Trabalho 2 da disciplina de Tradutores*

Ana Paula Martins Tarchetti[1] - 17/0056082

Departamento de Ciência da Computação, Universidade de Brasília (UnB) - Brasília, DF, 70910-900
aptarchetti@gmail.com

1 Introdução e motivação

Um compilador consiste em traduzir um programa em uma linguagem de programação fonte para uma linguagem de programação alvo. Além disso, um compilador também relata possíveis erros que possam estar presentes no código-fonte. A compreensão do processo de compilação/tradução permite a utilização das técnicas aprendidas em aplicações futuras. Exemplos desse uso são: editores de texto, sistemas de recuperação de informações, reconhecimento de padrões, construção de novas linguagem de programação, desenho digital e verificadores [ALSU03].

Este trabalho tem como objetivo fixar de forma prática o entendimento desse processo de construção de compiladores computacionais. Dessa forma, este documento escrito traz os detalhes da implementação que foi feita com base nos processos envolvidos na compilação/tradução de programas. A linguagem de programação fonte que será utilizada é a C-IPL, que se baseia em um subconjunto da linguagem C com algumas alterações e uma nova primitiva que foi criada com o intuito de facilitar a manipulação de listas. Em adição a isso, a linguagem de programação alvo da compilação é a *Three Address Code*, que foi baseada na linguagem de três endereços criada pelos autores do livro [ALSU03].

1.1 Analisador léxico

A primeira etapa que foi implementada é o analisador léxico, também chamado de analisador linear. A função dessa etapa é identificar em um código as sequências de caracteres que formam *tokens*, sendo esse a unidade mínima que compõe o vocabulário de um programa e é composto da seguinte forma: <nome do token, atributos do token>.

Para realizar a implementação dessa etapa foi utilizada a ferramenta *Flex*, [PEM07], cujo intuito é gerar programas que realizam correspondência de padrões em um dado texto. Então, por meio dessa ferramenta, foram declaradas expressões regulares (que foram detalhadas no Apêndice A) para indicar as seguintes definições:

- LETRA: qualquer caractere alfabético;
- DIGITO: qualquer algarismo;

* Professora: Cláudia Nalon

- ID: identificadores de funções e variáveis;
- INTEGER: reconhece números inteiros;
- FLOAT: reconhece números de ponto flutuante;
- STRING_LITERAL: cadeias de texto delimitadas por aspas duplas;
- ESPAÇO: qualquer espaço em branco;
- Outras expressões regulares para reconhecimento de erros específicos.

Após isso, foram definidas regras de identificação de *tokens*. As regras declaradas são divididas em:

- Palavras-chave: **int**, **float**, **list**, **read**, **write**, **writeln**, **main**, **return**, **if**, **else**, **for**, **NIL**;
- Novas operações primitivas: "**?**", "**%**", "**»**", "**«**", ":", "**!**" (sobrecarregado, podendo também denotar negação);
- Operadores: **||**, **&&**, >, <, >=, <=, ==, **!**=, +, **-**, *****, /, =;
- Delimitadores e outros: **vírgula**, **ponto e vírgula**, **parênteses** e **chaves**;
- Regras auxiliares: Ignorar espaços, indicar comentários e indicar erros léxicos e suas linhas e colunas correspondentes.

1.2 Analisador sintático

O Analisador sintático, também chamado de analisador gramatical, corresponde à segunda etapa da compilação. Essa etapa se baseia em uma Gramática Livre de Contexto (GLC) para expressar regras recursivas ou não, que indicam as estruturas de agrupamento de *tokens* que podem ser aceitas pelo compilador. A Gramática Livre de Contexto utilizada neste trabalho está descrita no apêndice B deste documento. Ademais, é nessa etapa em que se cria/popula a árvore sintática abstrata, essa estrutura servirá de auxílio para as próximas etapas do compilador/tradutor.

Para realizar a implementação dessa etapa foi utilizada a ferramenta Bison, [DS15], cujo intuito é gerar programas que realizam essa análise supracitada a partir da declaração de uma GLC em um arquivo de extensão "y". Em adição a isso, foram implementadas, em linguagem C, funções e estruturas auxiliares para que fosse possível a construção da tabela de símbolos e da árvore sintática. Ambas estruturas foram feitas por meio do uso de *struct's* para representar nós de uma árvore e ponteiros para *struct's* afim de poder relacionar esses nós. Na **Figura 1** é possível compreender visualmente como essas estruturas foram implementadas.

