Back to the future – the history of programming languages

Robert Stroud

Adelard LLP

rjs@adelard.com

Abstract

- What can we learn from the history of programming languages?
- Are we still seeing technical innovation after 50-60 years of evolution or is there nothing new under the sun?
- What can the past tell us about the future?
- In this talk, I will review some of the key arguments and debates that have informed the development of programming language and speculate about current and future trends.

Overview of talk

- My first programming language
- A brief history of programming languages
- Technical debates
- What does the future hold?

“Real Programmers don't eat quiche”

Anon, c.1984

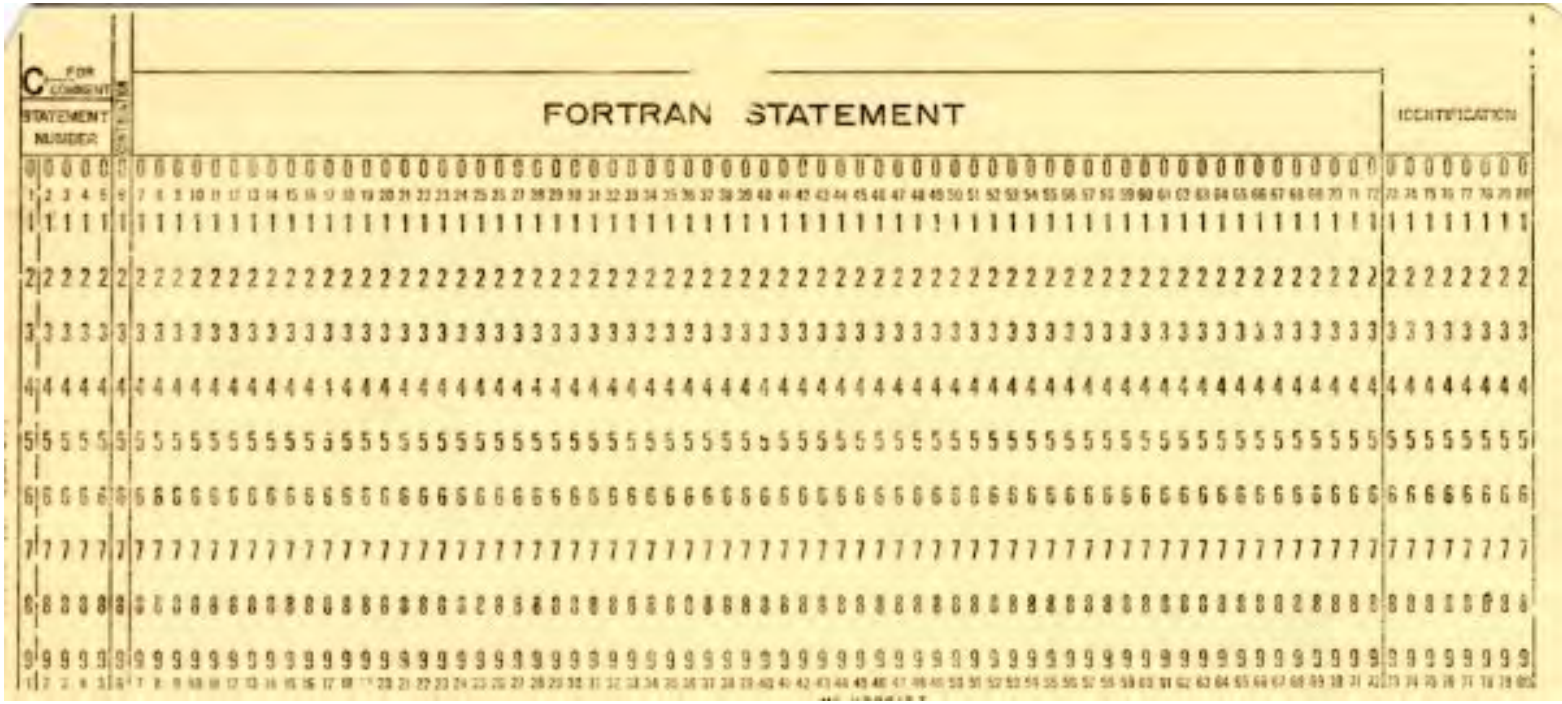
MY FIRST PROGRAMMING LANGUAGE

Real Programmers

- Real Programmers don’t write application programs - they program right down on the bare metal. Applications programming is for the dullards who can’t do system programming.
- Real Programmers don’t document. Documentation is for simpletons who can’t read listings or the object code from the dump.
- Real Programmers don't write in PASCAL, or BLISS, or ADA, or any of those pinko computer science languages. Strong typing is for people with weak memories.

