Please add your name, a proposed title/topic of your paper, and a one or two paragraph abstract describing the topic and the key points of your paper (300 words max).

The abstracts should be in alphabetical order by last name.

Abstracts are due Wednesday, April 11th at 11:59pm (note the corrected date).

Stephen Adams (3) - Comparing and contrasting the concurrency features provided by the Go and Clojure programming languages.

In a 2005 interview Gordon Moore predicted that his namesake law (Moore's law states that the "number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years") will reach a fundamental limit in ten to twenty years [1]. Our transistors are starting to approach the size of individual atoms which is a boundary that will not be broken. With this in mind, the challenge to continue increase the speed of computing will soon fall to computer scientists as our abilities to create programs that utilize multiple processing cores will be where the futures speed gains are created.

In the past five years two languages were created (Go and Clojure)[2] with support for easy concurrency being a primary tenet of these languages. Both of these languages have received significant levels of interest (both languages are in the top 100 programming languages in the Tiobe index [3]) to think that they may either see large scale adoption for their concurrency features or influence the next generation of concurrency control methods.

In this paper I will briefly cover the basics of both Go and Clojure syntax and then I will cover these two languages' concurrency features. Finally there will be a discussion about the merits of both of these concurrency systems.

Sources -

[1] Moore's Law Wiki page - http://en.wikipedia.org/wiki/Moore's_law#Ultimate_limits_of_the_law

[2] Timeline of Programming Languages - http://en.wikipedia.org/wiki/Timeline_of_programming_languages#2000s

[3] Tiobe Software index - http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Chris Aga - JavaScript

Designed by Netscape's own Brendan Eich, JavaScript is a java influenced, pseudo-object oriented language, being such since it provides polymorphism and inheritance but not encapsulation. It is a fairly strong cross-platform language used to control the DOM of HTML pages and implements objects on both a client and server side. It has the downside of having very inconsistent library implementations when jumping from browser to browser, but the use of JQuery can fix such issues that were previously horrible to deal with when writing in pure JavaScript. JavaScript uses dynamic scoping and is a quite weakly typed language that supports both imperative and functional programming. When used as a functional language, functions are first class and programmers have the ability to use closures. This paper will talk about various aspects of JavaScript like its security issues such as cross-site scripting (XSS). For example, people can force an insecure site to provide a "malicious script" to its clients which can gain access to said client's privileges. XSS can be particularly hard to protect against when such a script is stored on the database side. Also it will inform the general audience to incorrect labels on JavaScript such as it being used for purely controlling a web page's DOM.

Drake Arvidson - Jython

The paper will look into the language implementation of Python that runs on the JVM. There will be an overview of Jython. It will look at Jython and how many parts of the language do not differ from Python, but also how it provides features that Python does not as a result of it being extended to run on the Java platform. These features will be considered and the benefits will be explained showing how much Jython gains from being able to use the Java library and running on the JVM.

Vincent Borchardt - Simplifying Java Concepts Through a New Language
Tentative Abstract:

Java appears to be a language that combines the best of the old, new, and future paradigms, starting with the syntax and structure of C, making objects the focus of the language instead of a bolted-on addition, and looking to the future with relatively easy concurrent functionality (for an imperative language). However, I feel that Java puts too many options on the table in many cases, which manifests itself in many ways and hurts it as a language.

The most obvious way Java's options hurt is the ceremony involved with writing even the simplest methods, meaning Java is not as suitable as a first language as others. Java also allows many bad coding practices because of its flexibility, allowing for many mistakes, such as confusing variable names. Finally, Java has been around for many iterations, and even though people have added new features, those features need to be implemented in complex ways to retain the majority of compatibility with old virtual machines (most notably with generics).

Reducing the ceremony of Java primarily comes through differentiating a class for an object from a class for static methods, as well as making public the default method access type. Eliminating variable shadowing forces better names in large classes, and prevents variable collisions. Making everything an object allows types to be handled consistently, even if it requires a minor decrease in performance.

Brandon Botzet - Comparison of Java and C Sharp

In this piece I will look the programming languages C# by Microsoft and Java by Oracle. Both languages are popular in commercial uses and I will compare features that can make them unique, but you will find that they are more similar than you think. Features of both languages that make them popular include semi-interpretation, garbage-collection, use of objects and the option of libraries. I will compare these features between both languages to show if one language exceeds the other in a feature. In the end both languages are so very similar and the differences shown are not conclusive in choosing one over the other. The choice of Java or C# usually come down to personal preferences, systems set up, or legacy code being used.

