CMSC 330: Organization of Programming Languages

A Brief History of Programming Languages

Babylon

- Founded roughly 4000 years ago
  - Located near the Euphrates River, 56 mi south of Baghdad, Iraq
- Historically influential in ancient western world
- Cuneiform writing system, written on clay tablets
  - Some of those tablets survive to this day
  - Those from Hammurabi dynasty (1800-1600 BC) include mathematical calculations
    - (Also known for Code of Hammurabi, an early legal code)

A Babylonian Algorithm

A [rectangular] cistern.
The height is 3, 20, and a volume of 27, 46, 40 has been excavated.
The length exceeds the width by 50.
You should take the reciprocal of the height, 3, 20, obtaining 18.
Multiply this by the volume, 27, 46, 40, obtaining 8, 20.
Take half of 50 and square it, obtaining 10, 25.
Add 8, 20, and you get 8, 30, 25
The square root is 2, 55.
Make two copies of this, adding [25] to the one and subtracting from the other.
You find that 3, 20 [i.e., 3 1/3] is the length and 2, 30 [i.e., 2 1/2] is the width.
This is the procedure.

– Donald E. Knuth, *Ancient Babylonian Algorithms*, CACM July 1972

The number n, m represents $n*(60^k) + m*(60^{k-1})$ for some k

More about Algorithms

- Euclid's Algorithm (Alexandria, Egypt, 300 BC)
  - Appeared in *Elements*
  - Computes gcd of two integers

```
let rec gcd a b =
    if b = 0 then a else gcd b (a mod b)
```

- Al-Khwarizmi (Baghdad, Iraq, 780-850 AD)
  - *Al-Khwarizmi Concerning the Hindu Art of Reckoning*
  - Translated into Latin (in 12th century?)
    - Author's name rendered in Latin as *algoritmi*
    - Thus the word *algorithm*

The Analytical Engine (Babbage)

- Charles Babbage (1791-1871, London, England)
  - – Developed a mechanical calculator
  - – Then during 1830's developed plans for the *Analytical Engine*
    - But plans only discovered in 1937
    - Built in 1991 at the Science Museum of London
  - – Included branching, looping, arithmetic, and storage
  - – Programmed using punch cards

Alonzo Church (1903-1995)

- Mathematician at Princeton Univ.
- Three key contributions:
  - – The lambda calculus (lectures in 1936, publ. 1941)
  - – Church's Thesis
    - All effective computation is expressed by recursive (decidable) functions
  - – Church's Theorem
    - First order logic is undecidable

Alan Turing (1912-1954)

- The father of modern computer science
  - – Dissertation work advised by Church at Princeton
  - – Formulated the Turing machine (~1936)
    - Σ – A finite alphabet
    - Q – a set of states
    - $s \in Q$ – A start state
    - F □ Q – The final states
    - $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$
      - – If $\delta(q, a) = (q', a', d)$, then if we're in state q and see a on the tape, then replace it by a', move to state q', and move the position of the tape either left or right
  - – A formal definition of a computable algorithm

Other Early Computers

- ABC (1939-1942)
  - – Atanasoff and Berry Computer, at Iowa State Univ.
  - – First electronic digital computer
    - As decided by a judge in 1973! (Invalidated ENIAC patent)
- Z3 (1945)
  - – Konrad Zuse, essentially isolated from everyone else
  - – Used Plankalkül, a sophisticated programming lang.
    - But no one knew about his results, so not influential

Other Early Computers (cont'd)

- Harvard Mark I (1944)
  - Aiken, IBM
  - Electronic, used relays
- ENIAC (1946)
  - Electronic Numerical Integrator and Computer
  - Developed by Eckert and Mauchly at UPenn
  - Electronic, general purposes
  - Used vacuum tubes

