SI413: Programming Languages and Implementations

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: *ADA*

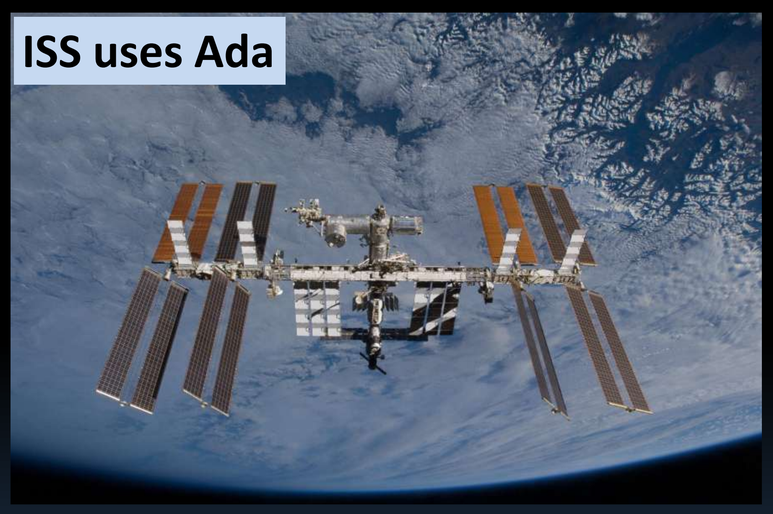
MIDN Kevin Lees, USN
MIDN Alek Ogden, USN

LANGUAGE HISTORY

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

ADA HELLO WORLD

```
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**

Applications

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

SI413: Programming Languages and Implementation

#!/bin/bash

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

Code Examples

```
#!/bin/bash
echo -n "Which fibonacci number do u want to see? "
read nSerial
a=0
b=1
count=2
fibonacci_number=$a
while [ $count -le $nSerial ]; do
        fibonacci_number=$[$a+$b]
        a=$b
        b=$fibonacci_number
        count=$[$count + 1]
done
echo "Fibonacci $nSerial = $fibonacci_number"
```

```
basil[115][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 1
Fibonacci 1 = 0
basil[116][~/413/proj2]$ ./fib.sh 2
Which fibonacci number do u want to see? 2
Fibonacci 2 = 1
basil[117][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 1
Fibonacci 1 = 0
basil[118][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 2
Fibonacci 2 = 1
basil[118][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 3
Fibonacci 3 = 2
basil[118][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 4
Fibonacci 4 = 3
basil[118][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 5
Fibonacci 5 = 5
basil[118][~/413/proj2]$ ./fib.sh
Which fibonacci number do u want to see? 6
Fibonacci 6 = 8
basil[118][~/413/proj2]$
```

```
#!/bin/bash
echo Hello World
```
→ Hello world script

```
diff <(find dir1) <(find dir2)
```
→ Find difference between the contents of 2 directories

```
if [ $file1 -nt  $file2 ]
```
→ Checks if file1 has been modified more recently than file2

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

Cool Stuff

- 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

Gotchas

- Use of whitespace in variable assignments
- Mixing up –eq and =
- Assuming uninitialized variables are zero

SI 413: Programming Languages and Implementation

b r a i n 70 85 c k

What brain**ck has been used for:
Self interpreters!

```
>>>+[[-]>>[-]++>+>+++++++[<++++>>++<-
]++>>+>+>+++++[>++>++++++<<-]+>>>,<++[[>[ -
>>]<[>>]<<-]<[<]<+>>[>]>[<+>-[[<+>-]>]<[[[-]<]++<-
[<+++++++++>[<->-]>>]>>]]<<
]<]<[[<]>[[>]>>[>>]+[<<]<[<]<+>>-]>[>]+[-
>>]<<<<[[<<]<[<]+<<[+>+<<-[>-->+<<-[> +<[>>+<<-
]]]]>[<+>-]<]++>>--
>[>]>>[>>]]<<[>>+<[[<]<]>[[<<]<[<]+[-<+>>-
[<<+>++>- [<->[<<+>>-]]]<[>+<-
]>]>[>]>]>[>>]>>]<<[>>+>>+>>]<<[-
>>>>>>>>]<<[>.>>>>>>>]<<[ >-
>>>>>]<<[>,>>>]<<[>+>]<<[+<<]<]
```

[Daniel B Cristofani (cristofdathevanetdotcom)
http://www.hevanet.com/cristofd/brainfuck/]

Creator: Urban Muller

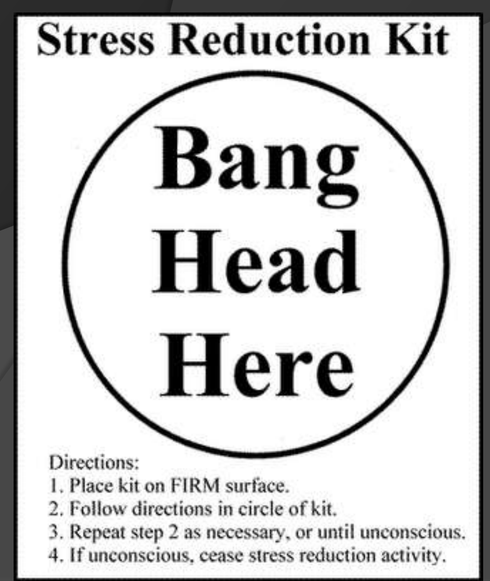
Original Distribution

