**COURSE OVERVIEW:** This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

**Language:** \textbf{ADA}

MIDN Kevin Lees, USN  
MIDN Alek Ogden, USN

**ADA HELLO WORLD**

```ada
with Ada.Text_IO;  
use Ada.Text_IO;  
procedure Hello is  
begin  
    Put_Line("Hello, world!");  
end Hello;
```

**Strengths**

- Interoperability with Other Languages
- Concurrency Support
- Real-Time Support
- Safety-Critical Support
- Reusability

**LANGUAGE HISTORY**

- Developed for the DoD in 1980s by CII Honeywell Bull
- DoD Standard Language 1983-1997
- Versions: Ada83, Ada95, Ada05

**Applications**

Ada has a set of unique technical features that make it highly effective for use in large, complex and safety-critical projects.
Overview

• Written by Brian Fox for the GNU Project in 1989
• BASH stands for Bourne Again Shell
• BASH is a shell scripting language, perfect for writing
command line programs
• Huge amount of online support
• Used to easily automate complex series of commands
for easy reuse

# Features

• No explicit types
• Supports arrays: no size declaration required
• Redirect stdin and stdout to files
• Flexible parameter passing with functions
• Extensive string manipulation
  • `tr` command
• Tight integration with operating system
  • Commands executed on the command line can be
    executed in the shell script
• Variables global unless declared otherwise
• Read and write to sockets
• Process substitution
• Multifunctional test command
• Debugging: `#!/bin/bash` – `x`
• Can execute most Bourne shell scripts without
  modification
• Doesn’t support floating point math
• Only supports 1-D arrays

# Code Examples

```
#!/bin/bash

echo -n "Which fibonacci number do you want to see? "
read serial

n=0
count=2
fibonacci_number=0
while [ $count -le $serial ]; do
  fibonacci_number=$[fibonacci_number + fibonacci_number]
  n+=1
done

echo "Fibonacci $n=serial = $fibonacci_number"
```

```
#!/bin/bash

echo Hello World
```

```
# !/bin/bash

diff <(find dir1) <(find dir2)
```

```
if [ $file1 -nt $file2 ]
```

```
Find difference between the contents of 2 directories
```

```
Checks if file1 has been modified more recently than file2
```

# Gotchas

• Use of whitespace in variable assignments
• Mixing up `–eq` and `=`
• Assuming uninitialized variables are zero
This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

Creator: Urban Muller

Other uses:

Original Distribution

Hello world!

Who can program anything useful with it? :)

This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

Created by: Brandon Tinkham William McCrone
Erlang was developed by Joe Armstrong for Ericsson. The name itself is a shortened version of its full name, Ericsson Language. Erlang was designed to cope with the needs of a telephone network, giving it many unique characteristics. Erlang:

- Is a functional language; variables may only have one value
- Encourages concurrency by passing information between threads in messages, eliminating locks
- Allows sections of code to be modified while it is running
- Has no built in string manipulation

Erlang may have been designed for Ericsson, but it has many uses now, including:

- Facebook Chat
- Databases
- MMORPGs

Code Example

```erlang
main(A, B, N, X, Pid) ->
  C = A + B,
  Pid ! {"~s lines of text on the screen.~n", C},
  io:format("~s lines of text on the screen.~n",
          [?i2l(C)]),
  if
    N < X -> main(C, A, N+1, X, Pid);
    true -> Pid ! stop
  end.
```
SI 413: Programming Languages and Implementation

The Course

This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

You will learn the skills necessary to quickly learn and begin programming in any new language you may encounter.

During the semester you will become familiar with how a programming language works and how you can write and modify your own language.

Fortran

Designed by John Backus and his IBM team in 1957. It was the first high level assembly language and is still used today, mostly in the scientific community.

Fortran is still used today, primarily by scientists, especially within the astrophysics community. This is because Fortran is good at handling math and numbers.

