600.120 Intermediate Programming, Spring 2017*

Misha Kazhdan

**Much of the code in these examples is not commented because it would otherwise not fit on the slides. This is bad coding practice in general and you should not follow my lead on this.*

Announcements

- Final Project: Today 11pm @ BlackBoard
  - Only one person per group should be submitting
- Individual "contributions": Sunday 11pm @ BlackBoard
  - Each member submits
- Review session: Monday 10-11:30am @ Gilman 50
- Final exam: Wednesday 6-9:00pm @ Hodson 110

Outline

- Iterators

Iterators

<u>Recall</u>:

- We can enrich a container class with an (nested) iterator class
  - iterator / const_iterator defines:
    - dereference
    - increment
    - comparison
  - container defines:
    - begin
    - end

container.h

```
template< typename T >
class Container
{
public:
        …
        class iterator
        {
                public:
                        T& operator * ( );
                        iterator& operator ++ ();
                        bool operator != ( const iterator& i ) const;
        };
        iterator begin( void );
        iterator end( void );
        …
};
```

Iterators

<u>Recall</u>:

- We can enrich a container class with an (nested) iterator class

⇒ Code for iterating through the container's elements becomes consistent across different container types

container.h

```
template< typename T >
class Container
{
public:
        …
        class iterator
        {
                public:
                        T& operator * ( );
                        iterator& operator ++ ();
                        bool operator != ( const iterator& i ) const;
        };
        iterator begin( void );
        iterator end( void );
```

```
void Print( const Container< int >& c )
{
        for( Container< int >::const_iterator i=c.begin() ; i!=c.end() ; ++i )
                cout << *i << endl;
}
```

Iterators

Recall:

- We can enrich a container class with an (nested) iterator class

⇒ Code for iterating through the container's elements becomes consistent across different container types

main.cpp

```
#include <iostream>
#include <vector>
using namespace std;

void Print( const vector< int >& v )
{
	for( vector< int >::const_iterator i=v.begin() ; i!=v.end() ; ++i )
		cout << *i << endl;
}
int main( void )
{
	vector< int > v( 3 );
	for( size_t i=0 ; i<v.size() ; i++ ) v[i] = i;
	Print( v );
	return 0;
}
```

```
>> ./a.out
1
2
3
>>
```

Iterators

- C++ supports a special **for** loop to iterate over container contents:

  for(<range declaration > : <range expression >) <loop statement>

  - **<range declaration>**:
    Declaration of range variable (should be of the same type as container elements)
  - **<range expression>**:
    The container
  - **<loop statement>**:
    What to do with each element

main.cpp

```
#include <iostream>
#include <vector>
using namespace std;

void Print( const vector< int >& v )
{
	for( int i : v ) cout << i << endl;
}
int main( void )
{
	vector< int > v( 3 );
	for( size_t i=0 ; i<v.size() ; i++ ) v[i] = i;
	Print( v );
	return 0;
}
```

```
>> ./a.out
1
2
3
>>
```

Note:
The type of the range variable is *not* the type of the iterator, it's the type of the contents

Iterators

- C++ supports a special **for** loop to iterate over container contents:

  for(<range declaration > : <range expression >) <loop statement>

  - **<range declaration>**:
    Declaration of range variable (should be of the same type as container elements)
  - **<range expression>**:
    The container
  - **<loop statement>**:
    What to do with each element

main.cpp

```
#include <iostream>
#include <vector>
using namespace std;

void Print( const vector< int >& v ) { for( int i : v ) cout << i << endl; }
void Increment( vector< int >& v ) { for( int& i : v ) i++; }
int main( void )
{
	vector< int > v( 3 );
	for( size_t i=0 ; i<v.size() ; i++ ) v[i] = i;
	Increment( v );
	Print( v );
	return 0;
}
```

```
>> ./a.out
1
2
3
>>
```

Note:
The type of the range variable is *not* the type of the iterator, it's the type of the contents

Iterators

- C++ supports a special **for** loop to iterate over container contents:

  for(<range declaration > : <range expression >) <loop statement>

- To support more general code, C++ can determine the type of an object based on the return type of a function / method

main.cpp

```
#include <iostream>
#include <vector>
using namespace std;

void Print( const vector< int >& v ) { for( int i : v ) cout << i << endl; }
void Increment( vector< int >& v ) { for( int& i : v ) i++; }
int main( void )
{
	vector< int > v( 3 );
	for( size_t i=0 ; i<v.size() ; i++ ) v[i] = i;
	Increment( v );
	Print( v );
	return 0;
}
```

```
>> ./a.out
1
2
3
>>
```

Iterators

- C++ supports a special **for** loop to iterate over container contents:

  for(<range declaration > : <range expression >) <loop statement>

- To support more general code, C++ can determine the type of an object based on the return type of a function / method
  - We can use the keyword **auto** to have C++ fill in the type
    - We can add "**const**" and / or "**&**" to get a constant / reference to the object instead of a copy

main.cpp

```
#include <iostream>
#include <vector>
using namespace std;

template< typename T >
void Print( const vector< T >& v ) { for( auto i : v ) cout << i << endl; }
template< typename T >
void Increment( vector< T >& v ) { for( auto& i : v ) i++; }
int main( void )
{
	vector< int > v( 3 );
	for( size_t i=0 ; i<v.size() ; i++ ) v[i] = i;
	Increment( v );
	Print( v );
	return 0;
}
```

```
>> ./a.out
1
2
3
>>
```