**Part I**

**Amir Vaxman**

---

**THE EVOLUTION OF COMPUTER LANGUAGES**

- Babbage initiated programmable computers
- Invented the difference engine
  - Polynomial actions
  - Using finite differences
  - Supported programmable output formatting!

Still not a freely programmed device…

---

**Ada Lovelace (Nee Byron)**

- Born Augusta Ada King, daughter of Lord Byron
- Known for having written a description of Babbage’s early mechanical GP computer
- An adept scientific author, she devised a method to compute Bernoulli’s numbers.

1842

---

**Railroad Train Conductors**

- Herman Hollerith encoded train timetables on punched cards.
- First usage of codes to express information separately.

1889

---

**Hollerith**

- Got the idea from train conductor punch tickets
- Used to read and collate the American census of 1890
- The machine originated with Joseph-Marie Jacquard in his textile factory
- The CTR (later: IBM) census machine utilized these cards

---

**Invention of λ-calculus**

- Introduced by Church and Kleene in the (19)30’s
- A formal system – a logical language – defining the set of computable functions
- Equivalent to the Turing Machine formalism

- Examples (f is the incremental function)
  - PLUS := \( \lambda m n f x. n f (m f x) \)
  - MULT := \( \lambda m n. m \) \( \) (PLUS \( n \) 0)
Turing is considered the father of computer science.
Invented the idea of an Algorithm, and the notions of computability
Both Turing and Church proved that Hilbert's Entscheidungsproblem has a negative answer.

Can any truth be computationally acknowledged? Guess not!

Konrad Zuse Built the Z3 – the first general purpose program-stored computer.
Proved to be Turing complete in 1998.

German: "Plan Calculus"
First high-level non-Von-Neumann "programming language"
A set of notions invented by Zuse
Was never implemented.

Invented by Mauchly, and implemented for the BINAC computer.
A "Hand Compiled" Language

X3 = (X1+Y1)X1*Y1 =>
X3 03 09 X1 07 Y1 02 04 X1 =>
07Y10204X1Y1.
0000X30309X1

A machine that implements Turing’s universal Machine
The separations between CPU, CU and Memory allows for general purpose programming

Architecture dependant.
Examples:
- MIPS
- Sun SPARC
- PDP 11
**Introduction of Formal Grammars 1956**

- Noam Chomsky invented generative grammars, i.e., a method to parse and realize strings.
- He also created the Chomsky Hierarchy

```
1. S → aBSc
2. S → abc
3. Ba → aB
4. Bb → b
```