Alex Decker - An Analysis of the Frink Programming Language

The Frink language was created by Alan Eliasen around 2001, and originally designed as a calculating tool to help him solve problems. To this end, he implemented one of the most uniquely useful parts of the language early on: it tracks the units of measurement along with a value in its variables. Frink also smartly interprets mixed-type calculations based on mathematical rules, storing the result with its appropriate unit of measurement. Thus, you can say that the language is strongly typed. The language could also be called dynamically typed, though constraints may be placed on variables, requiring values to be even/odd, less than 100, primes larger than 50, etc.

Frink is implemented in Java and runs on top of the JVM. It can call Java methods, and will automatically convert its internal types to and from Java types. It can use external Java libraries, as well as be embedded into a Java program.

Joe Einertson (5) - Class- vs. Prototype-Based Objects

Nearly all major object-oriented languages, including Java and C++, use a class-based object system, where objects are defined as belonging to a class of similar objects. Some languages, however, use a prototype-based object system, where objects are defined in terms of other objects. Most prominent among prototype-based languages is JavaScript.

Many programmers do not understand or even know about prototype-based objects, and this leads to many misconceptions. For example, it may, at first glance, seem like an object system where classes cannot be defined lacks inheritance. To the contrary, prototyping does lead to inheritance, just a very different type than class-based objects.

In my paper I will compare and contrast class- and prototype-based object systems, specifically discussing object creation and inheritance in each. I will use Java and ECMAScript (JavaScript) as the basis for my comparisons, as both are widely-used languages. Additionally, I will discuss the strengths and shortcomings of each system, and provide examples of where each system is most powerful.

Lucas Ellgren - Python

For my paper I will explore the programming language Python in depth. Python is an interpreted language with dynamic, less strict typing. It supports multiple programming paradigms including imperative, functional and object-oriented. Python's syntax is designed to be simple and elegant with an emphasis on readability, and is based on the design philosophy that there should be multiple approaches to a problem. I will explore the history of Python's development, including the reasons behinds its unique design choices. The aspects of the language, such as its syntax, usage of types and support for polymorphism will be detailed here as well. I will investigate the recent rise in popularity of the language and compare the ways in which it is used in modern programming environments. Overall, the paper will give an in-depth look into the features of the language and provide a better understanding of the ways in which Python can be used.

Josh Johnson (3) - Go

Go is a new systems programming language created by three software engineers employed by Google. While it was originally an experimental language, it has recently been used for “real stuff“ at Google. Go was designed to make software design fast and efficient with emphasis on high-speed compile times. It is compiled, concurrent, garbage-collected, and requires explicit declaration of dependencies. Its syntax is very clean and minimalistic in order to support numerous styles of programming. Even though it is statically typed and compiled language, Go was designed to feel like a dynamically typed, interpreted one. The paper will go in depth about all these paradigms and features of the language, but it will also focus on the concurrency of Go since that was one of the focal design points for creating the language.

Ashley Koch - Visual Basic
Visual Basic was designed to rapidly develop graphical user interface (GUI) applications and also to be a program that is simple to learn and to use. Visual Basic is an Object Oriented programming language that includes parametric polymorphism. Visual Basic has some features and an interesting history that sets it apart from many other programming languages.

One feature that sets Visual Basic apart from other programming languages is that it allows the programmer to alter the compiler by resetting compiler options so that the programmer can choose between having a strongly typed program or a “typeless” program. Instead of choosing a different programming language to get the desired type discipline, the programmer can make a few alterations to the compiler and write the code in the same language.

Visual Basic also has an interesting history of altering the programming language to meet the desires of the general programming audience. For example, the early versions of Visual Basic used a “duck typing” type system. As programming attitudes and trends changed, newer versions of Visual Basic became statically typed. Now, trends have changed again and Visual Basic 10 is dynamically typed.

All in all, Visual Basic is an easy-to-use programming language that also allows programmers a great deal of control over how they want their code to work. The developers of Visual Basic also adapt the language over time so that it meets the needs of its users.

Eric Laska (3) - PHP vs. Ruby (on Rails) for web-development — Why all the hatred?
As many CSci students have inferred, PHP is an inferior web-developing language. My paper will be on why such predispositions may exist or arise. From personal preference and popularity to succinctity and standards, I will investigate the pros and cons of each language on the level of usability and practicality. As a web developer, I involve my own experiences with each as well.

Andrew Latterner (3) - Haskell
In 1987, a group of computer scientists decided that, due to the large amount of purely functional programming languages, a common language needed to be created in order to synthesize the existing functional languages’ strengths as well as develop a unified community. Out of this synthesis, Haskell was created. Haskell is a strongly-typed, functional programming language. Haskell was designed with specific ideas in mind so as to stay true to functional programming concepts while developing a language that could produce more stable, faster code than its competitors. These ideas included concepts such as the computational method of using monads, laziness, and pure functions.

