Week 15

Generic Programming

Class Templates

Class Templates

- A single class template provides functionality to operate on different types of data
- Facilitates reuse of classes
- Definition of a class template follows
  - template< class T > class Xyz { … }; or
  - template< typename T > class Xyz { … };

Example – Class Template

- A **`Vector`** class template can store data elements of different types
- Without templates, we need a separate **`Vector`** class for each data type

…Example – Class Template

```
template< class T >
class Vector {
private:
  int size;
  T* ptr;
public:
  Vector<T>( int = 10 );
  Vector<T>( const Vector< T >& );
  ~Vector<T>();
  int getSize() const;
```

…Example – Class Template

```
  const Vector< T >& operator =(
                const Vector< T >& );
  T& operator [](  int );
};
```

…Example – Class Template

```
template< class T >
Vector<T>::Vector<T>( int s ) {
  size = s;
  if ( size != 0 )
     ptr = new T[size];
  else
     ptr = 0;
}
```

…Example – Class Template

```
template< class T >
Vector<T>:: Vector<T>(
          const Vector<T>& copy ) {
  size = copy.getSize();
  if (size != 0) {
     ptr = new T[size];
     for (int i = 0; i < size; i++)
          ptr[i] = copy.ptr[i];
  }
  else ptr = 0;
}
```

…Example – Class Template

```
template< class T >
Vector<T>::~Vector<T>() {
  delete [] ptr;
}

template< class T >
int Vector<T>::getSize() const {
  return size;
}
```

…Example – Class Template

```
template< class T >
const Vector<T>& Vector<T>::operator
        =( const Vector<T>& right) {
  if ( this != &right ) {
     delete [] ptr;
     size = right.size;
```

…Example – Class Template

```
    if ( size != 0 ) {
         ptr = new T[size];
         for(int i = 0; i < size;i++)
              ptr[i] = right.ptr[i];
    }
    else
         ptr = 0;
  }
  return *this;
}
```

…Example – Class Template

```
template< class T >
T& Vector< T >::operator [](
                          int index ) {
  if ( index < 0 || index >= size ) {
     cout << "Error: index out of
                               range\n";
     exit( 1 );
  }
  return ptr[index];
}
```

…Example – Class Template

- A customization of above class template can be instantiated as

```
Vector< int > intVector;
…
Vector< char > charVector;
```

Member Templates

- A class or class template can have member functions that are themselves templates

…Member Templates

```
template<typename T> class Complex {
  T real, imag;
public:
  // Complex<T>( T r, T im )
  Complex( T r, T im ) :
     real(r), imag(im) {}
  // Complex<T>(const Complex<T>& c)
  Complex(const Complex<T>& c) :
     real( c.real ), imag( c.imag ) {}
  …
};
```

…Member Templates

```
int main() {
  Complex< float > fc( 0, 0 );
  Complex< double > dc = fc; // Error
  return 0;
}
```

Because

```
class Complex<double> {
  double real, imag;
public:
  Complex( double r, double im ) :
     real(r), imag(im) {}
  Complex(const Complex<double>& c) :
     real( c.real ), imag( c.imag ) {}
  …
};
```

…Member Templates

```
template<typename T> class Complex {
  T real, imag;
public:
  Complex( T r, T im ) :
     real(r), imag(im) {}
  template <typename U>
  Complex(const Complex<U>& c) :
     real( c.real ), imag( c.imag ) {}
  …
};
```

…Member Templates

```
int main() {
  Complex< float > fc( 0, 0 );
  Complex< double > dc = fc; // OK
  return 0;
}
```

Because

```
class Complex<double> {
  double real, imag;
public:
  Complex( double r, double im ) :
     real(r), imag(im) {}
  template <typename U>
  Complex(const Complex<U>& c) :
     real( c.real ), imag( c.imag ) {}
  …
};
```

<float> Instantiation

```
class Complex<float> {
  float real, imag;
public:
  Complex( float r, float im ) :
     real(r), imag(im) {}
   // No Copy Constructor
  …
};
```

Resolution Order

- Compiler searches target of a function call in the following order
  - Ordinary Function
  - Complete Specialization
  - Partial Specialization
  - Generic Template

Class Template Specialization

- Like function templates, a class template may not handle all the types successfully
- Explicit specializations are provided to handle such types

Review