Peter Naur의 생애와 업적

이원열 (POSTECH)

4회 열린튜링 (2026.05.21, KAIST)

[도움: 2회 열린튜링 자료, 이윤규 학부생 (POSTECH)]

튜링상 (2005)

PETER NAUR

Denmark – 2005

CITATION

For fundamental contributions to programming language design and the definition of Algol 60, to compiler design, and to the art and practice of computer programming.

다음에 대한 근본적 기여로 수상.

- 프로그래밍언어 설계
- ALGOL 60의 정의
- 컴파일러 설계
- 프로그래밍의 기술과 실제

출처: https://amturing.acm.org/award_winners/naur_1024454.cfm

생애

어린 시절 (1928~47)

독일의 덴마크 점령

출처: https://ko.wikipedia.org/wiki/유럽_전구_(제2차_세계_대전)

어린 시절 (1928~47)

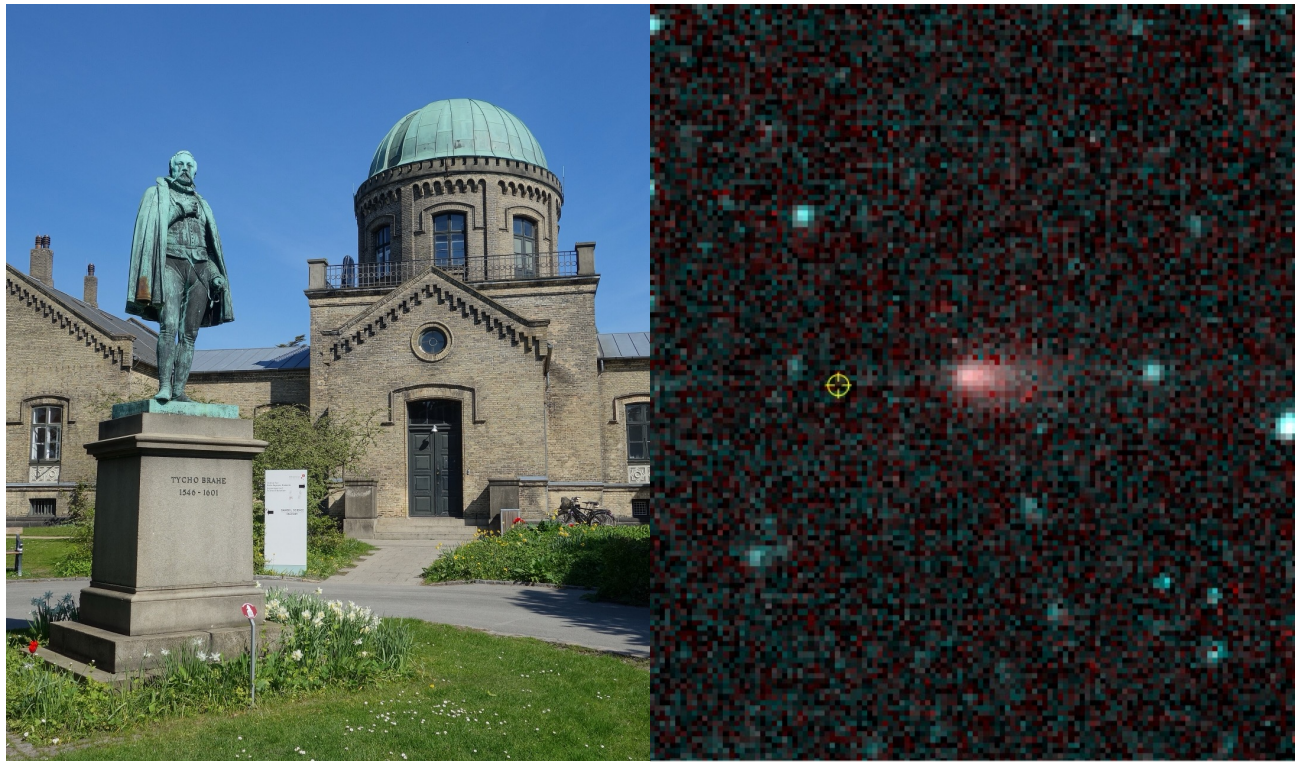
독일의 덴마크 점령

별 보기 좋은 환경

(불빛 없음)

출처: https://ko.wikipedia.org/wiki/유럽_전구_(제2차_세계_대전)

어린 시절 (1928~47)

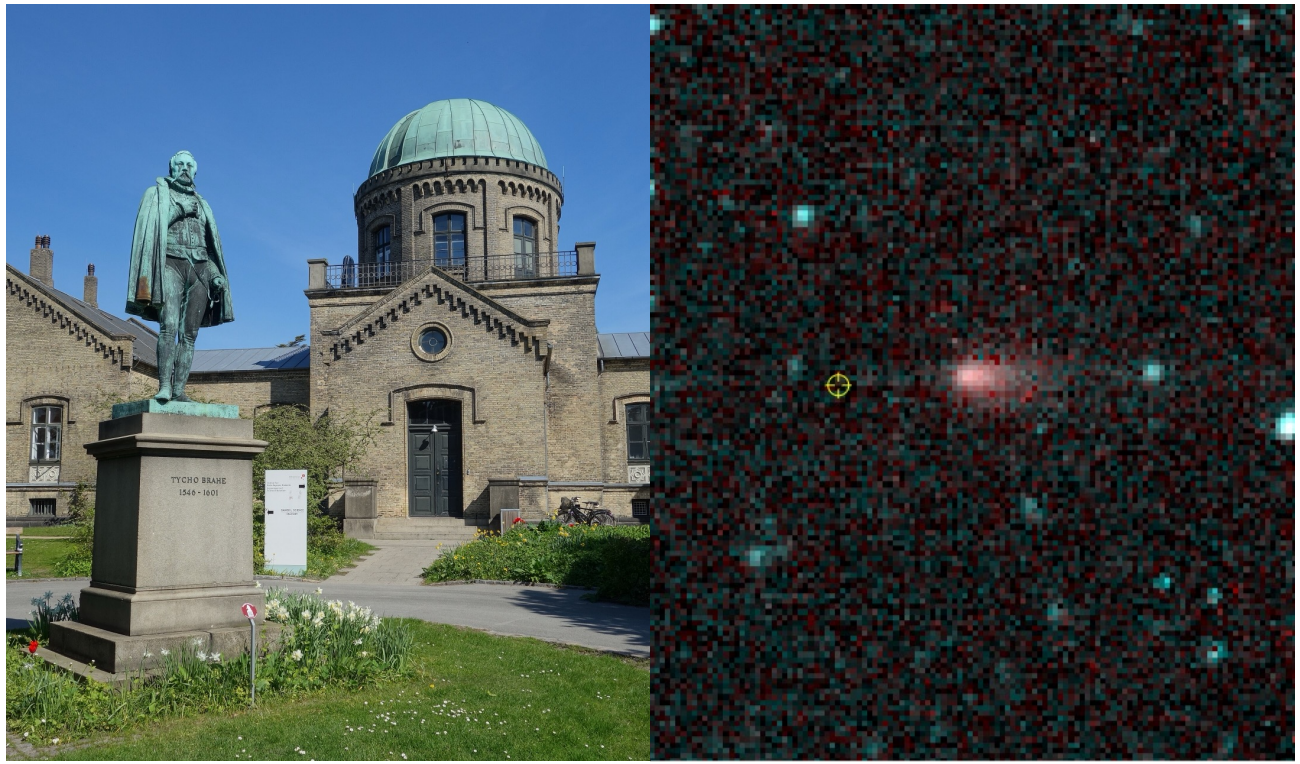


별/행성/혜성 관측

(코펜하겐 대학교 천문대)

출처: https://en.wikipedia.org/wiki/Ostervold_Observatory | https://ui.adsabs.harvard.edu/scan/manifest/1945PCopO.141....1N /

어린 시절 (1928~47)



별/행성/혜성 관측

(코펜하겐 대학교 천문대)

