MODERN C++ DESIGN PATTERNS

Mixing C and C++ code

by Prasanna Ghali

Mixing C and C++ Code: First Steps

- Problem similar to cobbling together C or C++ program out of object files produced by more than one compiler
  - Size and alignment of `int`s and `double`s
  - Mechanism by which parameters are passed from caller to callee and who orchestrates the passing
- Make sure your C++ and C compilers generate compatible object files
- Then, four other things to worry about

Four Things To Worry About

- Name mangling
- Initialization of statics
- Dynamic memory allocation
- Data structure compatibility

Name Mangling (1/4)

- Necessary in C++ but unnecessary in C
- If you stay within confines of C++, name mangling is not of concern
- But suppose `draw_line` is C function and you call this function from C++, then what happens?

Name Mangling (2/4)

- You can tell your C++ compiler to suppress name mangling

```
// function implemented in non-C++ language
// and is meant to be imported by C++ linker
extern "C"
void draw_line(int, int, int, int);
```

- You can also tell your C++ compiler to suppress name mangling for certain C++ function names

```
// function implemented in C++ and is to be
// exported to clients using other languages
extern "C"
void draw_line(int, int, int, int);
```

Name Mangling (3/4)

- □ You can `extern "C"` set of functions like this:

```
extern "C" {
  // disable name mangling for following functions
  void draw_line(int, int, int, int);
  unsigned int twiddle_bits(unsigned int, unsigned int);
  void simulate_rope(int iterations);
}
```

Name Mangling (4/4)

- □ You want extern "C" when compiling for C++ but not for C
- □ Polyglot header files can be structured like this:

```
// disable C++ name mangling for following functions
#ifdef __cplusplus
extern "C" {
#endif

  void draw_line(int, int, int, int);
  unsigned int twiddle_bits(unsigned int, unsigned int);
  void simulate_rope(int iterations);

#ifdef __cplusplus
}
#endif
```

Four Things To Worry About

- Name mangling
- Initialization of statics
- Dynamic memory allocation
- Data structure compatibility

Initialization of Statics (1/4)

- In comparison to C, C++ has lots of code that gets executed before and after `main`
  - Static initialization occurs before body of `main` gets executed
  - Objects created thro’ static initialization must have their dtors called after `main` gets executed

Initialization of Statics (2/4)

```
// C++ main looks like this:
int main() {
  // C++ implementation performs
  // static initialization here

  // statements in main go here

  // C++ implementation performs
  // static destruction here
}
```

Initialization of Statics (3/4)

- When mixing C and C++ code, if you can't write `main` in C++, the program is toast
- What if C program is calling C++ functions?
  - `main` must still be written in C++!!!
  - Rather than rewriting your C code, you could do this neat hack:

Initialization of Statics (4/4)

```
// rename C's main to real-main
extern "C"
int real_main(int argc, char *argv[]);

// write a new main in C++
int main(int argc, char *argv[]) {
  return real_main(argc, argv);
}
```

Four Things To Worry About

- Name mangling
- Initialization of statics
- Dynamic memory allocation
- Data structure compatibility

Dynamic Memory

- Simple but consistent rule: C++ parts of program always use `new` and `delete`; C parts use `malloc` [and its variants] and `free`

Four Things To Worry About

- Name mangling
- Initialization of statics
- Dynamic memory allocation
- Data structure compatibility

Data Structure Compatibility

- Lowest common denominator is what C can do:
  - Can safely exchange normal pointers to C-style objects and pointers to non-member functions or `static` functions
  - Structures and variables of built-in types can also freely cross C/C++ border

Mixing C and C++ Code: Summary

- □ Make sure C++ and C compilers produce compatible object files
- □ Declare functions to be used by both languages with `extern "C"`
- □ Write `main` in C++
- □ Always use `delete` with memory from `new`; always use `free` with memory from `malloc`
- □ Limit what you pass between two languages to data structures that compile under C