C/C++ Refresher Session

Michael DiBernardo and Lloyd Markle
September 18th, 2007

You always hurt the one you love

- “If C++ has taught me one thing, it's this: Just because the system is consistent doesn't mean it's not the work of Satan.” (Andrew Plotkin)
- “Within C++, there is a much smaller and cleaner language struggling to get out.” (Bjarne Stroustrup)
- “I invented the term Object-Oriented, and I can tell you I did not have C++ in mind.” (Alan Kay)
- “There are only two things wrong with C++: The initial concept and the implementation.” (Bertrand Meyer)
- “C++ is like jamming a helicopter inside a Miata and expecting some sort of improvement.” (Drew Olbrich)

So why do we use it?

- “I now understand better why C++ is the way it is. Even though C++ is monstrously complex, the ability to write code that moves smoothly from a bit-level of abstraction to a fairly high level of abstraction is extremely valuable for many projects, especially ones where efficiency is paramount.” (Mike Vanier)

Outline

- Philosophy and Origins
- C++: A Confederation of Languages
- The *NIX C++ toolchain
- Programming “into” C++, not “in” C++
- Resources and finale

Outline

- **Philosophy and Origins**
- C++: A Confederation of Languages
- The *NIX C++ toolchain
- Programming “into” C++, not “in” C++
- Resources and finale

Heritage

- The C OOP family
  - The statics (give commands):
    - Algol led to
    - Simula, which led to
    - C++
  - The dynamics (send messages):
    - Lisp led to
    - Smalltalk, which led to
    - Objective-C

Philosophy

- Knuth says it best:
  - ‘Whenever the C++ language designers had two competing ideas as to how they should solve some problem, they said, "OK, we'll do them both".’
- You can select from a grab-bag of language features to address a problem
- You should probably not use all of them
- So, a lot of the work in C++ involves choosing what you will *not do*
  - i.e. How you will protect yourself from the rest of the language that you’re not using
  - and how you will keep things understandable by omitting parts
- “A C program is like a fast dance on a newly waxed dance floor by people carrying razors.” (Waldi Ravens)

Outline

- Philosophy and Origins
- **C++: A Confederation of Languages**
- The *NIX C++ toolchain
- Programming “into” C++, not “in” C++
- Resources and finale

Confederation of languages

- C
- C++ OOP extensions
- Generics / templates

Confederation of libraries

- The C++ standard library is:
  - Stream library
  - C stdlib
    - e.g. <assert.h> becomes <cassert>
    - <ctype.h> becomes <cctype>
  - Strings
  - The Standard Template Library (STL)

Be aware of borders

- Why does this matter?
- As part of the restriction effort:
  - It helps to program with foreknowledge of what “country” you’re in
  - Strive for apartheid
  - Or write your own wrappers to unify
- So you have to be able to identify these borders in order to work within them

Outline

- Philosophy and Origins
- C++: A Confederation of Languages
- **The *NIX C++ toolchain**
- Programming “into” C++, not “in” C++
- Resources and finale

The toolchain

- The usual suspects (i.e. stuff you'll see most often at UBC):
  - gcc
  - make
  - cvs or svn
  - gdb / xxgdb / ddd
- Other stuff that might help you:
  - cscope
  - scons
  - valgrind, purify
  - pprof and variants

Debugging!

- Debo vs. Lloyd

Outline

- Philosophy and Origins
- C++: A Confederation of Languages
- The *NIX C++ toolchain
- **Programming “into” C++, not “in” C++**
- Resources and finale

Programming “into” a language

- “Programmers who program *in* a language limit their thoughts to the constructs that the language directly supports. If the language tools are primitive, the programmer’s thoughts will also be primitive”
- “Programmers who program *into* a language first decide what thoughts they want to express, and then they determine how to express those thoughts using the tools provided by their specific language.” (Steve McConnell, paraphrasing David Gries).

Programming “into” C++

- C++:
  - provides little library support
  - syntax is rather verbose
  - the actual constructs are fairly primitive
  - ... and basically provides few conveniences
- Thus, it’s a good idea to get used to programming “into” C++
- Corollary to this is that having other language experience *really* helps

Programming “into” C++

- Programming into C++ entails:
  - Including things you want in the language that it doesn’t have
  - Excluding the rest so that it doesn’t “creep” in and increase complexity

Adding support for circular dependencies

- e.g. in a producer/consumer problem, producer must import the consumer and vice-versa
- In, say, Java, this is easy, as dependencies are managed by compiler (sort of)
- How to do this in C++?
  - Header guards
  - Not including headers in other headers
  - ... which requires forward declarations
  - ... which requires class-typed member variables to be pointers to instances

Adding memory / resource management

- The most oft-quoted difference between C/C++ and most other modern languages is that there is no support for garbage collection
- How to do this in C++?
  - Resource Allocation Is Initialization (RAII)
  - Other gotchas:
    - delete vs delete[]
    - Always make destructors virtual
    - Use const to enforce ownership

Handling exceptional conditions

- In most languages, you'd just throw an exception
- Exceptions in C++, erm, suck -- Many people choose not to use them (e.g. STL, anything portable)
- How to do this in C++?
  - Use return values to signal exceptional conditions
    - ... requires formal parameters to hold return types
    - ... requires careful signature design to make the distinction clear
  - Arbitrate object creation through static factory functions, making constructors private

No style conventions

- e.g. like those suggested by Sun for Java, or enforced by Python
- The solution here is simple -- adopt some!
- There are many on the net
- It is less important at first about exactly *what* restrictions the conventions suggest - anything is almost always better than nothing

Examples of things one might exclude

- friend
- exceptions
- operator overloading
- defaults
- stack allocation for member variables
- macros
- C libraries
- C-style casts

Take-home message

- To be effective in C++ you have to:
  - Be aware of the available constructs
  - Actively select which ones you will use and not use

Resources

- Style guides / examples
  - *Splash* and programming pearls @ http://www.mikedebo.ca
  - http://www.possibility.com/Cpp/CppCodingStandard.html
- Good books:
  - Bruce Eckel's *Thinking in C++* (for beginners)
  - Scott Meyers' *Effective C++* (after some experience)
- comp.lang.c++
- #c++ on freenode
- Boost (boost.org)

Questions and discussion