THE ORBIT OF COMET
DU TOIT - NEUJMIN - DELPORTE
(1941 e)

BY

PETER NAUR

No.	t—light time	$\alpha_{1950.0}$	$\delta_{1950.0}$	$\Delta\delta\cos\delta$	$\Delta\delta$	p	
31	Sept. 11.05975	$21^h19^m10^s.24$	$-5°41'52''.9$	$-0^s.42$	$-6\ .2$	3	Yerkes
32	11.81540	20 20 .76	43 20 .1	$+2\ .08$	$+2\ .6$	0	Uccle
33	11.84068	20 22 .30	43 22 .5	$+1\ .34$	$+3\ .3$	0	—
34	12.06600	20 41 .05	43 56 .5	$-0\ .21$	$-2\ .6$	3	Yerkes
35	15.81748	26 19 .57	50 34 .5	$-0\ .80$	$+23\ .1$	0	Uccle

$+19.98787dM_0$	$+41.1398d\mu$	$+0.51493de$	$-0.04044dp$	$+1.00704dq$	$+4.29344dr$	$= +2''.1$
+15.93193	+316.770	+4.03241	−0.45270	+0.94600	+3.53893	= +8 .9
+14.45154	+349.583	+4.44802	−0.52274	+0.87456	+3.26146	= +8 .3
+11.77827	+378.797	+4.80981	−0.60802	+0.72250	+2.75767	= +9 .5
+10.33399	+381.586	+4.83633	−0.63447	+0.62995	+2.48351	= +12 .3

혜성의 궤도 계산

(미분방정식 풀이 = 손 + 계산기)

출처: https://en.wikipedia.org/wiki/Ostervold_Observatory | https://ui.adsabs.harvard.edu/scan/manifest/1945PCopO.141....1N /

대학 시절 (1947~53)

코펜하겐 대학교 / 천문학 학사

(5년 과정 → 2년 졸업)

출처: https://en.wikipedia.org/wiki/University_of_Copenhagen | https://en.wikipedia.org/wiki/King's_College,_Cambridge

대학 시절 (1947~53)

코펜하겐 대학교 / 천문학 학사

(5년 과정 → 2년 졸업)

케임브리지 대학교 / 천문학 연구원

(2년 근무)

출처: https://en.wikipedia.org/wiki/University_of_Copenhagen | https://en.wikipedia.org/wiki/King's_College,_Cambridge

대학 시절 (1947~53)

별 보기 안 좋은 환경

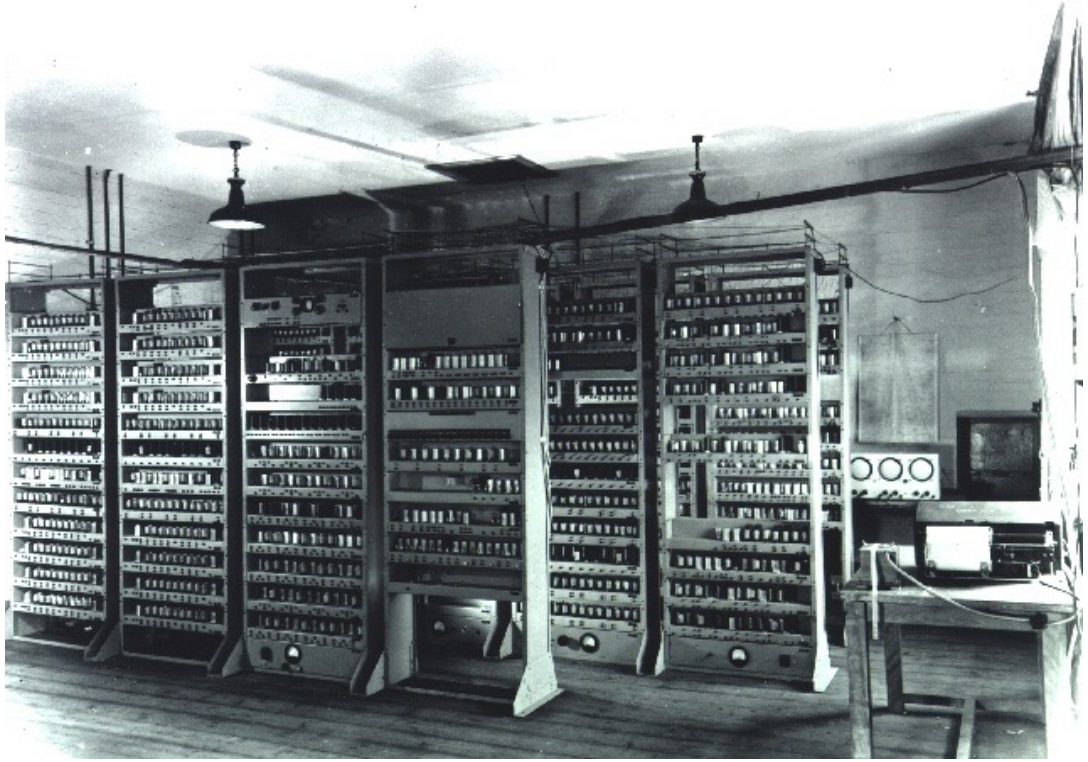
(구름/비 많음)

출처: https://en.wikipedia.org/wiki/Precipitation | https://en.wikipedia.org/wiki/EDSAC

대학 시절 (1947~53)

별 보기 안 좋은 환경

(구름/비 많음)



EDSAC 컴퓨터

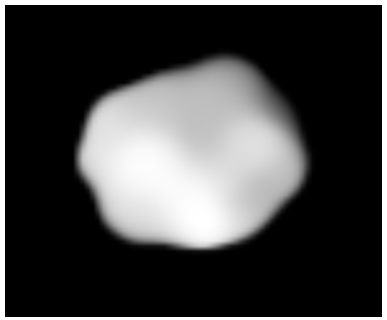
(2시간 고생 → 20초 딸깍)

출처: https://en.wikipedia.org/wiki/Precipitation | https://en.wikipedia.org/wiki/EDSAC

대학원과 이후 (1953~2016)

Minor Planet 51 Nemausa and
the Fundamental System of Declinations

Peter Naur



코펜하겐 대학교 / 천문학 박사

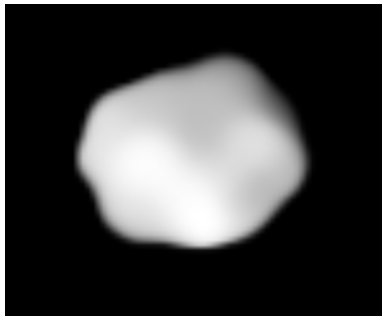
(1953년~1959년)

출처: https://en.wikipedia.org/wiki/51_Nemausa | https://www.danishcomputers.com

대학원과 이후 (1953~2016)

Minor Planet 51 Nemausa and
the Fundamental System of Declinations

Peter Naur



코펜하겐 대학교 / 천문학 박사

(1953년~1959년)

Regnecentralen / 프로그래머

(덴마크 최초의 컴퓨터 회사)

출처: https://en.wikipedia.org/wiki/51_Nemausa | https://www.danishcomputers.com

대학원과 이후 (1953~2016)

PRELIMINARY REPORT—INTERNATIONAL ALGEBRAIC LANGUAGE

국제 수학식 기반 언어

Note:

In the interest of immediate circulation of the ;ee work on an algebraic programming language, this preliminary report is presented. The language described naturally enough represents a compromise, but one based more upon differences of taste than on content or fundamental ideas. Even so, it provides a natural and simple medium for the expression of a large class of algorithms. This report has not been thoroughly examined for errors and inconsistencies. It is anticipated that the committee will prepare a more complete description of the language for publication.

For all scientific purposes, reproduction of this report is explicitly permitted without any charge.

