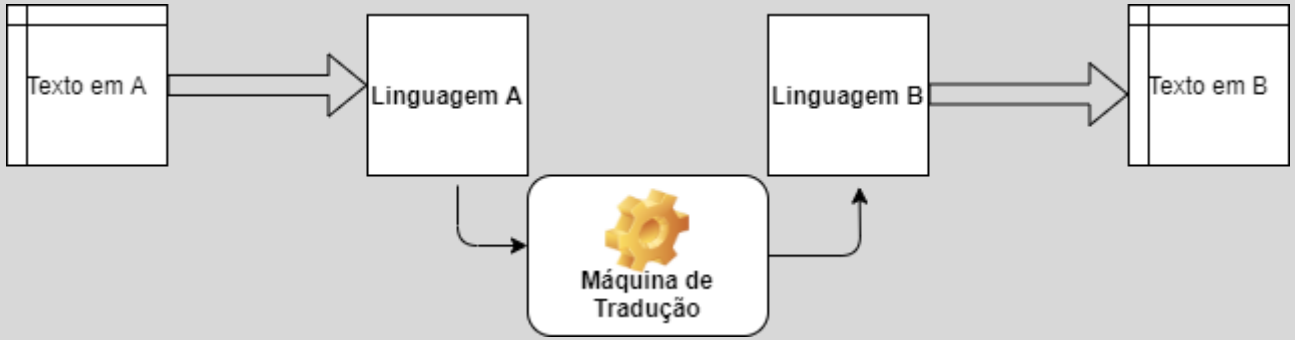
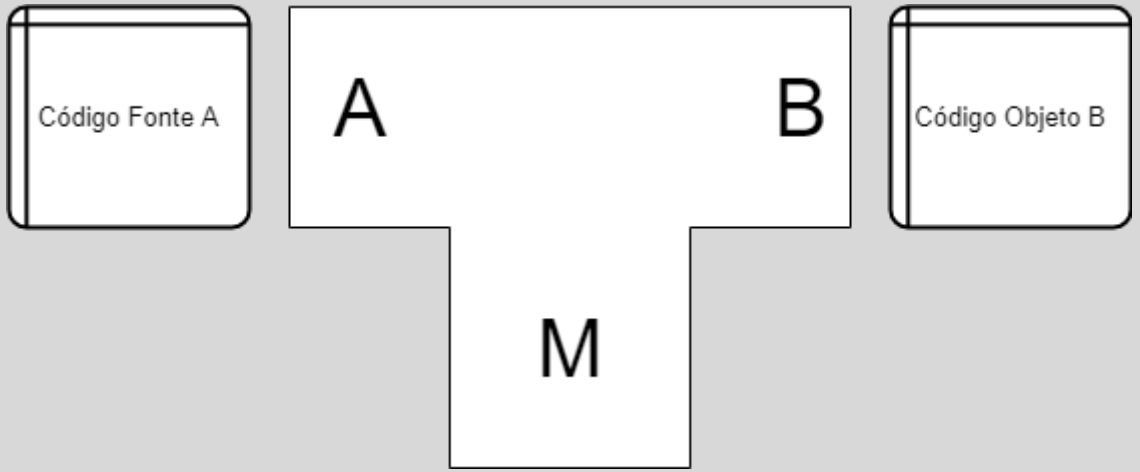
Professor Eraldo Pereira Marinho