Fortran has some downsides that keep it from being mainstream. Input and output are incredibly difficult to format if you want anything other than simple read/write. Also, two dimensional arrays or stored differently than in C++, so you have to be mindful that while they have similar syntax, array calls mean entirely different things in Fortran.
This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

**Some languages used**

**Imperative:** Ada, C  
**Object-Oriented:** Java, C++  
**Functional:** Haskell, Lisp  
**Scripting:** Bash, Perl  
**Logic-based:** Prolog

---

**Imperative vs Functional Programming**

Imperative programming is a list of step-by-step instructions for the program to follow in order to execute. The programmer tells the computer exactly how to solve the problem. In functional programming, the programmer defines functions that are very similar to mathematical functions, defining what is computed, not how. For example, adding 1 to each element of a list or array is very different in a functional vs imperative language.

**IMPERATIVE (C)**

```c
int x = 0;
while( x < arraySize ){
    array[x] = array[x] + 1;
    x = x + 1;
}
```

**FUNCTIONAL (Haskell)**

```haskell
map (1+) [LIST]
```
This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

- **Array based programming of J allows for loopless code.**
- **Verbs** are short rules that are applied to an array from right to left.
- **Nouns** are objects such as integers, that verbs operate on.
- There are two kinds of verbs, **monads** and **dyads**. Dyads have arguments before and after the verb while monads are only followed by a noun.
- Monads and Dyads change the meaning of verbs which allow for more ways objects/nouns in arrays can be manipulated.

<table>
<thead>
<tr>
<th>J Term</th>
<th>Other Language Term/Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb</td>
<td>Function or operator</td>
</tr>
<tr>
<td>Noun</td>
<td>Object or variable or constant</td>
</tr>
<tr>
<td>Copula</td>
<td>Assignment</td>
</tr>
<tr>
<td>Punctuation</td>
<td>Separator</td>
</tr>
<tr>
<td>Adverb</td>
<td>n/a</td>
</tr>
<tr>
<td>Conjunction</td>
<td>n/a</td>
</tr>
<tr>
<td>Sentence</td>
<td>Executable unit</td>
</tr>
</tbody>
</table>

Table from "A Casual J Tutorial"  
http://jeffzellner.com/miidaj/
SI 413: Programming Languages and Implementation

This course examines basic concepts underlying the design of modern programming languages: types, control structures, abstraction mechanisms, inheritance, and constructs for programming. This course will include programming assignments in several languages.

OCAML – A functional paradigm programming language that combines object oriented and imperative techniques with static typing to ensure strict type safety. Major Uses: MLDonkey p2p client, Airbus A340 Control Software

Did you know: OCAML is an abbreviation for Objective Categorical Abstract Machine Language?
Overview

Smalltalk is a programming language based on message passing, dynamic strong typing, reflection, and object orientation.

Messages and Methods

**Message**: which action to perform

- `aWorkstation accept: aPacket`
- `aMonster eat: aCookie`

**Method**: how to carry out the action

```plaintext
accept: aPacket
(aPacket isAddressedTo: self)
ifTrue:[
  Transcript show:
    'A packet is accepted by the Workstation ',
    self name asString ]
ifFalse: [super accept: aPacket]
```

Features

- Small and uniform language
- Large library of reusable classes
- Advanced development tools

Smalltalk vs. C++ vs. Java

<table>
<thead>
<tr>
<th></th>
<th>Smalltalk</th>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object Model</strong></td>
<td>Pure</td>
<td>Hybrid</td>
<td>Hybrid</td>
</tr>
<tr>
<td><strong>Garbage Collection</strong></td>
<td>Automatic</td>
<td>Manual</td>
<td>Automatic</td>
</tr>
<tr>
<td><strong>Inheritance</strong></td>
<td>Single</td>
<td>Multiple</td>
<td>Single</td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td>Dynamic</td>
<td>Static</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Reflection</strong></td>
<td>Fully reflective</td>
<td>Introspection</td>
<td>Introspection</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>Semaphores, Monitors</td>
<td>Some libraries</td>
<td>Monitors</td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>Categories, Namespaces</td>
<td>Namespaces</td>
<td>Packages</td>
</tr>
</tbody>
</table>