Acknowledgements:

The members of the conference wish to express their appreciation to the Association for Computing Machinery, the "Deutsche Forschungsgemeinshaft" and tl of Technology for substantial help in making this conference and resultant rep

튜링상, 1966

A. J. PERLIS
K. SAMELSON
for the ACM-GAMM Committee

대학원과 이후 (1953~2016)

PRELIMINARY REPORT—INTERNATIONAL ALGEBRAIC LANGUAGE

국제 수학식 기반 언어

Note:

In the interest of immediate circulation of the ;ee work on an algebraic programming language, this preliminary report is presented. The language described naturally enough represents a compromise, but one based more upon differences of taste than on content or fundamental ideas. Even so, it provides a natural and simple medium for the expression of a large class of algorithms. This report has not been thoroughly examined for errors and inconsistencies. It is anticipated that the committee will prepare a more complete description of the language for publication.

For all scientific purposes, reproduction of this report is explicitly permitted without any charge.

Acknowledgements:

The members of the conference wish to express their appreciation to the Association for Computing Machinery, the "Deutsche Forschungsgemeinshaft" and tl of Technology for substantial help in making this conference and resultant rep

튜링상, 1966

A. J. PERLIS
K. SAMELSON
for the ACM-GAMM Committee

대학원과 이후 (1953~2016)

PRELIMINARY REPORT—INTERNATIONAL ALGEBRAIC LANGUAGE

Note:

In the interest of immediate circulation of the 국제 수학식 기반 언어 ee work on an algebraic programming language, this preliminary report is presented. The language described naturally enough represents a compromise, but one based more upon differences of taste than on content or fundamental ideas. Even so, it provides a natural and simple medium for the expression of a large class of algorithms. This report has not been thoroughly examined for errors and inconsistencies. It is anticipated that the committee will prepare a more complete description of the language for publication.

For all scientific purposes, reproduction of this report is explicitly permitted without any charge.

Acknowledgements:

The members of the conference wish to express their appreciation to the Association for Computing Machinery, the "Deutsche Forschungsgemeinshaft" and tl of Technology for substantial help in making this conference and resultant rep

튜링상, 1966

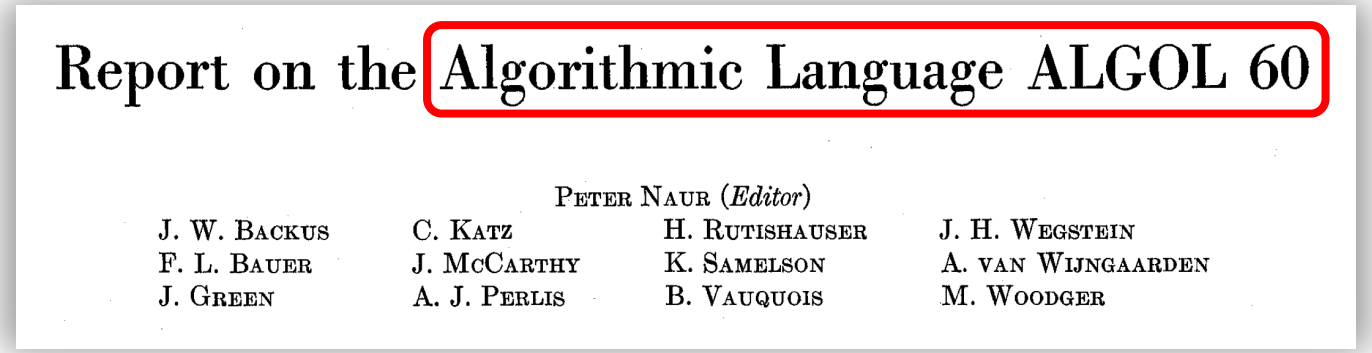
A. J. PERLIS
K. SAMELSON
for the ACM-GAMM Committee

(1) 언어의 설계가 꽤나 복잡한데?

(2) 언어의 정의가 너무 모호한데?

업적 1: 언어 설계

ALGOL 60



Report on the Algorithmic Language ALGOL 60

PETER NAUR (*Editor*)

J. W. BACKUS, F. L. BAUER, J. GREEN, C. KATZ, J. MCCARTHY, A. J. PERLIS, H. RUTISHAUSER, K. SAMELSON, B. VAUQUOIS, J. H. WEGSTEIN, A. VAN WIJNGAARDEN, M. WOODGER

출처: https://dl.acm.org/doi/pdf/10.1145/800025.1198354

ALGOL 60

Report on the Algorithmic Language ALGOL 60

PETER NAUR (*Editor*)

J. W. BACKUS, F. L. BAUER, J. GREEN, C. KATZ, J. MCCARTHY, A. J. PERLIS, H. RUTISHAUSER, K. SAMELSON, B. VAUQUOIS, J. H. WEGSTEIN, A. VAN WIJNGAARDEN, M. WOODGER

출처: https://dl.acm.org/doi/pdf/10.1145/800025.1198354

ALGOL 60

Report on the Algorithmic Language ALGOL 60

PETER NAUR (*Editor*)

J. W. BACKUS, F. L. BAUER, J. GREEN, C. KATZ, J. MCCARTHY, A. J. PERLIS, H. RUTISHAUSER, K. SAMELSON, B. VAUQUOIS, J. H. WEGSTEIN, A. VAN WIJNGAARDEN, M. WOODGER

출처: https://dl.acm.org/doi/pdf/10.1145/800025.1198354

당시 언어

FORTRAN (1956~)

- John Backus (튜링상, 1977) 등이 개발
- Formula Translator

LISP (1958~)

- John McCarthy (튜링상, 1971) 등이 개발
- List Processor

COBOL (1960~)

- Grace Hopper 등이 개발
- Common Business-Oriented Language

• • •

당시 언어

FORTRAN (1956~)
- John Backus (튜링상, 1977) 등이 개발
- Formula Translator

LISP (1958~)
- John McCarthy (튜링상, 1971) 등이 개발
- List Processor

COBOL (1960~)
- Grace Hopper 등이 개발
- Common Business-Oriented Language

• • •

[예제] 자연수 n을 입력 받아서 $n! = n \times (n-1) \times \cdots \times 2 \times 1$을 계산하라.

당시 언어의 문제점

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

- 문법: 이해 어려움
- 구조: 평면 구조

당시 언어의 문제점

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

- 문법: 이해 어려움
- 구조: 평면 구조
- 흐름: GO TO에 의존
- 함수: 자기호출 X

당시 언어의 문제점

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
   10 FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
   20 FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
  100 IF (I - 1) 110, 110, 120
  110 RETURN
  120 FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

- 문법: 이해 어려움
- 구조: 평면 구조
- 흐름: GO TO에 의존
- 함수: 자기호출 X

당시 언어의 문제점

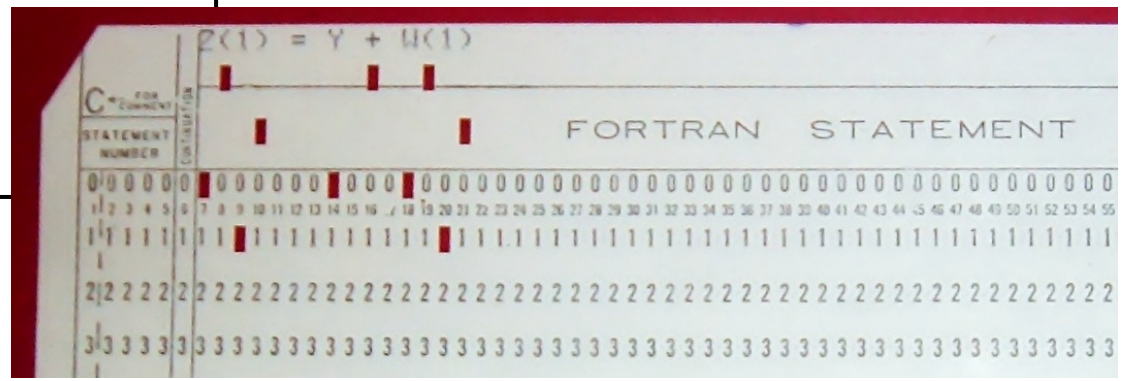
COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
   10 FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
   20 FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
  100 IF (I - 1) 110, 110, 120
  110 RETURN
  120 FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

- 문법: 이해 어려움
- 구조: 평면 구조
- 흐름: GO TO에 의존
- 함수: 자기호출 X



출처: https://en.wikipedia.org/wiki/Punched_card

당시 언어의 문제점

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N         PIC 99.
01  I         PIC 99.
01  RESULT    PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
   10 FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
   20 FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
  100 IF (I - 1) 110, 110, 120
  110 RETURN
  120 FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instr
branch instruction, conditional clause, alternative clause,
itive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

"GO TO 구문은 해롭다."

