CptS 355- Programming Language Design

Lecture 2
Programming Languages
Introduction

Instructor: Jeremy E. Thompson

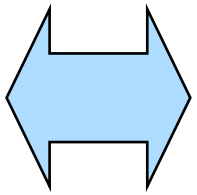
Spr 2021

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Programming languages:

- What is a programming language?
  - abstraction of virtual machine
    - a set of detailed instructions for specifying what you want the computer to do without getting down into the bits (languages from the implementer's point of view)
  - way of expressing algorithms (languages from the user's point of view)
  - this course tries to balance coverage of these two angles
    - we will talk about language features for their own sake, and how they can be implemented

```
int sum(int[] x) {
    int sum = 0;
    n = 0;
    while (n < x.length) {
        sum += x[n];
    }
    return sum;
}
```



```
0010101010101010
1010101111111010
1110101010101110
0010101010101010
...
```

Programming languages:

- What makes a programming language a programming language?
- What do you find in a programming language?

Programming languages:

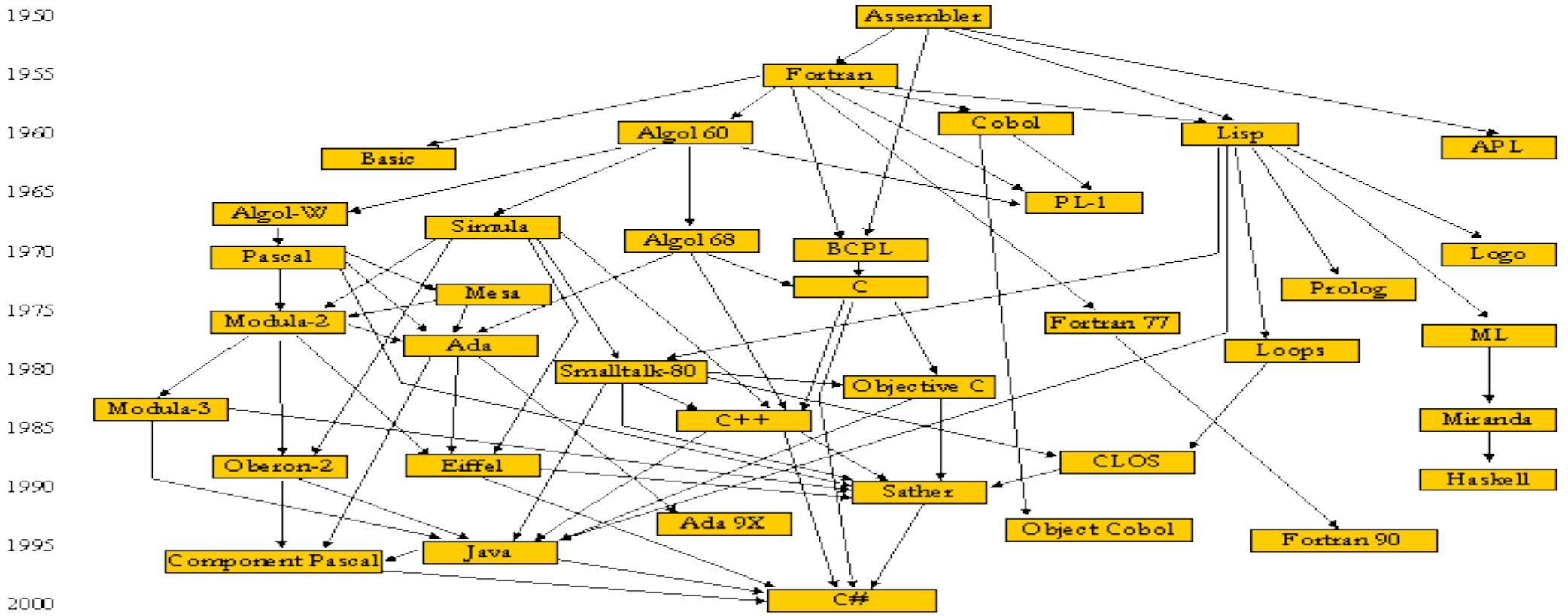
- What is a (computer) program?
  - (Some) Strings of characters are programs
  - (Some) Numbers are programs
  - Programs == Data == Programs
    - Programs are data, some numbers/strings are programs, some aren't
    - Which strings are programs depend on what programming language we talk about
    - How do we go about as Computer scientists specify which strings are programs
    - That is a field called grammars, and we will not talk about grammars in 355 but you have or will cover it in 317
- When does data become a program?
  - When you interpret it. When numbers/strings can be interpreted as a program
  - Given a programming language, not every number is a program and not every string is a program
  - What are strings after all (in computers)?
    - Arrays of numbers. We see a sequence of characters