The First Programming Languages

- Early computers could be "programmed" by rewiring them for specific applications
  - Tedious, error prone
- John von Neumann (1903-1957)
  - Three CS contributions (famous for lots of other stuff)
    - von Neumann machine – the way computers are built today
      - A *stored program architecture*
        - » Program stored in memory, so can be modified
      - (Unclear that he actually invented this...)
    - "Conditional control transfer" – if and for statements
      - Allows for reusable code, like subroutines
    - Merge sort algorithm

Pseudocodes (Assembly Interpreter)

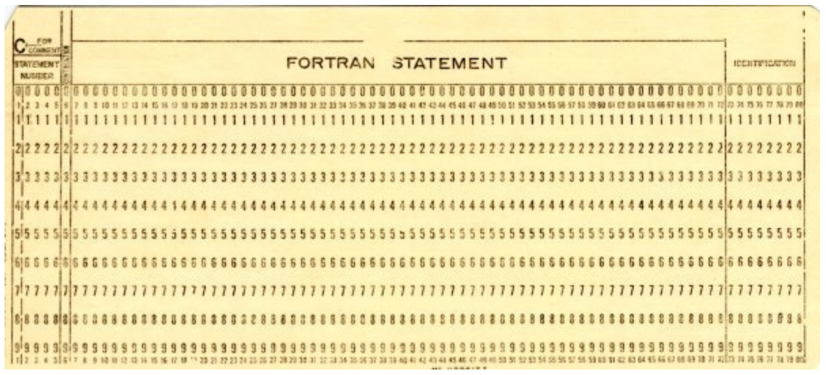
- Short Code (1949)
  - John Mauchly
  - Interpreted instructions
    - E.g., X0 = sqrt(abs(Y0)) becomes 00 X0 03 20 06 Y0
      - 06 = abs, 20 = sqrt, 03 = assignment
  - But needed to translate by hand
- A-0 Compiler (1951; Grace Murray Hopper)
  - Translated symbolic code into machine code
    - Sounds like an assembler...
  - Assigned numbers to routines stored on tape
    - Which would then be retrieved and put in memory

FORTRAN (1954-1957)

- FORmula TRANslator
- Developed at IBM by John Backus et al
  - Aimed at scientific computation
  - Computers slow, small, unreliable
    - So FORTRAN needed to produce efficient code
- Features (FORTRAN I)
  - Variable names (up to 6 chars)
  - Loops and Arithmetic Conditionals
    - IF (ICOUNT-1) 100, 200, 300
  - Formatted I/O
  - Subroutines

Writing FORTRAN Programs

- Programs originally entered on punch cards
  - Note bevels on top-left corner for orientation
  - First five columns for comment mark or statement number
  - Each column represents one character



Punch Card Programming

- Not interactive!
  - Feed the deck into the machine
    - Or give it to someone to put in
  - Eventually get back printout with code and output
    - Could take a couple of hours if machine busy
- Long test-debug cycle
  - Debugging by hand critical to not wasting time
    - Don't want to wait several hours to find you made a typo
- What happens if you drop your deck of cards?
  - Could put sequence number in corner for ordering
  - Hard to maintain this as you keep modifying program

Example (FORTRAN 77)

C = "comment"

All-caps

For loop; I goes from 2 to 12 in increments of 1

Cols 1-6 for comment or stmt label

```
C A PROGRAM TO COMPUTE MULTIPLICATION TABLES
      PROGRAM TABLES
      DO 20 I = 2,12
      PRINT *,I,' TIMES TABLE'
      DO 10 J = 1,12
10      PRINT *,I,' TIMES',J,' IS',I*J
20      CONTINUE
      END
```

Source: University of Strathclyde Computer Centre, Glasgow, Scotland

End of program

FORTRAN Parsing Example

- Blanks don't matter in FORTRAN
  - And identifiers can have numbers in them
- Consider parsing the following two statements

```
DO 20 I = 2,12
DO 20 I = 2.12
```

  - The first is a loop
  - The second is an assignment to "DO20I"
  - Notice that we need many characters of lookahead to tell the difference between the two for a parser