My first programming language (1974)

- FORTRAN, written on punched cards, running on an IBM 1130 in the sub-basement of Claremont Tower



My first computer (1974)

- IBM 1130, with up to 32K memory (3.6µs), 1MB disk.
- Fortran compiler ran in 8K
- Card reader could read up to 400 cards/minute
- Printer ran at 120 characters, 80 lines/minute

Fortran example - solving quadratic equations

```
      SUBROUTINE QUADR(A,B,C,DISCR,X1,X2)
      REAL A,B,C,DISCR,X1,X2,VX,VY,FL,FPY

C DISCRIMINANT
      DISCR = B**2.0 - 4.0*A*C

C COMPUTE THE ROOTS BASED ON THE DISCRIMINANT
      IF(DISCR) 110,120,130

C -VE DISCRIMINANT, TWO COMPLEX ROOTS, REAL=X1, IMG=+/-X2
110   X1 = -B / (2.0*A)
      X2 = SQRT(-DISCR) / (2.0*A)
      RETURN

C ZERO DISCRIMINANT, ONE REAL ROOT
120   X1 = -B / (2.0*A)
      X2 = X1
      RETURN

C +VE DISCRIMINANT, TWO REAL ROOTS
130   X1 = (-B + SQRT(DISCR)) / (2.0*A)
      X2 = (-B - SQRT(DISCR)) / (2.0*A)
      RETURN
```

Example - continued

```
      DATA ICARD,IPRT /2,3/
      REAL A,B,C,DISCR,XR1,XR2
C READ A B C, IF A=0 THEN EXIT
100   READ(ICARD,801)A,B,C
801   FORMAT(3F8.3)
C     EXIT WHEN A IS ZERO
      IF (A) 110,9000,110
C PRINT A B C
110   WRITE(IPRT,902)A,B,C
902   FORMAT(' QUADRATIC A=',F8.3,' B=',F8.3,' C=',F8.3)
C COMPUTE AND PRINT THE CRITICAL VALUES
      CALL QUADR(A,B,C,DISCR,XR1,XR2,VX,VY,FL,FPY)
      WRITE(IPRT,903) DISCR
903   FORMAT(' DISCRIMINANT=',F9.4)

      IF (DISCR) 120,130,140
C SEE NEXT SLIDE…
```

Example - continued

```
C -VE DISCRIMINANT, TWO COMPLEX ROOTS
120   WRITE(IPRT,913) XR1, XR2
913   FORMAT(' COMPLEX ROOTS =(',F9.4,'  +/-',F9.4,'I)')
      GO TO 200

C ZERO DISCRIMINANT, ONE REAL ROOT
130   WRITE(IPRT,912) XR1
912   FORMAT(' ROOT  X =',F9.4)
      GO TO 200

C +VE DISCRIMINANT, TWO REAL ROOTS
140   WRITE(IPRT,911) XR1, XR2
911   FORMAT(' ROOTS X1=',F9.4,'   X2=',F9.4)
      GO TO 200

C END OF QUAD
200   WRITE(IPRT,901)
      GO TO 100

C END OF PROGRAM
9000  CALL EXIT
      END
```

“Those who cannot remember the past are condemned to repeat it”

