Some Programming Language History

Akim Demaille Étienne Renault Roland Levillain
first.last@lrde.epita.fr

EPITA — École Pour l'Informatique et les Techniques Avancées

April 25, 2019

Some Programming Language History

The Tower of Babel [Pigott, 2006]

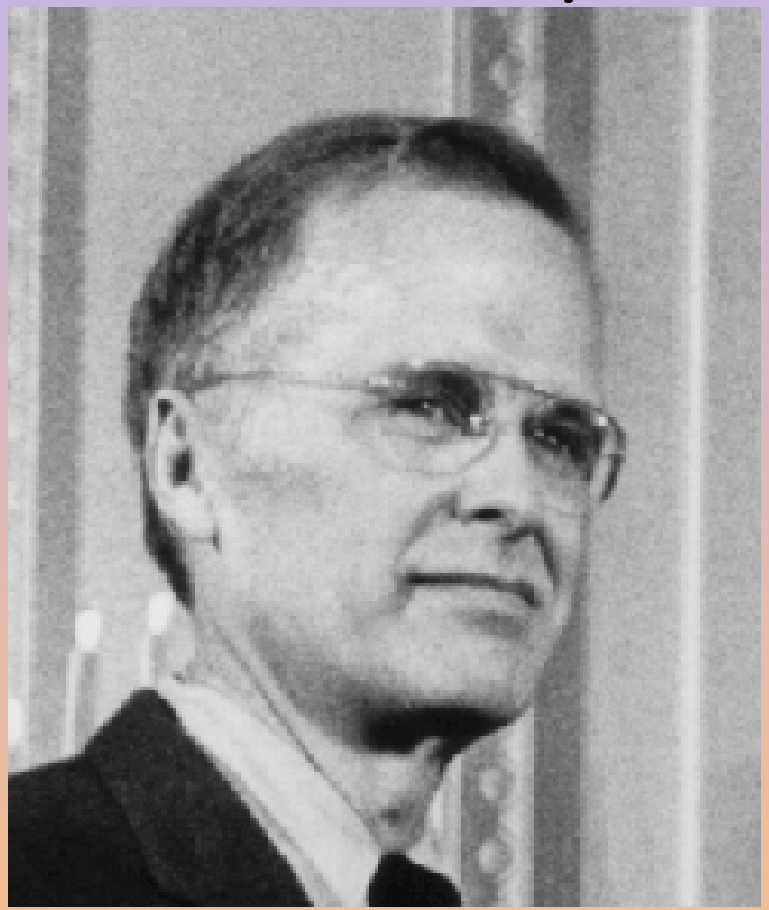
The Very First Ones

FORTRAN

IBM 704 (1956)

IBM Mathematical Formula Translator system

Fortran I, 1954-1956, IBM 704, a team led by John Backus.



IBM Mathematical Formula Translator System

The main goal is user satisfaction (economical interest) rather than academic. Compiled language.

- a single data structure : arrays
- comments
- arithmetics expressions
- `DO` loops
- subprograms and functions
- I/O
- machine independence

FORTRAN's success

Because:

- programmers productivity
- easy to learn
- by IBM
- the audience was mainly scientific
- simplifications (e.g., I/O)

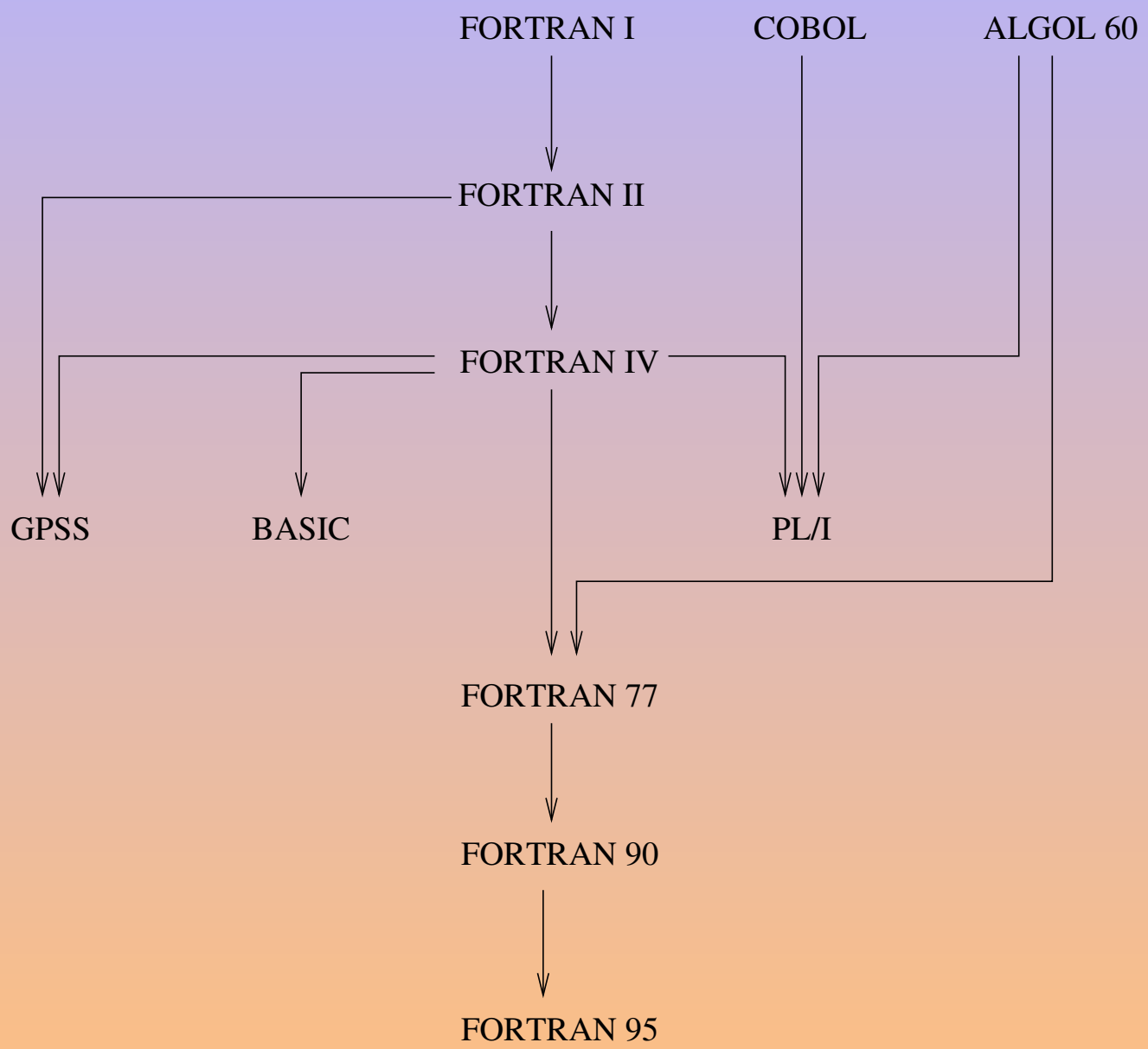
FORTRAN I

```
C       FIND THE MEAN OF N NUMBERS AND THE NUMBER OF
C       VALUES GREATER THAN IT
        DIMENSION A(99)
        REAL MEAN
        READ(1,5)N
 5      FORMAT(I2)
        READ(1,10)(A(I),I=1,N)
 10     FORMAT(6F10.5)
        SUM=0.0
        DO 15 I=1,N
 15        SUM=SUM+A(I)
        MEAN=SUM/FLOAT(N)
        NUMBER=0
        DO 20 I=1,N
           IF (A(I) .LE. MEAN) GOTO 20
           NUMBER=NUMBER+1
 20     CONTINUE
        WRITE (2,25) MEAN,NUMBER
 25     FORMAT(11H MEAN = ,F10.5,5X,21H NUMBER SUP = ,I5)
```

