Potted History of Programming Languages

Vitaly Shmatikov

Algorithm

- Abu Ja’far Muhammad ibn Musa al-Khorezmi (“from Khorezm”)
  - Lived in Baghdad around 780 – 850 AD
  - Chief mathematician in Khalif Al Mamun’s “House of Wisdom”
  - Author of “A Compact Introduction To Calculation Using Rules Of Completion And Reduction”

Removing negative units from the equation by adding the same quantity on the other side (“al-gabr” in Arabic)

“Calculus of Thought”

- Gottfried Wilhelm Leibniz
  - 1646 - 1716
  - Inventor of calculus and binary system
  - “Calculus ratiocinator”: human reasoning can be reduced to a formal symbolic language, in which all arguments would be settled by mechanical manipulation of logical concepts
  - Invented a mechanical calculator

Formalisms for Computation (1)

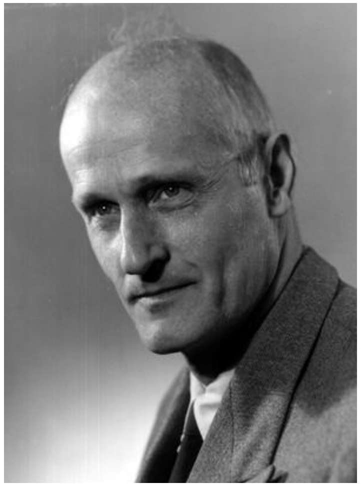
◆ Predicate logic
- Gottlöb Frege (1848-1925)
- Formal basis for proof theory and automated theorem proving
- Logic programming
  - Computation as logical deduction

◆ Turing machines
- Alan Turing (1912-1954)
- Imperative programming
  - Sequences of commands, explicit state transitions, update via assignment

Formalisms for Computation (2)

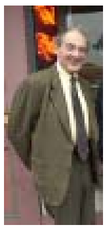
- Lambda calculus
  - Alonzo Church (1903-1995)
  - Formal basis for all functional languages, semantics, type theory
  - Functional programming
    - Pure expression evaluation, no assignment operator

- Recursive functions & automata
  - Stephen Kleene (1909-1994)
  - Regular expressions, finite-state machines, PDAs



Church's Legacy

Alonzo Church (PhD Princeton 1927)



Hartley Rogers (PhD Princeton 1952)

Recursion theory

Albert Meyer (PhD Harvard 1972)

Semantics, concurrency

John Mitchell (PhD MIT 1984)

Theory of object-oriented languages

Vitaly Shmatikov (PhD Stanford 2000)

1916 other academic descendants

Church's Thesis

- All these different syntactic formalisms describe the same class of mathematical objects
  - Church's Thesis: "Every effectively calculable function (effectively decidable predicate) is general recursive"
  - Turing's Thesis: "Every function which would be naturally regarded as computable is computable by a Turing machine"
- Recursion, lambda-calculus and Turing machines are equivalent in their expressive power
- Why is this a "thesis" and not a "theorem"?

Formalisms for Computation (3)

- **Combinatory logic**
  - Moses Schönfinkel (1889-1942??)
  - Haskell Curry (1900-1982)

- **Post production systems**
  - Emil Post (1897-1954)

- **Markov algorithms**
  - Andrey Markov (1903-1979)

Programming Language

- Formal notation for specifying computations
  - Syntax (usually specified by a context-free grammar)
  - Semantics for each syntactic construct
  - Practical implementation on a real or virtual machine
    - Translation vs. compilation vs. interpretation
      - C++ was originally translated into C by Stroustrup's Cfront
      - Java originally used a bytecode interpreter, now native code compilers are commonly used for greater efficiency
      - Lisp, Scheme and most other functional languages are interpreted by a virtual machine, but code is often precompiled to an internal executable for efficiency
    - Efficiency vs. portability

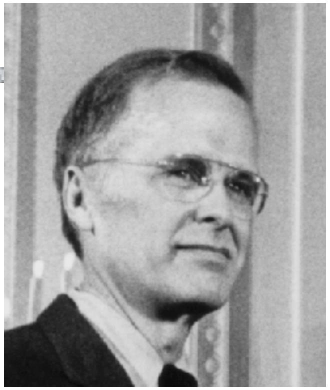
Assembly Languages

- Invented by machine designers the early 1950s
- Mnemonics instead of binary opcodes

```
push ebp
mov ebp, esp
sub esp, 4
push edi
```