FORTRAN Today

- Success of FORTRAN led to modern compilers
- Up to FORTRAN 95
  - Heavily modernized version of the original language
  - Includes pointers, recursion, more type checking, dynamic allocation, modules, etc.
- Only (?) major language with column-major arrays
- Still popular for scientific computing applications
  - Because FORTRAN compilers produce good code
    - C has pointers, which can hurt optimization
      - FORTRAN has arrays, but compiler can assume that if x and y are arrays then x and y are unaliased
    - Java interpreted or JIT'd, and uses dynamic dispatch heavily

COBOL (1959)

- COmmon Business Oriented Language
  - Project led by Hopper (again!)
- Design goals
  - Look like simple English
  - Easy to use for a broad base
  - Notice: *not* aimed at scientific computing
    - Aimed at business computing instead, very successfully
- Key features
  - Macros
  - Records
  - Long names (up to 30 chars), with hyphen

COBOL Example (Part 1)

```
     $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  Iteration-If.
AUTHOR.  Michael Coughlan.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  Num1           PIC 9  VALUE ZEROS.
01  Num2           PIC 9  VALUE ZEROS.
01  Result        PIC 99 VALUE ZEROS.
01  Operator       PIC X  VALUE SPACE.
```

Program data (variables)

Single-digit number

Initial value

Level number; can be used to express hierarchy (records)

Character

The source of Y2K bugs

Source: http://www.csis.ul.ie/COBOL/examples/conditn/IterIf.htm

COBOL Example (Part 2)

```
PROCEDURE DIVISION.
Calculator.
   PERFORM 3 TIMES
      DISPLAY "Enter First Number      : " WITH NO ADVANCING
      ACCEPT Num1
      DISPLAY "Enter Second Number     : " WITH NO ADVANCING
      ACCEPT Num2
      DISPLAY "Enter operator (+ or *) : " WITH NO ADVANCING
      ACCEPT Operator
      IF Operator = "+" THEN
         ADD Num1, Num2 GIVING Result
      END-IF
      IF Operator = "*" THEN
         MULTIPLY Num1 BY Num2 GIVING Result
      END-IF
      DISPLAY "Result is = ", Result
   END-PERFORM.
   STOP RUN.
```

Iteration

Data Layout

- Variables have fixed position in memory
  - Same with early FORTRAN compilers
- Advantages?
  - Compilers easier to write
  - Efficient at runtime
- Disadvantages?
  - No dynamic memory alloc (i.e., no data structures)
  - No recursive functions

COBOL Today

- Was a DoD requirement at one time
  - Important element of its success
- Still used today
  - Legacy mainframe applications
  - New standard in 2002
  - Language has been updated for new features
    - Object-oriented?!
    - Unicode support
    - XML support

LISP (1958)

- LISt Processing
- Developed by John McCarthy at MIT
  - Designed for AI research
- Key ideas:
  - Symbolic expressions instead of numbers
  - Lists, lists, lists
  - Functions, functions, functions
    - Compose simpler functions to form more complex functions
    - Recursion
  - Garbage collection

LISP Code

- Scheme looks like LISP

```
(defun factorial (n)
      (cond ((zerop n) 1)
            (t (times n (factorial (sub1 n)))))
```

- Implemented on IBM 704 machine
  - Machine word was 36 bits
    - Two 15-bit parts, “address” and “decrement,” distinguished
    - car = “Contents of the Address part of Register number”
    - cdr = “Contents of the Decrement part of Register number”
- Invented maplist function
  - Same as map function in OCaml
- Used Church's lambda notation for functions

LISP Code as Data

- Notice that LISP programs are S-expressions
  - Which represent lists
- So LISP programs can easily manipulate LISP programs
  - Just do list operations to put programs together
  - Probably the first high-level language with this feature

LISP Machines (Later Development)