Fortran on Cards

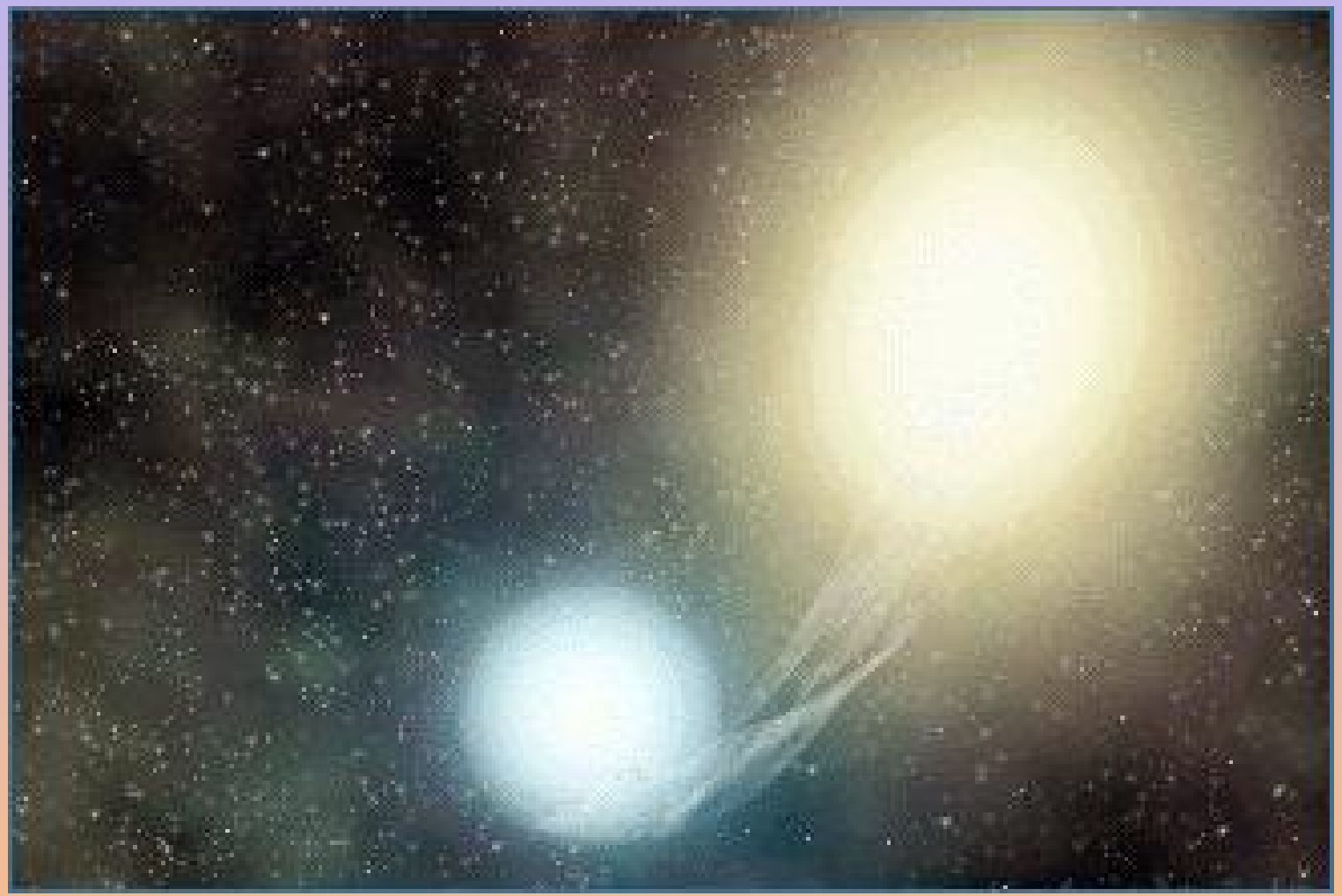
C ← FOR COMMENT	STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)	IDENTIFICATION (73–80)
C			PROGRAM FOR FINDING THE LARGEST VALUE	
C		X	ATTAINED BY A SET OF NUMBERS	
			DIMENSION A(999)	
			FREQUENCY 30(2,1,10), 5(100)	
			READ 1, N, (A(I), I = 1,N)	
	1		FORMAT (I3/(12F6.2))	
			BIGA = A(1)	
	5		DO 20 I = 2,N	
	30		IF (BIGA-A(I)) 10,20,20	
	10		BIGA = A(I)	
	20		CONTINUE	
			PRINT 2, N, BIGA	
	2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
			STOP 77777	

Fortrans


FORTRAN I
COBOL
ALGOL 60
FORTRAN II
FORTRAN IV
GPSS
BASIC
PL/I
FORTRAN 77
FORTRAN 90
FORTRAN 95

ALGOL

The Very First Ones



ALGOL, Demon Star, Beta Persei, 26 Persei

ALGOL 58

ALGOL 58

Originally, IAL, International Algebraic Language.
Goals:

1. Usable for algorithm publications in scientific reviews
2. As close as possible to the usual mathematical notations
3. Readable without assistance
4. Automatically translatable into machine code

Meeting between 8 Americans and Europeans in Zurich. ALGOL 58.

ALGOL 58

Originally, IAL, International Algebraic Language.
Goals:

1. Usable for algorithm publications in scientific reviews
2. As close as possible to the usual mathematical notations
3. Readable without assistance
4. Automatically translatable into machine code

Meeting between 8 Americans and Europeans in Zurich. ALGOL 58.

ALGOL 58

Originally, IAL, International Algebraic Language.
Goals:

1. Usable for algorithm publications in scientific reviews
2. As close as possible to the usual mathematical notations
3. Readable without assistance
4. Automatically translatable into machine code

Meeting between 8 Americans and Europeans in Zurich. ALGOL 58.

ALGOL 58

Originally, IAL, International Algebraic Language.
Goals:

1. Usable for algorithm publications in scientific reviews
2. As close as possible to the usual mathematical notations
3. Readable without assistance
4. Automatically translatable into machine code

Meeting between 8 Americans and Europeans in Zurich. ALGOL 58.

ALGOL 58

Originally, IAL, International Algebraic Language.
Goals:

1. Usable for algorithm publications in scientific reviews
2. As close as possible to the usual mathematical notations
3. Readable without assistance
4. Automatically translatable into machine code

Meeting between 8 Americans and Europeans in Zurich. ALGOL 58.

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

ALGOL 58 [Wikipedia, 2005]

- In May 1958, IAL was rejected as an "'unspeakable' and pompous acronym"
- Introduced the fundamental notion of compound statement
  - restricted to control flow only
  - not tied to identifier scope
- Used during 1959 to publish algorithm in CACM
  use of ALGOL notation in publication many years
- Primary contribution was to later languages: a basis for JOVIAL Quick, MAD, and NELIAC.
- Early compromise design soon superseded by ALGOL 60

JOVIAL [Wikipedia, 2005]

- "Jules Own Version of the International Algorithmic Language."
- Developed to write software for the electronics of military aircraft by Jules Schwartz in 1959.
- Runs the Advanced Cruise Missile, B-52, B-1, and B-2 bombers, C-130, C-141, and C-17 transport aircraft, F-15, F-16, F-18, and F-117 fighter aircraft, LANTIRN, U-2 aircraft, E-3 Sentry AWACS aircraft, Special Operations Forces, Navy AEGIS cruisers, Army Multiple Launch Rocket System (MLRS), Army UH-60 Blackhawk helicopters, F-100, F117, F119 jet engines, and RL-10 rocket engines.

JOVIAL [Wikipedia, 2005]

- "Jules Own Version of the International Algorithmic Language."
- Developed to write software for the electronics of military aircraft by Jules Schwartz in 1959.
- Runs the Advanced Cruise Missile, B-52, B-1, and B-2 bombers, C-130, C-141, and C-17 transport aircraft, F-15, F-16, F-18, and F-117 fighter aircraft, LANTIRN, U-2 aircraft, E-3 Sentry AWACS aircraft, Special Operations Forces, Navy AEGIS cruisers, Army Multiple Launch Rocket System (MLRS), Army UH-60 Blackhawk helicopters, F-100, F117, F119 jet engines, and RL-10 rocket engines.

JOVIAL [Wikipedia, 2005]

- "Jules Own Version of the International Algorithmic Language."
- Developed to write software for the electronics of military aircraft by Jules Schwartz in 1959.
- Runs the Advanced Cruise Missile, B-52, B-1, and B-2 bombers, C-130, C-141, and C-17 transport aircraft, F-15, F-16, F-18, and F-117 fighter aircraft, LANTIRN, U-2 aircraft, E-3 Sentry AWACS aircraft, Special Operations Forces, Navy AEGIS cruisers, Army Multiple Launch Rocket System (MLRS), Army UH-60 Blackhawk helicopters, F-100, F117, F119 jet engines, and RL-10 rocket engines.

