compiler: a compiler is a translator which takes as input a program written in a high-level programming language (the **source**) and transforms it into an equivalent functional program written in another language (the **target** language)

basic functions of a compiler: (input → tokens) ← * by char, * ints

- **lexical analysis:** input statements are broken into syntactical units or tokens
- **parsing:** each token is checked within its context to verify its structural correctness ← (verify tokens)
- **code generation:** produce the assembly or machine code necessary to run on the host computer

precompiler: a precompiler is a translator whose input is a program written in an "extended" version of a high-level programming language and transform it into an equivalent functional program written in the standard version of the high-level programming language.

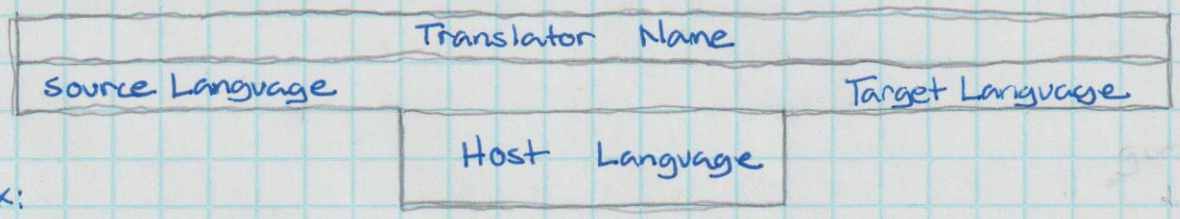
* precompilers are useful to extend a language with internal organizational conventions that facilitate enforcing company or government standards
* the basic function of a precompiler are generally limited to lexical analysis and parsing. they also perform some string substitutions

a compiler is characterized by three languages:

* the source language (what it accepts as input)
* the target language (what is produced)
* the translating or host language (the language in which it is written)

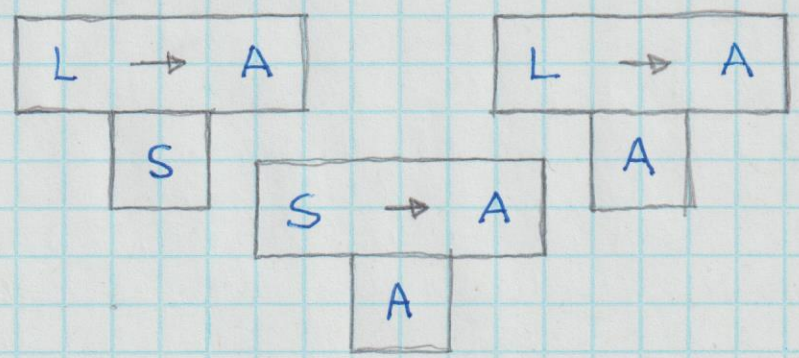
if the compiler is written in its own source language and the compiler is a program, who compiled the first compiler?

* we will use T-diagrams to illustrate the process of bootstrapping



ex:

L language
S subset of language
A assembly



30.08.2021 cms.395.1.91061 [introduction to grammar and languages]

- alphabet: any nonempty finite set of symbols, $\Sigma \neq \emptyset$, $|\Sigma| = n$
- strings: a sequence of elements from a given alphabet
- terminal symbols: certain indivisible symbols (like, begin, end, while) are also elements of the alphabet
  * formally a string can be written as:
    $x = a_1 a_2 \ldots a_k$ where $k \geq 0$, $a_i \in \Sigma$ where $1 \leq i \leq k$
  * the length of a string is often written as: $|x|$, sometimes as: $lg(x)$
- length: number of characters in a string (appearances)
- empty string: unique string with no symbols (denoted by: ε)
- sigma plus: Σ^+ the set of all strings of length one or more consisting of members of Σ (does not contain the empty string ε)
- sigma star: Σ^* the set of all strings of length zero or more consisting of members of Σ (contains the empty string ε)
  * $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ ← set notation
- concatenation: connection/combination of two strings
  - implicit operation (no special symbol)
  - concatenation is associative, $x(yz) = (xy)z$
  - empty string is invisable, $\varepsilon x = x$, $y = \varepsilon y$
  - Σ^* has left and right cancellation, $\begin{cases} zx = zy \rightarrow x = y \\ xz = yz \rightarrow x = y \end{cases}$
  - $|xy| = |x| + |y|$

