

COMP 381

Programming Languages

What is a programming language?

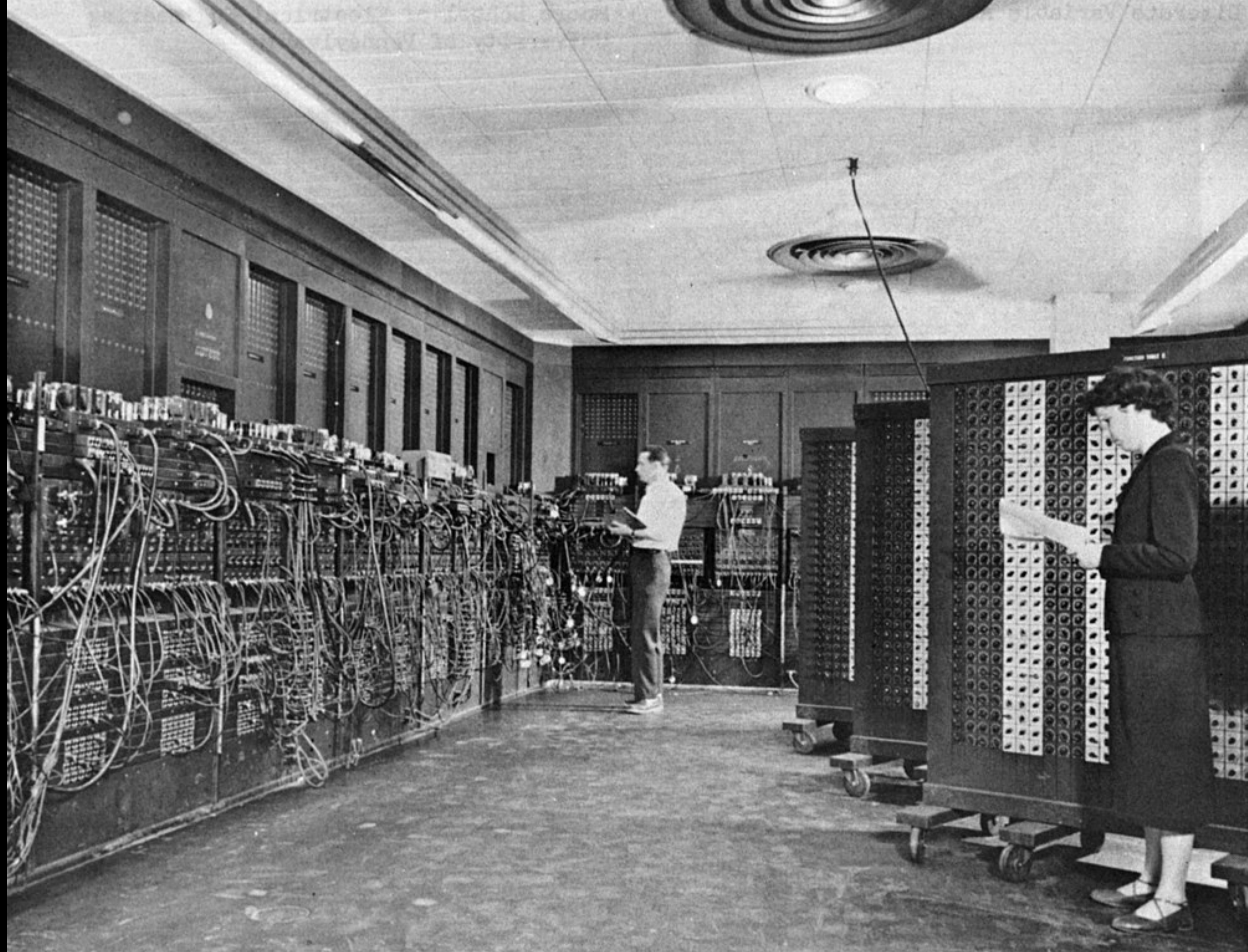
What are the essential distinguishing characteristics of a programming language?

What are the essential distinguishing characteristics of a programming language?

machine readable

What are the essential distinguishing characteristics of a programming language?