George Santayana, “The Life of Reason”, 1905

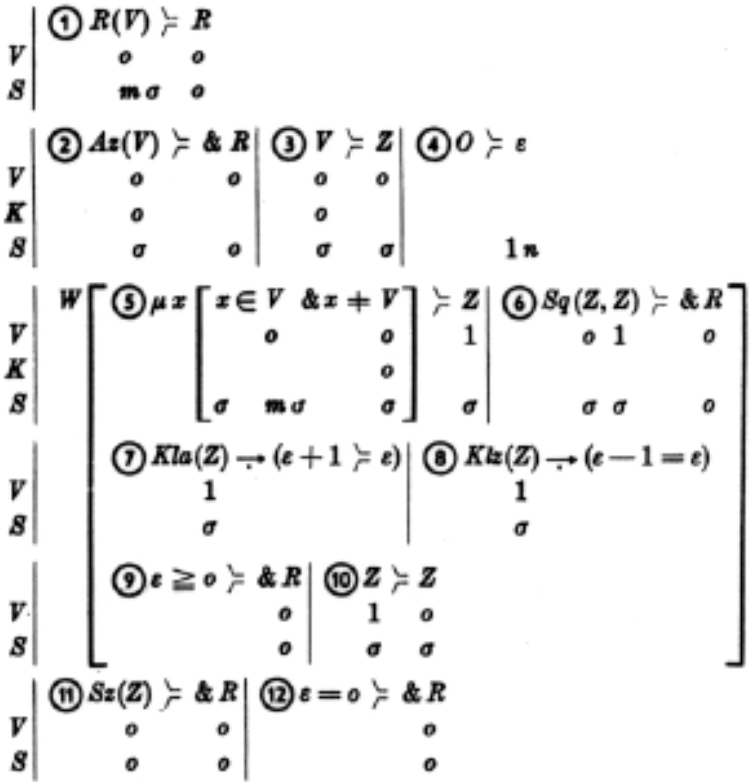
A BRIEF HISTORY OF PROGRAMMING LANGUAGES

1940-1950

- First programmable computers became operational
- Konrad Zuse invented the first high level programming language (Plankalkül) in 1945
- However, his work was not published until 1972, and not implemented until 2000!
- Meanwhile, all programming was done in binary machine code

Plankalkül – syntax checking boolean expressions

Fig. 3.



1950-1955

- Idea of generators - programs to write programs
- Codes for abstract machines
- Translation of algebraic expressions
- Initial proposals for high-level programming languages

1955-1960

- First compilers for high level programming languages become available
- Division into scientific computing and business computing
- FORTRAN for numerical scientific programming
- COBOL for business users
- ALGOL for describing algorithms
- LISP for symbolic computing and AI research
- APL as a mathematical notation

FORTRAN

A high-level language for scientific programs (introduced 1954)

```
C      FORTRAN PROGRAM FOR
PRINTING A TABLE OF CUBES
       DO 5  I = 1, 64
       ICUBE = I * I * I
       PRINT 2, I, ICUBE
    2  FORMAT (1H , I3,I7)
    5  CONTINUE
       STOP
```

John Backus, IBM (b1924)

Image courtesy of Columbia University

COBOL

A business data processing language (1959) strongly backed as a standard by the US Department of Defense

```
MAIN.
    PERFORM INPUT-ADD 10 TIMES.
    DIVIDE 10 INTO SUM-VALUE
GIVING AVERAGE-VALUE.
    MOVE AVERAGE-VALUE TO EDIT-FIELD.
    DISPLAY "THE AVERAGE IS ", EDIT-FIELD.
    STOP RUN.
INPUT-ADD.
    DISPLAY "TYPE IN AN INTEGER (2  DIGITS)".
    ACCEPT INPUT-VALUE FROM SYSIN.
    ADD INPUT-VALUE TO SUM-VALUE.
```

Rear Admiral Grace Hopper (1906-1992)

Images courtesy of US Navy and Computer History Museum

Edsger Dijkstra

- “The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.”

- “FORTRAN, "the infantile disorder", by now nearly 20 years old, is hopelessly inadequate for whatever computer application you have in mind today: it is now too clumsy, too risky, and too expensive to use.”

Edsger Dijkstra,
“How do we tell truths that might hurt?”, 1968

Cobol today

- In the late 1990s, the *Gartner Group* estimated that of the 300 billion lines of computer code that existed, eighty percent – or 240 billion lines – were COBOL.
- They also reported that more than half of all new mission-critical applications were still being created using COBOL – an estimated 5,000,000,000 net new lines of COBOL code annually.

