Chapter 1
Overview

Sherri Shulman

Math-linux.com

Oct 1, 2013

Very brief Intro C++ – part 3

1. So ... so far you should be able to create a simple class (IntCell)
2. You can either create the class implementation and the interface at the same time (similar to Java)
3. Or you can separate the class interface into a file called IntCell.h and the implementation
4. You can compile and load separately (using the -c option on the compiler)
5. Or you can compile and load at the same time. The default name is then a.out. Or you can use the -o filename option to specify a different filename
6. Eg *g++ TestIntCell.cpp IntCell.cpp -o TestIntCell*
7. Eg *g++ -c TestIntCell.cpp IntCeol.cpp*
8. The second version just leaves .o files in your directory.

1. Templates: C++ does provide for type-like variables through the template.
2. As in Java, we often want to describe data structures (or other langauge elements) that are poly-morphic.
3. The text calls these "type independent" algorithms and data structures OR "generic" ...
4. There are two basic kinds of templates: function and class

1. A function template is a pattern that describes what a funciton will look like, but is not really a function yet.
2. When you provide the template argument, the function is instantiated (generated).
3. If you call it multiple times with new types, you will get multiple copies ... so it isn't polymorphic in the same way that Haskell is!
4. You might say that it is just overloaded.
5. Here is findMax and the main program to call findMax.
6. The argument can be instantiated with any class type, so one should assume that you do not have primitive types and use and return constant references.

```
/**
 * Return the maximum item in array a.
 * Assumes a.size( ) > 0.
 * Comparable objects must provide operator< and operator=
 */
template <typename Comparable>
const Comparable & findMax( const vector<Comparable> & a )
{
    int maxIndex = 0;

    for( int i = 1; i < a.size( ); i++ )
        if( a[ maxIndex ] < a[ i ] )
            maxIndex = i;
    return a[ maxIndex ];
}
```

```
int main( )
{
    vector<int>     v1( 37 );
    vector<double>  v2( 40 );
    vector<string>  v3( 80 );
    vector<IntCell> v4( 75 );

    // Additional code to fill in the vectors not shown

    cout << findMax( v1 ) << endl;  // OK: Comparable = int
    cout << findMax( v2 ) << endl;  // OK: Comparable = double
    cout << findMax( v3 ) << endl;  // OK: Comparable = string
    cout << findMax( v4 ) << endl;  // Illegal; operator< undefined

    return 0;
}
```

1. What happens if you call a templated function and there isn't an exact match for the function parameters/type args?
2. What if there is a templated version and a non-templated version?
3. In the last case the non-templated version has priority.
4. There are a complex list of type coercions that apply and the best match wins.
5. If there are two equally good "matches", then the compiler can't decide,
6. Note also: the type argument is explicitly provided by the type of the argument, since the typename Comparable is used in the type of the argument. and generates a type error.

1

1. Class templates
2. A class can be templated
3. Since you don't know the typename you need to make sure that the class type argument has a default (0-arg) constructor so the compiler knows how much space to allocate and how to initialize.
4. MemoryCall is just like IntCell but is templated so it doesn't have to be an int.

```
/**
 * A class for simulating a memory cell.
 */
template <typename Object>
class MemoryCell
{
  public:
    explicit MemoryCell( const Object & initialValue = Object( ) )
      : storedValue( initialValue ) { }
    const Object & read( ) const
      { return storedValue; }
    void write( const Object & x )
      { storedValue = x; }
  private:
    Object storedValue;
};
```

```
int main( )
{
    MemoryCell<int>    m1;
    MemoryCell<string> m2;( "hello" );

    m1.write( 37 );
    m2.write( m2.read( ) + "world" );
    cout << m1.read( ) << end1 << m2.read( ) << end1;

    return 0;
}
```

1. Again ... MemoryCell is not a class until the argument is provided.
2. StoredValue is now an Object, and it is initialized with by calling its constructor.
3. BUT now the default value is just Object(), which is the 0-arg constructor.

1. We've spoken about separate compilation, but templates don't always work well with separate compilation.
2. This requires that the entire class (interface and implementation) are p in the header file.
3. Looking at qan example with Object and Comparable.
4. Figure 1.21 introduces opeartor overloading
5. We should also discuss friend.

```
class Employee
{
  public:
    void setValue( const string & n, double s )
      { name = n; salary = s; }

    const string & getName( ) const
      { return name; }
    void print( ostream & out ) const
      { out << name << " (" << salary << ")"; }
    bool operator< ( const Employee & rhs ) const
      { return salary < rhs.salary; }

    // Other general accessors and mutators, not shown
  private:
    string name;
    double salary;
};

  // Define an output operator for Employee
ostream & operator<< ( ostream & out, const Employee & rhs )
{
    rhs.print( out );
    return out;
}

int main( )
{
    vector<Employee> v( 3 );

    v[0].setValue( "George Bush", 400000.00 );
    v[1].setValue( "Bill Gates", 2000000000.00 );
    v[2].setValue( "Dr. Phil", 13000000.00 );
    cout << findMax( v ) << endl;
```

1. Function objects.
2. Our findMax requires the operator$<$ be defined for the argument provided.
3. Even though we called the arg Comparable, it wasn't specified in the type.
4. Also what if we want the capability to have several operator$<$ dynamically determined?
5. One way (which we use in Haskell a lot) is to allow function objects.
6. We'll pass in the desired function.

```
// Generic findMax, with a function object, Version #1.
// Precondition: a.size( ) > 0.
template <typename Object, typename Comparator>
const Object & findMax( const vector<Object> & arr, Comparator cmp )
{
   int maxIndex = 0;

   for( int i = 1; i < arr.size( ); i++ )
        if( cmp.isLessThan( arr[ maxIndex ], arr[ i ] ) )
            maxIndex = i;

   return arr[ maxIndex ];
}

class CaseInsensitiveCompare
{
  public:
    bool isLessThan( const string & lhs, const string & rhs ) const
      { return stricmp( lhs.c_str( ), rhs.c_str( ) ) < 0; }
};

int main( )
{
    vector<string> arr( 3 );
    arr[ 0 ] = "ZEBRA"; arr[ 1 ] = "alligator"; arr[ 2 ] = "crocodile";
    cout << findMax( arr, CaseInsensitiveCompare( ) ) << endl;

    return 0;
}
```

```
code{text}
// Generic findMax, with a function object, C++ style.
// Precondition: a.size( ) > 0.
template <typename Object, typename Comparator>
const Object & findMax( const vector<Object> & arr, Comparator isLessThan )
{
    int maxIndex = 0;

    for( int i = 1; i < arr.size( ); i++ )
        if( isLessThan( arr[ maxIndex ], arr[ i ] ) )
            maxIndex = i;

    return arr[ maxIndex ];
}

// Generic findMax, using default ordering.
#include <functional>
template <typename Object>
const Object & findMax( const vector<Object> & arr )
{
    return findMax( arr, less<Object>( ) );
}

class CaseInsensitiveCompare
{
  public:
    bool operator( )( const string & lhs, const string & rhs ) const
      { return stricmp( lhs.c_str( ), rhs.c_str( ) ) < 0; }
};

int main( )
{
    vector<string> arr( 3 );
    arr[ 0 ] = "ZEBRA"; arr[ 1 ] = "alligator"; arr[ 2 ] = "crocodile";
    cout << findMax( arr, CaseInsensitiveCompare( ) ) << endl;
```

1