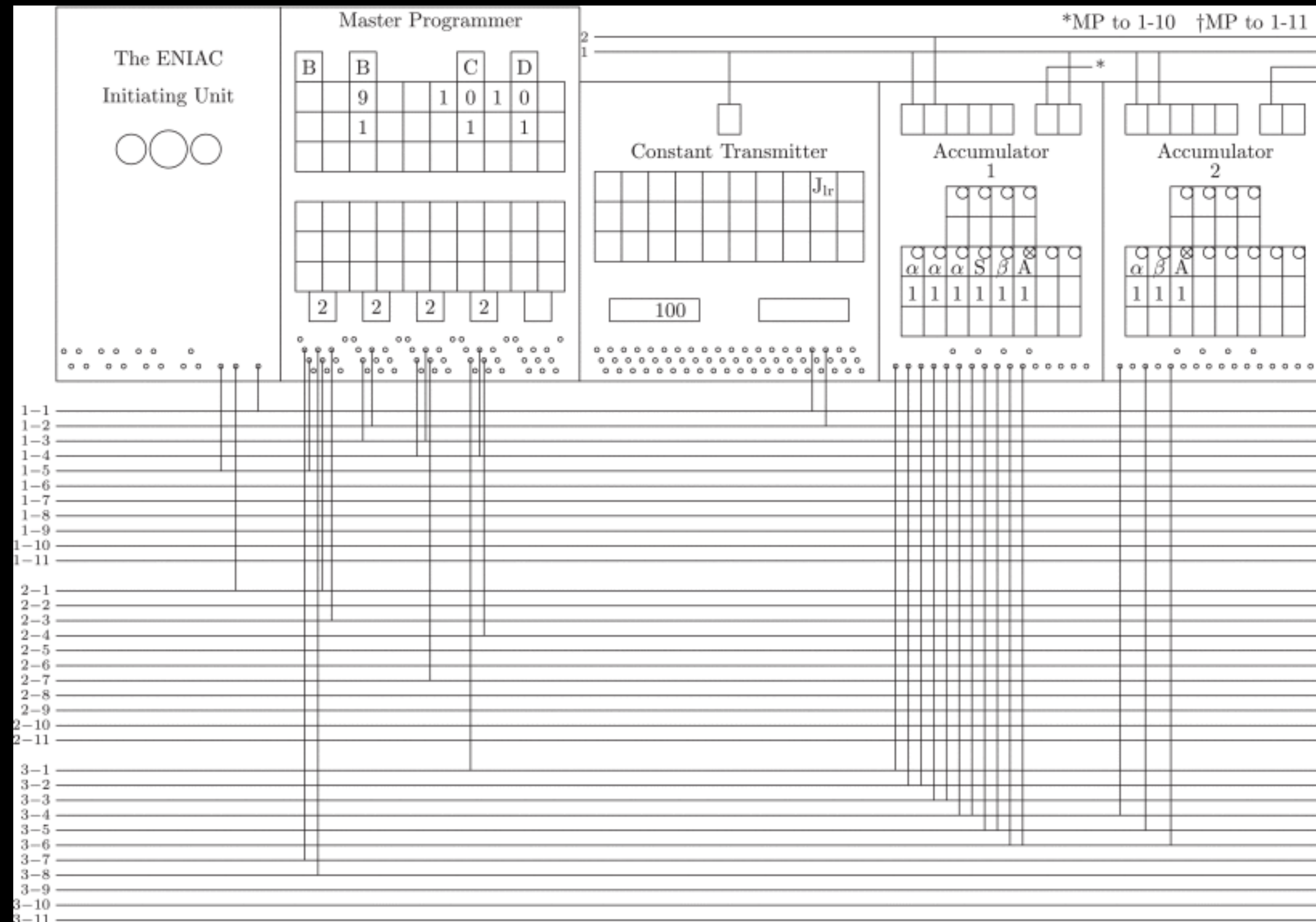
machine readable → unambiguous

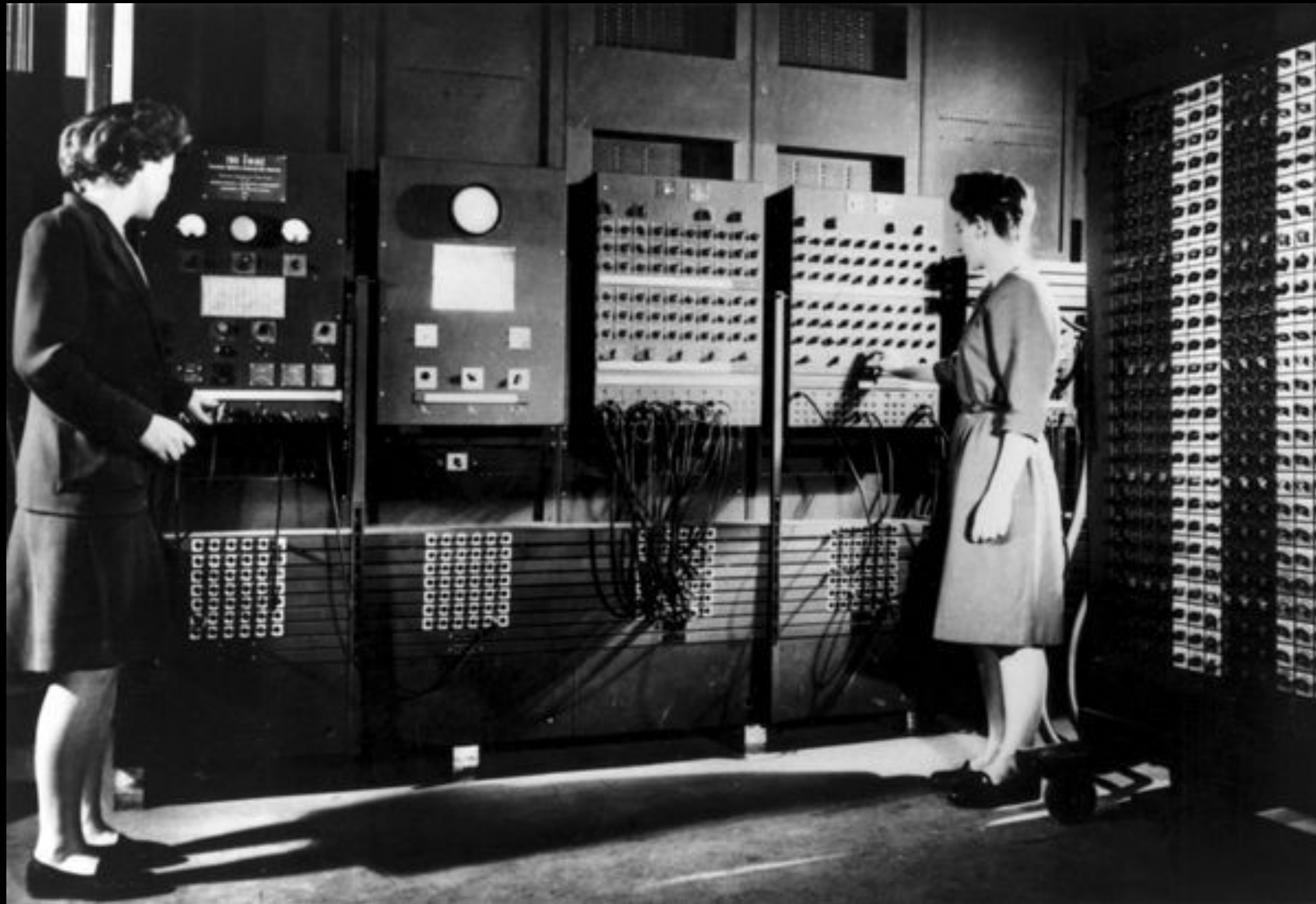


ENIAC, mid-1940s (Glen Beck and Betty Snyder)

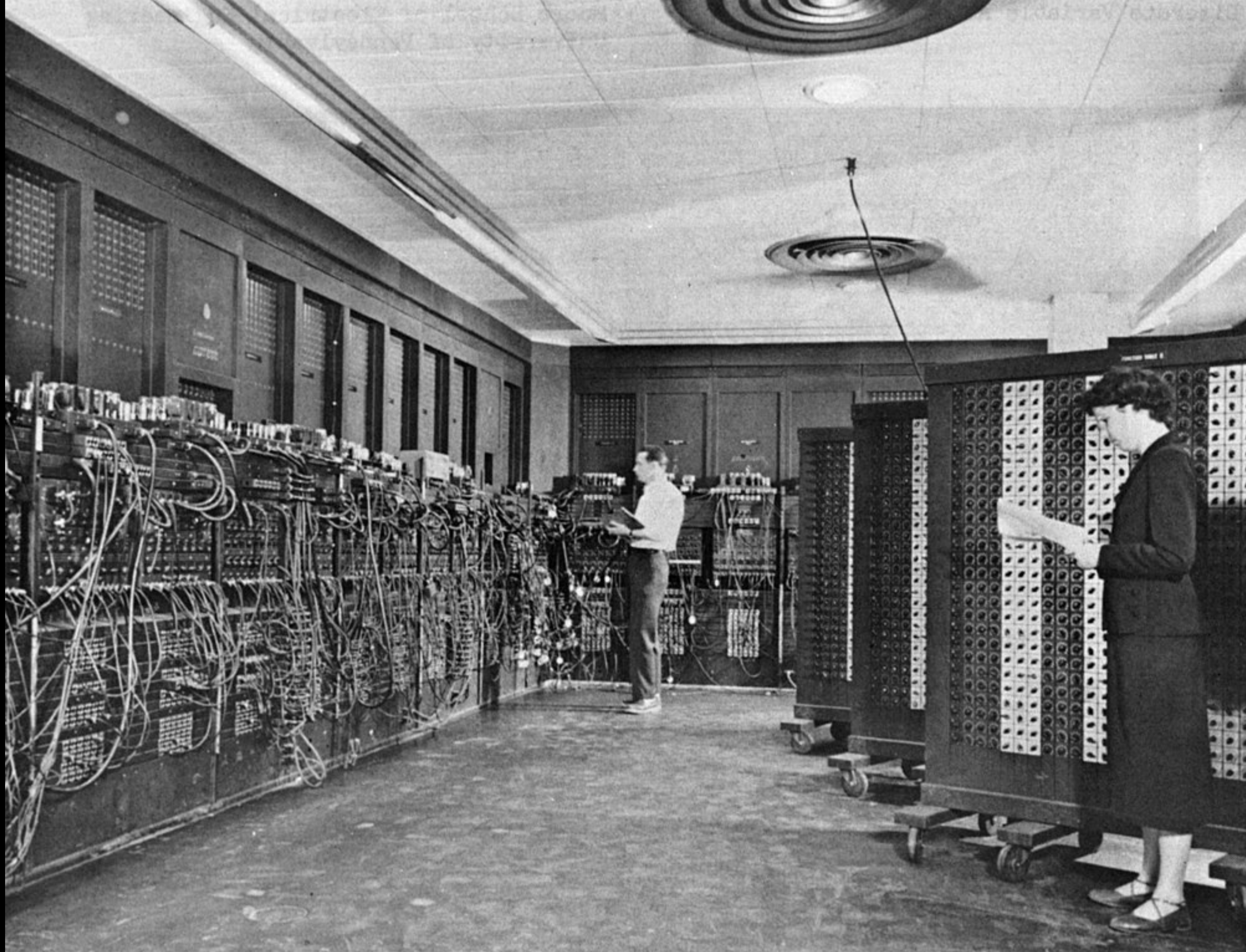
What did an ENIAC program look like?

What did an ENIAC program look like?