Fortran today

- Fortran remains the language of choice for high performance numerical computing
- The language has been tuned for scientific and numeric work, and can be run very efficiently on parallel computers
- Thus, Fortran is the primary language for some of the most intensive super-computing tasks, including weather and climate modeling
- Since Fortran has been around for nearly sixty years, there is a vast body of Fortran in daily use throughout the scientific community

Fortran 2000

- “I don’t know what programming language people will be using in the year 2000, but I know it will be called Fortran”
  - (Anon, circa 1980?)

- “As I said in my comments to the committee, [Fortran 90' would be a] nice language, too bad it's not Fortran”

ALGOL

highly-influential precursor to Pascal, Simula, etc.

```
begin
      integer i, cube;
      for i := 1 step 1 until 64 do
      begin
            cube := i ** 3;
            print (i, cube);
      end;
end;
```

Peter Naur, editor of the Algol Report (1960)

Image courtesy of Computer History Museum

Algol 60

- “The more I ponder the principles of language design, and the techniques which put them into practice, the more is my amazement and admiration of ALGOL 60.
- Here is 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.”

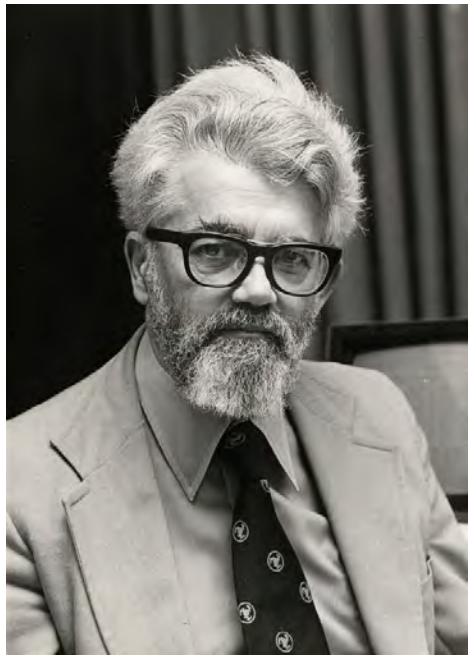
Tony Hoare,
“Hints on Programming Language Design”, December 1973

LISP

List processing language based on λ-calculus,
still widely used for Artificial Intelligence programming

```
(define (cubes N)
  (cond
    ((= N 0) nil)
    (else
      (append (list N (* N N N))
              (cubes (- N 1))))))

(cubes 64)
```



John McCarthy,
inventor of Lisp and originator of the term
"Artificial Intelligence"

Cubes in Haskell (Lisp style)

LISP

```
(define (cubes N)
   (cond
      ((= N 0) nil)
      (else
         (append (cubes (- N 1)
                 (list N (* N N N)))))))
```

Haskell

```
cubes :: Int -> [(Int,Int)]
cubes n
   | n == 0        = []
   | n > 0         = cubes(n-1) ++ [(n, n^3)]
```

APL

array programming language that used its own distinctive character set based on mathematical notation

Program to print cubes:
Φ2,64⍴⍳64,(⍳64)*3

Program to print primes:
(~R∈R∘.×R)/R←1↓⍳R

Ken Iverson, inventor of
"A Programming Language (APL)"

Cubes in Haskell (APL style)

APL

```
      L ← cubes N
[1]   L ← Φ2,NριN,(ιN)*3
```

Haskell

```
cubes :: Int -> [(Int,Int)]
cubes n = zip [1..n] (map (^3) [1..n])
```

Edsger Dijkstra

- “APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.”
  “How do we tell truths that might hurt?”, 1968

- “Lisp has jokingly been called "the most intelligent way to misuse a computer". I think that description is a great compliment because it transmits the full flavor of liberation: it has assisted a number of our most gifted fellow humans in thinking previously impossible thoughts.”
  “The humble programmer”, 1972

Edsger Dijkstra,

1960-1970

- Development of ALGOL-like languages:
  - Algol 60, Simula, Algol/W, Algol 68
- Birth of structured programming:
  - “Goto statement considered harmful”
- Arguments about successor to Algol 60 led to a split in the program language community
- Attempt by IBM to define a universal language PL/I, for both business and scientific users
- Object-oriented programming invented by designers of Simula 67

Simula’67

- The Simula programming language was designed in Norway by Kristen Nygaard and Ole-Johan Dahl
- Simula was an extension to Algol that was intended as a simulation language
- As a side-effect of thinking about how to model real-world entities, the designers of Simula invented object-oriented programming!
- However, this wasn’t widely recognised until much later…