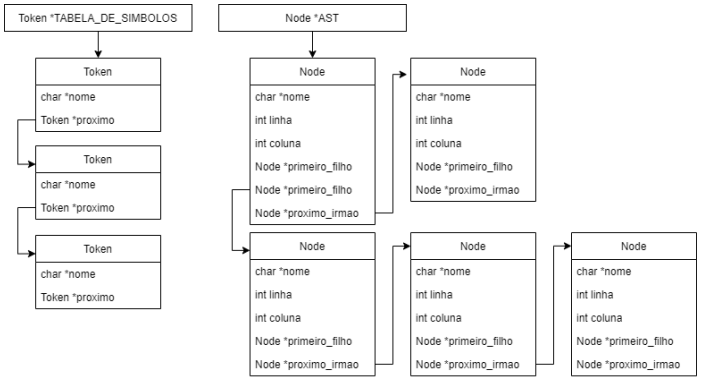

Figura 1. Fluxograma das estruturas utilizadas para a tabela de símbolos e para a árvore sintática abstrata.

2 Compilação, Execução e Testes

Os passos para compilação e execução do analisador podem ser encontrados no arquivo **README.txt** do diretório principal do projeto, ou, a partir do diretório principal, siga os seguintes comandos:

```
$ make
$ ./tradutor < tests/lexico/<nome_do_teste>
$ ./tradutor < tests/sintatico/<nome_do_teste>
```

Tendo isso, foram criados os seguintes **arquivos de teste** para validar o funcionamento esperado em cada etapa da implementação:

1. Testes da análise léxica
   - (i) **lexico/correto_teste_1.c** e (ii) **lexico/correto_teste_2.c** que não apresentam erros;
   - (iii) **lexico/errado_teste_1.c**: apresenta dois erros, um de símbolo não reconhecido na linha 2 e um comentário não finalizado na linha 5;
   - (iv) **lexico/errado_teste_2.c**: apresenta dois erros, um de sufixo inválido na linha 2 e um de demasiados pontos decimais na linha 3.
2. Testes da análise sintática
   - (i) **sintatico/correto_teste_1.c** e (ii) **sintatico/correto_teste_2.c** que não apresentam erros;
   - (iii) **sintatico/errado_teste_1.c**: Nesse teste, falta um ponto e vírgula no final da linha 2 e o operador *header* (?) na linha 4 não deveria haver um parâmetro antes dele;
   - (iv) **sintatico/errado_teste_2.c**: Nesse teste, há um *else* não conseguinte de *if* na linha 6 e faltou uma espreção antes do operador *constructor* (:) na linha 9.

2.1 Ferramenta extra para a visualização da árvore sintática abstrata

Caso seja necessário, a implementação provê uma opção de uso de uma ferramenta para gerar um visualização da árvore sintática abstrata de cada teste, para isso siga os seguintes passos:

1. Execute o teste escolhido;
2. Pegue o conteúdo do arquivo "tree_output_file.txt";
3. Use o texto desse arquivo como entrada para a seguinte ferramenta [ME20]: `https://ironcreek.net/syntaxtree/`

Um exemplo de uso dessa ferramenta pode ser encontrado na **Figura 2**.