ENIAC programmers Frances Bilas and Betty Jean Jennings, ~1946



ENIAC, mid-1940s (Glen Beck and Betty Snyder)

What are the essential distinguishing characteristics of a programming language?

machine readable → unambiguous

What are the essential distinguishing characteristics of a programming language?

machine readable → unambiguous

human readable

What are the essential distinguishing characteristics of a programming language?

machine readable → unambiguous

human readable → expressive

Early programming languages

Early programming languages

SHORT CODE (1950)

```
07Y10204X1Y1  
0000X30309X1
```

Early programming languages

SHORT CODE (1950)

X3 = (X1 + Y1) / X1 * Y1 desired expression

X3 03 09 X1 07 Y1 02 04 X1 Y1 operators encoded

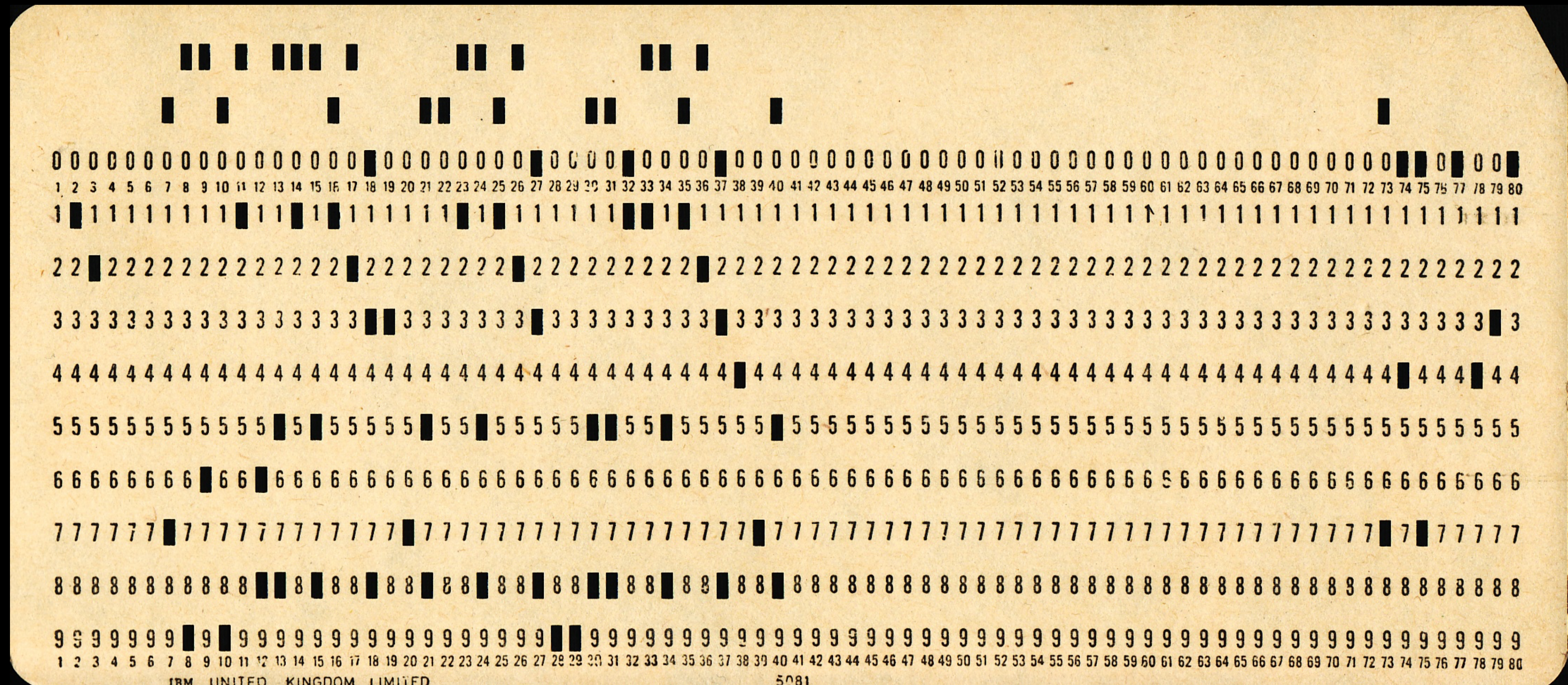
07Y10204X1Y1 result grouped into 12-byte words
0000X30309X1

Early programming languages

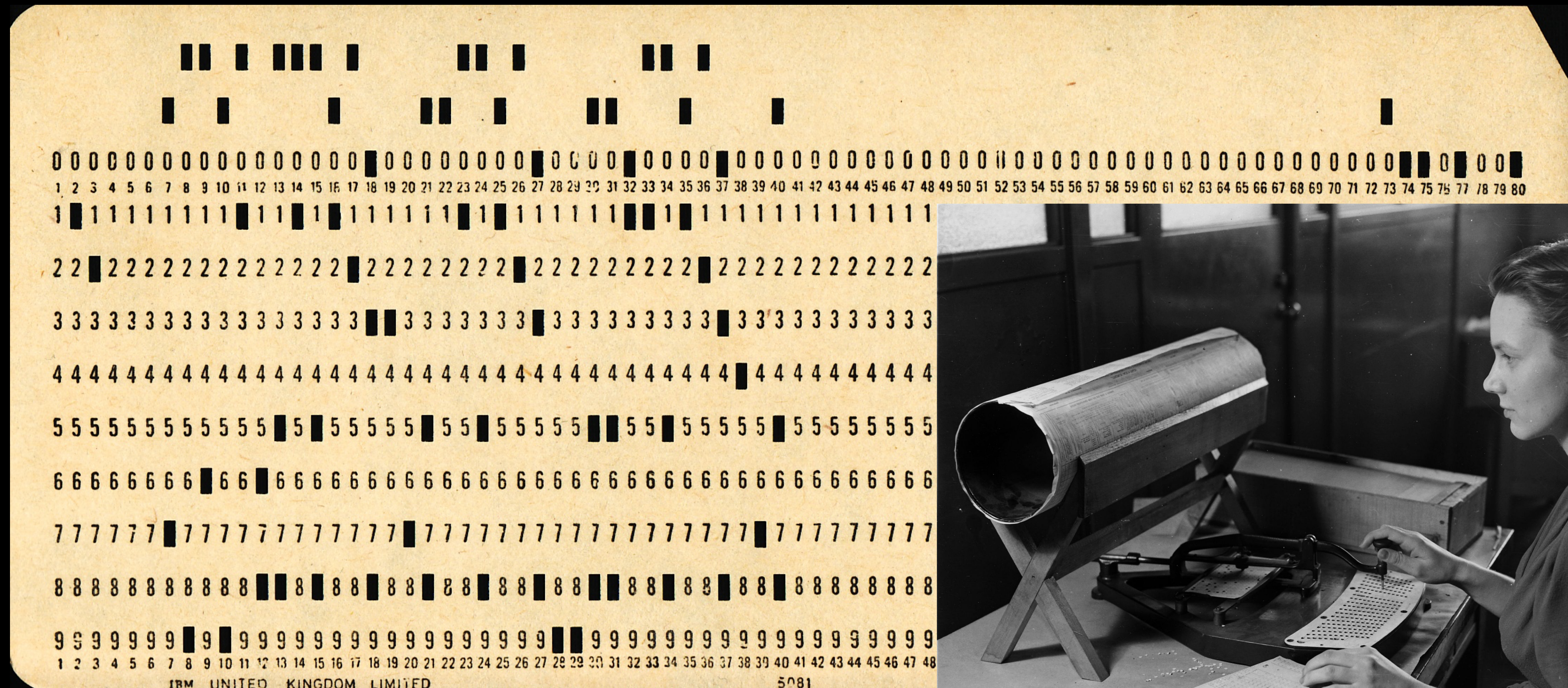
SHORT CODE (1950)

```
07Y10204X1Y1  
0000X30309X1
```

Early programming languages



Early programming languages



Early programming languages

SHORT CODE (1950)

```
0000X30309X1
07Y10204X1Y1
```

AUTOCODE (1952)

```
c@VA t@IC x@ $\frac{1}{2}$ C y@RC z@NC
INTEGERS +5 →C
           →t
+t       TESTA Z
-t
          ENTRY Z
```