01.09.2021 cms.395.1.91061 [introduction to grammar and language]

- meta (mathmatical) symbols: symbols used to explain language (not part of the language)
- substring: given a string 'w' over an alphabet 'Σ' - any part of the string 'w'
- prefix: 'x' of "xy" (ex: 1010ε, 1010ε, 1010)
- suffix: 'y' of "xy" (ex: ε1010, ε1010, 1010)
- phrase-structure grammar: a mathematical system for defining a language
  - a grammar is defined as a 4-tuple: $G = (N, \Sigma, P, S)$ where:
    - N: a finite nonempty set of nonterminal symbols
    - Σ: a finite nonempty set of terminal symbols or tokens
    - $N \cap \Sigma = \emptyset$, N and Σ do not have common elements
    - P: production rules
    - S: starting symbol

- expression → expression + term
- expression → term
- term → term * factor
- term → factor
- factor → (expression)
- factor → symbol

nonterminal symbols (N)

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid a$

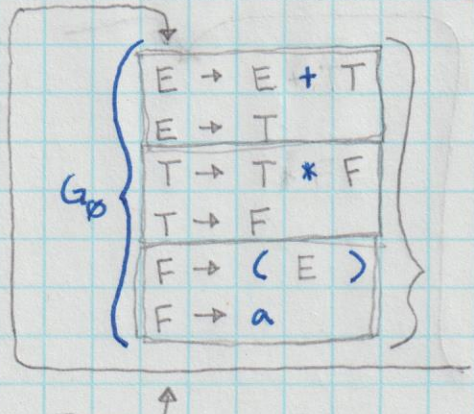
("g naught") G_0

* terminal symbols (Σ): {+, *, (,), a}

01.09.2021 cns.395.1.91061 [introduction to grammar and languages]

08.09.2021 cns.395.1.91061 [introduction to grammar and languages]

* tree is traversed in **preorder** (start at root, follow left child on branch)



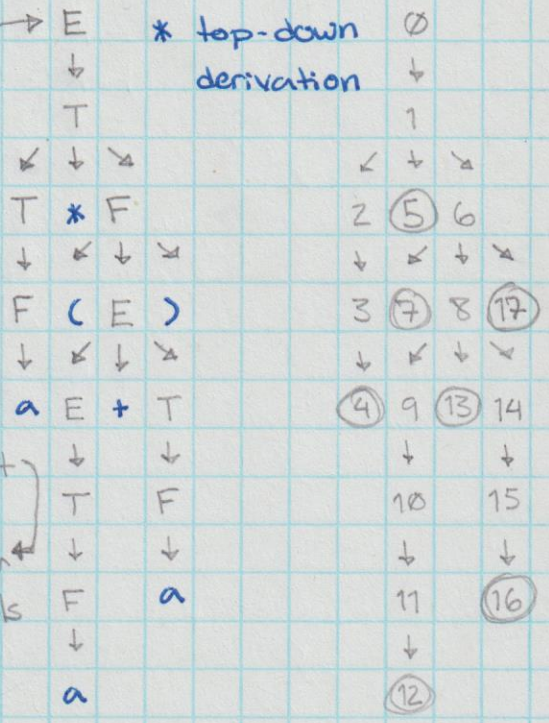
N: non terminal

Σ: terminal

P: production rules

S: starting symbol

a * (a + a)



- cartisian product: $A = \{1,2\}$ $B = \{a,b\}$ $A*B = \{(1,a),(1,b),(2,a),(2,b)\}$
* context sensitive grammar: terminals can be on left
* context-free grammar: only non-terminals on left (computer can't backtrack) cannot write a compiler for
- **sentence:** sentential form containing only terminals
- **sentential form:** "derivation" which contains both nonterminals and terminals
- **leftmost linear derivation:** any derivation where at each step the leftmost non terminal of the sentential form is substituted
- **rightmost linear derivation:** any derivation where at each step the rightmost nonterminal of the sentential form is substituted

a* a*( )

E → T → T * F → T → F * F → a * F → (E) → E + T →
a * (T + T) → a * (F + T) → a * (a + T) →
a * (a + F) → a * (a + a)

→ all terminals on left (finished)

- right linear grammars: $A \to xB \mid x$
- left linear grammars: $A \to Bx \mid x$
- regular grammars:
  - must be a right linear grammar
  - if $S \to \varepsilon$ is a member of P, S may not appear on the left of any P