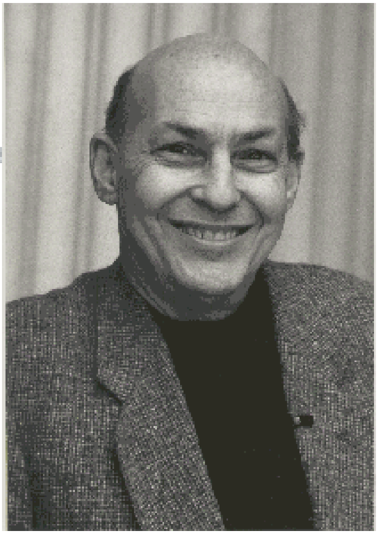
- Reusable macros and subroutines

FORTRAN



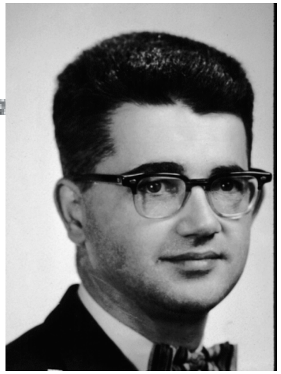
- Procedural, imperative language
  - Still used in scientific computation
- Developed at IBM in the 1950s by John Backus (1924-2007)
  - Backus's 1977 Turing award lecture (see course website) made the case for functional programming
  - On FORTRAN: "We did not know what we wanted and how to do it. It just sort of grew. The first struggle was over what the language would look like. Then how to parse expressions – it was a big problem…"
    - BNF: Backus-Naur form for defining context-free grammars

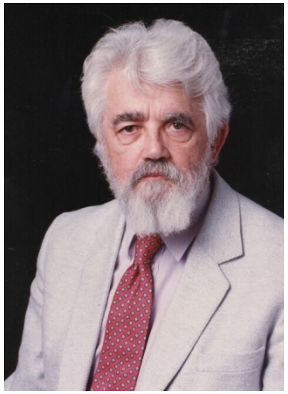
From FORTRAN to LISP



"Anyone could learn Lisp in one day, except that if they already knew FORTRAN, it would take three days"
- Marvin Minsky

LISP





- Invented by John McCarthy (b. 1927, Turing award: 1971)
  - See original paper on course website
- Formal notation for lambda-calculus
- Pioneered many PL concepts
  - Automated memory management (garbage collection)
  - Dynamic typing
  - No distinction between code and data
- Still in use: ACL2, Scheme, ...

LISP Quotes

- “The greatest single programming language ever designed” --Alan Kay
- “LISP being the most powerful and cleanest of languages, that's the language that the GNU project always prefer” -- Richard Stallman
- “Programming in Lisp is like playing with the primordial forces of the universe. It feels like lightning between your fingertips.” -- Glenn Ehrlich
- “Lisp has all the visual appeal of oatmeal with fingernail clippings mixed in” -- Larry Wall
- “LISP programmers know the value of everything and the cost of nothing” -- Alan Perlis

Algol 60

- Designed in 1958-1960
- Great influence on modern languages
  - Formally specified syntax (BNF)
    - Peter Naur: 2005 Turing award
  - Lexical scoping: begin ... end or {...}
  - Modular procedures, recursive procedures, variable type declarations, stack storage allocation

- "Birth of computer science" -- Dijkstra
- "A language so far ahead of its time that it was not only an improvement on its predecessors, but also on nearly all its successors" -- Hoare

Algol 60 Sample

```
real procedure average(A,n);
   real array A; integer n;          no array bounds
   begin
        real sum; sum := 0;
        for i = 1 step 1 until n do
                sum := sum + A[i];
        average := sum/n             no ; here
   end;
```

set procedure return value by assignment

Algol Oddity

- Question
  - Is x := x equivalent to doing nothing?
- Interesting answer in Algol

```
integer procedure p;
begin
        ...
        p := p
        ...
 end;
```