APL (1966)

```
grids←(N N)ρ⌈M, M≠M/(M M)ριN
```

Early programming languages

FLOW-MATIC (1955-1959)

```
(0)  INPUT INVENTORY FILE-A PRICE FILE-B ; OUTPUT PRICED-INV FILE-C UNPRICED-INV
      FILE-D ; HSP D .
(1)  COMPARE PRODUCT-NO (A) WITH PRODUCT-NO (B) ; IF GREATER GO TO OPERATION 10 ;
      IF EQUAL GO TO OPERATION 5 ; OTHERWISE GO TO OPERATION 2 .
(2)  TRANSFER A TO D .
(3)  WRITE-ITEM D .
(4)  JUMP TO OPERATION 8 .
(5)  TRANSFER A TO C .
(6)  MOVE UNIT-PRICE (B) TO UNIT-PRICE (C) .
(7)  WRITE-ITEM C .
(8)  READ-ITEM A ; IF END OF DATA GO TO OPERATION 14 .
(9)  JUMP TO OPERATION 1 .
(10) READ-ITEM B ; IF END OF DATA GO TO OPERATION 12 .
(11) JUMP TO OPERATION 1 .
(12) SET OPERATION 9 TO GO TO OPERATION 2 .
(13) JUMP TO OPERATION 2 .
(14) TEST PRODUCT-NO (B) AGAINST ; IF EQUAL GO TO OPERATION 16 ;
      OTHERWISE GO TO OPERATION 15 .
(15) REWIND B .
(16) CLOSE-OUT FILES C ; D .
(17) STOP . (END)
```

What are the essential distinguishing characteristics of a programming language?

machine readable → unambiguous

human readable → expressive

What are the essential distinguishing characteristics of a programming language?

machine readable



unambiguous

human readable



expressive

“Machine readable” vs “Human readable”

```
public static boolean isOdd(int n) {  
    return n % 2 == 1;  
}
```

isOdd(4) → false

isOdd(5) → true

isOdd(-4) → false

isOdd(-5) → false

What are the essential distinguishing characteristics of a programming language?

machine readable



unambiguous

human readable



expressive

Early programming languages

FORTRAN (1957–)

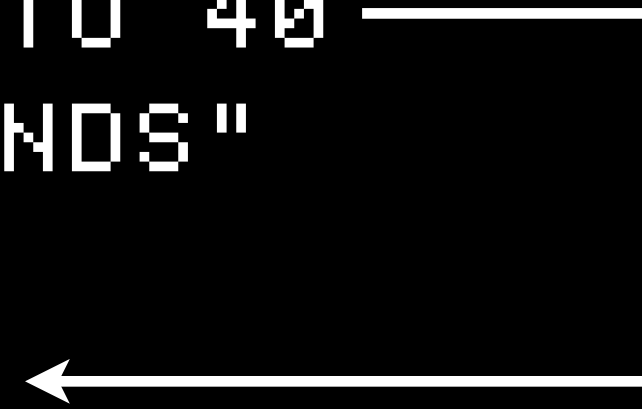
```
C AREA OF A TRIANGLE – HERON'S FORMULA
C INPUT – CARD READER UNIT 5, INTEGER INPUT, ONE BLANK CARD FOR END-OF-DATA
C OUTPUT – LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPAY ERROR MESSAGE ON OUTPUT
501 FORMAT(3I5)
601 FORMAT(4H A= ,I5,5H B= ,I5,5H C= ,I5,8H AREA= ,F10.2,
$13H SQUARE UNITS)
602 FORMAT(10HNORMAL END)
603 FORMAT(23HINPUT ERROR, ZERO VALUE)
      INTEGER A,B,C
10  READ(5,501) A,B,C
      IF(A.EQ.0 .AND. B.EQ.0 .AND. C.EQ.0) GO TO 50
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) GO TO 90
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C) )
      WRITE(6,601) A,B,C,AREA
      GO TO 10
50  WRITE(6,602)
      STOP
90  WRITE(6,603)
      STOP
      END
```

GOTO

```
10 IF X <= MAX THEN GOTO 40
20 PRINT "X OUT OF BOUNDS"
30 STOP
40 ...
```

GOTO

```
10 IF X <= MAX THEN GOTO 40  
20 PRINT "X OUT OF BOUNDS"  
30 STOP  
40 ...
```



GOTO

```
10 IF X <= MAX THEN GOTO 40  
20 PRINT "% OUT OF BOUNDS"  
30 STOP  
40 ...
```



```
if x > max  
    print "x out of bounds"  
    exit(0)
```

GOTO

```
10 IF X <= MAX THEN GOTO 40  
20 PRINT "X IS LARGE"  
30 GOTO 50  
40 PRINT "X IS SMALL"  
50 ...
```



```
graph TD; L10[10 IF X <= MAX THEN GOTO 40] --> L40[40 PRINT "X IS SMALL"]; L20[20 PRINT "X IS LARGE"]; L30[30 GOTO 50] --> L50[50 ...]; L40 --> L50;
```

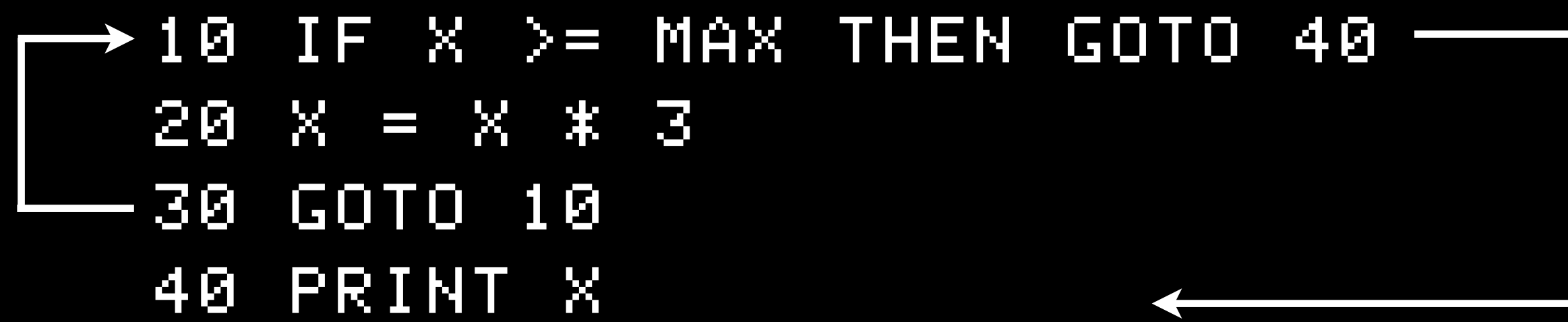
GOTO

```
10 IF X <= MAX THEN GOTO 40  
20 PRINT "X IS LARGE"  
30 GOTO 50  
40 PRINT "X IS SMALL"  
50 ...
```



```
if x <= max  
    print("x is small")  
else  
    print("x is large")
```

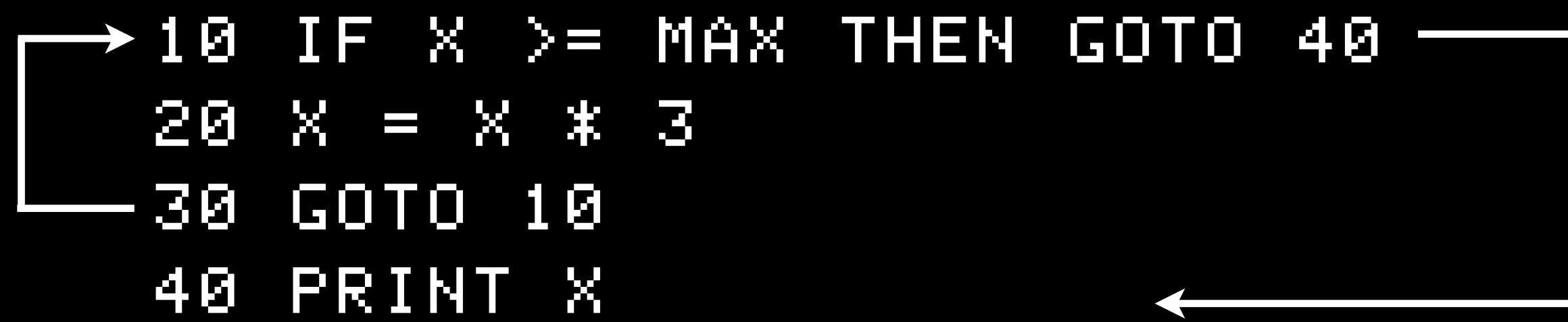
GOTO



```
graph TD; L1[10 IF X >= MAX THEN GOTO 40] --> L2[20 X = X * 3]; L2 --> L3[30 GOTO 10]; L3 --> L4[40 PRINT X]; L4 --> L1;
```

