Introduction

CSE 410/510 ETH: Interactive Programming Environments

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo

Interactive Programming

Interactive programming is not necessarily a well-defined term.

For us, it will mean:

- Building a program a little bit at a time.
- Keeping the program runnable as much as possible.
- Querying and interacting with the running program.
- Modifying the running program!

Most of these points sound like normal Agile development.

The last one does not!

Residential Programming Environments

We will also look at residential programming environments.

In these environments:

- Code and data are part of an image.
- The image is loaded at the beginning of a session.
- The image is saved back out at the end of a session.
- The total environment is seldom restarted, just mutated.

This Course

Our goal is not so much to do specific programming.

Rather, we want to think about how we develop programs.

Why do we develop the way we do?

How could it be different?

The Languages

We will learn three languages:

- Lisp
- Smalltalk
- Forth

They are all highly interactive, but very different

Lisp

Lisp is one of the oldest languages still in use (1958).

It didn't start out nearly so interactive.

Its usage has changed a lot, the language hasn't as much.

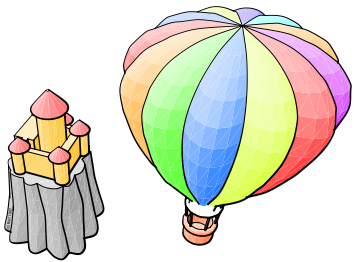
Lisp's foundations are in lambda calculus.

Lisp Example

```
(defun fib (n)
  (cond
    ((= n 1) 1)
    ((= n 2) 1)
    (t (+ (fib (1- n)) (fib (- n 2))))))

(fib 7)
```

Smalltalk



“Smalltalk syntax fits on a postcard.”

Its early focus was on education and accessibility to children.

The first Smalltalk implementations had a GUI and multimedia.

Everything in Smalltalk is an object.

Smalltalk Example

```
fib
    self = 1
    ifTrue: [^1]
    ifFalse: [
        self = 2
        ifTrue: [^1]
        ifFalse: [^(self - 1) fib + (self - 2) fib] ]

7 fib
```

Forth

Forth was developed to provide flexibility and simplicity.

It is designed to be tailored to its environment.

It is frequently implemented on-demand to an unusual degree.

Its simplicity is brutal but elegant.

Its defining feature is the stack.

Forth Example

```
: fib
  dup 1 = if drop 1 else
    dup 2 = if drop 1 else
      dup 1 - recurse swap 2 - recurse +
    then
  then ;

7 fib
```

Syntax

All three of these languages have extremely minimal syntax.

(I don't think this is *necessarily* related to our topic?)

This encourages domain-specific languages.

Often even flow control is built in the language itself![1]

[1] See PlanckForth for a wild ride through this.

Expectations

This will be a highly interactive course.

Lectures will involve:

- Discussion
- Student presentations
- Group work

Participation is graded.

Projects will expect significant implementation time.

There will be little structure to keep you on track.

Participation

We will have lots of discussion.

This means:

- Coming prepared
- Reading in a timely fashion
- Working ahead of deadlines

You will each make multiple (short) presentations.

Programming Assignments

You will do one programming assignment in each language.

They will require you to:

- Learn the language
- Accomplish some (simple) task

You will do lots of reading.

Semester Project

There will be a semester project.

You must:

- Work in a group
- Define the project within your group
- Provide evidence of work
- Participate in peer reviews

There will be a proposal, progress reports, and a final report.

You must present at CSE Demo Day on May 6.

Resources

You may use any necessary resources.

Generative AI/LLMs/*etc.*are forbidden.

You must attribute your sources.

You may not claim others' work as your own.

Your goal is learning the systems –
the project is the vehicle.

My Goals

My goals for this course:

- You learn some languages you didn't know.
- You think about programming in new ways.
- You take something with you in the end.
- We all have fun doing what we do.

Maximize fun and learning for all of us!

A Product of Their Times

These languages were all extravagant:
They imagined a computer dedicated to the programmer.

(Lisp came about this somewhat later.)

By the mid-1970s to mid-1980s this became feasible.

This kind of thinking changed language design.

Batch Computing

Early computers were extremely expensive.

Programmers edited offline and submitted jobs.

Time Sharing

With progress, computer costs came down.

A programmer would have a dedicated terminal.
(Sometimes only while programming!)

A Dedicated Computer

Eventually a programmer could use an entire computer.

This was, again, very expensive early on.

Exploring Productivity

This was a time of radical exploration in:

- Languages
- System design
- User interfaces
- *etc.*

These languages were at the intersection of several topics.

There was an implicit assumption that the user is a programmer.

Next Time…

- Introductions and experiences!
- Quick demos
- Q&A?
- TBD?

License

To retrieve a copy of this material, or related materials, see `https://www.cse.buffalo.edu/~eblanton/`.