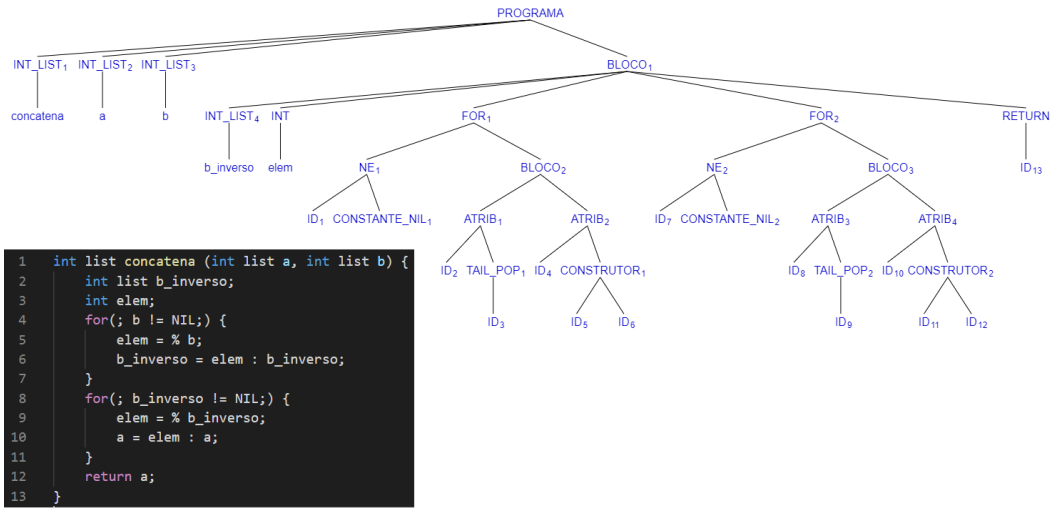

Figura 2. Exemplo de uso da ferramenta de visualização da árvore sintática abstrata, de acordo com o teste feito no programa mostrado acima que implementa a concatenação de duas listas.

Referências

ALSU03. Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: principles, techniques and tools (for Anna University), 2/e*. Pearson Education India, 2003.

DS15. C Donnely and Richard Stallman. Gnu bison–the yacc-compatible parser generator. *Free Software Foundation, Cambridge*, 2015.

ME20. André Eisenbach Mei Eisenbach. jssyntaxtree. `https://ironcreek.net/syntaxtree/`, 2003-2020. (Accessed on 09/17/2021).

PEM07. Vern Paxson, Will Estes, and John Millaway. Lexical analysis with flex. *University of California*, page 28, 2007.

Apêndice A Tabela de definições regulares

Nome	Expressões Regulares (Regex)	Padrão
LETRA	[a-zA-Z]	letras (maiúsculas e minúsculas)
DIGITO	[0-9]	dígitos de 0 a 9
ID	[_a-zA-Z][_a-zA-Z0-9]*	identificadores
INTEGER	{DIGITO}+	número inteiro
FLOAT	({DIGITO}*\.?{DIGITO}+ | $\{DIGITO\} + \backslash.$)	número com ponto flutuante
STRING_LITERAL	\"(\\.[^"\\])*\"	expressões entre aspas duplas
ESPACO	[\t\v\f\r]	todos os tipos de espaço em branco

Apêndice B Gramática

⟨*programa*⟩ ::= ⟨*lista_de_declaracoes*⟩

⟨*lista_de_declaracoes*⟩ ::= ⟨*lista_de_declaracoes*⟩ ⟨*declaracao*⟩
| ε

⟨*declaracao*⟩ ::= ⟨*declaracao_de_variavel*⟩
| ⟨*declaracao_de_funcao*⟩

⟨*declaracao_de_variavel*⟩ ::= ⟨*tipo_de_variavel_id*⟩ **PONT_VIRG**

⟨*tipo_de_variavel_id*⟩ ::= ⟨*tipo_de_variavel*⟩ ⟨*id*⟩

⟨*id*⟩ ::= **ID**

⟨*declaracao_de_funcao*⟩ ::= ⟨*tipo_de_variavel_id*⟩ **ABRE_PARENTESES**
⟨*parametros*⟩ **FECHA_PARENTESES** ⟨*definicao_de_funcao*⟩

⟨*definicao_de_funcao*⟩ ::= ⟨*bloco_de_comando*⟩

⟨*parametros*⟩ ::= ⟨*lista_de_parametros*⟩
| ε

⟨*lista_de_parametros*⟩ ::= ⟨*lista_de_parametros*⟩ **VIRGULA** ⟨*parametro*⟩
| ⟨*parametro*⟩

⟨*parametro*⟩ ::= ⟨*tipo_de_variavel_id*⟩

⟨*comando*⟩ ::= ⟨*bloco_de_comando*⟩
| ⟨*comando_unico*⟩

⟨*comandos*⟩ ::= ⟨*comandos*⟩ ⟨*comando*⟩
| ε

⟨*bloco_de_comando*⟩ ::= **ABRE_CHAVES** ⟨*comandos*⟩ **FECHA_CHAVES**

⟨*comando_unico*⟩ ::= ⟨*comando_condicional*⟩
| ⟨*comando_iterativo*⟩
| ⟨*declaracao_de_variavel*⟩
| ⟨*chamada_de_retorno*⟩
| ⟨*comando_de_atribuicao*⟩
| ⟨*expressao*⟩ **PONTO_VIRGULA**