How do we tell truths that might hurt?

Prof. Dr. Edsger W. Dijkstra
Burroughs Research Fellow
18th June 1975

"COBOL은 정신을 망가뜨린다.
이를 가르치는 일은 범죄행위다."

Edsger Dijkstra (튜링상, 1972)

ALGOL 60의 혁신

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
   10 FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
   20 FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
  100 IF (I - 1) 110, 110, 120
  110 RETURN
  120 FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

ALGOL 60

```
begin
  integer procedure fact(n);
    integer n;
  begin
    if n = 0 then
      fact := 1
    else
      fact := n * fact(n - 1)
  end;

  integer n, result;
  ininteger(0, n);
  result := fact(n);
  outinteger(1, result)
end
```

ALGOL 60의 혁신

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
 10   FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
 20   FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
100   IF (I - 1) 110, 110, 120
110   RETURN
120   FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

ALGOL 60

```
begin
  integer procedure fact(n);
    integer n;
  begin
    if n = 0 then
      fact := 1
    else
      fact := n * fact(n - 1)
  end;

  integer n, result;
  ininteger(0, n);
  result := fact(n);
  outinteger(1, result)
end
```

- 문법: 이해 쉬움
- 구조: 중첩블록 구조

  (begin … end)

ALGOL 60의 혁신

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
 10   FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
 20   FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
100   IF (I - 1) 110, 110, 120
110   RETURN
120   FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

ALGOL 60

```
begin
  integer procedure fact(n);
    integer n;
  begin
    if n = 0 then
      fact := 1
    else
      fact := n * fact(n - 1)
  end;

  integer n, result;
  ininteger(0, n);
  result := fact(n);
  outinteger(1, result)
end
```

- 문법: 이해 쉬움
- 구조: 중첩블록 구조
- 흐름: if … then … else 등
- 함수: 자기호출 O

ALGOL 60의 혁신

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT-PROG.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  N       PIC 99.
01  I       PIC 99.
01  RESULT  PIC 9(7).

PROCEDURE DIVISION.
MAIN.
  ACCEPT N.
  GO TO FACT.
MAIN-RETURN.
  DISPLAY RESULT.
  STOP RUN.

FACT.
  MOVE 1 to RESULT.
  MOVE N to I.
FACT-LOOP.
  IF I = 0 GO TO MAIN-RETURN.
  MULTIPLY I BY RESULT
  SUBTRACT 1 FROM I
  GO TO FACT-LOOP.
```

FORTRAN

```
      READ 10, N
   10 FORMAT(I3)
      IRESULT = FACT(N)
      PRINT 20, IRESULT
   20 FORMAT(I8)
      STOP
      END

      FUNCTION FACT(N)
      FACT = 1
      I = N
  100 IF (I - 1) 110, 110, 120
  110 RETURN
  120 FACT = FACT * I
      I = I - 1
      GO TO 100
      END
```

Pascal, C, C++, Java 등

- Niklaus Wirth (튜링상, 1984)
- Dennis Ritchie (튜링상, 1983)

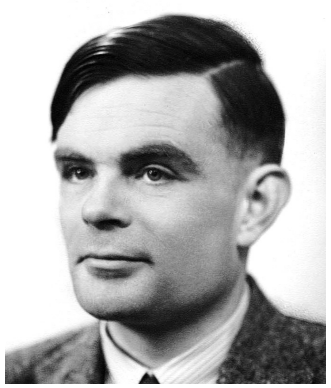
ALGOL 60

```
begin
  integer procedure fact(n);
    integer n;
  begin
    if n = 0 then
      fact := 1
    else
      fact := n * fact(n - 1)
  end;

  integer n, result;
  ininteger(0, n);
  result := fact(n);
  outinteger(1, result)
end
```

```
def fact(n: int) -> int:
  if n == 0:
    return 1
  else:
    return n * fact(n - 1)

n = int(input())
result = fact(n)
print(result)
```

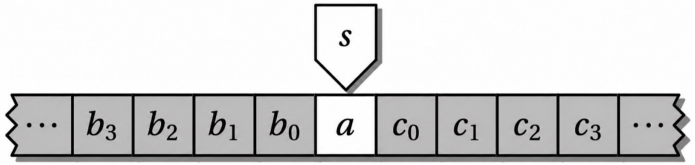
Python 등

튜링의 유산



앨런 튜링

- 컴퓨터 = 튜링 기계

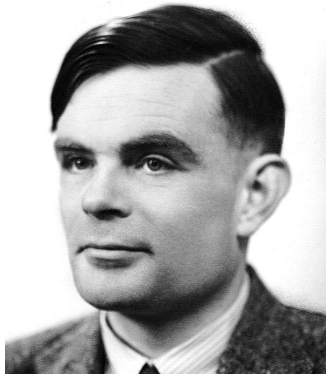


- 상태를 바꿔 가며 계산 = 명령 중심 언어

ALGOL 60 → C, C++, Java, …

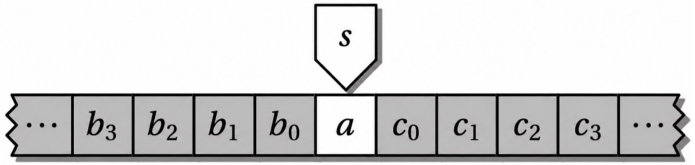
출처: https://en.wikipedia.org/wiki/Alan_Turing | https://en.wikipedia.org/wiki/Alonzo_Church

튜링의 유산



앨런 튜링

- 컴퓨터 = 튜링 기계



- 상태를 바꿔 가며 계산 = 명령 중심 언어

ALGOL 60 → C, C++, Java, …

알론조 처치
(튜링의 지도교수)

- 컴퓨터 = 람다 계산법 $\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$
- 식을 줄여 가며 계산 = 함수 중심 언어

LISP → Haskell, OCaml, Lean, …

출처: https://en.wikipedia.org/wiki/Alan_Turing | https://en.wikipedia.org/wiki/Alonzo_Church

업적 2: 언어 정의

언어 문법의 정의

ALGOL 60

```
begin
  integer procedure fact(n);
    integer n;
  begin
    if n = 0 then
      fact := 1
    else
      fact := n * fact(n - 1)
  end;

  integer n, result;
  ininteger(0, n);
  result := fact(n);
  outinteger(1, result)
end
```

문법을 엄밀하게
정의하고 싶은데...

언어 문법의 정의

ALGOL 60