**Backus-Naur Form 1959**

- BNF is a metasyntax used to describe formal languages.
- Was developed to express the context-free grammars.
- Was used to create ALGOL.
- Example (U.S. postal address):
  `<postal-address> ::= <name-part> <street-address> <zip-part>
  <name-part> ::= <personal-part> <last-name> <opt-jr-part>
  <EOL> | <personal-part> <name-part> <EOL> <personal-part>
  ::= <first-name> | <initial> ""
  <street-address> ::= <opt-apt-num> <house-num> <street-name>
  <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>
```

**3rd Generation – The Fore-Fathers**

- A few languages set the basic template for known languages today.
  - Fortran
  - ALGOL
  - COBOL
  - Lisp
  - Simula

**1956 – Fortran**

- FORmula TRANslator.
- Invented by IBM team lead by John W. Backus.
- Not the first high-level programming language, nor the first compiled language, but the first successful one.
- Fathered GOTO languages (such as BASIC)

**ALGOL 1958**

- ALGOrithmic Language
- Designed specifically for scientific calculations.
- Structure similar to C
- Like Fortran – machine independent

**COBOL 1959**

- CCommon Business-Oriented Language.
- Still in active use! Mostly for banking and (old) DB accounting.
- Did not support local variables, recursion, dynamic memory allocation, or structured programming constructs.
LISP 1959
- LISt Processing
- A family of functional programming languages.
- Built as a model to utilize Church's \(\lambda\)-calculus.
- Fundamental operations CAR (head of list) and CDR (tail of list) originate in the structure of the IBM 704
- Examples:
  - \((\text{lambda} \ (\text{arg}) \ (+ \ \text{arg} \ 1)) \ 5\) – Evaluates to 6
  - \((\text{append} \ (1 \ 2) \ (3 \ 4))\) - Output: \(1 \ 2 \ 3 \ 4\)

Simula 1967
- The first language to utilize the ideas of Object-Oriented Programming, introducing objects, classes, garbage collection and so forth.

Offsprings of the Fore-Fathers 1964
- Some languages utilized the ideas of these fundamental languages.
- BASIC: A tutorial language which became popular in PCs (and in ATARIs) because of its simplicity.
- inserting lines in the middle was a headache.
- Not all implementations supported GOSUB recursion.

APL 1962
- An Array Programming language.
- May be considered the father of MATLAB.
- Used incomprehensible characters.
- The following example assigns the set of primes in the range \([1, R]\) to the array PRIMES!
  \[
  \text{PRIMES : } (\sim R \in R^\circ \times R) / R \leftarrow 1 \downarrow R
  \]

PL/1 1964
- Was widely used as a tutorial language for 1st semester students.
- Totally free-form – No reserved keywords.

Approaches to programming
- With the development of languages, several principles guided the needs:
  - Application vs. system programming
  - Imperative languages (OOP or non-OOP), functional languages, logic.
  - Semantics: scopes, recursion, encapsulation, modularity, platform-independence
  - Memory abstraction
  - Type checking
  - Compiled vs. interpreted.
A basic part of Turing-completeness (can be realized with loops)
Had to submit to the invention of the function call stack.
Early languages, like FORTRAN on the HP 210 computer did not support recursion.
There are many variants, some are newer.
Call-by-Value – most common, used in C and scheme, for example. The expression is evaluated and then passed as argument.
Call-by-reference – using “L-Value”. Used by most modern languages, and some middle ones (e.g. ML, and to some extent FORTRAN).
The same syntax may cause different effect in different languages (and sometimes, in the same language with a different compiler!)
Commonly-affected areas:
- Order of evaluation in an IF statement
- Default initialization of variable.
- Variable casting (example in next slide).
Example #1:
(String) x
Semantics:
- C++: Blind Cast (always succeeds)
- Java: Checked cast (may fail at run-time)
"Ignoring inessential details"
A computer is a complex machine
Many views of computers
- (of various level of abstraction)
A programming language: A simple yet comprehensive view
A mapping from integers onto \{ 0, 1 \}
- This is a bit-by-bit view
The programming model:
- Read a bit (from a location in memory)
  - Optionally change its value (0 \leftrightarrow 1)
- Write a bit (to a location in memory)
Very difficult for most tasks
The Imperative Model
- A computation - Reading/Writing values from/into cells
- A Turing-machine based approach
- Additional abstraction evolved
  - Stack, types, ...
- Example: Computing factorial numbers
  For n=1 to k
  temp = n
  x = x * temp
  End for

The Functional Model
- A computation – Evaluating a formula
  - Almost like in the 7-th grade...
  - ... but with recursion
- A Lambda-calculus based approach
- Example: computing next factorial number
  \[ f(n) = nf(n-1) \]

The Logical Model
- A computation – Finding all tuples which satisfy certain conditions
- A Relational-algebra based approach
  - ... but with recursion
- Example: Computing next factorial number
  \[(x, n) \in F \Rightarrow (x \cdot (n + 1), n + 1) \in F\]

Paradigms
- Functional
- Logical
- Imperative
  - Object-oriented

HoPL: The History Of Programming Languages
- Part 2

Itay Maman
298801 Seminar lecture – 9 Jul 2007

The Stormy 70's
- Some of the languages from this decade are still live and kicking
- E.g.: ML, C, SQL, Smalltalk
- Older languages are rarely used today
### 1970 - Pascal
- **Designer:** Niklaus Wirth
- **Paradigm:** Imperative
- **Type system:** static, strong
- **Based on:** Algol
- **Intended to be used as a teaching language**
- **First Pascal compiler was written in Fortran**
- **Subsequent versions were bootstrapped**

#### Fibonacci: Pascal