  - Assignment here is actually a recursive call

Some Trouble Spots in Algol 60

- Type discipline improved by later languages
  - Parameter types can be array
    - No array bounds
  - Parameter type can be procedure
    - No argument or return types for procedure parameter
- Parameter passing methods
  - Pass-by-name had various anomalies
    - "Copy rule" based on substitution, interacts with side effects
  - Pass-by-value expensive for arrays
- Some awkward control issues
  - Goto out of block requires memory management

Algol 60 Pass-by-Name

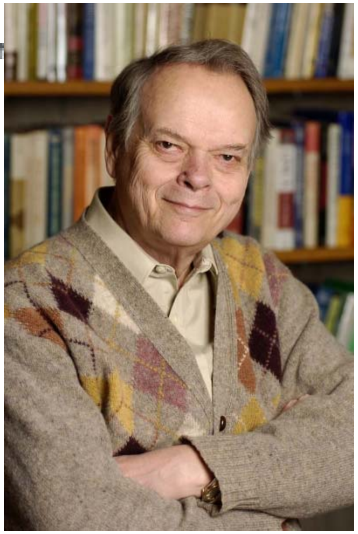
- Substitute text of actual parameter
  - Unpredictable with side effects!
- Example

```
procedure inc2(i, j);
   integer i, j;
   begin
      i := i+1;
      j := j+1
   end;
inc2 (k, A[k]);
```

```
begin
   k := k+1;
   A[k] := A[k] +1
end;
```

Is this what you expected?

Algol 60 Legacy



"Another line of development stemming from Algol 60 has led to languages such as Pascal and its descendants, e.g., Euclid, Mesa, and Ada, which are significantly lower-level than Algol. Each of these languages seriously restricts the block or procedure mechanism of Algol by eliminating features such as call by name, dynamic arrays, or procedure parameters."

- John C. Reynolds

Algol 68

- Very elaborate type system
  - Complicated type conversions
  - Idiosyncratic terminology
    - Types were called “modes”
    - Arrays were called “multiple values”
- vW grammars instead of BNF
  - Context-sensitive grammar invented by A. van Wijngaarden
- Eliminated pass-by-name
- Considered difficult to understand

Pascal

- Designed by Niklaus Wirth
  - 1984 Turing Award
- Revised type system of Algol
  - Good data structure concepts
    - Records, variants, subranges
  - More restrictive than Algol 60/68
    - Procedure parameters cannot have procedure parameters
- Popular teaching language
- Simple one-pass compiler

Limitations of Pascal

- Array bounds part of type

  illegal

  ```
  procedure p(a: array [1..10] of integer)
  procedure p(n: integer,  a: array [1..n] of integer)
  ```

  - Attempt at orthogonal design backfires
    - Parameter must be given a type
    - Type cannot contain variables

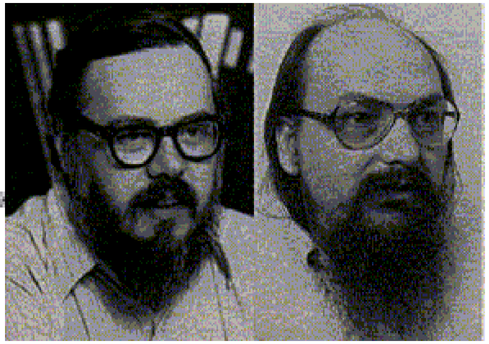
  How could this have happened? Emphasis on teaching!

- Not successful for "industrial-strength" projects
  - See Kernighan's "Why Pascal is not my favorite language" on the course website

SIMULA 67

- Ole-Johan Dahl (1931-2002)
- Kristen Nygaard (1926-2002)
  - Joint 2001 Turing Award
- First <u>object-oriented</u> language
  - Objects and classes
  - Subclasses and inheritance
  - Virtual procedures

BCPL / B / C Family



- Born of frustration with big OSes and big languages (Multics, PL/I, Algol 68)
- Keep lexical scope and recursion
- Low-level machine access
  - Manual memory management
  - Explicit pointer manipulation
  - Weak typing (introduced in C)
- Systems programming for small-memory machines
  - PDP-7, PDP-11, later VAX, Unix workstations and PCs
  - C has been called a "portable assembly language"

BCPL

- Designed by Martin Richards (1966)
- Emphasis on portability and ease of compilation
  - Front end: parse + generate code for virtual machine
  - Back end: translate code for native machine
- Single data type (word), equivalence of pointers and arrays, pointer arithmetic – this is unusual!