- LISP is a fairly high-level language
  - Small pieces of LISP code can require a fair amount of time to execute
  - Running a LISP program could kill a timeshared machine
- Idea: design a machine specifically for LISP
  - Well-known examples are from Symbolics
    - Everything on the machine was written in LISP
  - Killed by the PC revolution

Scheme (1975)

- Dialect of LISP
- Developed buy Guy L. Steele and Gerald Jay Sussman
  - Guy Steele is heavily involved in Java
- Goal: Minimalist language
  - Much smaller than full LISP
  - First dialect of LISP to include static scoping
  - Used by people experimenting with languages

LISP Today

- LISP still used in AI programming
  - Common LISP is standard version
- Scheme still popular in academics
  - Good vehicle for teaching introductory programming
  - E.g., the TeachScheme! project

Algol (1958)

- ALGOrithmic Language
  - Designed to be a universal language
  - For scientific computations
- Never that popular, but extremely important
  - Led to Pascal, C, C++, and Java
    - "Algol-like" languages
  - Had formal grammar (Backus-Naur Form or BNF)
  - Algol 60 added block structures for scoping and conditionals
  - Imperative language with recursive functions

Example Code

```
procedure Absmax(a) Size:(n, m) Result:(y)
                    Subscripts:(i, k);
    value n, m; array a;
    integer n, m, i, k; real y;
comment The absolute greatest element of the
matrix a, of size n by m is transferred to
y, and the subscripts of this element to i
and k;
begin integer p, q;
    y := 0; i := k := 1;
    for p:=1 step 1 until n do
    for q:=1 step 1 until m do
        if abs(a[p, q]) > y then
            begin y := abs(a[p, q]);
            i := p; k := q
            end
end Absmax
```

Source: http://en.wikipedia.org/wiki/ALGOL

Algol 68

- Successor to Algol 60
  - But bloated and hard to use
  - And very hard to compile
    - E.g., variable names can include blanks!
- Included many important ideas
  - User-defined types
  - Code blocks that return the value of the last expr
  - struct and union
  - parallel processing (in the language)

Example Code

User-defined types

```
BEGIN MODE NODE = STRUCT (INT k, TREE smaller, larger),
      TREE = REF NODE;

   TREE empty tree = NIL;

   PROC add = (REF TREE root, INT k) VOID:
      IF root IS empty tree
      THEN root := HEAP NODE := (k, NIL, NIL)
      ELSE IF k < k OF root
         THEN add (smaller OF root, k)
         ELSE add (larger OF root, k)
         FI
      FI;
END
```

Identifier w/blank

Malloc

Field access

Source: http://www.xs4all.nl/~jmvdveer/algol68g-mk8/doc/examples/quicksort.a68.html

Algol Discussion

- Good points:
  - Standard for writing algorithmic pseudocode
  - First machine-independent language
  - C, C++, Java, etc. all based on it
  - Used BNF to describe language syntax
- Bad points:
  - Never widely used
  - Hard to implement
  - FORTRAN much more popular
  - Not supported by IBM

APL (early 1960s)

- A Programming Language
  - Developed by Kenneth Iverson at Harvard
  - Goal is a very concise notation for mathematics
    - Focuses on array manipulation, interactive environment
  - Was very popular
    - In 1969, 500 attendees at APL conference
    - Still some devotees
- Notable features
  - Programs evaluated right-to-left
  - Fast execution of array operations
  - Required special keyboard to enter programs

Example Code

```
(~ R ϵ R o.× R) / R←1↓ιR
```

Source: http://en.wikipedia.org/wiki/APL_programming_language

- Finds prime numbers from 1 to R

  ιR creates vector containing 1..R
  - 1↓ drops first element
  - R← assigns result to R
  - R o.× R computes (outer) product 2..R × 2..R
  - R ϵ produces vector the same size as R with 1 in position i if i is in R o.× R
    - I.e., if there is a product of two numbers equal to i
  - ~ negates the resulting vector; call it S
  - / returns a vector of items in R that have 1's in same pos in S

PL/I (1964)