```pascal
function fib(N : integer) : longint;
var
  tmp, first, second : integer;
begin
  while n <> 0 do
    begin
      n := n - 1;
      tmp = first + second;
      first = second;
      second = tmp;
    end
  fib := first;
end
```

### 1972 - C
- **Designer:** Dennis Ritchie
- **Paradigm:** Imperative
- **Type system:** static, weak
- **Based on:** ALGOL, Assembly
- **Designed for system-programming tasks**
  - Till 1973: Unix kernel written in assembly (PDP-7, 11)
- **Philosophy:**
  - Simple language => simple compiler
  - Many language constructs are based on machine instructions
  - Fine-grain control over memory, I/O
  - Minimal run-time support

#### Standardization Process of C
- **1973 - First robust implementation**
- **1978 - “The C Programming Language” (AKA: K&R)**
  - The de-facto standard
  - No type checking of parameters
  - Old-style:
    ```c
    int main(argc, argv) { ... }
    ```
- **1983 - C’s ANSI standard committee formed**
- **1988 - 2nd edition published**
- **1989 - ANSI standard ratified (AKA: C89)**
  - Function prototypes
  - Void pointers/functions, enum types
- **1990 - ANSI C adopted by ISO**
  - AKA: C90 (== C89)

### 1972 - Smalltalk
- **Designers:** Alan Kay, Dan Ingalls, Adele Goldberg
- **Paradigm:** Object-oriented
- **Type system:** dynamic, strong
- **Based on:** Simula, Lisp
- **Philosophy:**
  - Everything is an object
  - Program is a data structure in memory
    - The program can examine/modify itself
  - Three primitive constructs:
    - send a message to a receiver
    - return a value
    - assign a value to a variable

#### Smalltalk: The Cookie Monster Class
- **Class:** CookieMonster
- **Superclass:** Monster
- **Category:** Sesame Street
- **Instance variables:** hunger
- **Methods:**
  ```smalltalk
  nag
  | item |
  item := self askForCookie.
  (item isKindOf: Cookie)
  ifTrue: [self eat: item]
  ifFalse: [self complainAbout: item].
  ```
**1972: Prolog**
- Designers: Alain Colmerauer
- Paradigm: Logic programming
- Type system: varies
- Designed for AI tasks (searching in trees/graphs)
- Philosophy:
  - A rule defines an implication (right implies left)
  - Predicate – A set of rules with the same left-hand side term
  - Predicate invocation:
    - Returns all tuples matching the value passed-in

**The Whitehouse Tenants**
- `pred('Washington', 'Adams').`
- `pred('Adams', 'Jefferson').`
- `pred('Jefferson', 'Madison').`
- `pred('Madison', 'Monroe').`
- `pred('Monroe', 'Adams').`
- `before(X,Z) :- pred(X,Z).`
- `before(X,Z) :- pred(X,Y), before(Y,Z).`
- `before(A,'Madison')? < 'Jefferson' < 'Adams' < 'Washington'.`

**1973: ML**
- Designer: Robin Milner
- Paradigm: Functional
- Type system: static, strong, inferred
- A functional language w/ static type checking
  - Lisp is functional but dynamically typed
  - Not purely functional => Allows side effects

```
fun step x y 0 = x
    | step x y n = step y (x+y) (n-1);
val fib = step 0 1;
```

**The Energetic 80’s**
- Large scale programs are here
- Language features for coping with the complexity of the code

**1983: Ada**
- Designer: Jean Ichbiah, S. Tucker Taft
- Paradigm: Imperative
- Type system: static, strong
- Based on: Algol 68, Pascal
- Developed for the US Department of Defence
- Requirements
  - Modular, safe, high level of abstraction
  - High maintainability
  - Should replace ad-hoc hardware-dependent languages
  - Frequent in embedded systems

**Fibonacci: Ada**
```
function fib(n : integer) return integer is
  f : integer := 0;
  s i integer := 1;
  tmp : integer;
begin
  for i in 1..n loop
    tmp := f + s;
    f := s;
    s := tmp;
  end loop;
  return f;
end fib;
```
Ada’s Story

- Designed using the “waterfall” approach
  - The same design procedure as in weapon systems
- 1975- Working group for a new computer language formed
- 1977- Ideal language specification, "Ironman", published
- 1979- The Green proposal is chosen
- 1980- Ada’s reference manual approved
- 1981- First implementation validated
  - Ada becomes an ANSI standard
- 1982- DoD adopts the Ada mandate
  - Requires the use of Ada in projects with > 30% new code
- 1983- DoD adopts the COTS policy
  - Commercial Off-The-Shelf technologies

C++

- Designer: Bjarne Stroustrup
- Paradigm: Object-oriented
- Type system: static, weak
- Based on: C, Simula
- Philosophy:
  - C’s performance
  - Simula’s features
- Three major stages: 1985, 1989, 1998 (First standard)
  - Next version: C++0x (...Hopefully x <= 9)
- Drawbacks: C, poor standard library, no GC

Eiffel

- Designer: Bertrand Meyer
- Paradigm: Object-oriented
- Type system: static, strong
- Based on: Simula, Ada
- Software Engineering oriented
  - Design by Contract
  - Automatic documentation

Fibonacci: Eiffel

```eiffel
fib (n: INTEGER): INTEGER is
  require
    -- Design By Contract:
    pre_fib: n > 0    -- Precondition
  local
    i, tmp, f, s: INTEGER
  do
    from
    f := 0; s := 1; i := 1;
    until
    i = n;
  loop
    tmp := f + s; f := s; s := tmp;
    i = i + 1;
  end;
  Result := s;
  end;
```

The Neurotic 90’s

- Performance is less of a problem
- Hybrid software systems
  - Interoperability
  - Rapid prototyping
- The Internet
  - Web servers
  - Browser-side programs
  - Accessibility to open source libraries
**Haskell**
- Designer: Simon Peyton-Jones, Paul Hudak, Philip Wadler
- Paradigm: Functional
- Type system: static, strong, inferred
- Based on: ML, Miranda
- Philosophy: Purely functional
  - No side effects (unless you ask nicely)