10 IF X >= MAX THEN GOTO 40
20 X = X * 3
30 GOTO 10
40 PRINT X

GOTO



```
graph TD; L1[10 IF X >= MAX THEN GOTO 40] --> L2[20 X = X * 3]; L2 --> L3[30 GOTO 10]; L3 --> L4[40 PRINT X]; L4 --> L1;
```

10 IF X >= MAX THEN GOTO 40
20 X = X * 3
30 GOTO 10
40 PRINT X

```
while x < max  
    x *= 3  
print(x)
```

GOTO

```
10 IF X >= MAX THEN GOTO 40
20 X = X * Y
30 IF Y < X THEN GOTO 10
40 Y = Y / 2
50 IF Y > 1 THEN GOTO 20
60 IF X = Y THEN GOTO 30
70 ...
```

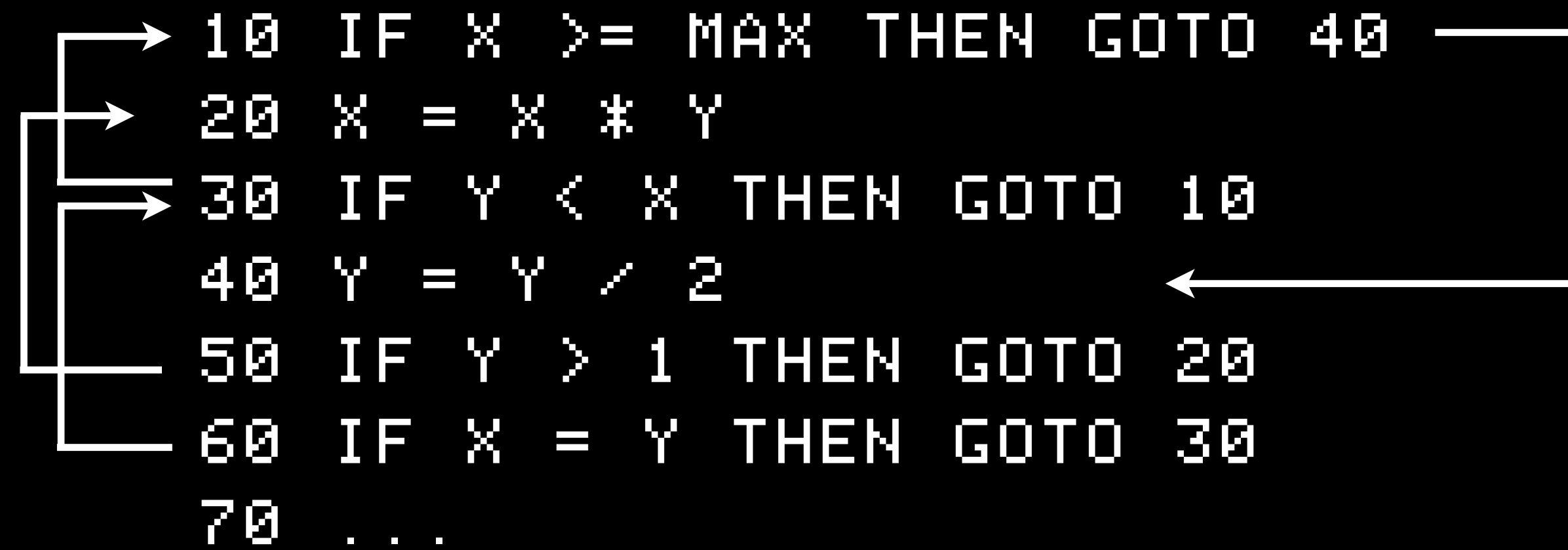
GOTO

```
graph LR; L10[10 IF X >= MAX THEN GOTO 40] --> L40[40 Y = Y / 2]; L30[30 IF Y < X THEN GOTO 10] --> L10; L60[60 IF X = Y THEN GOTO 30] --> L30;
```

```
10 IF X >= MAX THEN GOTO 40  
20 X = X * Y  
30 IF Y < X THEN GOTO 10  
40 Y = Y / 2  
50 IF Y > 1 THEN GOTO 20  
60 IF X = Y THEN GOTO 30  
70 ...
```

GOTO

```
10 IF X >= MAX THEN GOTO 40  
20 X = X * Y  
30 IF Y < X THEN GOTO 10  
40 Y = Y / 2  
50 IF Y > 1 THEN GOTO 20  
60 IF X = Y THEN GOTO 30  
70 ...
```



A flowchart illustrating the execution of the code. It shows a series of jumps between line numbers: from line 10 to 40, from line 30 to 10, from line 50 to 20, and from line 60 to 30. A bracket on the left groups lines 10 through 60, and another bracket on the right groups lines 40 through 60.

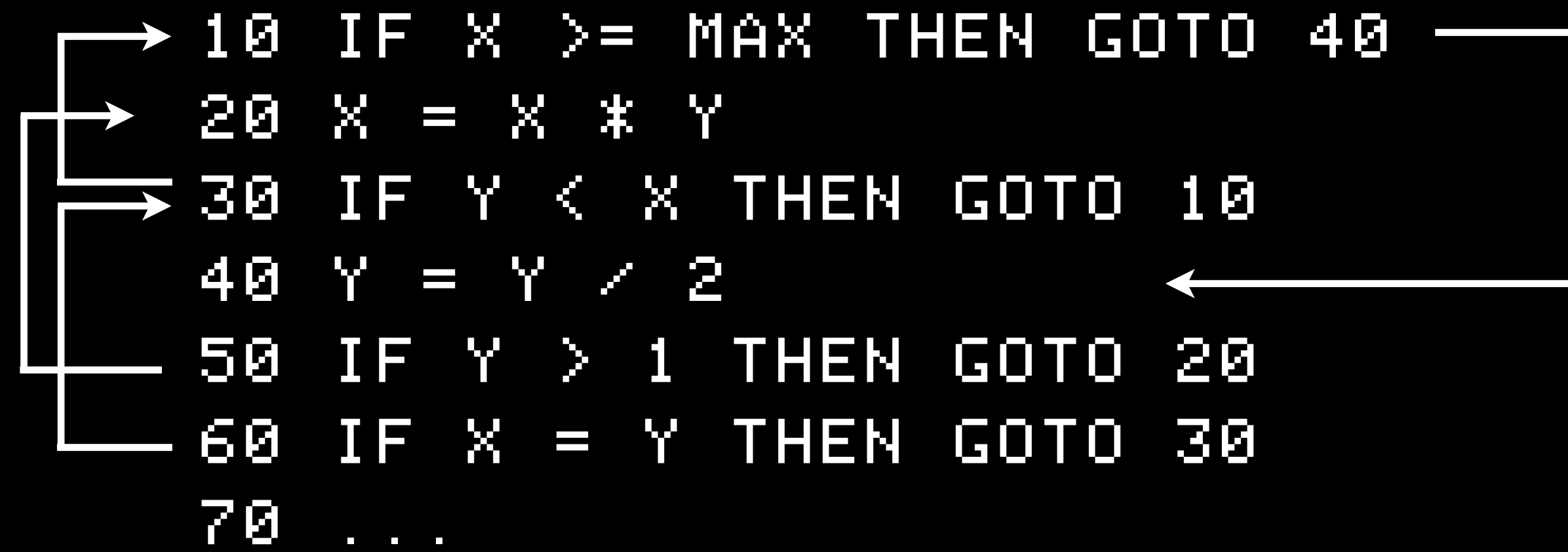
???!?!?!?!!?!!?!!?!!?

Early programming languages

ALGOL (1958–late 60s)

```
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  
end Absmax
```

GOTO



???!?!?!?!!?!?!?!?!!?!?!?!?