ALGOL 60

ALGOL 60 Participants at HOPL, 1974

Figure: John Mac Carthy, Fritz Bauer, Joe Wegstein. Bottom row: John Backus, Peter Naur, Alan Perlis [Mac Carthy, 2006]

ALGOL 60: Novelties

- Use of BNF to describe the syntax
- Informal semantics
- Block structure
- Dynamic arrays
- Advanced control flow (`if`, `for`...)
- Recursivity

ALGOL 60: Novelties

- Use of BNF to describe the syntax
- Informal semantics
- Block structure
- Dynamic arrays
- Advanced control flow (`if`, `for`...)
- Recursivity

ALGOL 60: Novelties

- Use of BNF to describe the syntax
- Informal semantics
- Block structure
- Dynamic arrays
- Advanced control flow (`if`, `for`...)
- Recursivity

ALGOL 60: Novelties

- Use of BNF to describe the syntax
- Informal semantics
- Block structure
- Dynamic arrays
- Advanced control flow (`if`, `for`...)
- Recursivity

ALGOL 60: Novelties

- Use of BNF to describe the syntax
- Informal semantics
- Block structure
- Dynamic arrays
- Advanced control flow (`if`, `for`...)
- Recursivity

ALGOL 60: Novelties

- Use of BNF to describe the syntax
- Informal semantics
- Block structure
- Dynamic arrays
- Advanced control flow (`if`, `for`...)
- Recursivity

ALGOL 60: One syntax, three lexics [Mohr, 2004]

Reference language (used in the ALGOL-60 Report)

a[i+1] := (a[i] + pi x r^2) / $6.02_{10}23$;

Publication language

$a_{i+1} \leftarrow \{a_i + \pi \times r^2\}/6.02 \times 10^{23}$;

Hardware representations – implementation dependent

```
    a[i+1] := (a[i] + pi * r^2) / 6.02E23;
or  a(/i+1/) := (a(/i/) + pi * r ** 2) / 6,02e23;
or  A(.I+1.) .= (A(.I.) + PI * R 'POWER' 2) / 6.02'23.,
```

ALGOL 60: One syntax, three lexics [Mohr, 2004]

Reference language (used in the ALGOL-60 Report)

a[i+1] := (a[i] + pi x r^2) / $6.02_{10}23$;

Publication language

$a_{i+1} \leftarrow \{a_i + \pi \times r^2\}/6.02 \times 10^{23}$;

Hardware representations – implementation dependent

```
    a[i+1] := (a[i] + pi * r^2) / 6.02E23;
or  a(/i+1/) := (a(/i/) + pi * r ** 2) / 6,02e23;
or  A(.I+1.) .= (A(.I.) + PI * R 'POWER' 2) / 6.02'23.,
```

ALGOL 60: One syntax, three lexics [Mohr, 2004]

Reference language (used in the ALGOL-60 Report)

a[i+1] := (a[i] + pi x r^2) / $6.02_{10}23$;

Publication language

$a_{i+1} \leftarrow \{a_i + \pi \times r^2\}/6.02 \times 10^{23}$;

Hardware representations – implementation dependent

```
    a[i+1] := (a[i] + pi * r^2) / 6.02E23;
or  a(/i+1/) := (a(/i/) + pi * r ** 2) / 6,02e23;
or  A(.I+1.) .= (A(.I.) + PI * R 'POWER' 2) / 6.02'23.,
```

ALGOL 60: For Loops

for loop syntax

```
<for statement>
            ::= <for clause> <statement>
              | <label>: <for statement>

<for clause> ::= for <variable> := <for list> do

<for list> ::= <for list element>
             | <for list> , <for list element>

<for list element>
 ::= <arithmetic expression>
   | <arithmetic expression> step <arithmetic expression>
                             until <arithmetic expression>
   | <arithmetic expression> while <Boolean expression>
```

ALGOL 60: For Loops

for step until

```
for i := 1 step 2 until N do
      a[i] := b[i];
```

for while

```
for newGuess := Improve (oldGuess)
      while abs (newGuess - oldGuess) > 0.0001 do
   oldGuess := newGuess;
```

for enumerations

```
for days := 31,
            if mod( year, 4 ) = 0 then 29 else 28,
            31, 30, 31, 30, 31, 31, 30, 31, 30, 31 do
   . . .
```

ALGOL 60: For Loops

for step until

```
for i := 1 step 2 until N do
      a[i] := b[i];
```

for while

```
for newGuess := Improve (oldGuess)
      while abs (newGuess - oldGuess) > 0.0001 do
   oldGuess := newGuess;
```

for enumerations

```
for days := 31,
            if mod( year, 4 ) = 0 then 29 else 28,
            31, 30, 31, 30, 31, 31, 30, 31, 30, 31 do
   . . .
```

ALGOL 60: For Loops

for step until

```
for i := 1 step 2 until N do
      a[i] := b[i];
```

for while

```
for newGuess := Improve (oldGuess)
      while abs (newGuess - oldGuess) > 0.0001 do
   oldGuess := newGuess;
```

for enumerations

```
for days := 31,
            if mod( year, 4 ) = 0 then 29 else 28,
            31, 30, 31, 30, 31, 31, 30, 31, 30, 31 do
   . . .
```

ALGOL 60: For Loops

for complete

```
for i := 3, 7,
      11 step 1 until 16,
      i / 2 while i >= 1,
      2 step i until 32 do
        print (i);
```

ALGOL 60: Loss

- FORTRAN was occupying too much room
- Richer than FORTRAN, so more difficult
- IBM tried to impose ALGOL, but clients refused, and even threatened IBM
- FORTRAN compilers were more efficient and smaller
- No standardized I/O

ALGOL 60: Loss

- FORTRAN was occupying too much room
- Richer than FORTRAN, so more difficult
- IBM tried to impose ALGOL, but clients refused, and even threatened IBM
- FORTRAN compilers were more efficient and smaller
- No standardized I/O

ALGOL 60: Loss

- FORTRAN was occupying too much room
- Richer than FORTRAN, so more difficult
- IBM tried to impose ALGOL, but clients refused, and even threatened IBM
- FORTRAN compilers were more efficient and smaller
- No standardized I/O

ALGOL 60: Loss

- FORTRAN was occupying too much room
- Richer than FORTRAN, so more difficult
- IBM tried to impose ALGOL, but clients refused, and even threatened IBM
- FORTRAN compilers were more efficient and smaller
- No standardized I/O

ALGOL 60: Loss

- FORTRAN was occupying too much room
- Richer than FORTRAN, so more difficult
- IBM tried to impose ALGOL, but clients refused, and even threatened IBM
- FORTRAN compilers were more efficient and smaller
- No standardized I/O

ALGOL 60

```
begin
  comment The mean of numbers and the number of greater values;
  integer n;
  read(n);
  begin
    real array a[1:n];
    integer i, number;
    real sum, mean;
    for i := 1 step 1 until n do read (a[i]);
    sum := 0;
    for i := 1 step 1 until n do sum := sum + a[i];
    mean := sum / n;
    number := 0;
    for i := 1 step 1 until n do
      if a[i] > mean then
        number := number + 1;
    write ("Mean = ", mean, "Number sups = ", number);
  end
end
```

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

ALGOL 60: Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.

> *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.*
>
> — C.A.R. Hoare

ALGOL W

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of case (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of case (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of case (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of case (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of case (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of case (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

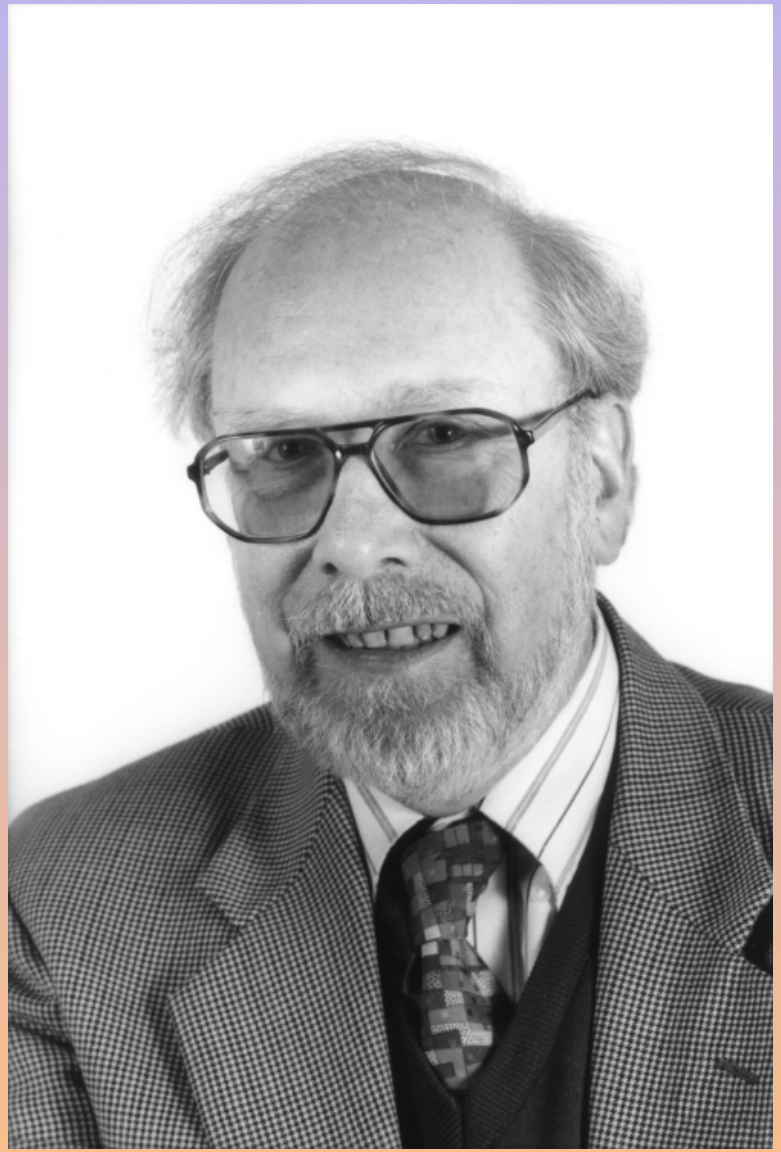
- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of `case` (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