```
module Main where
import System.Environment
fib = 1 : 1 : zipWith (+) fib (tail fib)
main = do
  args <- getArgs
  print (fib !! (read(args!!0) - 1))
```

**Java**
- Designer: James Gosling
- Paradigm: Object-oriented
- Type system: static, strong
- Based on: C++, Smalltalk, Eiffel
- Philosophy
  - Compile once run anywhere
  - Safe yet dynamic (applets)
  - => Internet ready!
- As of 2006, mostly open sourced (!)

**Java - OOP to the Masses**
Is Java's popularity due to its OO nature?
- Powerful runtime system
  - JVM, JIT, GC
- A huge standard library
  - Networking, JDBC, GUI, Threading, Cryptography, ...
- Enterprise Applications
  - J2EE, hotswapping
- Unicode

**Fibonacci: Haskell**
```
module Main where
import System.Environment
fib = 1 : 1 : zipWith (+) fib (tail fib)
main = do
  args <- getArgs
  print (fib !! (read(args!!0) - 1))
```

- 2D grammar
- Lazy evaluation: fib is recursive but has no if's
- do construct for sequential execution
- Type inference

**Interpreters: The Resurrection**
Interpreters: The Resurrection
Coming soon to an IDE near you

**1987- Perl**
- Designer: Yukihiro Matsumoto
- Paradigm: Object-oriented
- Type system: dynamic, strong
- Based on: Smalltalk, Lisp, Python
- Philosophy: "programmer productivity and fun"
- Highlights
  - Implicit declaration of setter, getters
  - Mixings, open classes/objects
  - RoR: Ruby-on-Rails
    - Framework for DB-based web application

**1990- Python**

**1991- Visual Basic**

**1995- Ruby**
- Designer: Rasmus Lerdorf
- Paradigm: Imperative
- Type system: dynamic, strong
- Based on: C, Perl
- Highlights
  - Server side scripting
  - HTML text embedded within the source code
  - PHP = HTP: Hypertext Preprocessor

```ruby
class Someone
  def initialize(name)
    @name = name
  end
end

b = Someone.new("Bart")
puts b.name # Output: 'Bart'
```

**1995- Javascript**

**1997- PHP**
- Designer: Rasmus Lerdorf
- Paradigm: Imperative
- Type system: dynamic, strong
- Based on: C, Perl
- Highlights
  - Server side scripting
  - HTML text embedded within the source code
  - PHP = HTP: Hypertext Preprocessor

```php
<?php
  echo 'Side Scripting';
?>
```
1996 - Squeak

2002 - Visual Studio .NET

2007 - Eclipse 3.2.2

<conclusions>

Main Battlefields - Objective

- Productivity
- Performance

Seems to be decided in favor of productivity

Main Battlefields - Execution Mode

- Machine code
- Virtual machine
- Interpreter

Strongly related to the question of portability...
Levels of Portability

1) Machine specific languages: Assembly
   - Almost no portability

2) Compiler + O/S isolate from hardware: Algol, C
   - Source compatibility on same O/S
   - Binary compatibility on same O/S & CPU

3) Comprehensive standard libraries: Eiffel
   - Source compatibility on all platforms

4) Virtual machine: Java, Smalltalk
   - Full source + binary compatibility

Main Battlefields – Type Checking

- Static
- Dynamic

Currently, the hottest debate
- Manifested by the “performance is not an issue” claim

Main Battlefields – Paradigm

- Functional
- Imperative

A 71-years old debate

So, Which Language Is Better?

- No absolute answer
- Too many contradicting factors
- Too many players (languages)
  - Very easy to produce a new language
  - Extremely difficult to measure the market
- Still, here are a few attempts...

Popularity: Open Source Projects

Source: http://www.cs.berkeley.edu/~flab/languages.html

Popularity: O’Reilly Book Sells

Software is cheaper the Hardware (to a certain degree)

A researcher can easily develop new language constructs

=> Wealth of languages
  - Compared to hardware technologies
  - Each making one small step forward

=> Languages evolve

It is hard to detect the major trends

References

http://www.cs.fit.edu/~ryan/ada/ada-hist.html
http://www.tiobe.com/index.htm?tiobe_index
http://thomer.com/vi/vi.html
http://www.squeak.org/Smalltalk/
http://acs.ucsd.edu/info/dbx.debug.php
http://www.research.att.com/~bs/homepage.html
http://pascalprogramming.byethost15.com
http://www.levenez.com/lang/history.html
http://en.wikipedia.org
http://www.sppn.nl/logo.html

http://www.scriptol.org/fibonacci-any-programming-language.html
http://www.whitehouse.gov/history/presidents/chronological.html
http://www.dcs.ed.ac.uk/home/stg/NOTES/node59.html
There are only two kinds of languages:
the ones people complain about
and the ones nobody uses

BS