GOTO

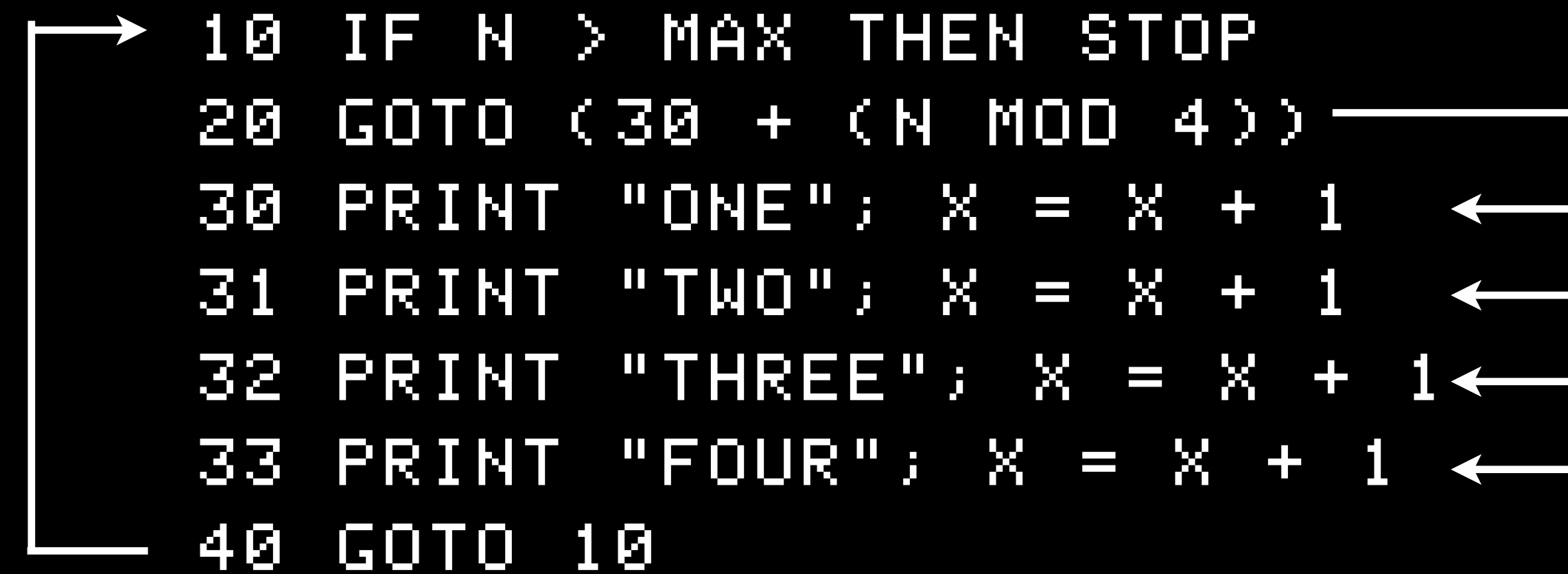
```
10 IF X >= MAX THEN GOTO 40
20 X = X * Y
30 IF Y < X THEN GOTO 10
40 Y = Y / 2
50 IF Y > 1 THEN GOTO 20
60 IF X = Y THEN GOTO 30
70 ...
```

```
skipXCheck = false
skipYCheck = false
while true
    if x < max || skipXCheck
        skipXCheck = false
        x *= y
        if y < x || skipYCheck
            skipYCheck = false
            continue
    y /= 2
    if y > 1
        skipXCheck = true
elif x == y
    skipXCheck = true
    skipYCheck = true
else
    break
```

GOTO

```
10 IF N > MAX THEN STOP
20 GOTO (30 + (N MOD 4))
30 PRINT "ONE"; X = X + 1
31 PRINT "TWO"; X = X + 1
32 PRINT "THREE"; X = X + 1
33 PRINT "FOUR"; X = X + 1
40 GOTO 10
```

GOTO



```
graph TD; L1[10 IF N > MAX THEN STOP] --> L2[20 GOTO (30 + (N MOD 4))]; L2 --> L3[30 PRINT "ONE"; X = X + 1]; L3 --> L4[31 PRINT "TWO"; X = X + 1]; L4 --> L5[32 PRINT "THREE"; X = X + 1]; L5 --> L6[33 PRINT "FOUR"; X = X + 1]; L6 --> L7[40 GOTO 10]; L7 --> L1;
```

```
10 IF N > MAX THEN STOP
20 GOTO (30 + (N MOD 4))
30 PRINT "ONE"; X = X + 1
31 PRINT "TWO"; X = X + 1
32 PRINT "THREE"; X = X + 1
33 PRINT "FOUR"; X = X + 1
40 GOTO 10
```

“Go To Statement Considered Harmful”
Edsger W. Dijkstra, 1968

“Go To Statement Considered Harmful”

Edsger W. Dijkstra, 1968

```
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
```

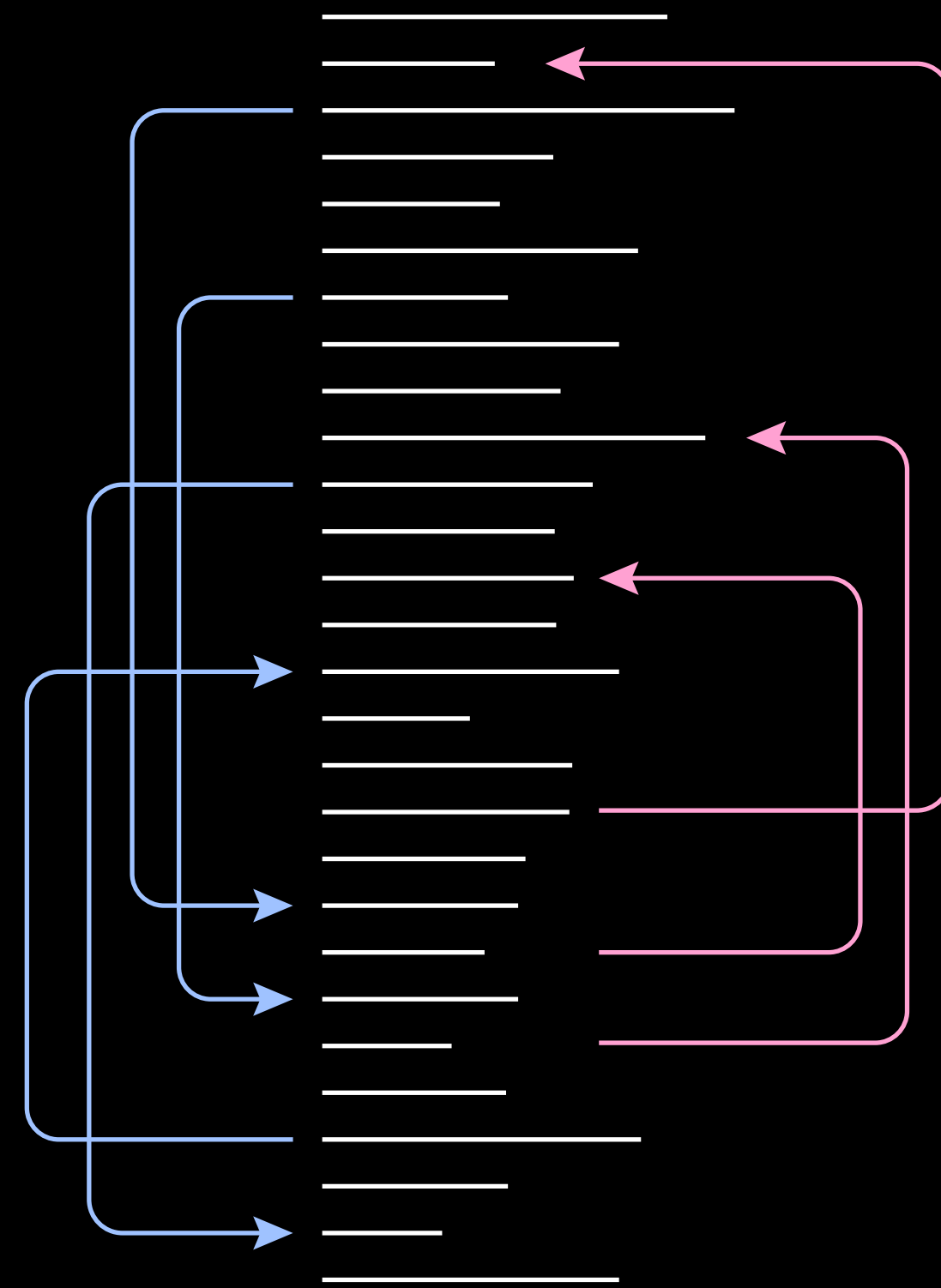
“Go To Statement Considered Harmful”
Edsger W. Dijkstra, 1968

“Go To Statement Considered Harmful”
Edsger W. Dijkstra, 1968

“...we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.”