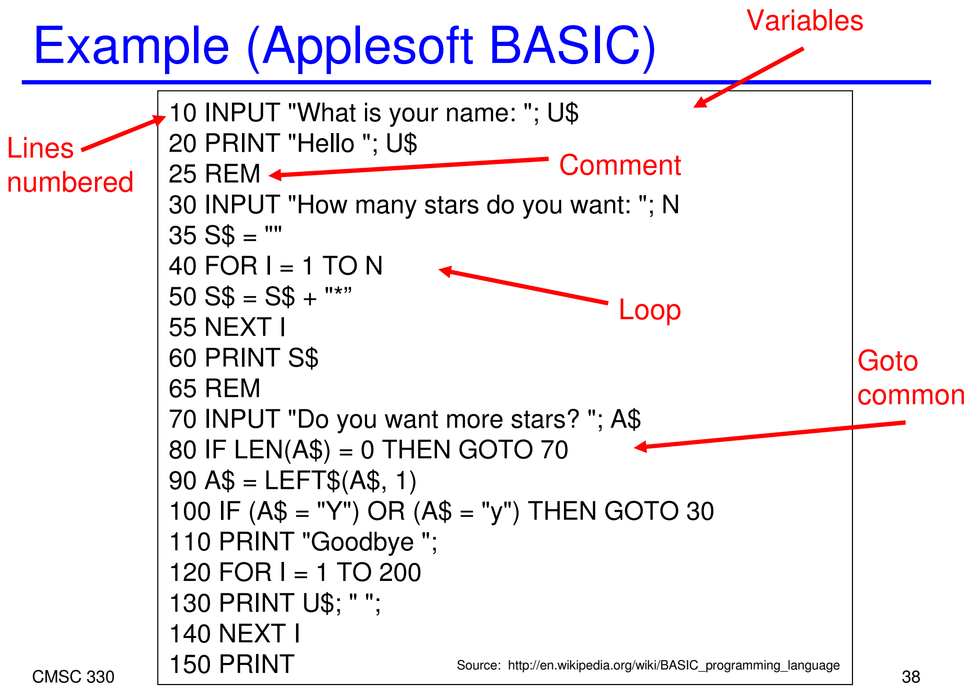
- One language to replace FORTRAN and COBOL
  - Went along with System/360, one mainframe to replace mainframes for science, business
  - Also include structured programming from ALGOL
- Famous parsing problems

  IF THEN THEN THEN = ELSE; ELSE ELSE = THEN
  - No keywords
- In the end, not that successful
  - Compilers hard to write
    - Not just because of parsing; language is complex
  - Looked down on by both camps (science, business)

BASIC (1964)

- Beginner’s All purpose Symbolic Instruction Code
  - John Kemeny and Thomas Kurtz (Dartmouth)
    - Kemeny’s PhD advised by Church
    - Kemeny also developed the first time sharing system
      - Multiple users could access shared mainframe as if they were the only user
- Goals of BASIC:
  - Easy to use for beginners
  - Stepping-stone to more powerful languages
  - *User time is more important than computer time*
  - First program run May 1, 1964 at 4:00 am

Example (Applesoft BASIC)



```
10 INPUT "What is your name: "; U$
20 PRINT "Hello "; U$
25 REM
30 INPUT "How many stars do you want: "; N
35 S$ = ""
40 FOR I = 1 TO N
50 S$ = S$ + "*”
55 NEXT I
60 PRINT S$
65 REM
70 INPUT "Do you want more stars? "; A$
80 IF LEN(A$) = 0 THEN GOTO 70
90 A$ = LEFT$(A$, 1)
100 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 30
110 PRINT "Goodbye ";
120 FOR I = 1 TO 200
130 PRINT U$; " ";
140 NEXT I
150 PRINT
```