Simula class

- A Simula class is a generalisation of an Algol block structure
- A class can contain declarations and code
- However, unlike a block, a class continues to exist after the code has been executed
- Instances of classes are created dynamically and managed by references
- Other features of Simula classes:
  - Class prefixes for inheritance
  - Virtual methods and dynamic binding
  - Support for concurrent execution

Stack class

```
class Stack;
begin
  integer top;
  integer array contents(0:9);

  procedure push(i); integer i;
  begin
    contents(top) := i;
    top := top + 1;
  end;

  integer procedure pop;
  begin
    top := top - 1;
    pop := contents(top);
  end;
end
```

Program using Stack

```
begin
  external class Stack;

  ref(Stack) stack :- new Stack;
  stack.push(100);
  outtext("Result = ");
  outint(stack.pop);
  outimage;
end
```

1970-1980

- Pascal language embodies current wisdom about structured programming and data types:
  - “Algorithms + Data Structures = Programs”
- But Pascal didn’t address all the issues:
  - Programming in the large - modules
  - Abstract data types (object-based programming)
  - Concurrent programming
- Systems programming - writing operating systems in high level languages
- Development of Unix and C at Bell Labs
- Work at Xerox PARC on Smalltalk, Mesa

Pascal-like languages

- Use `begin … end` for blocks
- Use `:=` for assignment
- Use "`var : type`" for declarations
- Use "`procedure fun() : type`" to declare functions and procedures
- Arrays can have arbitrary dimensions
- Procedure declarations can be nested
- Examples:
  - Pascal, Module-2, Ada, Oberon

C-like languages

- Use `{ … }` for blocks
- Use = for assignment
- Use "`Type var`" to declare variables
- Use "`Type func()`" to declare functions and methods
- Arrays are indexed from zero
- Nested functions/methods are not allowed
- Examples:
  - C, C++, Java, C#

1980-1990

- Pascal widely adopted as introductory language for teaching programming
- Unix and C became widely known outside AT&T, and popularise the idea of scripting languages
- US DoD standardise on Ada as a language for developing embedded systems
- BYTE magazine publishes a special edition devoted to object-oriented programming and Smalltalk-80
- C++ becomes popular as an object-oriented extension to C

1990-2000

- Growth of the Internet and World Wide Web
- Unix-based scripting languages such as perl used to write web applications
- C++ widely criticised for its complexity and error-proneness
- Java developed as a safe alternative to C++ for object-oriented programming
- Growing interest in dynamically typed languages for rapid prototyping and software development
- Development of component-based programming models, and embedded scripting languages

2000-2005

- Microsoft develop their own version of Java called C# as part of the .Net platform
- Java continues to evolve, partly in response to competition from C#
- Sophisticated programming development environments are widely used
- Dynamically typed languages are increasingly used, particularly for web development
- Everyone wants closures…

“The language designer should be familiar with many alternative features designed by others, and should have excellent judgment in choosing the best, and rejecting any that are mutually inconsistent.”

C.A.R. Hoare, “Hints on Programming Language Design"

TECHNICAL DEBATES

Some technical debates

- Blocks, modules, classes, closures
- Strong typing vs. weak / dynamic typing
- Structured programming
- Reference vs. value semantics
- Exception handling
- Multiple inheritance
- Concurrency
- Event handling vs. delegates
- Reflection, metaobject protocols, aspects
- Syntactic sugar, extensibility, expressive power

“Lambda: The Ultimate …”

Guy Steele, 1976-1979

CLOSURES

History of Closures

1936 - Alan Turing invents every programming language that will ever be but is shanghaied by British Intelligence to be 007 before he can patent them.

1936 - Alonzo Church also invents every language that will ever be but does it better. His lambda calculus is ignored because it is insufficiently C-like. This criticism occurs in spite of the fact that C has not yet been invented.

James Iry, “A brief, incomplete and mostly wrong history of programming languages”

History of closures (cont.)

1958 - John McCarthy and Paul Graham invent LISP. Due to high costs caused by a post-war depletion of the strategic parentheses reserve LISP never becomes popular. In spite of its lack of popularity, LISP (now "Lisp" or sometimes "Arc") remains an influential language in "key algorithmic techniques such as recursion and condescension".