"The philosophy of BCPL is not one of the tyrant who thinks he knows best and lays down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering his services to the best of his ability without complaint, even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and is not hemmed in by petty restrictions."

Arrays and Pointers

- An array is treated as a pointer to first element
- BCPL: let V = vec 10

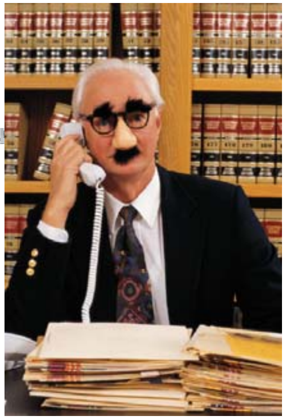
  V!i to index the i^{th} array element
- C: A[i] is equivalent to

  pointer dereference *((A) + (i))

B

- “BCPL squeezed into 8K bytes of memory & filtered through Ken Thompson’s brain”
- Very compact syntax
  - One-pass compiler on a small-memory machine
    - Generates intermediate “threaded code,” not native code
  - No nested scopes
  - Assignment: `=` instead of Algol-style `:=`
    - How many times have you written `if (a=b) { ... }` ?
  - Pre-/postfix notation: `x++` instead of `x:=x+1`
  - Null-terminated strings
    - In C, strings are null-terminated sequences of bytes referenced either by a pointer-to-char, or an array variable s[]

Lex the Language Lawyer



Can only be applied to l-value (more about this later in the course)

++x++

This is evaluated first

Increments x, returns old value

Not an l-value! This is illegal in C!

Now C++ ...

```
class DoublePlus {
public:
    // prefix operator
    DoublePlus operator++() { ... }
    // postfix operator
    DoublePlus operator++(int) { ... }
};
```

What is this for?

More Fun with Prefix and Postfix

What do these mean?

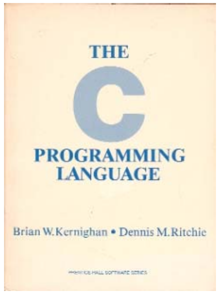
```
x+=x++
```

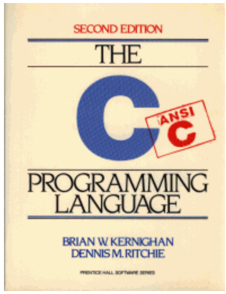
```
++x + x++
```

C

- Bell Labs 1972 (Dennis Ritchie)
- Development closely related to UNIX
  - 1983 Turing Award to Thompson and Ritchie

- Added weak typing to B
  - int, char, their pointer types
  - Typed arrays = typed pointers
    - int a[10]; … x = a[i]; means x = *(&a[0]+i*sizeof(int))
- Compiles to native code





Types in C

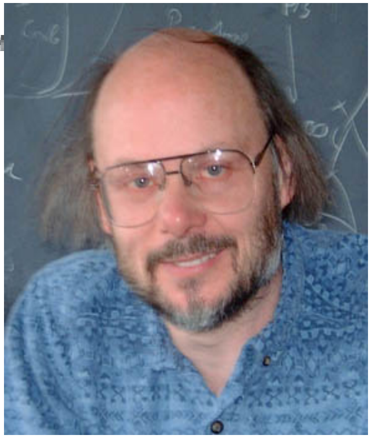
- Main difference between B and C
- Syntax of type rules influenced by Algol 68
  - int i, *pi, **ppi;
  - int f(), *f(), **f(), *(*pf)(), (*pf)(int);
  - int *api[10], (*pai)[10];
- Also structs and unions

What do these declarations mean?