Source: http://en.wikipedia.org/wiki/BASIC_programming_language

BASIC Today

- Still an extremely popular language
  - Microsoft Visual Basic convenient language for writing basic Windows applications
    - Lots of libraries available
    - Includes development enviroment for gui apps
- Modern dialects of BASIC are more structured
  - Eliminate most or all line numbers
  - Discourage use of “goto”

Pascal (1971)

- Developed by Niklaus Wirth
  - A response to complaints about Algol 68
- Teaching tool; not meant for wide adoption
  - Was popular for a long time
  - Best features of COBOL, FORTRAN, and ALGOL
  - (And used lowercase!)
    - Lowercase not introduced into ASCII until 1967
    - Even into 80’s some mainframes were 6-bit, no lowercase
- Pointers improved
- Case statement

Interesting Pascal Features

- Enumeration and subrange types

```
type day = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
type weekday = Mon..Fri;
type day_of_month = 1..31;
```

  - Key features:
    - Safe; values will always be within range (compare to C)
      - May require dynamic checks
- Array types with arbitrary ranges

```
var hours : array[Mon..Fri] of 0..24;
var M : array[Mon..Fri] of array[char] of real;
for day := Mon to Fri do
   M[day]['e'] := 0.0001;
```

Source: http://www.augustana.ca/~mohrj/courses/common/csc370/lecture_notes/pascal.html

Interesting Pascal Features (cont'd)

- Set types

```
var S, T : set of 1..10;
S := [1, 2, 3, 5, 7];
T := [1..6];
U := S + T;    { set union }
if 6 in S * T then ... { set intersection }
```

  - (Note comments in {}'s)

Source: http://www.augustana.ca/~mohrj/courses/common/csc370/lecture_notes/pascal.html

Pascal Today

- Kernighan's *Why Pascal is Not My Favorite Programming Language*
  - No polymorphism on types, including arrays
  - No separate compilation
  - No short-circuiting && and ||
  - Weak run-time environment
  - No escape from type system
  - and more...
- Most of these problems fixed
  - But differently in different compilers
  - Pretty much not used today

C (1972)

- Dennis Ritchie at Bell Labs
  - Ancestors B and BCPL
  - Tightly tied to Unix
- Two key features
  - Arrays and pointers closely related
    - `int *p` and `int p[]` are the same
    - Probably consequence of low-level view of memory
  - Type system lets you use values at any time
    - Type casts are necessary
    - Early compilers didn't complain about all sorts of things
      - Like assigning integers to pointers or vice-versa

Simula (1965)

- Developed at Norwegian Computing Center
  - By Ole-Johan Dahl and Kristen Nygaard
  - Goal was to simulate complex systems
  - Later used as a general purpose language
- Key features
  - Classes and objects
  - Inheritence and subclassing
  - Pointer to objects
  - Call by reference
  - Garbage collection
  - Concurrency via coroutines

Example

Parameter types

null pointer

Return value

Field access

```
class Point(x,y); real x,y;
  begin
    boolean procedure equals(p); ref(Point) p;
      if p =/= none then
        equals := abs(x - p.x) + abs(y - p.y) < 0.00001
    real procedure distance(p); ref(Point) p;
      if p == none then error else
        distance := sqrt((x - p.x)**2 + (y - p.y)**2);
end ***Point***

p :- new Point(1.0, 2.5);
q :- new Point(2.0, 3.5);
if p.distance(q) > 2 then ...
```