⟨*comando_condicional*⟩ ::= **IF ABRE_PARENTESES** ⟨*expressao*⟩ **FECHA_PARENTESES** ⟨*comando*⟩
| **IF ABRE_PARENTESES** ⟨*expressao*⟩ **FECHA_PARENTESES** ⟨*comando*⟩ **ELSE** ⟨*comando*⟩

⟨*comando_iterativo*⟩ ::= **FOR ABRE_PARENTESES** ⟨*expressao_for*⟩ **PONTO_VIRGULA** ⟨*expressao_for*⟩ **PONTO_VIRGULA** ⟨*expressao_for*⟩ **FECHA_PARENTESES** ⟨*comando*⟩

⟨*expressao_for*⟩ ::= ⟨*id*⟩ **ATRIB** ⟨*expressao*⟩
| ⟨*expressao*⟩

⟨*expressao*⟩ ::= ⟨*expressao*⟩ **VIRGULA** ⟨*exp*⟩
| ⟨*exp*⟩

⟨*exp*⟩ ::= ⟨*exp*⟩ **GT** ⟨*exp*⟩
| ⟨*exp*⟩ **LT** ⟨*exp*⟩
| ⟨*exp*⟩ **EQ** ⟨*exp*⟩
| ⟨*exp*⟩ **NE** ⟨*exp*⟩
| ⟨*exp*⟩ **LE** ⟨*exp*⟩
| ⟨*exp*⟩ **GE** ⟨*exp*⟩
| ⟨*exp*⟩ **AND** ⟨*exp*⟩
| ⟨*exp*⟩ **OR** ⟨*exp*⟩
| ⟨*exp_list*⟩

⟨*exp_list*⟩ ::= ⟨*exp_list*⟩ **CONSTRUTOR** exp_l⟨*exp_list*⟩ist
| ⟨*exp_list*⟩ **FILTER** ⟨*exp_list*⟩
| ⟨*exp_list*⟩ ⟨*exp_list*⟩ ⟨*exp_list*⟩
| ⟨*exp_aritmetica*⟩
| ε

⟨*exp_aritmetica*⟩ ::= ⟨*termo*⟩
| ⟨*exp_aritmetica*⟩ **SOMA** ⟨*termo*⟩
| ⟨*exp_aritmetica*⟩ **SUB** ⟨*termo*⟩

⟨*termo*⟩ ::= ⟨*fator*⟩
| ⟨*termo*⟩ **MULT** ⟨*fator*⟩
| ⟨*termo*⟩ **DIV** ⟨*fator*⟩

⟨*fator*⟩ ::= ⟨*constante*⟩
| **SUB** ⟨*fator*⟩

| **SOMA** ⟨*fator*⟩
| **TAIL_OR_NOT** ⟨*fator*⟩
| **TAIL_POP** ⟨*fator*⟩
| **HEADER** ⟨*fator*⟩
| **ID**
| **ABRE_PARENTESES** ⟨*exp*⟩ **FECHA_PARENTESES**

⟨*comando_de_atribuicao*⟩ ::= ⟨*id*⟩ **ATRIB** ⟨*expressao*⟩ **PONTO_VIRGULA**

⟨*exp_funcao*⟩ ::= ⟨*id*⟩ **ABRE_PARENTESES** ⟨*expressao*⟩ **FECHA_PARENTESES**
| **READ ABRE_PARENTESES** ⟨*expressao*⟩ **FECHA_PARENTESES**
| **WRITE ABRE_PARENTESES** ⟨*expressao*⟩ **FECHA_PARENTESES**
| **WRITELN ABRE_PARENTESES** ⟨*expressao*⟩ **FECHA_PARENTESES**

⟨*chamada_de_retorno*⟩ ::= **RETURN** ⟨*expressao*⟩ **PONTO_VIRGULA**

⟨*tipo_de_variavel*⟩ ::= **INT**
| **FLOAT**
| **INT LIST**
| **FLOAT LIST**

⟨*constante*⟩ ::= **INTEGER_CONST**
| **FLOAT_CONST**
| **CONSTANTE_NIL**
| **STRING_LITERAL**
| ⟨*exp_funcao*⟩