Evolution of C

- 1973-1980: new features; compiler ported
  - unsigned, long, union, enums
- 1978: K&R C book published
- 1989: ANSI C standardization
  - Function prototypes as in C++
- 1999: ISO 9899:1999 also known as “C99”
  - Inline functions, C++-like decls, bools, variable arrays
- Concurrent C, Objective C, C*, C++, C#
- “Portable assembly language”
  - Early C++, Modula-3, Eiffel source-translated to C

C++



- Bell Labs 1979 (Bjarne Stroustrup)
  - "C with Classes" (C++ since 1983)
- Influenced by Simula
- Originally translated into C using Cfront, then native compilers
  - GNU g++
- Several PL concepts
  - Multiple inheritance
  - Templates / generics
  - Exception handling

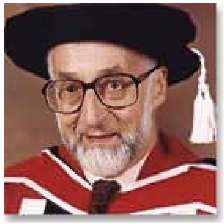
Java

- Sun 1991-1995 (James Gosling)
  - Originally called Oak,
    intended for set top boxes
- Mixture of C and Modula-3
  - Unlike C++
    - No templates (generics), no multiple inheritance, no operator overloading
  - Like Modula-3 (developed at DEC SRC)
    - Explicit interfaces, single inheritance, exception handling, built-in threading model, references & automatic garbage collection (no explicit pointers!)
- "Generics" added later

Other Important Languages

- Algol-like
  - Modula, Oberon, Ada

- Functional
  - ISWIM, FP, SASL, Miranda, Haskell, LCF, ML, Caml, Ocaml, Scheme, Common LISP



- Object-oriented
  - Smalltalk, Objective-C, Eiffel, Modula-3, Self, C#, CLOS

- Logic programming
  - Prolog, Gödel, LDL, ACL2, Isabelle, HOL

… And More

- Data processing and databases
  - Cobol, SQL, 4GLs, XQuery
- Systems programming
  - PL/I, PL/M, BLISS
- Specialized applications
  - APL, Forth, Icon, Logo, SNOBOL4, GPSS, Visual Basic
- Concurrent, parallel, distributed
  - Concurrent Pascal, Concurrent C, C*, SR, Occam, Erlang, Obliq

Forth

- Program BIOS, bootloaders, device firmware
  - Sun BIOS, Lockheed Martin's missile tracking, FedEx barcode readers ...

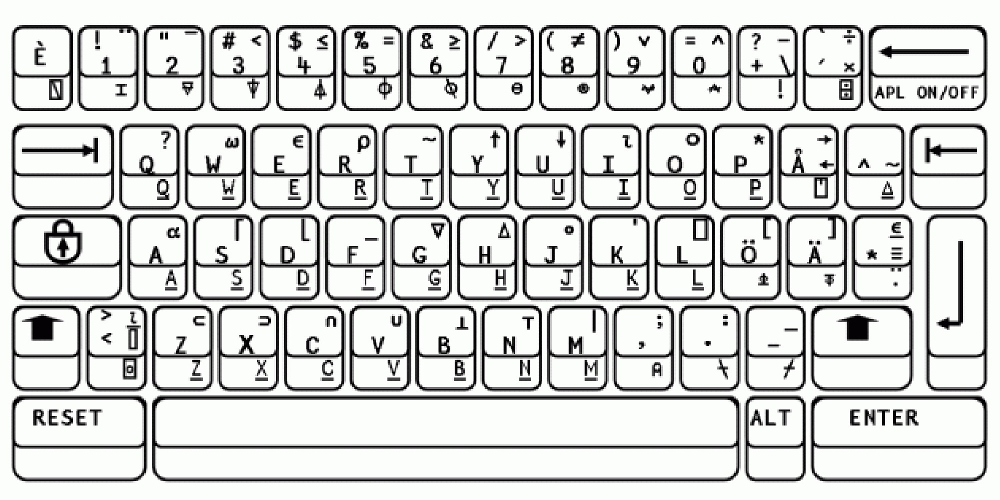
```
hex 4666 dup negate do i 4000 dup 2* negate
  do 2a 0 dup 2dup 1e 0 do 2swap * d >>a 4
  pick + -rot - j + dup dup * e >>a rot dup
  dup * e >>a rot swap 2dup + 10000 > if
  3drop 2drop 20 0 dup 2dup leave then loop
  2drop 2drop type 268 +loop cr drop 5de
  +loop
```

APL

- Computation-intensive tasks, esp. in finance
  - Mortgage cash flow analysis, insurance calculations, ...