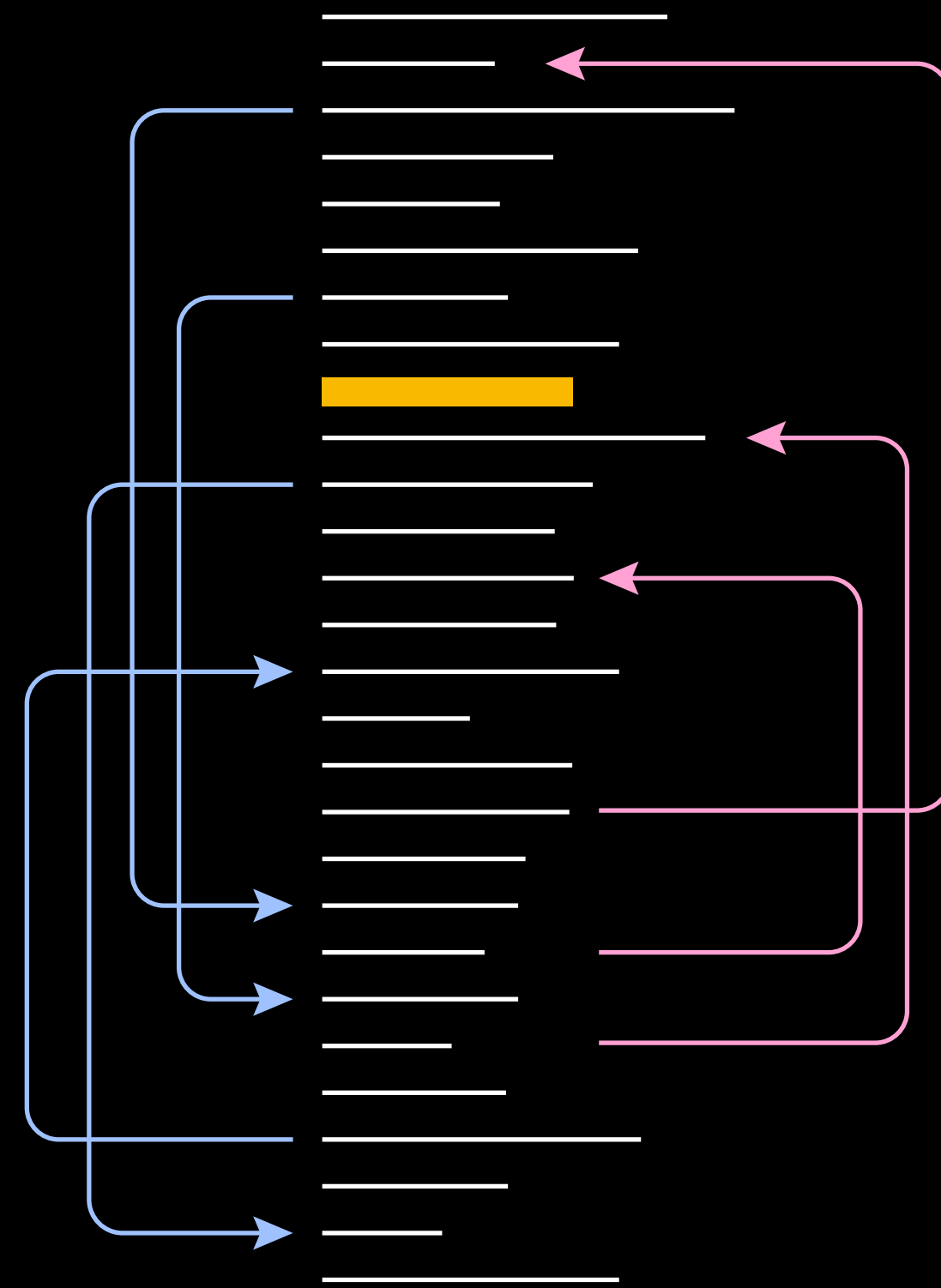
“Go To Statement Considered Harmful”

Edsger W. Dijkstra, 1968



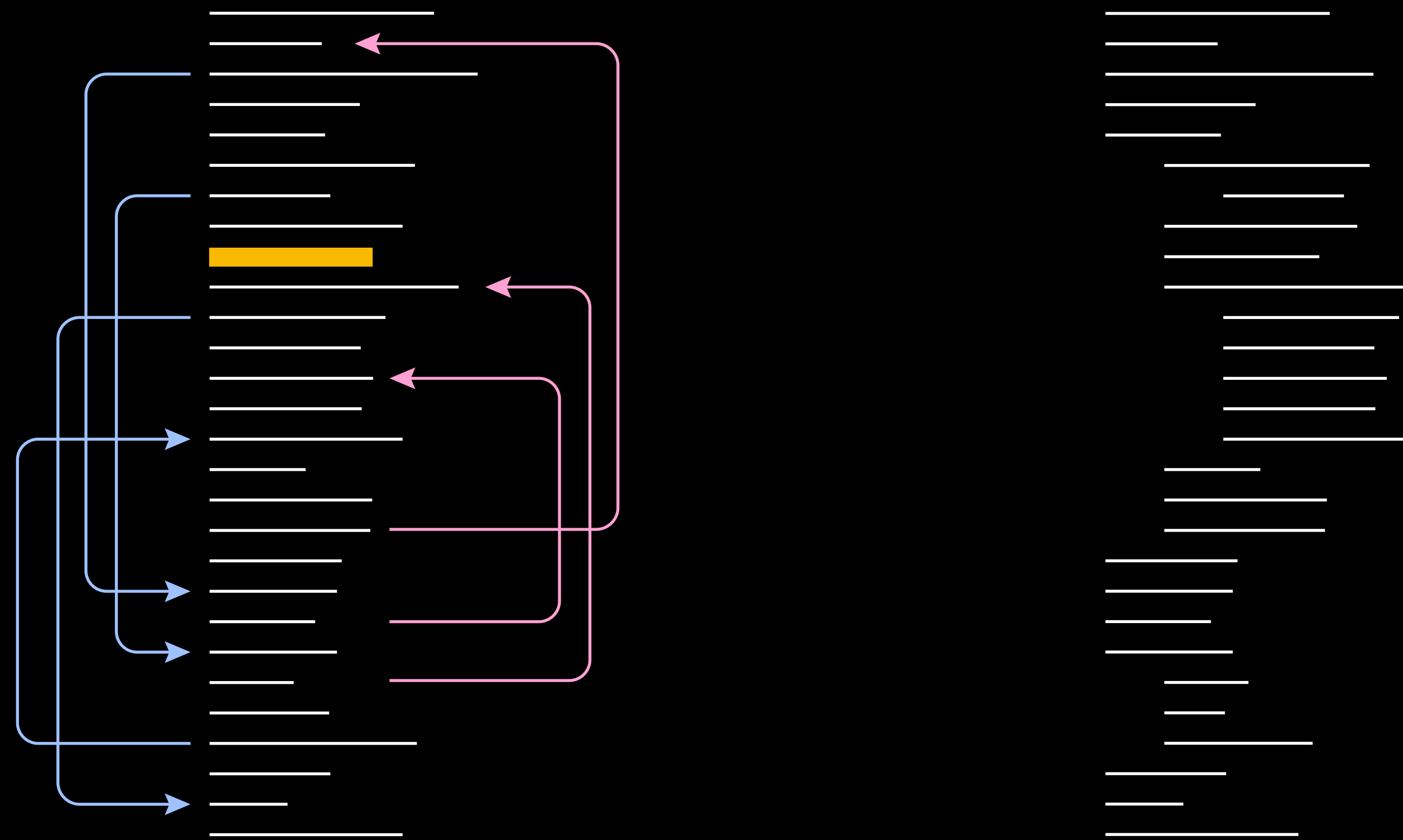
“Go To Statement Considered Harmful”