```
Short:        240 byte compiler. Fun, with src. OS 2.0
Uploader:     umueller amiga physik unizh ch
Type:         dev/lang
Architecture: m68k-amigaos


The brainfuck compiler knows the following instructions:

Cmd  Effect
---  ------
+    Increases element under pointer
-    Decrases element under pointer
>    Increases pointer
<    Decreases pointer
[    Starts loop, flag under pointer
]    Indicates end of loop
.    Outputs ASCII code under pointer
,    Reads char and stores ASCII under ptr

Who can program anything useful with it? :)
```

Hello world!

```
>+++++++++[<+++++++++>-]<.>+++++++[<++++>-
]<+.+++++++..+++.>>>++++++++[<++++>-]
<.>>>++++++++++[<+++++++++>-]<---.<<<<.+++.------
.--------.>>+.
```

Other uses:



Created by:
Brandon Tinkham
William McCrone

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

SI 413: Programming Languages and Implementation

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, inlcuding:

Facebook Chat *Databases* *MMORPGs*

Code Example

```
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.

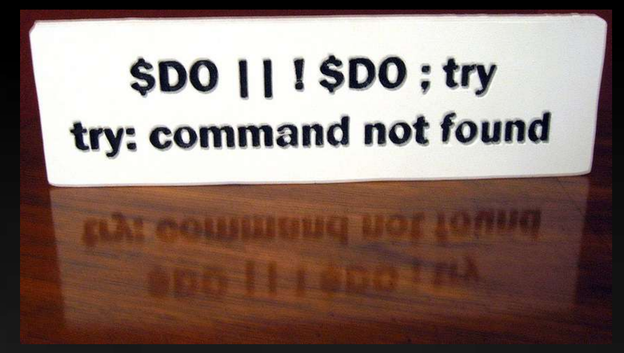
Fortran

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.

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.

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)

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

FUNCTIONAL (Haskell)

```
map (1+)[LIST]
```

SI413: Programming Languages and Implementation

Ethan Panal and Taylor Cooper

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.

J Language

- J is a mathematical language based on the APL language and invented by Kenneth Iverson and Roger Hui
- J language is terse but *powerful*
- J is used by several corporations such as Hewlett Packard and Intel

J Term	Other Language Term/Concept
Verb	Function or operator
Noun	Object or variable or constant
Copula	Assignment
Punctuation	Separator
Adverb	n/a
Conjuction	n/a
Sentence	Executable unit

Table from "A Casual J Tutorial" http://jeffzellner.com/miidaj/

Defining Features

- 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 followe by a noun.
- Monads and Dyads change the meaning of verbs which allow for more ways objects/nouns in arrays can be manipulated.

Example J language:

```
run=: 2 2  $ 1 2 3 4    <-- '$' creates a 2x2 matrix
                            named 'run' and fills it
                            with the integers 1,2,3,4
```

```
   ^run              <--- monad form of the verb '^'
2.71828 7.38906      <--- for every object in 'run' y,
20.0855 54.5982           e^y is outputed.
```

```
   run^2       <-- dyad form of the verb '^'
1  4           <-- use of this verb takes every
9 16                object of 'run' to the second
                    power to output 1,4,9,16
```

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?*

Smalltalk

SI413 – Programming Languages
MIDN La'Shaundra Collins, USN
MIDN Brian Real, USN

Overview

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

Features

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

Messages and Methods

Message: which action to perform

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

Method: how to carry out the action

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

Smalltalk vs. C++ vs. Java

	Smalltalk	C++	Java
Object Model	Pure	Hybrid	Hybrid
Garbage Collection	Automatic	Manual	Automatic
Inheritance	Single	Multiple	Single
Types	Dynamic	Static	Static
Reflection	Fully reflective	Introspection	Introspection
Concurrency	Semaphores, Monitors	Some libraries	Monitors
Modules	Categories, Namespaces	Namespaces	Packages