ALGOL W

Niklaus Wirth, 1966:

- Agregates (records, structures)
- References (hence lists, trees, etc.)
- Split `for` into `for` and `while`
- Introduction of `case` (`switch`)
- Call by value, result, value-result
- New types `long real`, `complex`, `bits`
- Introduction of `assert`
- String processing functions

Niklaus Wirth [Wirth, 1999]



ALGOL 68

ALGOL 68 Samples [Wikipedia, 2005]

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

Procedures

```
proc max of real (real a, b) real:
  if a > b then a else b fi;
```

Ternary Operator

```
proc max of real (real a, b) real: (a > b | a | b);
```

ALGOL 68 Samples [Wikipedia, 2005]

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

Procedures

```
proc max of real (real a, b) real:
  if a > b then a else b fi;
```

Ternary Operator

```
proc max of real (real a, b) real: (a > b | a | b);
```

ALGOL 68 Samples [Wikipedia, 2005]

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

Procedures

```
proc max of real (real a, b) real:
  if a > b then a else b fi;
```

Ternary Operator

```
proc max of real (real a, b) real: (a > b | a | b);
```

ALGOL 68 Samples [Wikipedia, 2005]

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

Procedures

```
proc max of real (real a, b) real:
  if a > b then a else b fi;
```

Ternary Operator

```
proc max of real (real a, b) real: (a > b | a | b);
```

ALGOL 68 Samples [Wikipedia, 2005]

Arrays, Functional Arguments

```
proc apply (ref [] real a, proc (real) real f):
  for i from lwb a to upb a do a[i] := f(a[i]) od;
```

User Defined Operators

```
prio max = 9;

op max = (int a,b) int: (a>b | a | b);
op max = (real a,b) real: (a>b | a | b);
op max = (compl a,b) compl: (abs a > abs b | a | b);

op max = ([]real a) real:
   (real x := - max real;
    for i from lwb a to upb a
       do (a[i]>x | x:=a[i]) od;
    x);
```

ALGOL 68 Samples [Wikipedia, 2005]

Arrays, Functional Arguments

```
proc apply (ref [] real a, proc (real) real f):
  for i from lwb a to upb a do a[i] := f(a[i]) od;
```

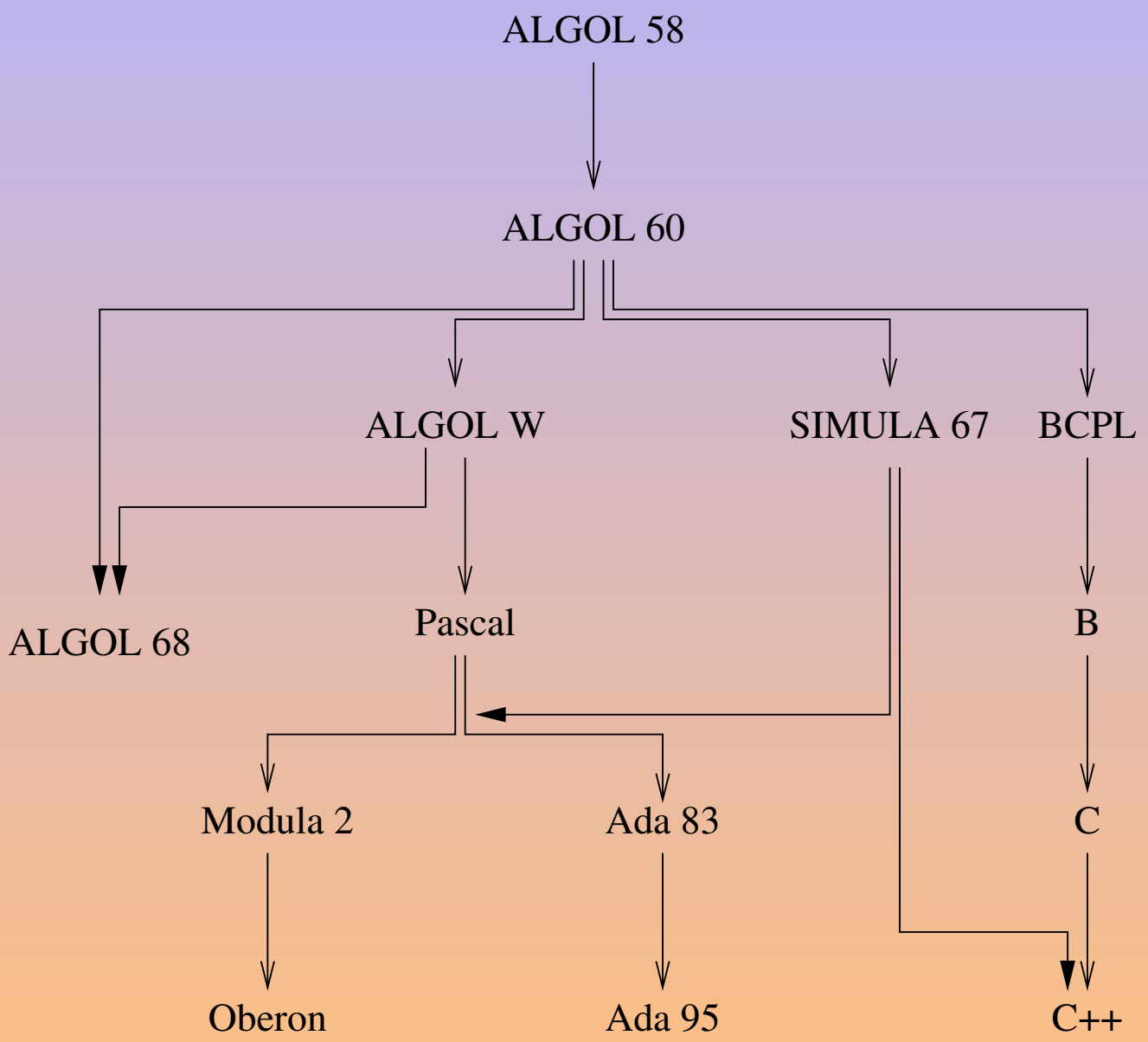
User Defined Operators

```
prio max = 9;

op max = (int a,b) int: (a>b | a | b);
op max = (real a,b) real: (a>b | a | b);
op max = (compl a,b) compl: (abs a > abs b | a | b);

op max = ([]real a) real:
   (real x := - max real;
    for i from lwb a to upb a
       do (a[i]>x | x:=a[i]) od;
    x);
```

ALGOL and its heirs



COBOL

Grace Murray, December 9, 1906 – January 1, 1992

Captain Grace Murray-Hopper

Rear Admiral Grace Murray-Hopper

Commodore Grace Murray-Hopper

Quotes from Grace Murray Hopper [Huggins, 2006]

> *Life was simple before World War II.*

> *I seem to do a lot of retiring.*

> *In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox.*
> *We shouldn't be trying for bigger computers, but for more systems of computers.*

> *Humans are allergic to change. They love to say, "We've always done it this way." I try to fight that. That's why I have a clock on my wall that runs counter-clockwise.*

Quotes from Grace Murray Hopper [Huggins, 2006]