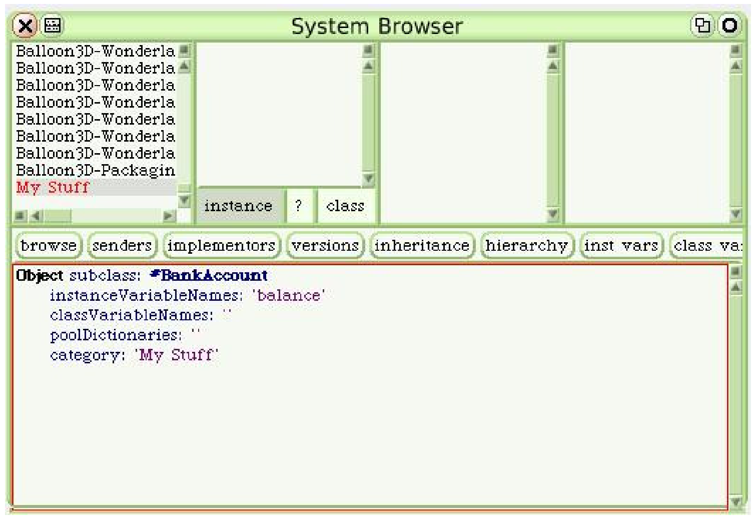
Pointer assignment

Source: http://www.stanford.edu/class/cs242/slides/2004/simula-smalltalk.pdf

Smalltalk (Early 1970s)

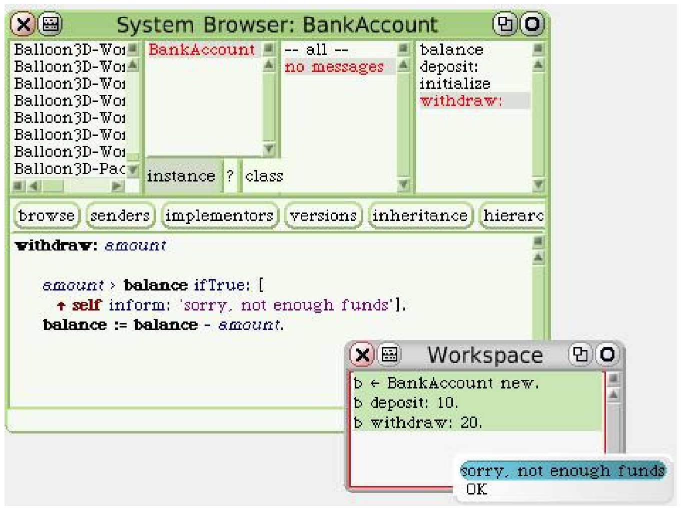
- Developed by Alan Kay et al at Xerox PARC
  - Goal: Build a small, portable computer
    - Remember, computers required separate rooms then
- Key ideas
  - Object oriented programming ideas from simula
  - Everything is an object
  - Intended for non-experts, especially children
  - Language and language editing environment integrated into operating system

Example



```
Object subclass: #BankAccount
    instanceVariableNames: 'balance'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'My Stuff'
```

Example (cont'd)



```
withdraw: amount

    amount > balance ifTrue: [
      ↑ self inform: 'sorry, not enough funds'].
    balance := balance - amount.
```

```
b ← BankAccount new.
b deposit: 10.
b withdraw: 20.
```

```
sorry, not enough funds
OK
```

C++ (1983)

- Bjarne Stroustrup, Bell Labs
- Began as "C with Classes" (~1980)
  - A preprocessor for C code
  - Added Simula-like classes
- Why use C as a base language?
  - C is *flexible* – can write any kind of program
  - C is *efficient* – can efficiently use hardware
  - C is *available* – C compilers exist for most hardware
  - C is *portable* – Porting requires some effort, but doable

[This and remaining quotes from Stroustrup, *The Design and Evolution of C++*]

Stroustrup's Language-Technical Rules

- No implicit violations of the static type system
- Provide as good support for user-defined types as for built-in types
- Locality is good
  - Abstraction and modularity
- Avoid order dependencies
- If in doubt, pick the variant of a feature that is easiest to teach
- Syntax matters (often in perverse ways)
- Preprocessor usage should be eliminated

Very Brief Comparison to Java

- C++ compiled, Java interpreted/JIT'd
  - C++ programs tend to be faster, but gap narrowing
- C++ does not guarantee safety
  - But makes an effort to be better than C
- C++ is much more complicated than Java
  - Copy constructors, assignment operator overloading, destructors, etc.
- Today, C++ is still used to build big, commercial applications
  - But Java making inroads because of big library, safety