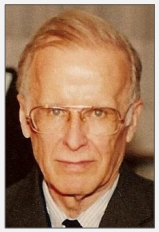
Applications

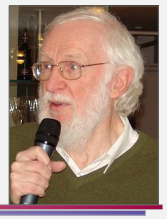
- aspects of real languages (natural languages, programming languages) can be expressed with context-free grammars
  - provides a precise definition of legal syntax
  - provides an algorithm for parsing
- *Backus-Naur Form* (BNF) is a notation typically used in these applications
  - there are variations

Backus and Naur

- John Backus, 1924-2007
  - American computer scientist
  - also known for Fortran (1950s)
    - first widely-used high-level programming language
  - received the 1977 Turing Award for "profound, influential, and lasting contributions to the design of practical high-level programming systems"



- Peter Naur, 1928-2016
  - Danish computer scientist
  - also known for ALGOL 60 (1960)
    - introduced many influential features (block structure, nested functions, lexical scope)
  - received the 2005 Turing Award for work on ALGOL 60



BNF

- non-terminals typically have meaningful names rather than being single symbols
  - written ⟨*thing*⟩ to distinguish from terminals
- terminals are the elements of the language
  - also typically multi-symbol units
- uses ::= instead of →
- offers a more compact representation for related rules

⟨*digit*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 — | denotes alternatives

⟨*declaration*⟩ ::= ⟨*type*⟩ ⟨*variable*⟩ [= ⟨*expression*⟩] ; — [ ] denotes optional

⟨*integer*⟩ ::= ⟨*digit*⟩ [⟨*digit*⟩]... — [ ]... denotes 0 or more repetitions

  - parens used for grouping

⟨*sentence*⟩ ::= ⟨*simple-sentence*⟩ [and ⟨*simple-sentence*⟩]...
⟨*simple-sentence*⟩ ::= ⟨*nout-part*⟩ ⟨*verb-part*⟩
⟨*noun-part*⟩ ::= ⟨*article*⟩ ⟨*noun*⟩ [who ⟨*verb-part*⟩]...
⟨*verb-part*⟩ ::= ⟨*intransitive-verb*⟩ | (⟨*transitive-verb*⟩ ⟨*noun-part*⟩)
⟨*article*⟩ ::= the | a
⟨*noun*⟩ ::= man | woman | dog | cat | computer
⟨*intransitive-verb*⟩ ::= runs | jumps | hides
⟨*transitive-verb*⟩ ::= knows | loves | chases | owns

⟨*sentence*⟩ ⟹ ⟨*simple-sentence*⟩
⟹ ⟨*noun-part*⟩ ⟨*verb-part*⟩
⟹ ⟨*article*⟩ ⟨*noun*⟩ ⟨*verb-part*⟩
⟹ the ⟨*noun*⟩ ⟨*verb-part*⟩
⟹ the man ⟨*verb-part*⟩
⟹ the man ⟨*transitive-verb*⟩ ⟨*noun-part*⟩
⟹ the man loves ⟨*noun-part*⟩
⟹ the man loves ⟨*article*⟩ ⟨*noun*⟩ who ⟨*verb-part*⟩
⟹ the man loves a ⟨*noun*⟩ who ⟨*verb-part*⟩
⟹ the man loves a woman who ⟨*verb-part*⟩
⟹ the man loves a woman who ⟨*intransitive-verb*⟩
⟹ the man loves a woman who runs

⟨*statement*⟩ ::= ⟨*block-statement*⟩ | ⟨*if-statement*⟩ | ⟨*while-statement*⟩
| ⟨*assignment-statement*⟩ | ⟨*null-statement*⟩
⟨*block-statement*⟩ ::= { [⟨*statement*⟩]... }
⟨*if-statement*⟩ ::= if "(" ⟨*condition*⟩ ")" ⟨*statement*⟩ [else ⟨*statement*⟩]

⟨*while-statement*⟩ ::= while "(" ⟨*condition*⟩ ")" ⟨*statement*⟩
⟨*assignment-statement*⟩ ::= ⟨*variable*⟩ = ⟨*expression*⟩ ;
⟨*null-statement*⟩ ::= ε

⟨*expression*⟩ ::= ⟨*term*⟩ [[+ | −] ⟨*term*⟩]...
⟨*term*⟩ ::= ⟨*factor*⟩ [[∗ | /] ⟨*factor*⟩]...
⟨*factor*⟩ ::= ***ident*** | ***number*** | "(" ⟨*expression*⟩ ")"

- quotes ("") are being used here to distinguish terminals [,], (,) in the language from the BNF notation [,], (,)
- **`ident`** refers to an identifier, **`number`** refers to a number

2. Rewrite the example BNF grammar for a subset of English as a context-free grammar.

⟨*sentence*⟩ ::= ⟨*simple-sentence*⟩ [and ⟨*simple-sentence*⟩]...
⟨*simple-sentence*⟩ ::= ⟨*nout-part*⟩ ⟨*verb-part*⟩
⟨*noun-part*⟩ ::= ⟨*article*⟩ ⟨*noun*⟩ [who ⟨*verb-part*⟩]...
⟨*verb-part*⟩ ::= ⟨*intransitive-verb*⟩ | (⟨*transitive-verb*⟩ ⟨*noun-part*⟩)
⟨*article*⟩ ::= the | a
⟨*noun*⟩ ::= man | woman | dog | cat | computer
⟨*intransitive-verb*⟩ ::= runs | jumps | hides
⟨*transitive-verb*⟩ ::= knows | loves | chases | owns

5. Variable references in the Java programming language can be rather complicated. Some examples include: x, $list.next$, $A[7]$, $a.b.c$, $S[i+1].grid[r][c].red$, ... Write a BNF production rule for Java variables. You can use the token ***ident*** and the non-terminal ⟨*expression*⟩ in your rule.

6. Use BNF to express the syntax of the try...catch statement in the Java programming language.

- write this BNF grammar using the standard context-free grammar notation

E ::= T [$+$ T]...
T ::= F [$*$ F]...
F ::= "(" E ")" | x | y | z

3. Write a single BNF production rule that is equivalent to the following context-free grammar:

$$S \longrightarrow aSa$$
$$S \longrightarrow bB$$
$$B \longrightarrow bB$$
$$B \longrightarrow \varepsilon$$