CS32 Summer 2013

Object-Oriented Programming in C++

Templates and STL

Victor Amelkin

September 12, 2013

Plan for Today

- PA5 – Due Date, Issues
- Templates
- Example Template Classes
- C++ Standard Library and STL
- Beyond CS32

PA5 – Due Date

Saturday, September 14, 08:59am

PA5 – Issues

`static`

From the slides
Discussion (Aug 21) — OOP in C++ / Advanced Topics

Static at C Level

- Static globals in C-files – hidden inside their translation units (“internal linkage”)
  - Translation unit – .c-file and everything `#include`'d

```
// file.c
static int global_var;
static int global_func() { … }
extern int global_var2; // visible to others (“external linkage”)
```

- Static globals in C-headers – each `#includ`'ing file gets its own copy

```
// header.h
static int another_var; // individual copy per inclusion (despite #include guards)
```

- Static local vars (“local global vars”) – retain value between calls

```
int func() {
  static int n = 0; // not the same as static int n; n = 0;
  return ++n;
}
func(); // returns 1
func(); // returns 2
func(); // returns 3
```

- In C, global vars are extern by default; in C++ – static, but const are extern

Static Class Fields

- Static field – belongs to class, not to object

```
// myclass.h
class MyClass {
private:
   char byte1;
   char byte2;
   static char my_static_field;

public:
   // instance members see static fields
   void print() { cout << my_static_field; }
};

// myclass.cpp
// must define class' static field:
char MyClass::my_static_field = 'x';

// main.cpp
MyClass obj;
cout << sizeof(obj); // prints 2
MyClass::my_static_field = 'y'; // error; var is private
```

Static Class Fields

- Static field can be of to the same class it is enclosed in

```
// myclass.h
class MyClass {
private:
   char byte1;
   char byte2;
public:
   static MyClass my_static_field;
public:
   MyClass(char bt1, char bt2) { … }
   void print() { cout << byte1 << “, “ << byte2; }
};

// myclass.cpp
char MyClass::my_static_field('x', 'y');

// main.cpp
MyClass obj;
cout << sizeof(obj); // prints 2
MyClass::my_static_field.print(); // prints “x, y”
```

Static Class Methods

- Static method – sees only static fields and other static methods

```
// myclass.h
class MyClass {
private:
   char nonstatic_field;
   static char static_field;
public:
   void method1() { cout << static_field; } // ok
   static void method2() { cout << static_field; } // ok
   static void method3() { method2(); } // ok
   static void method4() { method1(); } // error
   static void method5() { nonstatic_field = 'a'; } // error
};

// myclass.cpp
char MyClass::static_field = 'x';

// main.cpp
MyClass::method3();
```

Static Methods and (*this)

- Instance methods receive a pointer to the object

```
class MyClass {
   void instance_method(...args...);
};
obj.instance_method('x', 3);
```

translates into

```
class MyClass {
   void instance_method(MyClass *this, ...args...);
};
obj.instance_method(&obj, 'x', 3);
```

- Static methods do not

PA5 – Issues

`virtual~`

From the slides
Discussion (Aug 29) — OOP in C++ / Inheritance

Virtual Destructor

- Destructors are methods
- A non non-virtual destructor, like any other method, will not be called through a pointer/reference to a base class

```
class Base {
public:
   ~Base() { cout << "Base::~Base()\n"; }
};

class Derived : public Base {
public:
   ~Derived() { cout << "Derived::~Derived()\n"; }
};

Base *pobj = new Derived();
delete pobj; // only Base::~Base() is called
```

Virtual Destructor

- If a chain of destructors should be called (like on the slide with top-down destruction) when operating on pointers/references, destructor needs to be virtual

```
// http://cs.ucsb.edu/~victor/ta/cs32/disc4/code/virtdest.cpp
class Base {
public:
   virtual ~Base() { cout << "Base::~Base()\n"; }
};

class Derived : public Base {
public:
   ~Derived() { cout << "Derived::~Derived()\n"; }
};

Base *pobj = new Derived();
delete pobj;
>> Derived::~Derived()
>> Base::~Base()
```

Templates

- C++ templates allow to write generic code using types and values as parameters

```
template<typename TChar>
class String {
private:
    TChar *pchars;
    int len;
public:
    String();
    explicit String(const TChar *src);
    String(const String &other);
    TChar& operator[](int i) { return pchars[i]; }
    ...
};

using PlainString = String<char>;

PlainString plain_str;
String<wchar_t> unicode_str;
String<bool> boolean_str;
```

Templates

- Each time a template is used with a unique set of template arguments, a new class is generated by the compiler

```
// 3 different versions of class String are generated
String<char> plain_str;
String<wchar_t> unicode_str;
String<bool> boolean_str;
```

- This generating process is called *template instantiation*

- Each such class generated for a particular template argument list is called *template specialization*

```
vector<car> myvec; // instantiating vector<T>
```

Example Template Classes

http://cs.ucsb.edu/~victor/ta/cs32/disc6/code/template-utils/

C++ Standard Library

- The language itself is not enough for writing complex programs
- Need a library of commonly used classes and functions
  - utilities (e.g, memory utils, such as smart pointers)
  - data structures (stack, queue, hashtable, heap, ...)
  - algorithms (sort, search, shuffle, …)
  - threads (thread, mutex, …)
  - IO (<iostream>, ...)
  - …
- C++ Standard Library =
  - headers with declarations of classes and functions +
  - compiled library (automatically linked when you compile your code)
- C Standard Library – still available, but deprecated

C++ Standard Library

- Core of C++ Standard Library – STL
- **S**tandard **T**emplate **L**ibrary
  - proposed in 90's →
  - toolkit of template classes
    - vector<T>
    - pair<T1, T2>
    - stack<T>
    - …
    - ...
  - more importantly, proposed an idea of how to design highly reusable/universal template classes

Alex Stepanov @ HP Labs

The picture is borrowed from Dr. Dobb's Journal

C++ Standard Library

- Alternative – Boost C++ Libraries
  - http://www.boost.org, http://www.boost.org/doc/libs
  - wider than STL
  - will probably merge with STL in future

Beyond CS32

- Using Libraries (usually covered in CS32)
  - http://www.cs.ucsb.edu/~mikec/cs32/priorclasses/fall2012/slides/cs32wk10c.pdf
  - http://www.cs.ucsb.edu/~mikec/cs32/priorclasses/fall2012/labs/lab08/index.html
- Exception Handling
  - *"The C++ Programming Language"* by Bjarne Stroustrup or any well-written article
- C++11 (any article about new features of C++11)
- Templates
  - *"C++ Templates"* by Vandevoorde and Josuttis
- C++ Standard Library (smart pointers, container, algorithms, …)
  - *"The C++ standard library: a tutorial and reference"* by Nicolai Josuttis
- "C++ Advice":
  - books by Scott Meyers, Herb Sutter, Andrei Alexandrescu
- Software Design
  - refactoring: *"Refactoring: Improving the Design of Existing Code"* by Fowler et al.
  - design patterns: *"Design Patterns"* by Gamma et al.
  - unit-testing and TDD: *"Extreme Programming Explained"* by Kent Beck
  - building large projects: "Large-Scale C++ Software Design" by Lakos

~ Thanks ~