> *Life was simple before World War II. After that, we had systems.*

> *I seem to do a lot of retiring.*

> *In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox.*
> *We shouldn't be trying for bigger computers, but for more systems of computers.*

> *Humans are allergic to change. They love to say, "We've always done it this way." I try to fight that. That's why I have a clock on my wall that runs counter-clockwise.*

Quotes from Grace Murray Hopper [Huggins, 2006]

> *Life was simple before World War II. After that, we had systems.*

> *I seem to do a lot of retiring.*

> *In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox.*
> *We shouldn't be trying for bigger computers, but for more systems of computers.*

> *Humans are allergic to change. They love to say, "We've always done it this way." I try to fight that. That's why I have a clock on my wall that runs counter-clockwise.*

Quotes from Grace Murray Hopper [Huggins, 2006]

> *Life was simple before World War II. After that, we had systems.*

> *I seem to do a lot of retiring.*

> *In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox.*
> *We shouldn't be trying for bigger computers, but for more systems of computers.*

> *Humans are allergic to change. They love to say, "We've always done it this way." I try to fight that. That's why I have a clock on my wall that runs counter-clockwise.*

Quotes from Grace Murray Hopper [Huggins, 2006]

> *Life was simple before World War II. After that, we had systems.*

> *I seem to do a lot of retiring.*

> *In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox.*
> *We shouldn't be trying for bigger computers, but for more systems of computers.*

> *Humans are allergic to change. They love to say, "We've always done it this way." I try to fight that. That's why I have a clock on my wall that runs counter-clockwise.*

Quotes from Grace Murray Hopper [Huggins, 2006]

> *A business' accounts receivable file is much more important than its accounts payable file.*

> *We're flooding people with information. We need to feed it through a processor. A human must turn information into intelligence or knowledge. We've tended to forget that no computer will ever ask a new question.*

Quotes from Grace Murray Hopper [Huggins, 2006]

> *A business' accounts receivable file is much more important than its accounts payable file.*

> *We're flooding people with information. We need to feed it through a processor. A human must turn information into intelligence or knowledge. We've tended to forget that no computer will ever ask a new question.*

COBOL

- Common Business Oriented Language, end of the 50's.
- The most used language worldwide for a long time.
- Imposed by the DOD, thanks to Grace Hopper:
  - to have a contract, a COBOL compiler was required,
  - any material bought on governmental funding had to have a COBOL compiler.
- A program is composed of divisions.

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INOUT.
* Read a file, add information to records, and save
* as another file.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

       SELECT INP-FIL ASSIGN TO INFILE.
       SELECT OUT-FIL ASSIGN TO OUTFILE.
```

COBOL (CONT'D)

```
DATA DIVISION.
FILE SECTION.

FD        INP-FIL
          LABEL RECORDS STANDARD
          DATA RECORD IS REC-IN.
01        REC-IN.
           05 α-IN PIC A(4).
           05 SP-CH-IN PIC X(4).
           05 NUM-IN   PIC 9(4).
FD        OUT-FIL
          LABEL RECORDS STANDARD
          DATA RECORD IS REC-OUT.
01        REC-OUT.
           05 α-OUT PIC A(4).
           05 SP-CH-OUT PIC X(4).
           05 NUM-OUT   PIC 9(4).
           05 EXTRAS    PIC X(16).
```

COBOL (CONT'D)

```
WORKING-STORAGE SECTION.
01        EOF PIC X VALUE IS 'N'.

PROCEDURE DIVISION.
AA.
          OPEN INPUT INP-FIL
          OPEN OUTPUT OUT-FIL

          PERFORM CC
          PERFORM BB THRU CC UNTIL EOF = 'Y'

          CLOSE INP-FIL, OUT-FIL
          DISPLAY "End of Run"

          STOP RUN
```

COBOL (CONT'D)

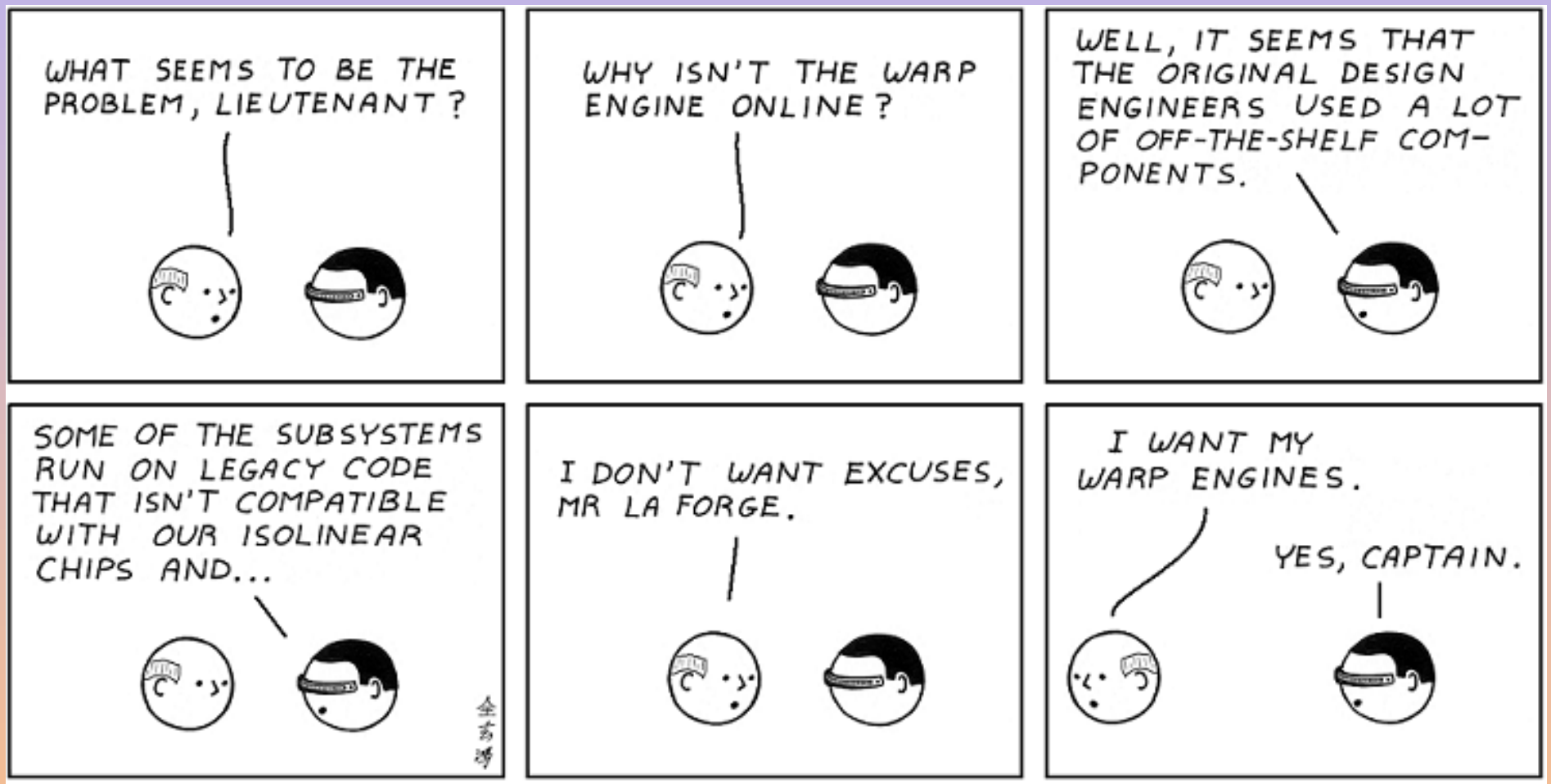
```
BB.
        MOVE REC-IN TO REC-OUT
        MOVE 'EXTRA CHARACTERS' TO EXTRAS
        WRITE REC-OUT.