Programming languages:

- Why are there so many programming languages?
  - evolution -- we've learned better ways of doing things over time
  - orientation toward special purposes
  - orientation toward special hardware
  - personal preference
  - socio-economic factors:
    - proprietary interests
    - commercial advantage

Evolution of Programming Languages

Programming Language Family Tree



1 November, 2000

(C) Fachhochschule Aargau für Technik, Wirtschaft und Gestaltung
Nordwestschweiz

10

Programming languages:

- What makes a good programming language? What makes a language successful?
  - easy to learn (BASIC, Pascal, Scheme)
  - "powerful“- easy to use once *fluent* (C++, Common Lisp, Perl)
  - easy to implement (BASIC, Python)
  - possible to compile to very good (fast/small) code (Fortran)
  - exceptionally good at something important (PHP, Ruby on Rails, R)
  - backing of a powerful sponsor (COBOL, Ada, Visual Basic, C#)

Programming languages:

- Programming languages *guide* the way we think about algorithmic problems
  - When you learn new programming languages you learn new ways of *thinking* about problems
  - Think about your own transitions (from C to C++, or ? to Java)
    - What was added technically there?
    - Once you learned about objects, did it change the way you solve problems?
  - You probably started to think about solutions using objects embedded in them, with states and behaviors
    - Knowing about objects affects how you program

Programming languages:

- Assume you program using an imperative language (C, C++, etc.) How do you approach a given programming problem?
- Example: Find the length of a list/array
  - Write a loop that iterates over a list and increments a counter

- How can we do this in functional programming?

Programming languages:

- **Quicksort in C:**

```
qsort( a, lo, hi ) {
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);
    t = a[l];
    a[l] = a[hi];
    a[hi] = t;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

- **Quicksort in Haskell:**

```
quicksort :: ( Ord a ) = > [ a ] -> [ a ]
quicksort [] = []
quicksort ( x : xs ) =
    let smallerSorted = quicksort [ a | a <- xs , a <= x ]
        biggerSorted = quicksort [ a | a <- xs , a > x ]
    in smallerSorted ++ [ x ] ++ biggerSorted
```

Why study programming languages?

1. Teach you to think in different programming paradigms
   - Knowing multiple languages allows problem to be looked at from a variety of perspectives
   - Multiple solutions can be compared
     - most natural solution for the problem can then be selected
2. Teach you strengths, weaknesses, and applicability of various programming languages
   - Use most *appropriate* programming language for your task
     - E.g. Java is great for writing applications
     - E.g. C is great for systems programming
3. Make it *easier* to learn new languages

Languages to solve problems:

1. FORTRAN:
2. COBOL:
3. C :
4. C++:
5. Java:
6. C#:

Languages to solve problems:

7. PYTHON:

8. RUBY:

9. JavaScript :

10. PHP:

11. ML:

12. Scheme:

Programming Paradigms

- A programming *paradigm* is a style, or "way," of programming

- There are several programming paradigms:
  - Imperative:
    - How to perform tasks (algorithms) and track changes in state
  - Functional
    - What information is desired and what transformations are required
  - Declarative
    - What outcome the program should accomplish (rather than how it should be accomplished)
  - ….

Examples:

- Object Oriented & Imperative: (Python)

```
array = [1, 2, 3, 4, 5, 6]
sum = 0

for element in array:
  if element % 2 == 1:
    sum += element
```

- Functional: (Haskell)

```
numbers = [1, 2, 3, 4, 5, 6]

odd :: Integral a => a -> Bool

      -- Defined in ‘GHC.Real’

s = sum (filter odd numbers)
```

Examples:

- Imperative:

?

- Declarative: (SQL)

```
SELECT sid,AVG(grade)
FROM StudentT
WHERE major in ('CptS','EE','SE', 'CptE')
GROUP BY major
```

Breakout

- Assume an array "`studentA`" stores the same information as the database table "`studentT`"

Questions?