Haskell’s type system and its use of type classes are only a couple of strengths that are brought up when discussing the benefits of Haskell. As previously mentioned, Haskell is a static, strongly-typed language. However, the type system is not so restrictive as to eliminate the possibility for polymorphism, which allows for greater code reuse and faster code production. This polymorphism is implemented through a very important feature of Haskell: type classes. These type classes are data representations which allow for ad hoc polymorphism, or overloading. These strengths, along with many others, are a topic for discussion when exploring Haskell.

Jeff Lindblom -
Alleviating the Stateful Insecurity of Java through Type Dependency Injection with Spring
(Title limit was unspecified :P)

As parallel architectures become a must for streamlining the efficiency of increasingly load intensive web applications, we are seeing an elevated need for synchronized functionality that we can trust. As Rich Hickey will explain at length, the problem with current language architectures in this regard is their lack of a concept of time. Hickey's solution for this negligence is clojure, a language built around the veracity of pure functions and immutable data.

While Hickey's approach does indeed solve for the impermanence and fluidity of data in a mutable world, it leaves open a big question for languages of which these constraints do not hold: How can we develop synchronized applications with the qualities that clojure holds dear, but through a language that manifests qualities quite the opposite? What better language to address this question with than Java, a hallmark example of the stateful insecurity of shared, mutable objects.

In Java, shared objects are one continuous stretch of data manipulation with little sense of when, what, where, or how. This confusion is exacerbated by our tendency to glob shared objects together through inheritance from interfaces, such as an observer, so as to update everything everywhere seamlessly as needed. The problem with this model is that there exists no innate means of determining the access timeline of any given object inheriting through our interfaces. Without such a record, the state of the data within those shared objects, at any given time, is shrouded in mystery.

This paper will elaborate upon a Java application framework called Spring that seems to address Java's stateless interface problem through type dependency injection. I will describe what type dependency injection is, how it is able to work around the interface inheritance web, and explore the intricacies of Spring's implementation from a web development perspective.

(297 words... Sorry ._.)

Todd Malone (3) - Two major languages in web development, Groovy and Ruby have much in common. Both use similar a type discipline and type system, both are purely object oriented. They draw inspiration from a similar set of languages, such as Perl and Smalltalk. But where Ruby was developed in the 1990s, Groovy is a much more recent language, version 1.0 being released in 2007. In fact, one of the languages Groovy credits in its development is Ruby.
So how different are they? I will explore both languages, touching on areas of similarity and focusing on areas of variation.

Michael Rislow (6) - FORTRAN and its Appeal to the Non-Computer Scientist
An acronym for formula translation, FORTRAN is an imperative programming language that

was designed by IBM in the 1950s for solving numerical computational problems for scientific computing. Modern FORTRAN looks quite different from its inaugural version as the allowance of recursion, object-oriented programming, and other features has been added in successive versions. FORTRAN's staying power lies in its backward compatibility and its appeal to disciplines beyond computer science. Huge amounts of old FORTRAN code is still in use today by the physics and engineering community in a wide array of applications. As a result, a vast amount of legacy code exists in each successive version of FORTRAN to support these programs. Its heavy usage in scientific computing also implies that much of FORTRAN code is written by non-computer scientists. In this paper, I will examine the main features of the language, and address how they do well to facilitate numerical computation. These include its type system, inheritance model, as well as its unique features. In addition, I will examine a FORTRAN code sample written by a physicist with no formal training in computer science and see how this fact affects his experience with the language and how his code contrasts with what we believe to be good or conventional programming practice.

Casey Robinson - Programmer vs CPU time - A comparison of C and Ruby

Ruby is an interpreted, dynamically-typed programming language meant to emphasize the speed with which code can be written over the speed with which it can be executed. This design philosophy runs almost directly contrary to that of low-level languages such as C and FORTRAN, and this dichotomy of programmer time vs CPU time will be a large focus of the paper. I will investigate how various features encourage rapid code generation while others seem to inhibit it, and how the inefficiencies in languages meant to hasten development might be alleviated.

David Ruprecht - Ruby v. Perl

The underpowered programming language, Perl, inspired the initial making of Ruby. Ruby took many features from Smalltalk, Eiffel, Lisp, and Perl and was developed in the mid-1990s. With automatic memory management, dynamic typing, variable scoping, and interpolation being just some of its features parallels are easily drawn between Ruby and one of its mother languages, Perl. Both Perl and Ruby have also been seen and used substantially in web development.

In this paper similarities and differences between Ruby and Perl will be compared and contrasted. I will identify what features of Ruby provide it an advantage to Perl as well as features shared between Ruby and Perl. I will also analyze and propose a possible reason for the use of either Perl or Ruby respectively in a few different applications.