# INTRODUÇÃO A COMPILADORES – AULA 01

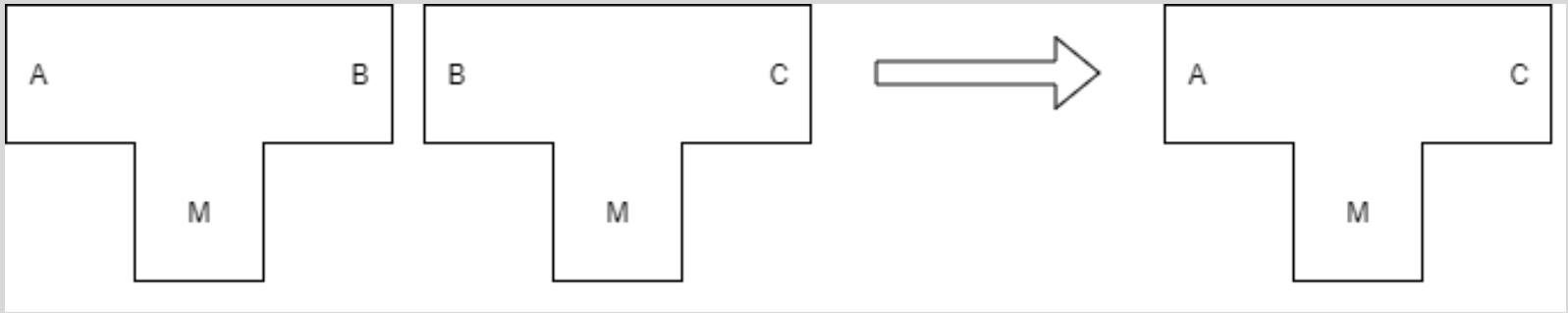
# Introdução – compilador, uma máquina de tradução