```
begin
  integer procedure fact(n);
    integer n;
  begin
    if n = 0 then
      fact := 1
    else
      fact := n * fact(n - 1)
  end;

  integer n, result;
  ininteger(0, n);
  result := fact(n);
  outinteger(1, result)
end
```


문법을 엄밀하게
정의하고 싶은데...

COBOL

COMPREHENSIVE RULES FOR FORMING ALGEBRAIC EXPRESSIONS

Arithmetic expressions may contain field names, constants, and literals joined by arithmetic operators. Subexpressions may be contained in parentheses as required. The rules for forming arithmetic expressions are as follows:

2. The ways in which symbol pairs may be formed are summarized in the following table:

		Second Symbol				
		Variable	+-*/**	Negation	(	)
FIRST SYMBOL	Variable	-	P	-	-	P
	+-*/**	P	-	-	P	-
	Negation	P	-	-	P	-
	(	P	-	P	P	-
	)	-	P	-	-	P

Where "P" indicates a permissible symbol pair, and "-" indicates a pair which is not permitted. Thus, "*(" is permissible, while "(*" is not.

3. When the hierarchy of operations in an expression is not completely specified by parentheses, the order of operations (working from inside to outside) is assumed to be exponentiation, then multiplication and division and finally, addition and subtraction. Thus, the expression A+B/C+D**E*F - G will be taken to mean A+(B/C)+(D^{E}·F) - G.

자연어로 모호하게 정의

예제

[예제] 자연수와 +, ×로 구성된 수식의 집합을 엄밀히 정의하라.

- 원하는 문자열: `0+001`, `4×3`, `8×10×2+5+1×9`, …
- 원하지 않는 문자열: `+1`, `3×`, `5+×10`, …

예제

[예제] 자연수와 +, ×로 구성된 수식의 집합을 엄밀히 정의하라.

- 원하는 문자열: `0+001`, `4×3`, `8×10×2+5+1×9`, …
- 원하지 않는 문자열: `+1`, `3×`, `5+×10`, …

[관찰] 위에 기술한 수식은 무한히 많다.

- 따라서 단순 나열로는 정의할 수 없다.
- 무한한 것을 유한하게 표현하는 방법은?

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

위 정의의 해석.

- <숫자>가 가능한 문자열: 0, 1, 2, …, 9

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`
- <자연수> ::= <숫자>

위 정의의 해석.

- <숫자>가 가능한 문자열: `0`, `1`, `2`, …, `9`
- <자연수>가 가능한 문자열: `0`, `1`, `2`, …, `9`

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <자연수> ::= <숫자> | <자연수><숫자>

위 정의의 해석.

- <숫자>가 가능한 문자열: 0, 1, 2, …, 9
- <자연수>가 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <자연수> ::= <숫자> | <자연수><숫자>
- <수식> ::= <자연수>

위 정의의 해석.

- <숫자>가 가능한 문자열: 0, 1, 2, …, 9
- <자연수>가 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …
- <수식>이 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …

아이디어

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <자연수> ::= <숫자> | <자연수><숫자>
- <수식> ::= <자연수> | <수식>+<수식> | <수식>×<수식>

위 정의의 해석.

- <숫자>가 가능한 문자열: 0, 1, 2, …, 9
- <자연수>가 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …
- <수식>이 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …,
  03+285, 03+285×2, …, 28×9, 28×9×037, …

Backus-Naur 표기법

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <자연수> ::= <숫자> | <자연수><숫자>
- <수식> ::= <자연수> | <수식>+<수식> | <수식>×<수식>

← Backus-Naur 표기법 (BNF)

= 문법을 엄밀히 정의하는 메타언어

위 정의의 해석.

- <숫자>가 가능한 문자열: 0, 1, 2, …, 9
- <자연수>가 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …
- <수식>이 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …,
  03+285, 03+285×2, …, 28×9, 28×9×037, …

Backus-Naur 표기법

[해결책] 귀납적 정의 = 바닥부터 쌓아올리기. (수식 ← 자연수 ← 숫자)

- <숫자> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <자연수> ::= <숫자> | <자연수><숫자>
- <수식> ::= <자연수> | <수식>+<수식> | <수식>×<수식>

← Backus-Naur 표기법 (BNF)

= 문법을 엄밀히 정의하는 메타언어

위 정의의 해석.

- <숫자>가 가능한 문자열: 0, 1, 2, …, 9
- <자연수>가 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …
- <수식>이 가능한 문자열: 0, 1, 2, …, 9, 03, 037, …, 28, 285, …,
  03+285, 03+285×2, …, 28×9, 28×9×037, …

수학적으로 엄밀히 정의 가능

[최소고정점 (least fixed point) 사용]

Backus-Naur 표기법

⟨digit⟩ ::= 0|1|2|3|4|5|6|7|8|9

⟨unsigned integer⟩ ::= ⟨digit⟩|⟨unsigned integer⟩⟨digit⟩
⟨integer⟩ ::= ⟨unsigned integer⟩|+⟨unsigned integer⟩|
−⟨unsigned integer⟩

⟨adding operator⟩ ::= +|−
⟨multiplying operator⟩ ::= ×|/|÷
⟨primary⟩ ::= ⟨unsigned number⟩|⟨variable⟩|
⟨function designator⟩|(⟨arithmetic expression⟩)
⟨factor⟩ ::= ⟨primary⟩|⟨factor⟩↑⟨primary⟩
⟨term⟩ ::= ⟨factor⟩|⟨term⟩⟨multiplying opreator⟩⟨factor⟩
⟨simple arithmetic expression⟩ ::= ⟨term⟩|
⟨adding operator⟩⟨term⟩|⟨simple arithmetic expression⟩
⟨adding operator⟩⟨term⟩
⟨if clause⟩ ::= if ⟨Boolean expression⟩then
⟨conditional statement⟩

⟨compound tail⟩ ::= ⟨statement⟩ **end** |⟨statement⟩ ;
⟨compound tail⟩
⟨block head⟩ ::= **begin**⟨declaration⟩|⟨block head⟩ ;
⟨declaration⟩
⟨unlabelled compound⟩ ::= **begin** ⟨compound tail⟩
⟨unlabelled block⟩ ::= ⟨block head⟩ ; ⟨compound tail⟩
⟨compound statement⟩ ::= ⟨unlabelled compound⟩|
⟨label⟩:⟨compound statement⟩
abelled block⟩|⟨label⟩:⟨block⟩

BNF로 정의한
ALGOL 60의 문법

Backus-Naur 표기법

```
⟨digit⟩ ::= 0|1|2|3|4|5|6|7|8|9

⟨unsigned integer⟩ ::= ⟨digit⟩|⟨unsigned integer⟩⟨digit⟩
⟨integer⟩ ::= ⟨unsigned integer⟩|+⟨unsigned integer⟩|
    −⟨unsigned integer⟩

⟨adding operator⟩ ::= +|−
⟨multiplying operator⟩ ::= ×|/|÷
⟨primary⟩ ::= ⟨unsigned number⟩|⟨variable⟩|
    ⟨function designator⟩|(⟨arithmetic expression⟩)
⟨factor⟩ ::= ⟨primary⟩|⟨factor⟩↑⟨primary⟩
⟨term⟩ ::= ⟨factor⟩|⟨term⟩⟨multiplying opreator⟩⟨factor⟩
⟨simple arithmetic expression⟩ ::= ⟨term⟩|
    ⟨adding operator⟩⟨term⟩|⟨simple arithmetic expression⟩
    ⟨adding operator⟩⟨term⟩
⟨if clause⟩ ::= if ⟨Boolean expression⟩then
    ⟨conditional statement⟩
⟨compound tail⟩ ::= ⟨statement⟩ end |⟨statement⟩ ;
    ⟨compound tail⟩
⟨block head⟩ ::= begin⟨declaration⟩|⟨block head⟩ ;
    ⟨declaration⟩
⟨unlabelled compound⟩ ::= begin ⟨compound tail⟩
⟨unlabelled block⟩ ::= ⟨block head⟩ ; ⟨compound tail⟩
⟨compound statement⟩ ::= ⟨unlabelled compound⟩|
    ⟨label⟩:⟨compound statement⟩
    abelled block⟩|⟨label⟩:⟨block⟩
```

BNF로 정의한
ALGOL 60의 문법

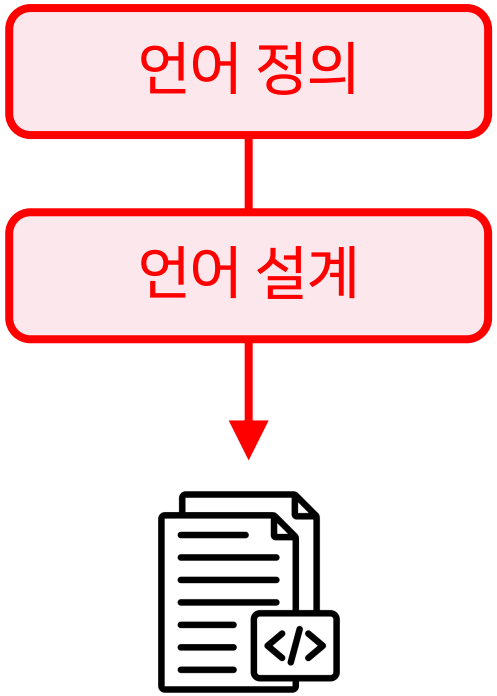
```
statements: statement+

statement: compound_stmt  | simple_stmts

statement_newline:
    | compound_stmt NEWLINE
    | simple_stmts
    | NEWLINE
    | ENDMARKER

simple_stmts:
    | simple_stmt !';' NEWLINE  # Not needed, there for speedup
    | ';'.simple_stmt+ [';'] NEWLINE