Adrian Schiller - FALSE: in depth Analysis

FALSE is a esoteric programming language created in '93, developed by Wouter van Oortmerssen, and was named after his favorite boolean variable. FALSE is based off of Forth, which inherits its properties of being structured, reflective, imperative, extensible, and stack-based, with the exception that it is intentionally obscured code. The purpose of FALSE was to see how small of a compiler van Oortmerssen could create (which was 1024 bytes large) while maintaining a very powerful language. Its small overhead and amusingly convoluted style has inspired several other esoteric languages (some of which have vulgar names). I will go into detail about the productivity of this language, its general properties, and how it is yet beneficial to the interested programmer.

Greg Schumacher - C Vs. Java (Focus on tools built in java that are not built in C)
C is an imperative systems implementation language that was created from 1969-1973 to be a relatively straightforward compiler. C provides low level access to memory and requires minimal run time support. Java was developed by James Gosling and was released in 1995 with similar syntax to C. Java however has a simpler object model and fewer low level facilities and Java applications are typically compiled to class file. Java and C are both popular languages that may have similar syntax, but differ greatly in the control they give to the programmer and the compile time. For example Java has automatic garbage collection, while in C garbage must manually be removed by the programmer, but C garbage collection compiles faster.

This paper will focus on the advantages that java provides the programmer with built in systems. I will also explain how these systems are done manually in C, but explain the advantage that C gives in the computation time that makes it an attractive code for things like kernels.

Reed Simpson - What is Perl?

My essay will be on the programming language Perl. According to Wikipedia, Perl is a high-level, general-purpose, interpreted, dynamic programming language. Perl is known for its text processing capabilities, but has otherwise continuously evolved over the last 25 years. The design of Perl has often been critiqued as promoting untidy code and slow compile times, while its proponents have claimed that its aim is to make the most efficient use of programmers rather than hardware resources. I would also like to mention an offshoot programming language designed for use in the game Minecraft called PerlStone.

Tim Snyder (6) - LOLCODE and chef: a comparison of esoteric languages designed for amusement.
Esoteric languages fall into several categories including Turing tarpits and joke languages. I will be comparing a pair of joke languages. The first is LOLCODE which is based off of the grammar and spelling style of the LOLCATS meme. The second is chef which is a language where the grammar is designed to look like recipes. Both are imperative languages without objects. Both have very little for type discipline but while LOLCODE is dynamically typed, chef is statically types if only because chef only has one type. LOLCODE does not allow overloading a function with a different number of variables, nor does chef since the only thing that you can, and must, pass to a function is the stacks of integers. These two languages are well outside of the mainstream of languages as well as difficult to program in to varying degrees. For instance, chef has no if statements, so it requires clever use of its only decrementing to 0 for-loops. For strengths, these languages are both amusing. Of course, since these are esoteric languages, they were not designed with ease of use in mind.

Seth Sorensen - Ada
The Ada programming language, named for Ada Lovelace (considered to be the first computer programmer), was initially developed throughout the late 1970's and early 1980's as a solution to the U.S. Department of Defense's lack of a single standardized language for use in DoD projects. Prior to its development, projects were implemented in hundreds of different non standard programming languages. This inconsistency in language use and coding practices

led to various problems for the department. In this paper I will discuss some of these problems which acted as motivation for the development of Ada, as well as how different features of the language were designed specifically to help resolve these problems. More specifically, I will address Ada's type system, what sort of memory management is available, how concurrency is handled, Generics and Polymorphism, and Inheritance.

Scott Steffes (7) - Perl: you can do what?!?!?!?
Perl was originally designed by Larry Wall in 1987 as a scripting language with features similar to those of common Unix command-line tools such as AWK and SED. Since then, Perl has undergone many significant changes, but has maintained a focus on powerful text processing. Modern Perl has syntax very similar to that of C. Unlike C, however, it is interpreted, dynamically typed, and includes numerous built in operations that can make Perl programs extremely short. A 2-3 line Perl program may perform operations that would take numerous lines in a more traditional language such as C or Java. The computer science community is sharply divided on the question of whether Perl is a good programming language. Some argue that in practice, Perl allows programmers to simply get things done quickly and is a powerful asset while others argue that Perl facilitates poor programming practice and that Perl programs are often unreadable and unmaintainable. For better or worse, Perl has been used extensively in web development, a sphere in which text processing is a major element. Websites such as Amazon.com, Craigslist, IMDb, LiveJournal, and Slashdot all make use of Perl. I will examine the unique features of Perl in detail, show how it can be used to rapidly develop applications, and examine arguments for and against the programming style associated with it.

Cody Sutherland - PHP
PHP is a server-side web language commonly used in web development of dynamic web pages. PHP is an imperative and object-oriented language. It is considered to be an interpreted language but it is similar to Java in that it is compiled to an intermediate bytecode which is then interpreted. PHP exhibits weak typing in that it does not require explicit types when declaring variables and it supports implicit type conversion and overloading. It is also dynamically typed as type checking is performed at run time.