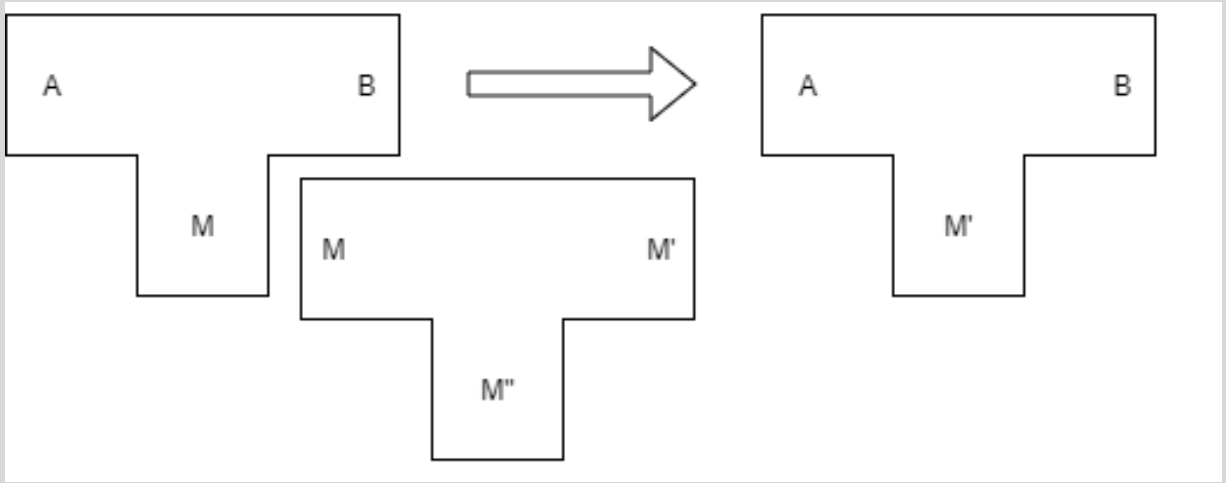
# Notação T



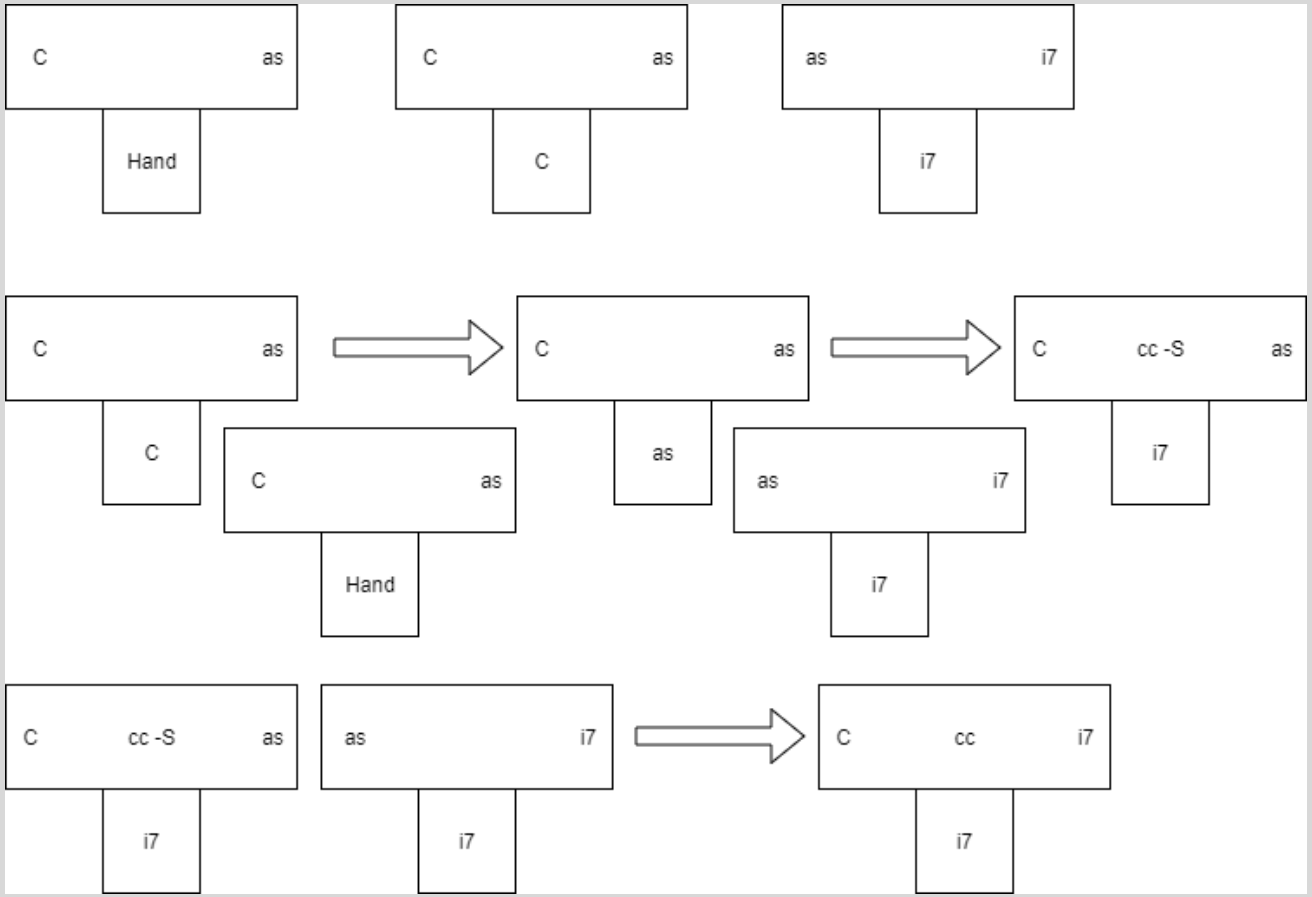
# Notação T – simples composição



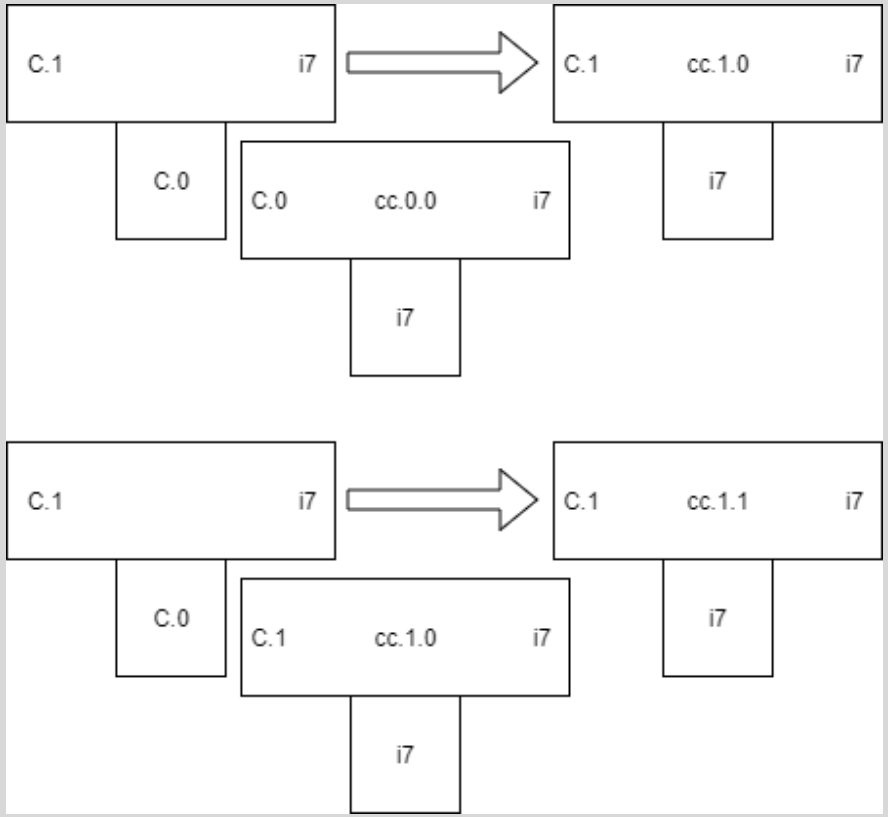
# Notação T – migração de arquitetura



# Notação T – bootstraping

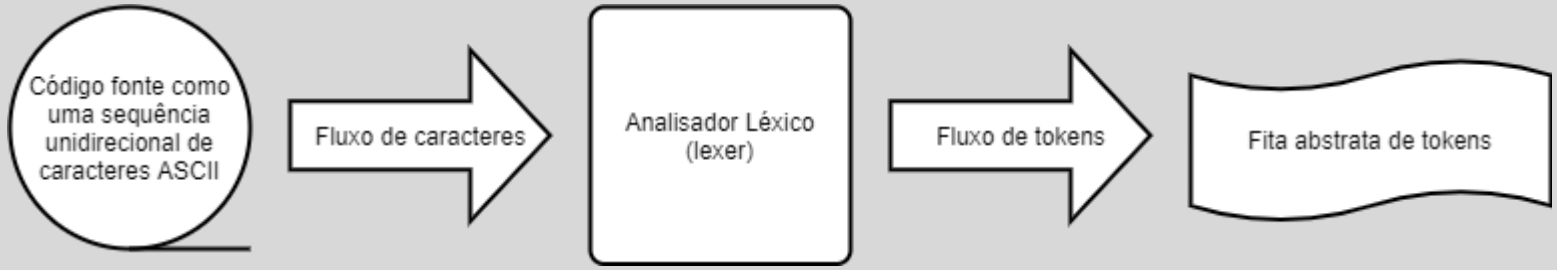


# Notação T – bootstraping e controle de versões C



# Componentes de um compilador – analisador léxico (lexer)

Abstração de um analisador léxico, que é basicamente a emulação de um autômato finito. Uma cadeia de entrada é vista como um fluxo contínuo (streaming) de caracteres ASCII. Esta sequência é convertida em tokens. Por exemplo, o segmento Ax123E seria classificado como identificador (ID) e, consequentemente, este seria o token de saída, um simples ID, sem denotar o que especificamente foi classificado como ID, pois é isso que interessa para a sintaxe, num estágio posterior.