> James Iry, “A brief, incomplete and mostly wrong history of programming languages”

History of closures (cont.)

1970 - Guy Steele and Gerald Sussman create Scheme. Their work leads to a series of "Lambda the Ultimate" papers culminating in "Lambda the Ultimate Kitchen Utensil." This paper becomes the basis for a long running, but ultimately unsuccessful run of late night infomercials. Lambdas are relegated to relative obscurity until Java makes them popular by not having them.

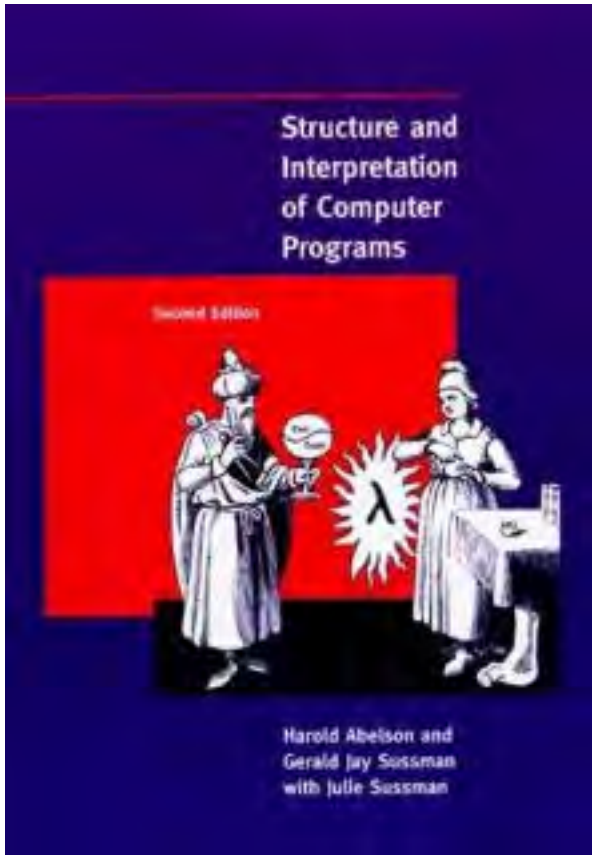
> James Iry, “A brief, incomplete and mostly wrong history of programming languages”

Structure and Interpretation of Computer Programs

"SICP has had a dramatic impact on CS curricula over the last decade […]

The book emphasizes the central role played by different approaches to dealing with time in computational models: objects with state, concurrent programming, functional programming and lazy evaluation, and nondeterministic programming."

http://mitpress.mit.edu/sicp/



Closures in Algol 60

- By default, Algol 60 used call by name for parameter passing, which was implemented using a ‘thunk’ (aka closure)
- Call by name parameters are re-evaluated in the caller's referencing environment every time the parameter is used.
- The effect is as if the called routine had been textually expanded at the point of call
- This allows a technique called Jensen’s device, which can be used to perform summation or integration of an expression over a range

Example

To evaluate the sum

$$y = \sum_{1 \le x \le 10} 3x^2 - 5x + 2$$

we can simply say

```
y := sum(3*x*x - 5*x + 2, x, 1, 10);
```

Jensen's device

```
real procedure sum (expr, i, low, high);
    value low, high;
        comment low and high are passed by value;
        comment expr and i are passed by name;
    real expr;
    integer i, low, high;
begin
    real rtn;
    rtn := 0;
    for i := low step 1 until high do
        rtn := rtn + expr;
        comment the value of expr depends on the value of i
    sum := rtn
end sum
```

Closures in Java

"Java 8 is expected in March 2014 and will include features that were planned for Java 7 but later deferred

> JSR 335: Language-level support for lambda expressions (officially, lambda expressions; unofficially, closures) under Project Lambda.