CC.
        READ INP-FIL
                AT END MOVE 'Y' TO EOF.
```

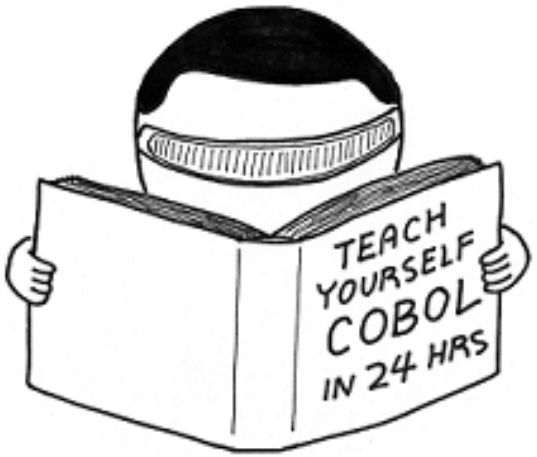
COBOL

> *The use of COBOL cripples the mind;*
> *its teaching should, therefore, be regarded as a criminal offense.*
>
> — Edsger Dijkstra

In the 24th century... [Goose, 2010]



In the 24th century. . . [Goose, 2010]



New technologies will come and go
but COBOL is forever.

The Second Wave

APL

The Second Wave

Kenneth E. Iverson

APL

> *APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard.*
>
> — David Given

> *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.*
>
> — Edsger Dijkstra, 1968

> *By the time the practical people found out what had happened; APL was so important a part of how IBM ran its business that it could not possibly be uprooted.*
>
> — Micheal S. Montalbano, 1982

APL

> *APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard.*
>
> — David Given

> *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.*
>
> — Edsger Dijkstra, 1968

> *By the time the practical people found out what had happened; APL was so important a part of how IBM ran its business that it could not possibly be uprooted.*
>
> — Micheal S. Montalbano, 1982

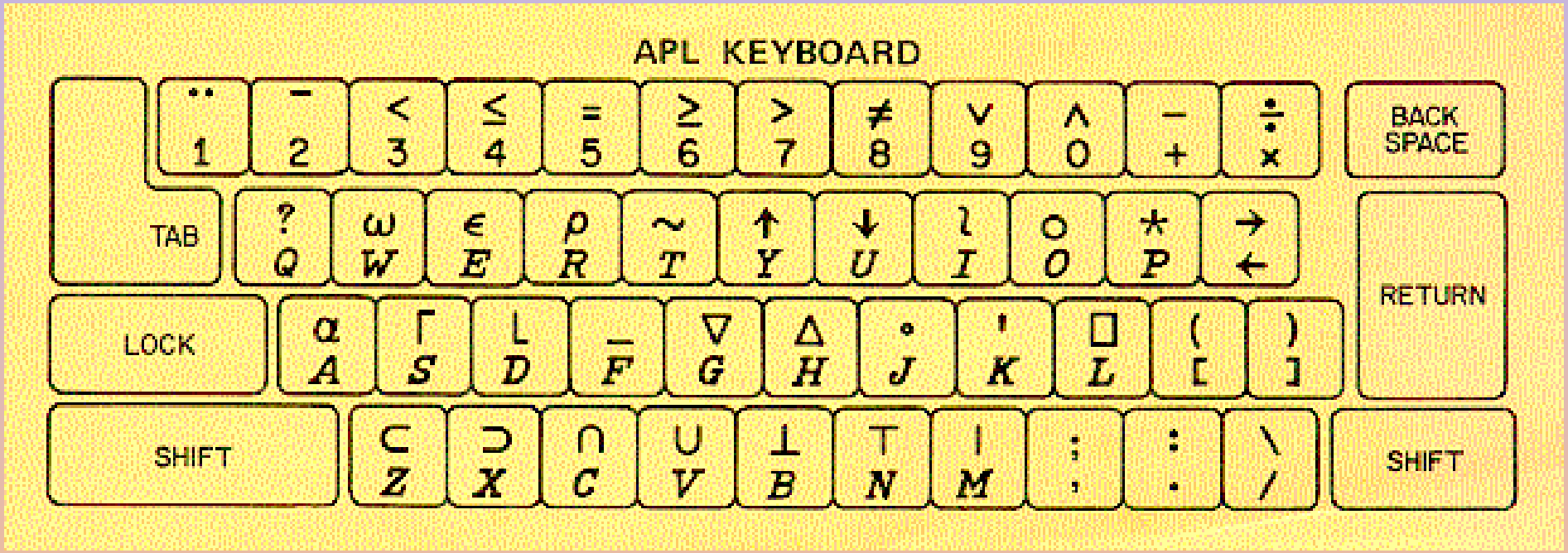
APL

> *APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard.*
>
> — David Given

> *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.*
>
> — Edsger Dijkstra, 1968

> *By the time the practical people found out what had happened; APL was so important a part of how IBM ran its business that it could not possibly be uprooted.*
>
> — Micheal S. Montalbano, 1982

APL Keyboard



APL Program

Prime Numbers up to R

$$(\sim R \in R \circ . \times R)/R \rightarrow 1 \downarrow \iota R$$

APL IDE

APL2 1001 - Object Editor - FFT

Object Edit Breakpoints Signals Options Windows Help

