BNF

Metalanguages

- A *metalanguage* is a language used to talk about a language (usually a different one)
- We can use English as its own metalanguage (e.g. describing English grammar in English)
- It is essential to distinguish between the metalanguage terms and the object language terms

<name> <group>

<vitoon> <myGroup> --> vitoon myGroup

BNF

- BNF stands for either Backus-Naur Form or Backus Normal Form
- BNF is a metalanguage used to describe the grammar of a programming language
- BNF is formal and precise
  - BNF is a notation for context-free grammars
- BNF is essential in compiler construction
- There are many dialects of BNF in use, but…
- …the differences are almost always minor

BNF

- < > indicate a *nonterminal* that needs to be further expanded, e.g. <variable>
- Symbols not enclosed in < > are *terminals;* they represent themselves, e.g. if, while, (
- The symbol ::= means *is defined as*
- The symbol | means *or;* it separates alternatives, e.g. <addop> ::= + | -

- This is *all there is* to "plain" BNF; but we will discuss *extended* BNF (EBNF) later in this lecture

BNF uses recursion

- <integer> ::= <digit> | <integer> <digit>

  or

  <integer> ::= <digit> | <digit> <integer>

- Recursion is all that is needed (at least, in a formal sense)
- "Extended BNF" allows repetition as well as recursion
- Repetition is usually better when using BNF to construct a compiler

BNF Examples I

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

- <if statement> ::=
  if (<condition>) <statement>
  | if (<condition>) <statement>
  else <statement>

BNF Examples II

- <unsigned integer> ::=

  <digit> | <unsigned integer> <digit>

- <integer> ::=

  <unsigned integer>

  | + <unsigned integer>

  | - <unsigned integer>

BNF Examples III

- <identifier> ::=
  ```
        <letter>
      | <identifier> <letter>
      | <identifier> <digit>
  ```
- <block> ::= { <statement list> }
- <statement list> ::=
  ```
        <statement>
      | <statement list> <statement>
  ```

BNF Examples IV

-
  ```
  <statement> ::=
        <block>
      | <assignment statement>
      | <break statement>
      | <continue statement>
      | <do statement>
      | <for loop>
      | <goto statement>
      | <if statement>
      | . . .
  ```

Extended BNF

- The following are pretty standard:
  - [ ] enclose an optional part of the rule
    - Example:

      ```
      <if statement> ::=
              if ( <condition> ) <statement> [ else <statement> ]
      ```

  - { } mean the enclosed can be repeated any number of times (including zero)
    - Example:

      ```
      <parameter list> ::= ( )
                        |  ( { <parameter> , } <parameter> )
      ```

Variations

- The preceding notation is the original and most common notation
  - BNF was designed before we had boldface, color, more than one font, etc.
  - A typical modern variation might:
  - Use boldface to indicate multi-character terminals
  - Quote single-character terminals (because boldface isn't so obvious in this case)
- Example:
  - *if_statement* ::=
    if "(" *condition* ")" *statement* [**else** *statement*]

Limitations of BNF

- No easy way to impose length limitations, such as maximum length of variable names
- No easy way to describe ranges, such as 1 to 31
- No way *at all* to impose distributed requirements, such as, a variable must be declared before it is used
- Describes ***only syntax, not semantics***
- Nothing clearly better has been devised

The End

http://cuiwww.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html