There was an ongoing debate in the Java community on whether to add support for lambda expressions, Sun later declared that lambda expressions would be included in Java and asked for community input to refine the feature."

http://en.wikipedia.org/wiki/Java_version_history

“Algol 60 was an improvement on almost all of its successors”

C.A.R. Hoare

ALGOL WARS

Evolution of Algol

“ALGOL / W was not only a worthy successor of ALGOL 60, it was even a worthy predecessor of PASCAL […]

I was astonished when the ALGOL working group, consisting of all the best known international experts of programming languages, resolved to lay aside the commissioned draft on which we had all been working and swallow a line with such an unattractive bait [ALGOL 68].

The best we could do was to send with it a minority report, stating our considered view that, "… as a tool for the reliable creation of sophisticated programs, the language was a failure.”

C.A.Hoare, “The Emporer’s New Clothes”

Algol 68 Minority Report

“We regard the current Report on Algorithmic Language ALGOL 68 as the fruit of an effort to apply a methodology for language definition to a newly designed programming language.

We regard the effort as an experiment and professional honesty compels us to state that in our considered opinion we judge the experiment to be a failure in both respects.”

Signed by: DIJKSTRA, DUNCAN, GARWICK, HOARE, RANDELL, SEEGMUELLER, TURSKI, WOODGER

The authors of the Algol 68 report beg to disagree…

"The writing of the Report was not only Work, it was also Fun, as should be apparent to all readers. […]

Just reading aloud certain lines of the syntax, slightly raising the voice for capitalized words, conveys a feeling of heroic and pagan fun.

```
REFETY ROWSETY ROWWSETY NONROW slice{860a} :
  weak REFETY ROWS ROWWSETY NONROW primary{81d},
  sub symbol{31e},
    ROWS leaving ROWSETY indexer{b,c,d,e},
    bus symbol{31e}.
```

Such lines can not be read or written with a straight face."

C.H.A. Koster, "The making of Algol 68"

Closures in Algol 68?

"It is striking to note the great concern for micro efficiency, which has in many respects hampered the development of Algol 68. […]

A discussion on procedures delivering procedures (in some cases limited by scope problems) did not lead to the obvious conclusion to do away totally with the (statically unenforceable) scope restrictions

Algol 68, which has higher-order functions, narrowly missed having Currying, which would have made it possess a complete functional sublanguage, even though Gerhard Goos saw no problem in implementing it."

C.H.A. Koster, "The making of Algol 68"

Algol 68 at Malvern

- At the first Algol 68 implementation conference in 1970, the group from the Royal Signals and Radar Establishment (RSRE), Malvern demonstrated a working Algol-68 R compiler
- They went on to relax the lexical scoping restrictions on procedures returning procedures, and used closures to implement an object-based capability architecture…

Back to the future?

- On November 10 2009, Google announced a garbage-collected Algol-descended language called Go…
- A blog article compare Go against a Brand X Algol-descended language and concluded that Brand X was better:
  - “Go has more in the way of features but Brand X is more consistent with its features.
  - “Brand X feels complete in a way that Go doesn’t”
- Brand X = Algol 68

  http://cowlark.com/2009-11-15-go/

“What has been will be again, what has been done will be done again; there is nothing new under the sun”

Ecclesiastes 1:9

WHAT DOES THE FUTURE HOLD?

Language wars

1996 - James Gosling invents Java. Java is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Sun loudly heralds Java's novelty.

2001 - Anders Hejlsberg invents C#. C# is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Microsoft loudly heralds C#'s novelty.

> James Iry, “A brief, incomplete and mostly wrong history of programming languages”

The march of progress?
(due to Cay Horstmann)

1980: C

```
printf("%10.2f", x);
```

1988: C++

```
cout << setw(10) << setprecision(2) << showpoint << x;
```

1996: Java

```
NumberFormat formatter = NumberFormat.getNumberInstance();
formatter.setMinimumFractionDigits(2);
formatter.setMaximumFractionDigits(2);
String s = formatter.format(x);
for (int i = s.length(); i < 10; i++)
    System.out.print(' ');
System.out.print(s);
```

2004: Java

```
System.out.printf("%10.2f", x);
```

Some observations

- Programming languages succeed or fail for both technical and non-technical reasons
- But good language features survive or get re-invented
- To a first approximation, most programming languages end up with more or less the same features eventually…
- However, evolving a programming language with an established user based becomes increasingly difficult over time
- Language designers should learn from each other's mistakes and try to find a combination of features that work well together

Current trends and predictions

- Broad agreement and convergence on a common set of desirable language features
- Integration of functional paradigms such as closures into mainstream programming languages
- Hybrid type systems that support both static and dynamic typing, with type inference used to reduce verbosity
- Increased use of syntactic sugar for expressive power
- Annotations and meta object protocols
- Sophisticated language-aware tool support for refactoring, automated testing, “drag and drop” programming
- Java and C* aren’t going to go away…
- But neither are Fortran or Cobol!