# NOTE: assignment MUST precede expression, else parsing a simple assignment
# will throw a SyntaxError.
simple_stmt:
    | assignment
    | type_alias
    | star_expressions
```

$$
\begin{array}{rrcl}
\textit{Real expressions} & E & ::= & c \mid x \mid f(E, \ldots, E) \\
\textit{Boolean expressions} & B & ::= & \text{true} \mid E < E \mid B \wedge B \mid \neg B \\
\textit{String expressions} & S & ::= & \alpha \mid g(S, \ldots, S, E, \ldots, E) \\
\textit{Commands} & C & ::= & \text{skip} \mid x := E \mid C; C \mid \text{if } B \{C\} \text{ else } \{C\} \mid \text{while } B \{C\} \\
 & & & \mid x := \text{sample}_{\text{norm}}(S, E, E) \mid \text{score}_{\text{norm}}(E, E, E)
\end{array}
$$

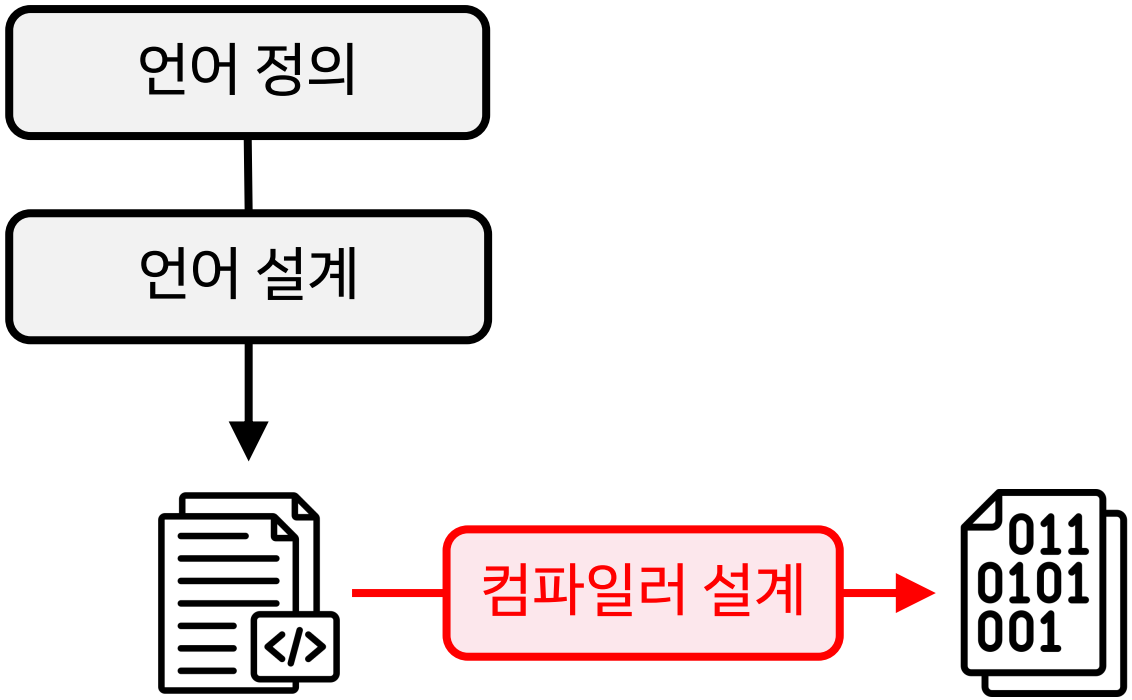
BNF로 정의한
Python / 발표자가 연구한 언어의 문법

그 밖의 업적

업적 1~2



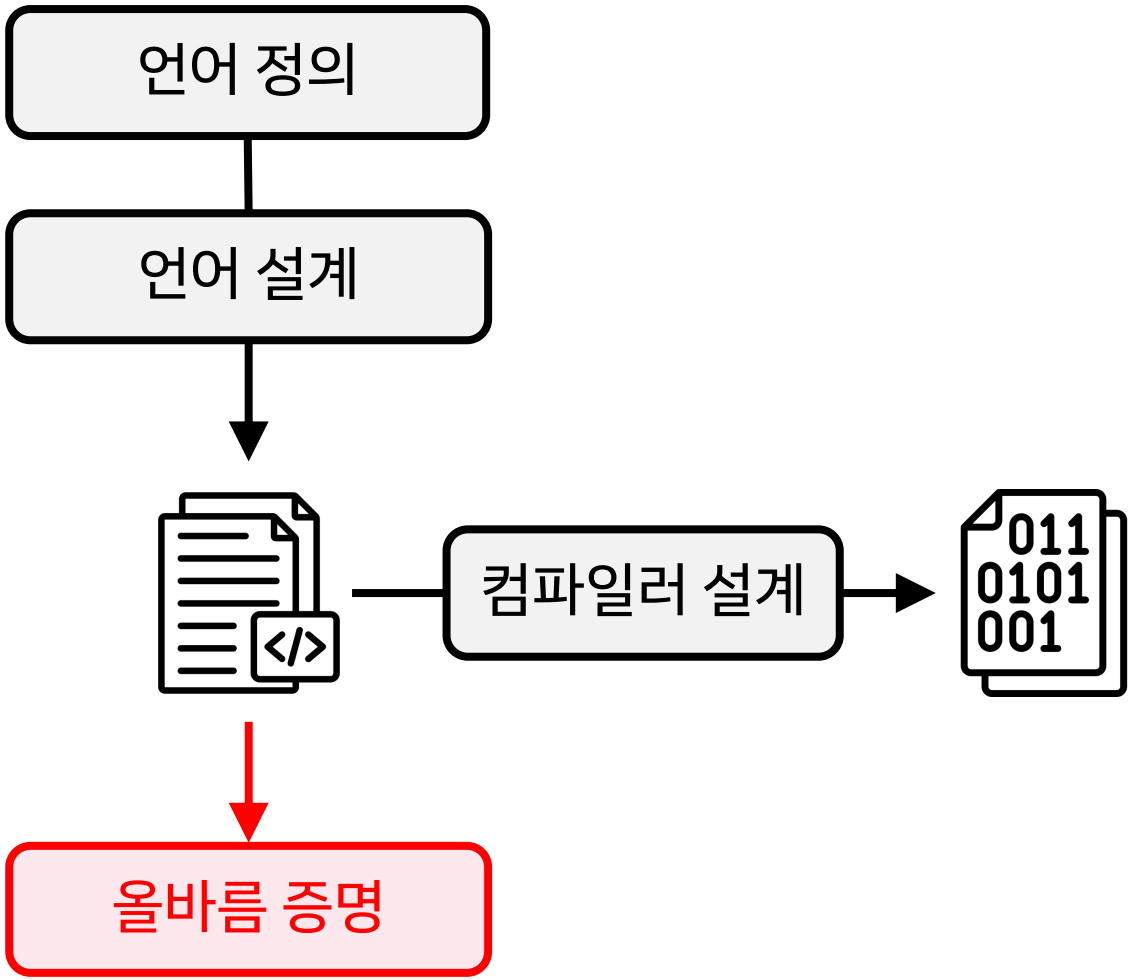
업적 3: 컴파일러 설계



Peter Naur의 업적

- ALGOL 60의 복잡한 특징을 올바르게 번역.
- 컴파일러를 체계적으로 설계 (주먹구구 X).

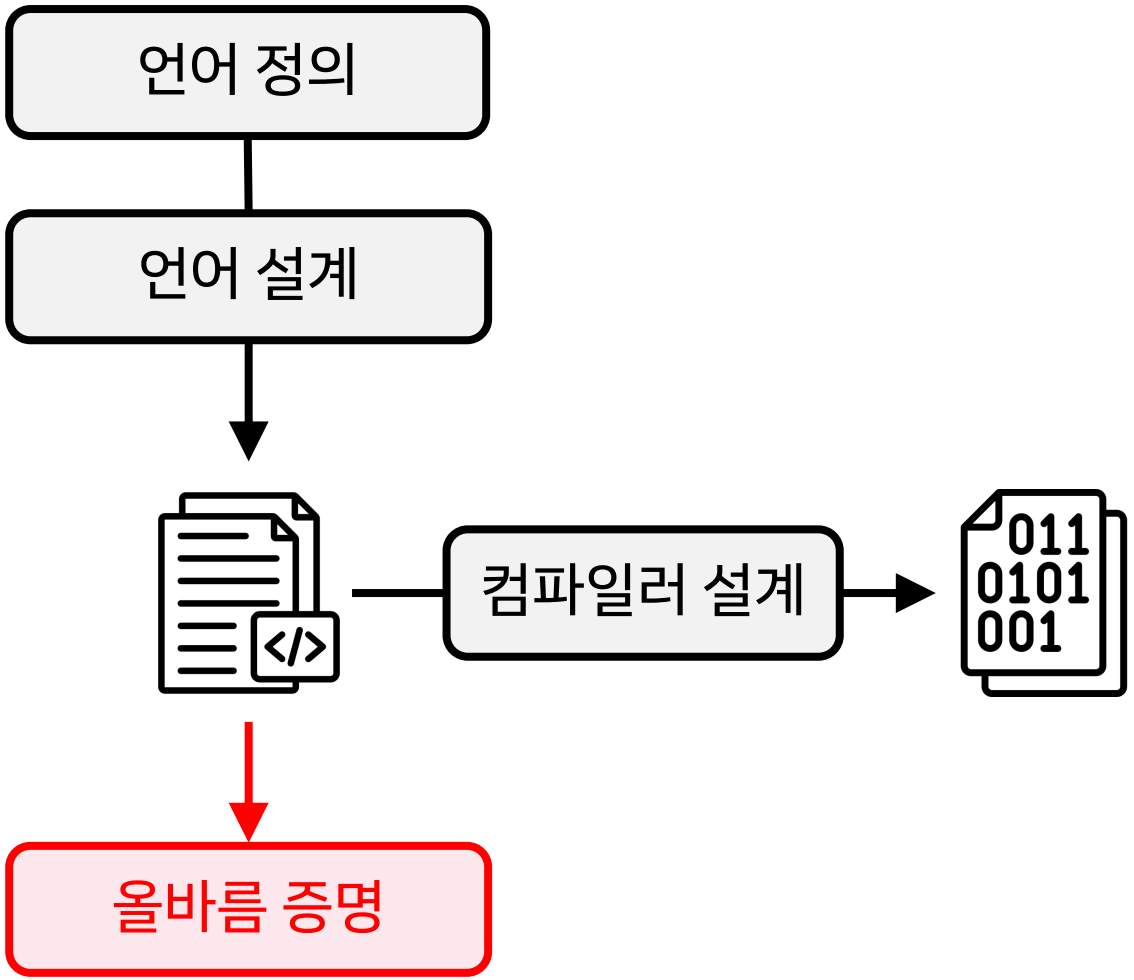
업적 4: 올바름 증명



Peter Naur의 업적

- 필수조건 (assertion)과
- 불변조건 (invariant)을 제안.

업적 4: 올바름 증명



Peter Naur의 업적

- 필수조건 (assertion)과
- 불변조건 (invariant)을 제안.

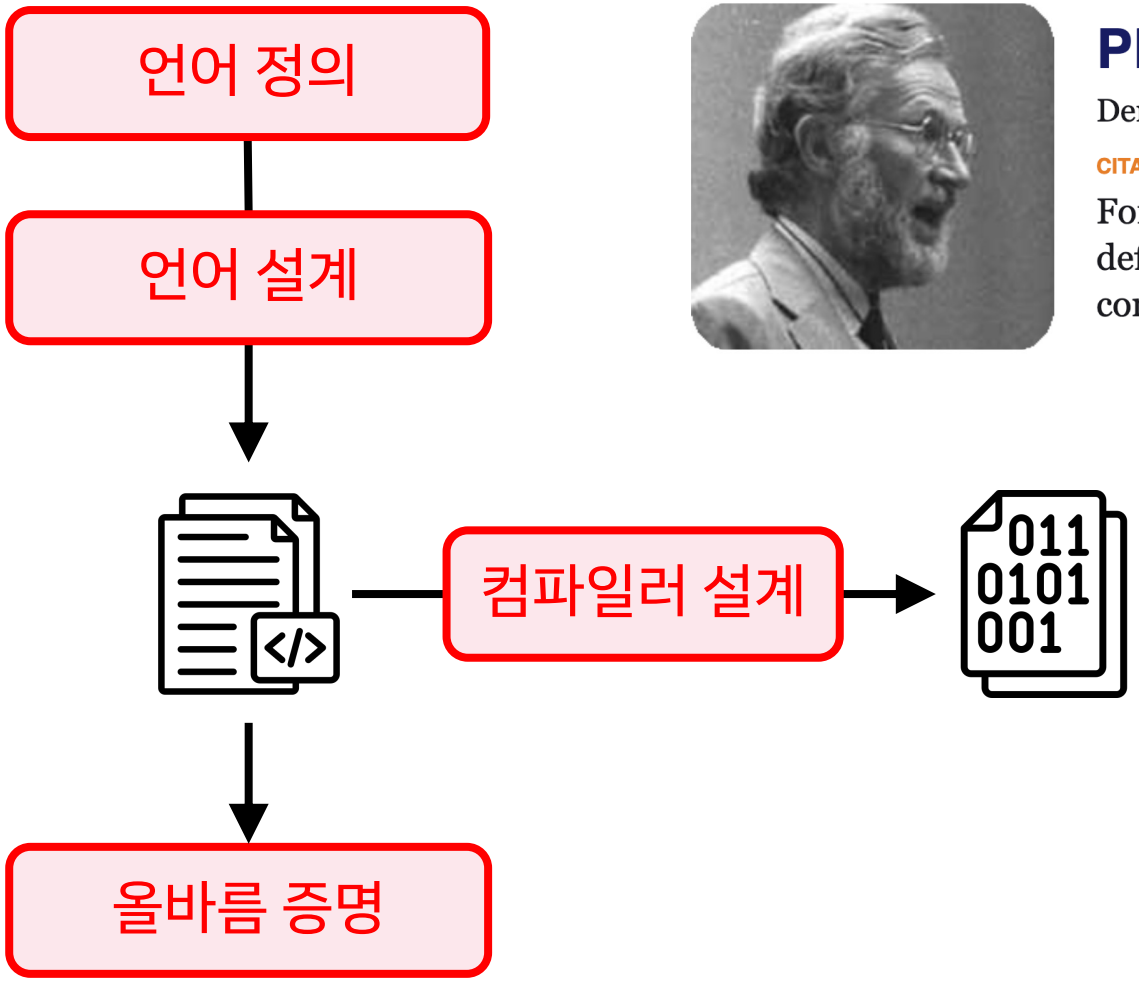
순차 프로그램

- Edsger Dijkstra (튜링상, 1972)
- Robert Floyd (튜링상, 1978)
- Tony Hoare (튜링상, 1980) 등

동시/분산 프로그램

- Amir Pnueli (튜링상, 1996)
- Edmund Clarke, et al. (튜링상, 2007)
- Leslie Lamport (튜링상, 2013) 등