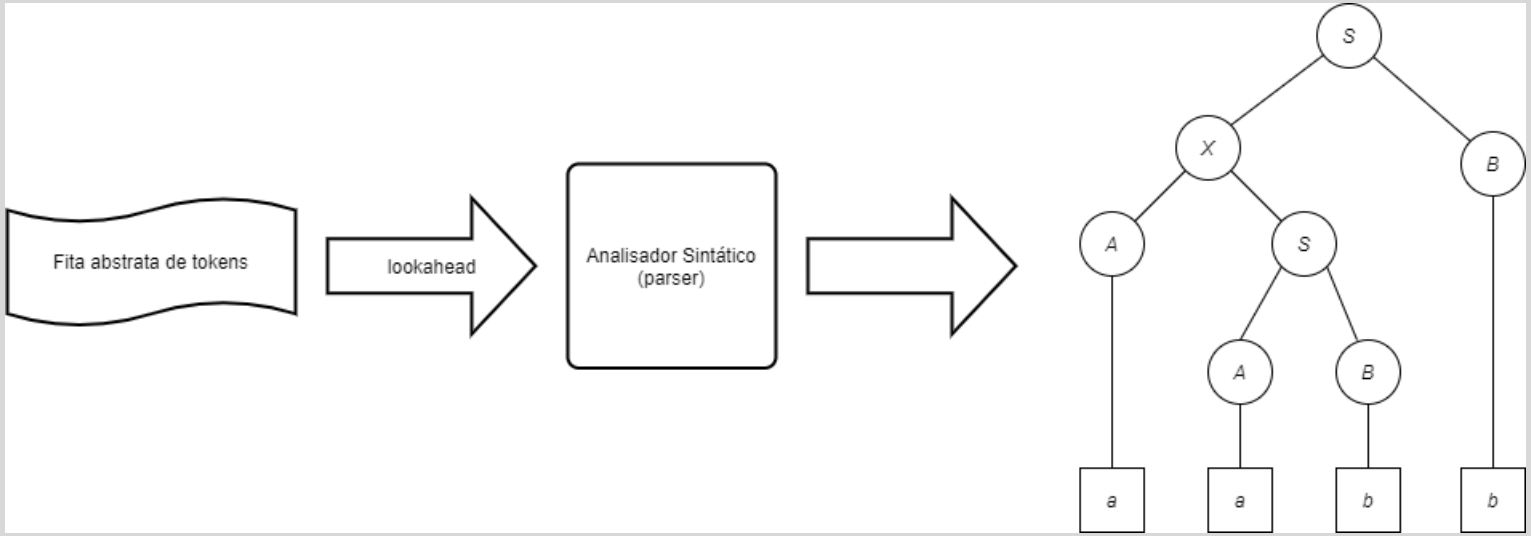


# Analisador léxico - continuação

- Uma sequência como "if x >= y then max := x else max := y;" seria segmentado pelo analisador léxico como $|IF|ID|GE|ID|THEN|ID|ASGN|ID|ELSE|ID|ASGN|ID|$; | onde cada célula corresponde a um tempo (passo) de análise léxica, solicitada passo a passo pelo analisador sintático (parser).
- Observemos que alguns componentes tiveram sua classificação como uma simples transliteração de minúsculo para maiúsculo como, por exemplo, if $\rightarrow IF$. Este é o caso em que o token $IF$ é uma chave única para a cadeia if, bem como as respectivas chaves para then e else. Este é o motivo de if, then e else serem ditas palavras chave (keywords). O mesmo não se dá para o token $ID$, que representa uma linguagem regular infinita, regida pela expressão regular, $ID = ALPHA\ (ALPHA \mid DIGIT)^*$, onde $ALPHA$ é qualquer uma das 26 letras maiúsculas e 26 letras minúsculas do alfabeto europeu sem acentos e $DIGIT$ é um dos 10 dígitos.
- A interação entre o lexer e o parser se dá através de uma função interface, denominada gettoken, implementada em C é definida como $\mathrm{int\ gettoken(FILE\ *source);}$ Na prática, os tokens são macro definições de constantes numéricas, que não se confudam com o código ASCII, que já são naturalmente os tokens padrão do sistema operacional Unix ou Linux.
- Com o decorrer das próximas aulas isso acima exposto ficará bem claro – é sempre bom revisitar esta apresentação para uma melhor compreensão dos conceitos aqui introduzidos.

# Analisador sintático – parser

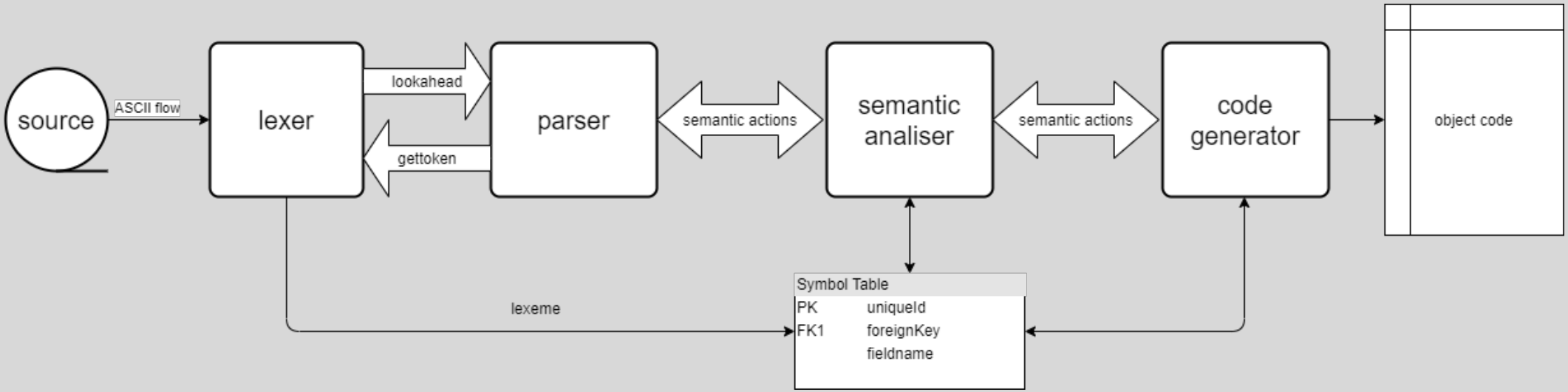
- O analisador sintático é visto como um estágio subsequente ao do lexer. É abstraído conforme o seguinte esboço:



# Parser - continuação

- Conforme o esboço no slide anterior, o parser solicita ao lexer um próximo token, que deve satisfazer a sintaxe da linguagem, geralmente regida por uma gramática livre do contexto. Como saída, o parser produz (virtual ou efetivamente) uma árvore abstrata, obtida da sequência de passos que acompanham o processo de derivação – *observe que, se você não fez ainda teoria da computação, nem fez um estudo prévio por conta própria, não vai estar entendendo muita coisa. Muito menos o que vem a ser um processo de derivação a partir de uma gramática qualquer*.
- Se o único objetivo é a verificação sintática de uma linguagem de programação, tal árvore abstrata não será de muito proveito. Contudo, se a próxima etapa é a análise semântica, que nem sempre é regida por uma gramática livre do contexto, então há uma componente subsequente que são a análise semântica e geração de código. Vimos em teoria da computação que a semântica geralmente é definida em termos de gramáticas sensíveis ao contexto e o processo de optimização na geração de código deve estar associado às gramáticas irrestritas. Contudo, iremos contornar a necessidade de se escrever tais gramáticas, bastante complexas, através do que chamaremos mais tarde de ações definidas por sintaxe, ou simplesmente semântica e geração de código dirigidos por sintaxe.

# Anatomia de um compilador convencional



# Anatomia de um compilador – continuação

- O diagrama no slide anterior é bastante intuitivo mas há um componente que requer explicações. Trata-se da tabela de símbolos. A tabela de símbolos faz a conexão entre os identificadores, ID, e seus conteúdos léxicos, abstraído pela variável global lexeme. Assim, um identificador chamado "abacaxi" terá sua classificação pelo lexer como simples ID. Contudo, a análise semântica requer uma amarra entre o símbolo e uma posição na memória, bem como seus atributos de tipo e forma de indexação, se array, ponteiro, função etc. A interação com o lexer é unidirecional, através de lexeme. Contudo, a interação com os demais componentes pode ser bidirecional no sentido de que atributos adicionais podem surgir no processo de análise sintática e mesmo de análise semântica e geração de código.
- Ao definirmos o escopo de uma função C, como por exemplo, int f(double x, int y); o parser recebe uma sequência de tokens |INT|ID|(|DOUBLE|ID|,|INT|ID|)| mas a tabela de símbolo tem que registrar que f é uma função inteira, que possui dois argumentos, o da esquerda é um double e o da direita é um int, denominados x e y, respectivamente. Assim, nenhuma outra função ou variável no mesmo escopo podem ter o mesmo nome f. Ademais, ao invocarmos f(3.14159, 423) isso está correto porque os parâmetros estão compatíveis declarados no escopo de declaração de f.

# Questionário

1. O que vem a ser um compilador? Podemos dizer que um compilador é uma máquina de Turing? Neste caso, faz sentido dizer que uma linguagem de programação deva ser uma linguagem decidível?
2. Que vem a ser um compilador bootstraping? Ilustre sua resposta com um exemplo prático de como se criar um compilador.
3. Qual o papel de um analisador léxico num compilador? Podemos dizer que um parser é um decisor sintático de uma linguagem de programação? Justifique.
4. Qual a importância da tabela de símbolos no processo de compilação? Ilustre um exemplo envolvendo uma variável C.
5. Com base no que foi explicado e do seu conhecimento de teoria da computação, faça a seguinte abstração: como a sequência lexer, parser, analisador semântico e gerador de código se classificaria na hierarquia de Chomsky?