Got this?

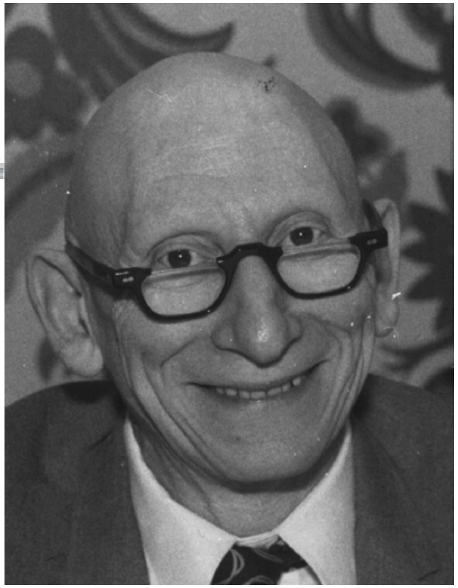
```
[6]    L←(L⍳':')↓L←,L
[7]    L←LJUST VTOM',',L
[8]    S←¯1++/∧\L≠'('
[9]    X←0⌈⌈/S
[10]   L←S⌽(-(⍴L)+0,X)↑L
[11]   A←((1↑⍴L),X)↑L
[12]   N←0 1↓DLTB(0,X)↓L
[13]   N←,'α',N
[14]   N[(N='_')/⍳⍴N]←' '
[15]   N←0 ¯1↓RJUST VTOM N
[16]   S←+/∧\' '≠⌽N
```



Brave New World

- Programming tool "mini-languages"
  - awk, make, lex, yacc, autoconf ...
- Command shells, scripting and "web" languages
  - sh, csh, tcsh, ksh, zsh, bash ...
  - Perl, JavaScript, PHP, Python, Rexx, Ruby, Tcl, AppleScript, VBScript ...
- Web application frameworks and technologies
  - ASP.NET, AJAX, Flash, Silverlight ...
    - **Note:** HTML/XML are markup languages, <u>not</u> programming languages, but they often embed executable scripts like Active Server Pages (ASPs) & Java Server Pages (JSPs)

Why So Many Languages?



"There will always be things we wish to say in our programs that in all languages can only be said poorly."

- Alan Perlis

What's Driving Their Evolution?

- Constant search for better ways to build software tools for solving computational problems
  - Many PLs are general purpose tools
  - Others are targeted at specific kinds of problems
    - For example, massively parallel computations or graphics
- Useful ideas evolve into language designs
  - Algol → Simula → Smalltalk → C with Classes → C++
- Often design is driven by expediency
  - Scripting languages: Perl, Tcl, Python, PHP, etc.
    - "PHP is a minor evil perpetrated by incompetent amateurs, whereas Perl is a great and insidious evil, perpetrated by skilled but perverted professionals." - Jon Ribbens

What Do They Have in Common?

- Lexical structure and analysis
  - Tokens: keywords, operators, symbols, variables
  - Regular expressions and finite automata
- Syntactic structure and analysis
  - Parsing, context-free grammars
- Pragmatic issues
  - Scoping, block structure, local variables
  - Procedures, parameter passing, iteration, recursion
  - Type checking, data structures
- Semantics
  - What do programs mean and are they correct

Core Features vs. Syntactic Sugar

- What is the core high-level language syntax required to emulate a universal Turing machine?
  - What is the core syntax of C?
    - Are ++, --, +=, -=, ?:, for/do/while part of the core?
- Convenience features?
  - Structures/records, arrays, loops, case/switch?
  - Preprocessor macros (textual substitution)
  - Run-time libraries
    - String handling, I/O, system calls, threads, networking, etc.
  - "Syntactic sugar causes cancer of the semicolons"

    \- Alan Perlis

Final Thoughts

- There will be new languages invented
  - You will have to spend time learning them on your own!
  - For now, enjoy the luxury of being able to take a class
- Conflicting goals for language design can lead to feature creep and hideous complexity
  - Exhibit A: PL/I
  - Exhibit B: C++
- Then someone gets fed up ...
  - A language that adopts the original simple and elegant ideas, while eliminating the complexity (e.g., Java)