Welcome to the ACM Student Lecture Series

ACM@UIUC is proud to announce the start of a new lectures series, for students, by students. We'll be featuring regular students on a regular basis giving talks, tutorials, or demos about things that interest them. We're calling it the Student Lecture Series, and we hope you'll stop by a talk to learn about something new (and eat some pizza). All talks will be held from 5:30 to 6:30 in 2405 Siebel Center. See you there! Any questions or comments can be directed to sls-l@acm.uiuc.edu

The APL Programming Language

Michael Ilsemann, Senior in CS

*Tuesday, 11/4 at 5:30pm in 2405 Siebel*

APL, originally a mathematical notation for expressing complex computations concisely, is a family of languages that facilitate different and novel approaches to solving problems. The language is known for its power, expressibility, being dynamic, and for being very quick to develop. It won it's creator a Turing award.  [video](#)

The Kerberos Authentication Protocol and what it can do for you

Christopher Clausen, Senior in CS

*Wednesday, 11/12 at 5:30pm in 2405 Siebel*

The Kerberos network authentication protocol can be used here at U of I to facilitate logging on to a variety of computers, including EWS, CSIL, CS, and ACM computers. This talk will briefly describe how Kerberos works and provide instructions for you to actually use this technology to save yourself time and increase security when accessing computers (usually through SSH.)  [video](#)

Computer Science, a Mac, and you: Taking Advantage of Apple’s Developer Tools

Kevin Cathey, Junior in CS

*Thursday, 11/20 at 5:30pm in 2405 Siebel*

Today, there are more Macs being used on campus than ever before, especially in computer science. However, most of Apple's core Developer Technologies remained unused, and learning even their basics can save you hours of time as a student or faculty member in computer science. This talk will look at these tools and the common scenarios that they are used in.

Pragmatic Regular Expressions

Matt Sparks, Senior in CS

*Thursday, 12/4 at 5:30pm in 2405 Siebel*

An exploration of the powerful pattern matching language, covering the fundamentals, syntax, and practical usage.
Computer Science, a Mac, and You

Kevin Cathey
Junior, Computer Science
Roadmap

Introduction to Apple’s Developer Tools — Xcode & Instruments

Implement three “MP’s”
Introduction to Developer Tools
Software Development Cycle
And how the tools integrate with it

Start with a project
Make core classes
Design an interface

Code Build Debug Write tests Implement

Profile
Software Development Cycle
And how the tools integrate with it

Start with a project

Make core classes

Design an interface

Code  Build  Debug  Write tests  Implement

Profile
Start with a project

• Xcode is project based (most of the time)
• Start an application with a project template
• What is a project?
  ▪ Collection of source code files (or resources)
  ▪ Definition of what that source code turns into (a product)
  ▪ Set of rules of how to get these (targets and configurations)
Main parts of a project

Source Code (resources)

Target

Product (Executable)
Creating core classes

• Class creation is easy
• Create a new file from the templates and give it a name
• Done.
Building an interface

- You use the most obvious tool: Interface Builder.
- Interface Builder is a drag and drop WYSIWYG editor for interfaces.
- IB is not a code generator and aims to minimize the amount of code you need to write.
- IB provides mechanisms to connect your code to your interface (and vice versa) called connections.
Software Development Cycle
And how the tools integrate with it

- Start with a project
- Make core classes
- Design an interface
- Code
- Build
- Debug
- Write tests
- Implement
- Profile
Implementation — Write code

• Xcode is a powerful editor.
• Does expected things like line numbers and color syntax.
• Also does “cool” things:
  ▪ Code focus
  ▪ Code folding
  ▪ Code sense
  ▪ Code completion
  ▪ Code refactoring
## Implementation — Write code

- Navigation cheat sheet

<table>
<thead>
<tr>
<th>Open Quickly</th>
<th>Command-Shift-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch between interface and implementation</td>
<td>Command-Option-Up-Arrow</td>
</tr>
<tr>
<td>Jump to definition of symbol</td>
<td>Command-Double-Click</td>
</tr>
<tr>
<td>Jump to documentation of symbol</td>
<td>Option-Double-Click</td>
</tr>
</tbody>
</table>
## Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Collection of resources (source files, images, text files, whatever) and all the rules and pieces needed to turn those resources into a running piece of software.</td>
</tr>
<tr>
<td>Target</td>
<td>Collection of rules and phases that turn some set of resources into an executable.</td>
</tr>
<tr>
<td>Product</td>
<td>The output of a target.</td>
</tr>
<tr>
<td>Executable</td>
<td>Any executable Xcode launches for debugging. This is usually the Product of your Target, but could be any executable (think of writing a dylib).</td>
</tr>
<tr>
<td>Configuration</td>
<td>Settings Target uses to build (compiler flags, warning settings, etc).</td>
</tr>
</tbody>
</table>
Implementation — Build

• This is what the **target** does.
• Xcode’s build system figures out dependencies for you, and what needs to be built.
• Drops down to gcc and ld for compiling and linking.
• Configurations
  ▪ A group of settings for a target.
  ▪ Default configurations are Debug and Release
    ▪ **Debug**: Has standard settings with debug symbols. Only setting you always need to change is to turn on warnings as errors.
    ▪ **Release**: Same as Debug, but without debug symbols.
Implementation — Debug

- Xcode sits on top of GDB for debugging, and brings it into the developer’s workspace.

- In-line debugging:
  - Roll over variables to see their values.
  - Step through your code without leaving it.

- Breakpoints:
  - To set them, just click on the source line you want to stop at.
  - Use Breakpoints window to set conditional breakpoints.

- Fully interactive gdb console.
“Just because Xcode sits on top of gcc, ld, and gdb, this does not give you an excuse not to know them.”

Useful links:
http://developer.apple.com
http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf
Implementation — Testing

• Xcode includes SenTest, an Objective-C and C/C++ unit testing framework.

• Create a SenTest target, and then set that target as a dependency of your others.

• If your unit tests fail, then your build fails, and you can quickly find the problem.

• Read Apple’s documentation for more.

• Another alternative is to write a Python script for unit tests, then add a build phase to a target that runs the Python script and fails the build if the script returns anything but zero. Then use same script when you upload to the Linux box.
Software Development Cycle
And how the tools integrate with it

1. Start with a project
2. Make core classes
3. Design an interface
4. Code
5. Build
6. Debug
7. Write tests
8. Implement
9. Profile
Profiling

• Instruments sits on top of dtrace.
• You can profile all different aspects of your program:
  ▪ CPU sampler (where are you spending the most time)
  ▪ Memory allocations
  ▪ Leaks
  ▪ File system usage
  ▪ Network usage
  ▪ Graphics performance
• From Xcode, instead of debugging, you run with a template from Instruments, or build your own.
Let’s code
Create a C program that counts the words in the first input argument. You will just turn in your "main.c" file.
Take MP1, and put your counting code into a function called num_words_in_string in a file called wordcounter.c. Also create a header file. Call with argv[1] from main. For building, use our makefile.
How do we go remote?
How to get changes remote?

• We need to test our stuff on the Linux boxes (dcllnx, csil-core, etc), how do we get our stuff back?

• A few options:
  • scp
  • SVN *
SCM For School

• Create repository in your home folder (or have one provided by school).
• Check out locally, then add your MP to the folder.
• In Xcode, setup your repository.
• Do all the rest of your integration with Xcode.
• Then, on Linux box, check-out, build.