```
[0]   Z←FFT A;L;M;P;W;⎕IO
[1]  ⍝ Calculate complex FFT (Fast Fourier Transform).
[2]   ⎕IO←0
[3]   A←((⌈2⍟W←⍴,A)⍴2)⍴A          ⍝ Structure data as 2 by 2 by ...  array
[4]   →(1 0=M)/L3,0                ⍝ If 2 points loop once, if 1 exit
[5]  ⍝ Compute first quadrant cosine,sine array
[6]  ⍝ Get second quadrant by replication
[7]   W←(1↓⍴A)⍴W,0J1×W←¯12○2×(⍳W÷4)÷W ⍝ ¯12○X is -0J1×X
[8]   P←M←0.5
[9]   L←1
[10]  →L2
[11] L1:W←⊃(⊂0 0)⌷[M-L]W          ⍝ Reduce order of W on each loop
[12] L2:A←(+/A),[P-L]W×-/A         ⍝ Do the transform
[13]  →(M>L←L+1)↑L1
[14] ⍝ Do last step separately since multiply is not needed
[15] L3:Z←,(+/A),[¯0.5]-/A
```

APL On Index [11;24] Fix time: 29/06/1991 11:00:00

PL/I

PL/I

Be able to address all the needs:

- scientific (floats, arrays, procedures, efficient computation)
- business (fixed points, fast asychronous I/O, string processing functions, search and sort routines)
- real time
- filtering
- bit strings
- lists

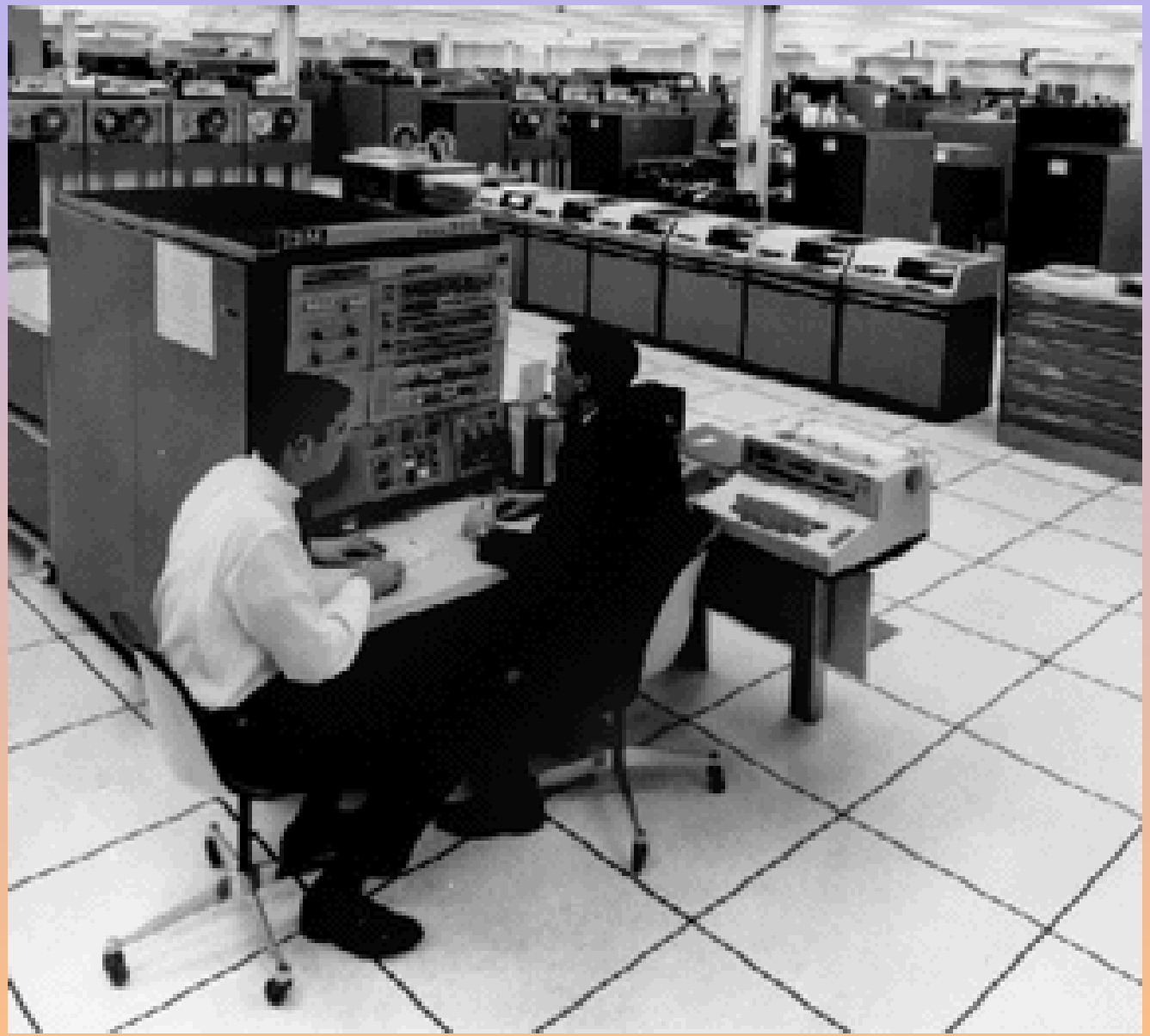
By IBM for IBM 360. "Includes" FORTRAN IV, ALGOL 60, COBOL 60 and JOVIAL. Introduction of `ON`, for exceptions.

PL/I

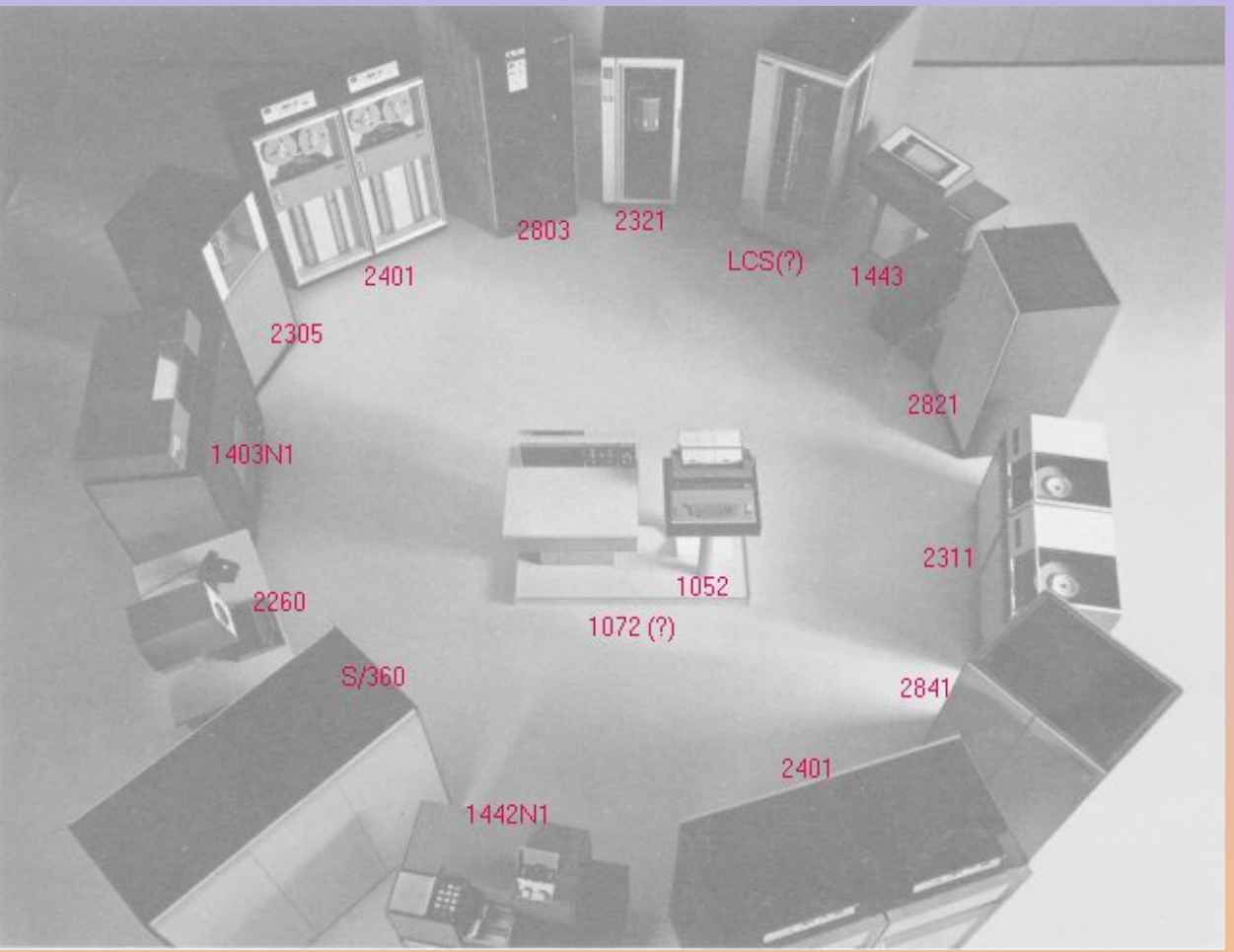
1963 FORTRAN VI

- They quickly dropped compatibility: NPL
- 1965, implementation in England of PL/I

IBM 360



IBM 360



1442N1	Card reader / punch
S/360	CPU, model 30(?)
2260	Display terminal
1403N1	Impact printer
2305	Drum storage
2401	Tape storage
2803	Tape control unit
2321	Data cell storage
LCS	Large core storage device
1443	Impact printer
2821	Control unit
2311	Disk storage
2841	DASD control unit
1052	Console typewriter
1072	Console station

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: DCL (not DEC) for DECLARE, PROC for PROCEDURE.
- 25 + 1/3 yields 5.33333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: DCL (not DEC) for DECLARE, PROC for PROCEDURE.
- 25 + 1/3 yields 5.333333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: DCL (not DEC) for DECLARE, PROC for PROCEDURE.
- 25 + 1/3 yields 5.33333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: `DCL` (not `DEC`) for `DECLARE`, `PROC` for `PROCEDURE`.
- 25 + 1/3 yields 5.33333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: DCL (not DEC) for DECLARE, PROC for PROCEDURE.
- 25 + 1/3 yields 5.33333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: `DCL` (not `DEC`) for `DECLARE`, `PROC` for `PROCEDURE`.
- 25 + 1/3 yields 5.33333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

PL/I Surprises [Holt, 1972]

- No reserved keywords in PL/I.

```
IF IF = THEN THEN THEN = ELSE ELSE ELSE = IF
```

- Abbreviations: `DCL` (not `DEC`) for `DECLARE`, `PROC` for `PROCEDURE`.
- 25 + 1/3 yields 5.333333333333333 ("undefined" says [IBM,])
- 25 + 01/3 behaves as expected...
- the loop

```
DO I = 1 TO 32/2 ,
Statements
END;
```

  is executed zero times.
- "Advanced" control structures

```
GOTO I,(1,2,3,92)
```

- Implementation of MULTICS

```
EXAMPLE : PROCEDURE OPTIONS (MAIN);
  /* Find the mean of n numbers and the number of
     values greater than it */
  GET LIST (N);
  IF N > 0 THEN
      BEGIN;
      DECLARE MEAN, A(N), DECIMAL POINT
              NUM DEC FLOAT INITIAL(0),
              NUMBER FIXED INITIAL (0)
      GET LIST (A);
      DO I = 1 TO N;
        SUM = SUM + A(I);
      END
      MEAN = SUM / N;
      DO I = 1 TO N;
        IF A(I) > MEAN THEN
          NUMBER = NUMBER + 1;
      END
      PUT LIST ('MEAM = ', MEAN,
                'NUMBER SUP = ', NUMBER);
END EXAMPLE;
```

PL/I by Dijkstra

> *When FORTRAN has been called an infantile disorder, full PL/1, with its growth characteristics of a dangerous tumor, could turn out to be a fatal disease.*
>
> — Edsger Dijkstra

PL/I by Dijkstra

> *Using PL/I must be like flying a plane with 7000 buttons, switches, and handles to manipulate in the cockpit. I absolutely fail to see how we can keep our growing programs firmly within our intellectual grip when by its sheer baroqueness, the programming language–our basic tool, mind you!–already escapes our intellectual control.*
> *And if I have to describe the influence PL/I can have on its users, the closest metaphor that comes to my mind is that of a drug.*
>
> — [Dijkstra, 1972]

BASIC

BASIC

Beginner's All-purpose Symbolic Instruction Code, J. Kemeny et T. Kurtz, 1965.
Made to be simple and interpreted (NEW, DELETE, LIST, SAVE, OLD, RUN).