Edsger W. Dijkstra, 1968



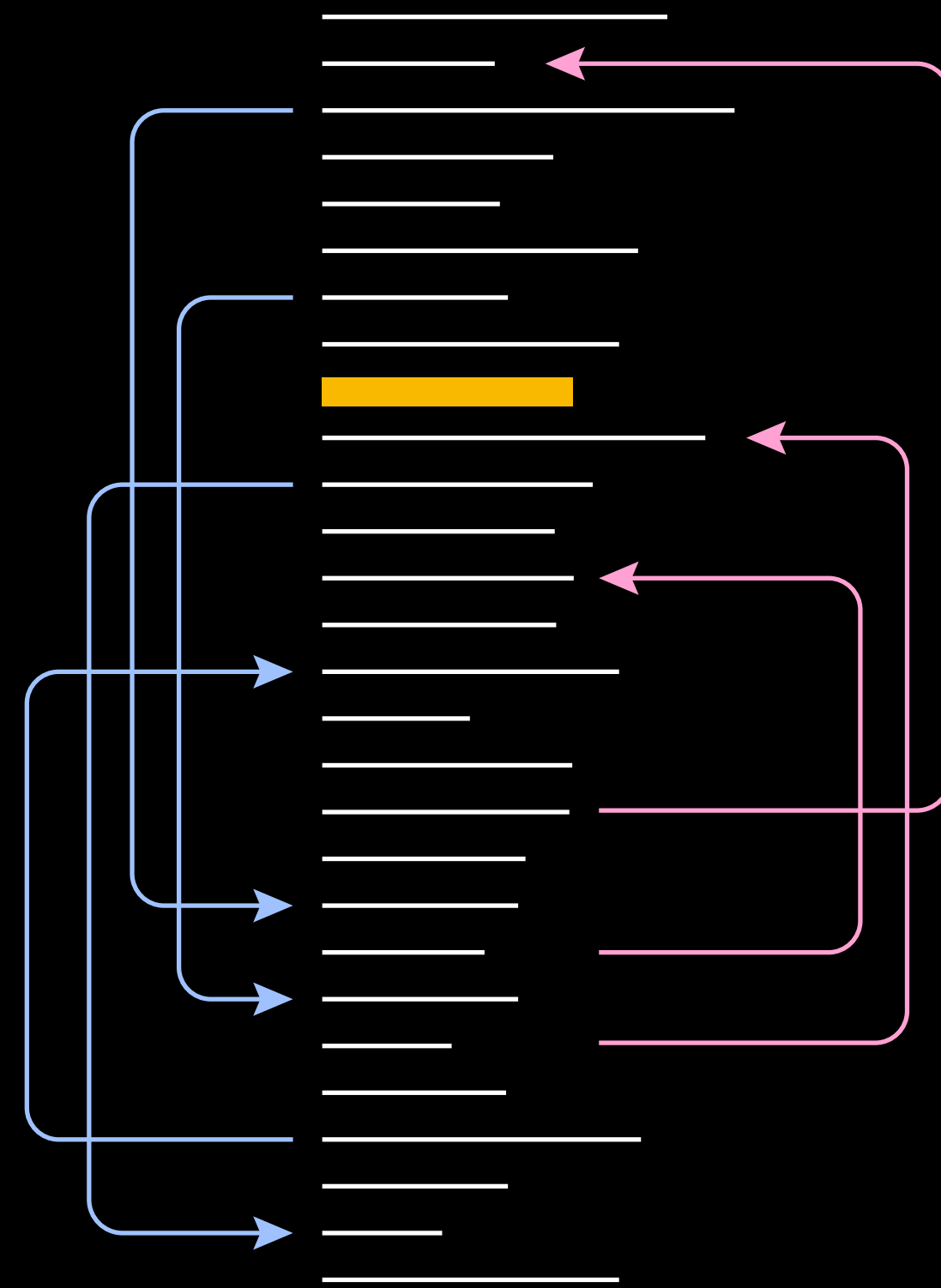
“Go To Statement Considered Harmful”

Edsger W. Dijkstra, 1968



“Go To Statement Considered Harmful”

Edsger W. Dijkstra, 1968



“Go To Statement Considered Harmful”
Edsger W. Dijkstra, 1968

Removing features can make
a programming language
more usable

“Go To Statement Considered Harmful”
Edsger W. Dijkstra, 1968

Removing features can give
a programming language
more capabilities

Big First-Day Lessons

1. Programming languages are made by humans.
2. Programming languages are for humans.
3. It's all tradeoffs.
4. We totally got rid of GOTO.

```
0000000010004eed4      add     x8, x23, x25, lsl #4
0000000010004eed8      ldp     x0, x1, [x8]
0000000010004eedc      cmp     x0, x22
0000000010004eee0      ccmp    x1, x21, #0x0, eq
0000000010004eee4      b.eq    loc_10004eefc
-----
0000000010004eee8      mov     x2, x22
0000000010004eec      mov     x3, x21
0000000010004eef0      mov     w4, #0x0
0000000010004eef4      bl      imp___stubs___$ss27_stringCompareWithSm
0000000010004eef8      tbz     w0, 0x0, loc_10004ef0c
-----
loc_10004eefc:
0000000010004eefc      mov     x0, x19
0000000010004ef00      bl      imp___stubs___swift_bridgeObjectRelease
0000000010004ef04      mov     w0, #0x1
0000000010004ef08      b       loc_10004ef64
-----
loc_10004ef0c:
0000000010004ef0c      mvn     x24, x24
-----
loc_10004ef10:
0000000010004ef10      add     x8, x25, #0x1
0000000010004ef14      and     x25, x8, x24
0000000010004ef18      lsr     x8, x25, #0x3
0000000010004ef1c      and     x8, x8, #0xffffffffffffffff
0000000010004ef20      ldr     x8, [x20, x8]
0000000010004ef24      lsr     x8, x8, x25
0000000010004ef28      tbz     w8, 0x0, loc_10004ef58
-----
0000000010004ef2c      add     x8, x23, x25, lsl #4
0000000010004ef30      ldp     x0, x1, [x8]
0000000010004ef34      cmp     x0, x22
0000000010004ef38      ccmp    x1, x21, #0x0, eq
0000000010004ef3c      b.eq    loc_10004eefc
-----
0000000010004ef40      mov     x2, x22
0000000010004ef44      mov     x3, x21
0000000010004ef48      mov     w4, #0x0
0000000010004ef4c      bl      imp___stubs___$ss27_stringCompareWithSm
0000000010004ef50      tbz     w0, 0x0, loc_10004ef10
-----
0000000010004ef54      b       loc_10004eefc
-----
loc_10004ef58:
0000000010004ef58      mov     x0, x19
0000000010004ef5c      bl      imp___stubs___swift_bridgeObjectRelease
-----
loc_10004ef60:
0000000010004ef60      mov     w0, #0x0
-----
loc_10004ef64:
0000000010004ef64      ldp     fp, lr, [sp, #0x90]
0000000010004ef68      ldp     x20, x19, [sp, #0x80]
0000000010004ef6c      ldp     x22, x21, [sp, #0x70]
0000000010004ef70      ldp     x24, x23, [sp, #0x60]
```

0000000010004ef24	lsr	x8, x8, x25
0000000010004ef28	tbz	w8, 0x0, loc_10004ef58
0000000010004ef2c	add	x8, x23, x25, lsl #4
0000000010004ef30	ldp	x0, x1, [x8]
0000000010004ef34	cmp	x0, x22
0000000010004ef38	ccmp	x1, x21, #0x0, eq
0000000010004ef3c	b.eq	loc_10004eeffc
0000000010004ef40	mov	x2, x22
0000000010004ef44	mov	x3, x21
0000000010004ef48	mov	w4, #0x0
0000000010004ef4c	bl	imp___stubs__\$ss27_stringCo
0000000010004ef50	tbz	w0, 0x0, loc_10004ef10
0000000010004ef54	l	loc_10004ef58

```
    cmp        x0,  
    ccmp      x1,  
    b.eq      loc_
```

```
    mov       x2,
```

```
    mov       x2,
```

comp

b.eq

Big First-Day Lessons

1. Programming languages are made by humans.
2. Programming languages are for humans.
3. It's all tradeoffs.
4. We ~~totally got rid of~~ GOTO.
 [^]
 mostly abstracted away

Why do people create different programming languages?

What characteristics do languages have in common?

What design decisions differentiate programming languages, and what tradeoffs motivate those decisions?

How do languages affect the style of code we write, our development processes, and the ways we think about software?