

• The new way with iterators:

```cpp
int zeroOutVector(std::vector<int> v) {
    std::vector<int>::iterator p;
    for (p = v.begin(); p != v.end(); p++) {
        std::cout << *p << std::endl;
        *p = 0;
    }
}
```

Declares an iterator for a vector of ints
Iterators

• The new way with iterators:

```cpp
int zeroOutVector(std::vector<int> v) {
    std::vector<int>::iterator p;
    for( p = v.begin(); p != v.end(); p++ ) {
        std::cout << *p << std::endl;
        *p = 0;
    }
}
```

Begin points to the first data element
End points to a sentinel just beyond the last data element
Iterators

• The new way with iterators:

```cpp
int zeroOutVector(std::vector<int> v) {
    std::vector<int>::iterator p;
    for( p = v.begin(); p != v.end(); p++ ) {
        std::cout << *p << std::endl;
        *p = 0;
    }
}
```

Dereference operator accessed the location pointed to by the iterator.
Iterators

• Random access: means you can access any element in one step
  – Eg. in an array \( x \), you can access the \( i^{th} \) element by \( x[i] \)

• You can perform random access with an iterator \( p \) on a vector in two ways (both get the item at index 2 of the vector):
  • \( p[2] \) or \( *(p+2) \)
Kinds of iterators

- Different containers have different kinds of iterators:
  1. Forward iterators: ++ works
  2. Bidirectional iterators: ++ and – works
  3. Random-access iterators: ++, -- and random access works
  4. Constant iterators: dereference operator produces a read-only version of the element
Constant Iterators

```cpp
int zeroOutVector(std::vector<int> v) {
    std::vector<int>::const_iterator p;
    for(p = v.begin(); p != v.end(); p++) {
        std::cout << *p << std::endl;
        *p = 0;
    }
}
```

This code will not compile because the line `*p = 0` violates the read-only property of a const_iterator.
int zeroOutVector(std::vector<int> v) {
    std::vector<int>::const_iterator p;
    for (p = v.begin(); p != v.end(); p++) {
        std::cout << *p << std::endl;
    }
}

This code compiles because it never modifies *p
Reverse Iterators

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

• This code will not work correctly. Why?
  – v.end() is not an actual element in the vector.
  – It points to beyond the last element.
Reverse Iterators

```cpp
int reversePrint(std::vector<int> v) {  
    std::vector<int>::reverse_iterator p;  
    for( p = v.rbegin() ; p != v.rend(); rp++ ) {  
        std::cout << *p << std::endl;  
    }  
}
```

- The correct way is to use the reverse iterator