```
 10 REM FIND THE MEAN OF N
 12 REM NUMBERS AND THE
 14 REM NUMBER OF VALUES
 16 REM GREATER THAN IT
 20 DIM A(99)
 30 INPUT N
 40 FOR I = 1 TO N
 50 INPUT A(I)
 60 LET S = S + A(I)
 70 NEXT I
 80 LET M = S / N
 90 LET K = 0
100 FOR I = 1 TO N
110 IF A(I) < M THEN 130
120 LET K = K + 1
130 NEXT I
140 PRINT "MEAN = ", M
150 PRINT "NUMBER SUP = ", K
160 STOP
```

Pascal & Heirs

Pascal

Niklaus Wirth, end of the 60's.

- Keep the ALGOL structure, but obtain FORTRAN's performances.
- `repeat`, `until`.
- Enumerated types.
- Interval types.
- Sets.
- Records.
- No norm/standard.

Ada (83)

A command from the DOD in the 70's. Embeded systems.

- Strawman, spec.
- Woodenman,
- Tinman, no satisfying language, hence a competition.
- Ironman,
- Steelman, Ada, the green language, wins. Jean Ichbiah, Honeywell-Bull.

Package, package libraries, rich control structures, `in`, `out`, `in out`, interruptions, exceptions, `clock`.

Modula-2, Oberon

Niklaus Wirth.
Modula-2 :

- Module, interface, implementation.
- Uniform syntax.
- Low level features (system programming).
- Processes, synchronization, co-routines.
- Procedure types.

Oberon : Inheritance.

The Finale

K. N. King

K. N. King & Jean Ichbiah [King, 1993]

K. N. King & Alan Kay [King, 1993]

K. N. King & Dennis Ritchie [King, 1993]

K. N. King & Bjarne Stroustrup [King, 1993]

K. N. King & Niklaus Wirth [King, 1993]

Quotes

Programming Languages

COBOL was designed so that managers could read code
BASIC was designed for people who are not programmers
FORTRAN is for scientists
ADA comes from a committee, a government committee no less
PILOT is for teachers
PASCAL is for students
LOGO is for children
APL is for Martians
FORTH, LISP and PROLOG are specialty languages

Programming Languages

COBOL was designed so that managers could read code
BASIC was designed for people who are not programmers
FORTRAN is for scientists
ADA comes from a committee, a government committee no less
PILOT is for teachers
PASCAL is for students
LOGO is for children
APL is for Martians
FORTH, LISP and PROLOG are specialty languages
C, however, is for programmers

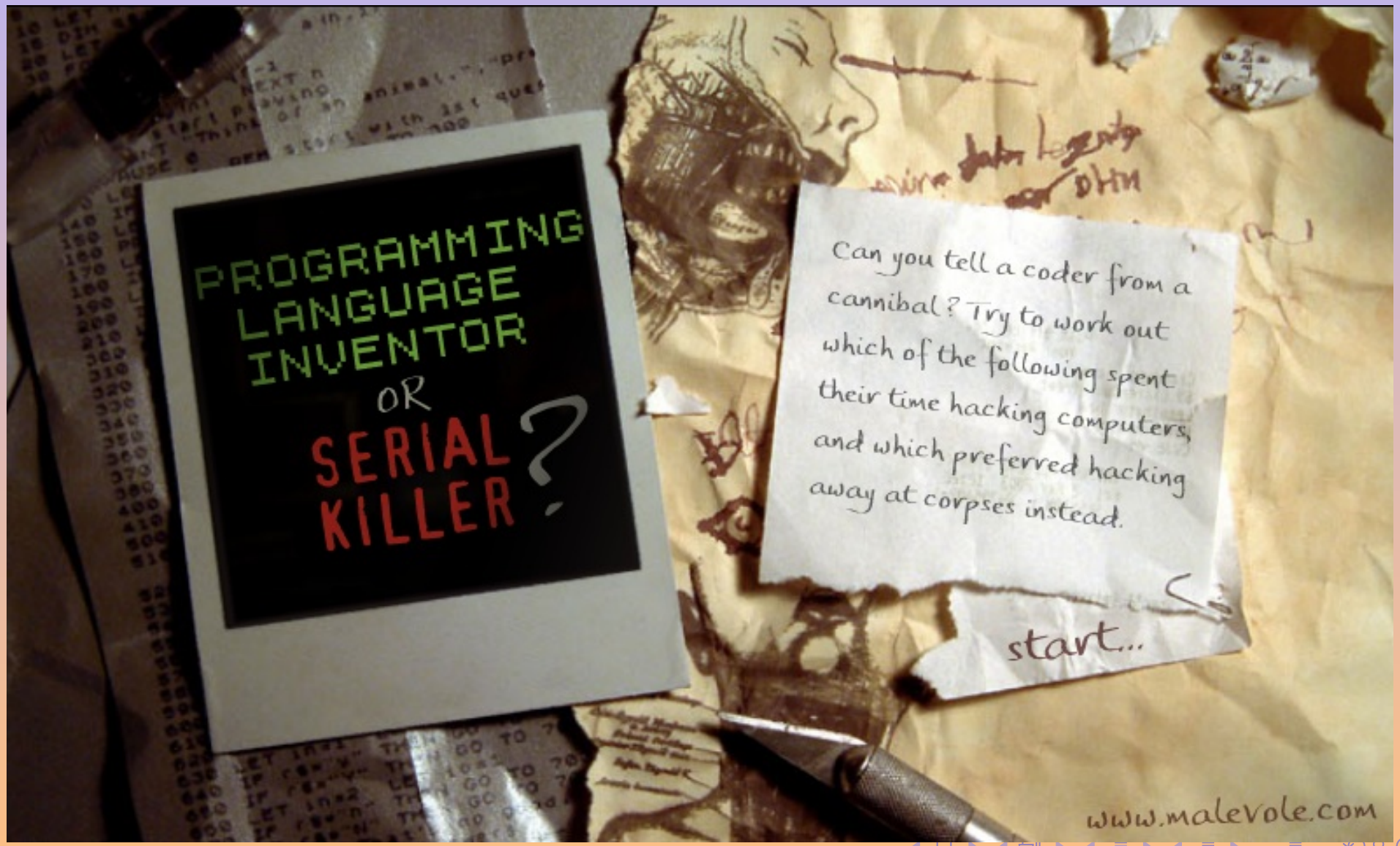
So far we've managed to avoid turning into APL. :-)

Larry Wall

So far we've managed to avoid turning Perl into APL. :-)

The Quiz

Programming Language Inventor or Serial Killer? [Round, 2006]



Bibliographic notes

[Pigott, 2006] An extensive dictionary of programming languages and their relations.

[Pixel, 2007] Syntactic comparison between programming languages.

Bibliography I

Dijkstra, E. W. (1972).
The humble programmer.
Commun. ACM, 15(10):859–866.
http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF.

Goose, A. (2010).
in the 24th century. . . — Abstruse Goose.
2010-12-03.

Holt, R. C. (1972).
Teaching the fatal disease (or) introductory computer programming using PL/I.
http://plg.uwaterloo.ca/~holt/papers/fatal_disease.html.

Huggins, J. (2006).
Grace Murray Hopper.
http://www.jamesshuggins.com/h/tek1/grace_hopper.htm.

Bibliography II

IBM.
Enterprise PL/I for z/OS, version 3.6, language reference, arithmetic operations.

King, K. N. (1993).
Photos from history of programming languages ii.
http://www2.gsu.edu/~matknk/hopl.html.

Mac Carthy, J. (2006).
jmc's web page.
http://www-formal.stanford.edu/jmc/index.html.

Mohr, J. (2004).
Computing science 370 programming languages.
http://www.augustana.ca/~mohrj/courses/2004.fall/csc370/index.html.

Bibliography III

Pigott, D. (2006).
HOPL: an interactive roster of programming languages.

Pixel (2007).
Syntax across languages.

Round, M. (2006).
Programming language inventor or serial killer quiz.
http://www.malevole.com/mv/misc/killerquiz/.

Wikipedia (2005).
Wikipedia, free encyclopedia.
http://en.wikipedia.org/wiki/Main_Page.

Wirth, N. (1999).
Miklaus Wirth Home Page.
http://www.cs.inf.ethz.ch/~wirth/.