INSTITUTO FEDERAL
Paraná
Campus Pinhais

Bacharelado em ciência da computação
Linguagens formais e autômatos — 26 de junho de 2025
Trabalho, parte 2: análise sintática — v1.0

1 Análise sintática

Um compilador é um programa que traduz código escrito em uma linguagem de programação para outra linguagem. Geralmente, a linguagem para a qual o código é traduzido é a linguagem de máquina (*Assembly*), que pode ser mais diretamente traduzida para um arquivo no formato executável pelo conjunto ⟨sistema operacional + processador⟩. Entretanto, vale ressaltar que nada impede que a tradução seja para outra linguagem de programação de alto nível.

O processo de compilação consiste de diversas etapas que, de certa forma, interpretam o código fonte em diferentes aspectos. Essa interpretação pode então ser usada para gerar o código na linguagem "de destino" da tradução.

A segunda etapa é chamada de *análise sintática*, que constroi uma *árvore sintática abstrata* a partir dos *tokens* da análise léxica e da gramática da linguagem. Com a árvore, é possível começar a gerar código na linguagem desejada.

Você não irá construir a árvore, mas usará os *tokens* e escreverá uma gramática para decidir se uma entrada é um programa válido (que poderia ser traduzido e efetivamente compilado) ou não.

2 Analisador sintático da linguagem C−−

Sua tarefa neste momento é reutilizar o analisador léxico e construir o analisador sintático para uma linguagem fictícia chamada *C−−*[1]. Para isso, você deverá utilizar seus conhecimentos de gramáticas e uma ferramenta chamada *Bison* [2].

Seu analisador deverá ler um código em C−− da entrada padrão e imprimir uma mensagem indicando que o programa é válido, ou então uma mensagem de erro indicando, *no mínimo* a linha com problema. Caso você encontre um problema de *tokens* desconhecidos, uma mensagem adequada deve ser impressa, como na parte 1 deste trabalho.

3 A linguagem C−−

Nossa linguagem será um subconjunto da linguagem C. Seu analisador deve identificar os seguintes *tokens*:

- Identificador: Nomes de variáveis ou funções, são sequências de letras, números e *underscores* (_). Um identificador não pode iniciar com números;
- Palavras-chave: A linguagem possui algumas palavras-chave usadas para construir os comandos e expressões da linguagem. São elas: `break, continue, else, for, if, return, struct, while`;

[1] Nossa linguagem, obviamente, foi baseada em C. Você pode conferir as definições dos elementos léxicos da linguagem C em `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Lexical-Elements`. A leitura da seção é recomendada para que o estudante perceba que, ao mesmo tempo que não tiramos muita coisa da linguagem, existem algumas situações que seriam mais complexas de tratar

[2] `https://www.gnu.org/software/bison/manual/bison.html`

- Tipos de dados: A linguagem possui os seguintes tipos de dados, usados para variáveis, argumentos de função e retorno de função: `char, int, long, short, void`;
- Constantes numéricas: A linguagem aceita números inteiros apenas;
- Constantes alfanuméricas: A linguagem deve aceitar constantes de caractere (entre aspas simples) e constantes de string (entre aspas duplas). Nessas constantes, são aceitos letras, números e os símbolos `!, @, #, $, %, ^ , &, *, (, ), -, =, +, \`. A linguagem não aceita strings que possuem quebra de linha;
- Operadores: São divididos em operadores aritméticos, lógicos e de comparação. São eles: `+, -, *, /, %, ||, &&, ==, !=, <, <=, >, >=, !`;
- Operador de atribuição: `=`;
- Delimitadores ou separadores: `(, ), [, ], {, }, ;, ,`

A linguagem deve ignorar espaços em branco e tabulações. Lembre-se de contar as linhas quando aparecerem quebras de linha.

Seu analisador deve suportar as seguintes construções sintáticas:

- `if, if-else, if-else-if, while, for`
- Declaração de variáveis com os tipos descritos acima
- Declaração de funções com ou sem argumentos, conforme tipos descritos acima
- Expressões aritméticas
- Expressões lógicas
- Atribuições
- Ponteiros

Você pode consultar uma gramática para a linguagem C em `https://gvcc.dev.br/teaching/bcc-automatos/c_bnf.html`. Você deve simplificar essa gramática para a nossa linguagem C−−.

3.1 O que seu analisador não precisa aceitar

- `struct, union`
- `auto, static, extern, typedef`
- Operador ternário (`expr ? true : false`)
- Constantes
- Auto-incremento (`++, −−`)
- Atribuições com operações (`+=, -=`)
- Enumeradores
- `switch, case`
- `goto`

Seu analisador *não precisa* dizer o tipo de erro sintático, converter o programa para outra linguagem, nem verificar problemas de semântica (se o código faz sentido ou é útil).

4 O que entregar

Você deverá entregar, até 07/julho, um arquivo *.zip* contendo uma pasta, que por sua vez contém:

- O arquivo `.lex` do Flex.
- O arquivo `.y` do Bison.
- Um arquivo Makefile que gera o seu compilador.
- Um arquivo de texto chamado `README.md` contendo o nome completo dos integrantes do grupo e quaisquer outras informações sobre o progresso do trabalho que julgarem relevantes;
  - Esse arquivo **deve conter** uma seção com a gramática que o grupo construiu para C−−, **no formato de gramáticas visto em sala.**
- Os arquivos que o grupo usou para testar o compilador (arquivos *.cmm*).

O arquivo compactado descrito acima deve ser nomeado com as iniciais do nome de cada integrante separados por um hífen. Por exemplo: se o Fulano da Silva fez o trabalho com o João de Souza, o arquivo deve ser nomeado `fs-js.zip`. A pasta, dentro do arquivo compactado, deve ter o mesmo nome, a não ser pela extensão.

O trabalho pode ser feito individualmente ou em duplas.