Key contributions

- Fortran - first optimising compiler
- Lisp - list processing, symbolic computation
- Algol - block structure, recursion, procedures
- Simula - object-oriented programming
- Pascal - structured data types
- C - system programming
- Smalltalk - pure OO language
- Mesa - separate compilation
- Algol 68 – orthogonal design

Closing thought

"I think conventional languages are for the birds. They're really low level languages. They're just extensions of the von Neumann computer, and they keep our noses pressed in the dirt of dealing with individual words and computing addresses, and doing all kinds of silly things like that, things that we've picked up from programming for computers; we've built them into FORTRAN; we've built them into PL/I; we've built them into almost every language. The only languages that broke free from that are LISP and APL, and in my opinion they haven't gone far enough."

John Backus, inventor of FORTRAN, ACM HOPL conference, 1981

“Of making many books there is no end; and much study is a weariness of the flesh”

Ecclesiastes 12:12

REFERENCES

History of programming languages

- Wikipedia article
  - http://en.wikipedia.org/wiki/History_of_programming_languages
- O'Reilly poster
  - http://oreilly.com/news/languageposter_0504.html
- Original poster
  - http://www.levenez.com/lang/
- Quotes about programming languages
  - http://www.scriptol.com/programming/quotes.php

Go To considered harmful

- Dijkstra, “A case against the Go To statement” (aka “Go To statement considered harmful”)
  - http://www.cs.utexas.edu/users/EWD/transcriptions/EWD02xx/EWD215.html
- Rubin, “‘GOTO considered harmful’ considered harmful”, CACM 30(3), March 1987
  - http://web.archive.org/web/20090320002214/http://www.ecn.purdue.edu/ParaMount/papers/rubin87goto.pdf
- Dijkstra, “On a somewhat disappointing correspondence”
  - http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1009.PDF

Algol

- Call by name and Jensen's Device
  - http://www.cs.helsinki.fi/u/wikla/OKP/Asiat/JensensD.html
- Hoare, "Record Handling"
  - http://archive.computerhistory.org/resources/text/algol/algol_bulletin/A21/P36.HTM
- Hoare, "Null references: the billion dollar mistake"
  - http://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare

Algol 68

- Algol 68 Minority Report
  - http://archive.computerhistory.org/resources/text/algol/algol_bulletin/A31/P111.HTM
- Koster, “The making of Algol 68”
  - http://www.cs.ru.nl/~kees/home/papers/psi96.pdf
- On Go – a comparison with Algol 68
  - http://cowlark.com/2009-11-15-go/

Lisp / Scheme

- Steele and Sussman, Lambda papers
  - http://library.readscheme.org/page1.html
- Steele & Sussman, “Structure and Interpretation of Computer Programs”
  - http://mitpress.mit.edu/sicp/
- Graham, Lisp essays
  - http://www.paulgraham.com/lisp.html

Miscellaneous

- Bauer and Wössner, “The Plankalkül of Konrad Zuse: A forerunner of today’s programming languages”
  - http://www.catb.org/retro/plankalkuel/
- Dahl, “The birth of object-orientation: the Simula languages”
  - http://www.olejohandahl.info/old/birth-of-oo.pdf
- Hoare, “Hints on Programming Language Design”
  - ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/73/403/CS-TR-73-403.pdf

Humor

- Real programmers don't use Pascal
  - http://www.pbm.com/~lindahl/real.programmers.html
- A brief, incomplete and mostly wrong history of programming languages
  - http://james-iry.blogspot.co.uk/2009/05/brief-incomplete-and-mostly-wrong.html
- The evolution of a programmer
  - http://www.ariel.com.au/jokes/The_Evolution_of_a_Programmer.html
- The evolution of a Haskell programmer
  - http://www.willamette.edu/~fruehr/haskell/evolution.html