정리 및 영향



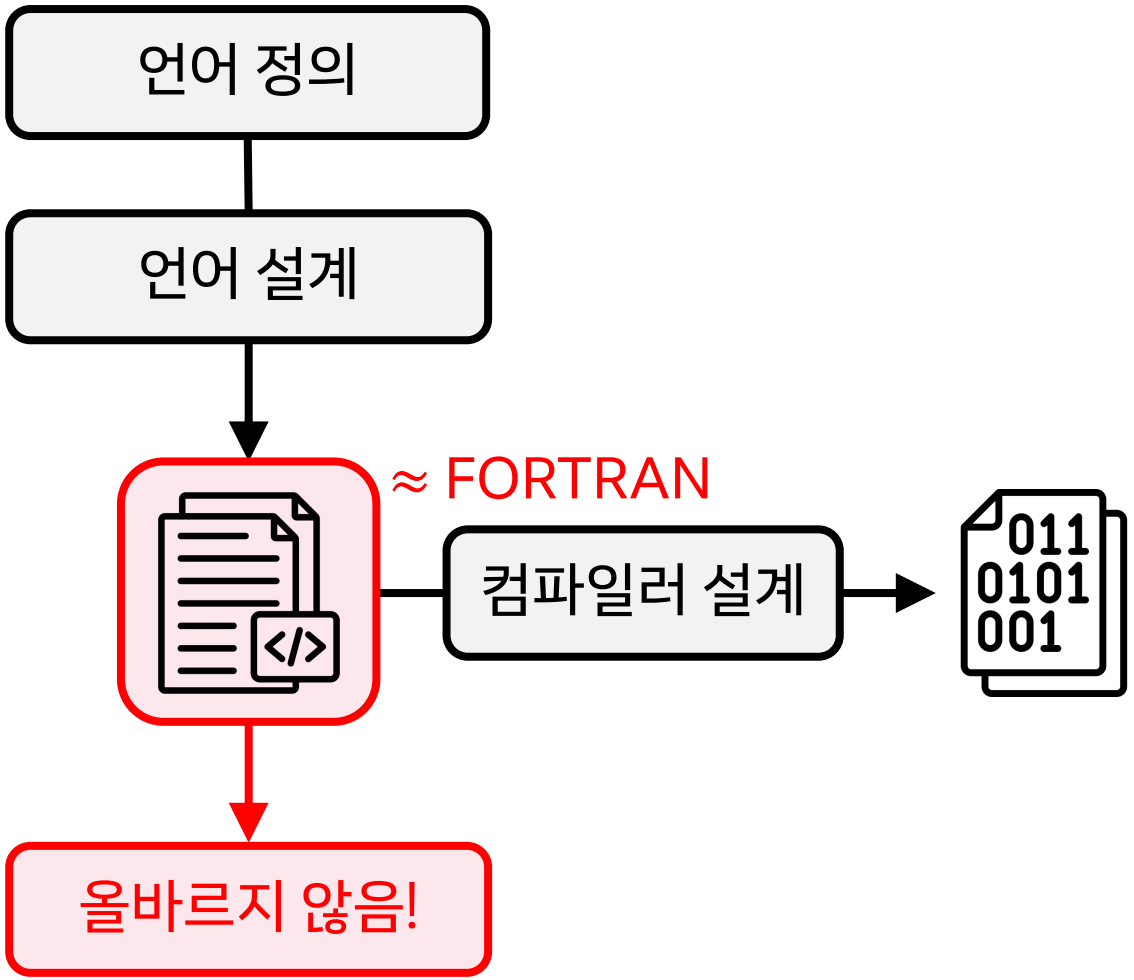
PETER NAUR

Denmark – 2005

CITATION

For fundamental contributions to programming language design and the definition of Algol 60, to compiler design, and to the art and practice of computer programming.

정리 및 영향

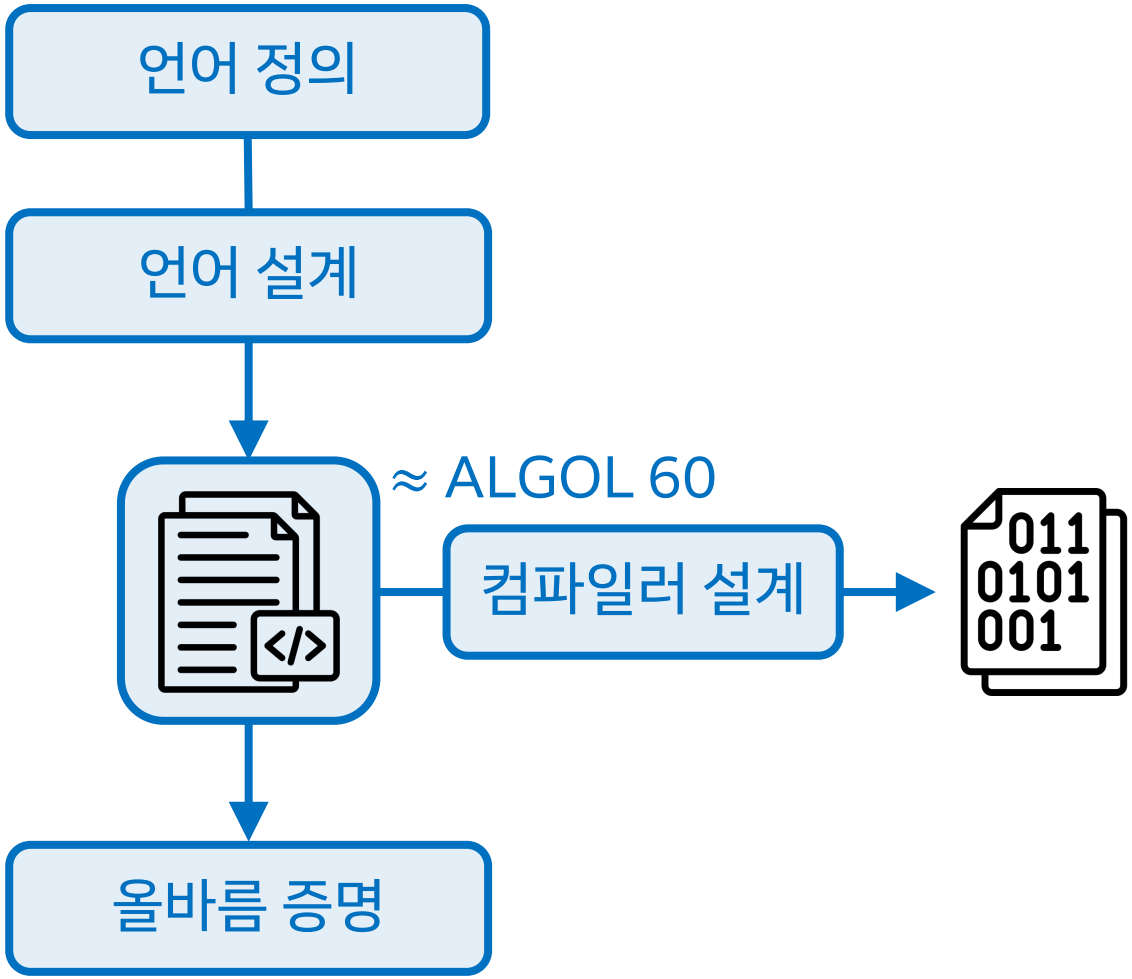


수학함수(sin 등)를 계산하는 라이브러리

- C/기계어로 작성
- 수만~수십만 줄 규모
- 다양한 오류가 계속 나옴
- 수학/암호학/보안 등에서 중요함

```
맞는 결과:   cos(4.8...e+9) = -0.3...
실제 결과:   cos(4.8...e+9) = +0.3...

맞는 결과:  acos(7.4...e-1) = +0.7...
실제 결과:  acos(7.4...e-1) = +1.4...
```

정리 및 영향



수학함수(sin 등)를 계산하는 라이브러리

- C/기계어로 작성
- 수만~수십만 줄 규모
- 다양한 오류가 계속 나옴
- 수학/암호학/보안 등에서 중요함

발표자의 최근 연구

감사합니다