

Universidade Federal de Goiás
Instituto de Informática
Compiladores - 1
2023-2

Compilador para a Linguagem Cafezino
Trabalho 1 - Análises Léxica e Sintática

Thierson Couto Rosa

1 Objetivo

Este projeto tem como objetivo a implementação de um compilador didático para a linguagem *Cafezinho*. O presente texto descreve a Etapa I do projeto que consiste na especificação do analisador léxico e do analisador sintático para a linguagem.

A Seção 2 descreve a gramática da linguagem *Cafezinho*. A Seção 3 descreve o trabalho de análise léxica, enquanto a Seção 4 corresponde à especificação do analisador sintático para a linguagem. A seção 5 descreve os detalhes de como a Etapa I do projeto deve ser entregue.

2 Gramática para a linguagem *Cafezinho*

A seguir, é apresentada a gramática para a linguagem *Cafezinho*. Os não-terminais da gramática iniciam-se com letra maiúscula e o símbolo inicial é *Programa*. Os terminais aparecem em negrito quando são formados por palavras ou por sequência de caracteres. Exemplo: **int**, **+**, **-**, **()**, **<=**, etc.

Gramática:

Programa \rightarrow *DeclFuncVar DeclProg*

DeclFuncVar \rightarrow *Tipo id DeclVar ; DeclFuncVar*
| Tipo id [intconst] DeclVar ; DeclFuncVar
| Tipo id DeclFunc DeclFuncVar
| ϵ

DeclProg \rightarrow **programa** *Bloco*

DeclVar \rightarrow **, id** *DeclVar*
| , id [intconst] DeclVar
| ϵ

<i>DeclFunc</i>	→ (<i>ListaParametros</i>) <i>Bloco</i>
<i>ListaParametros</i>	→ ε <i>ListaParametrosCont</i>
<i>ListaParametrosCont</i>	→ <i>Tipo id</i> <i>Tipo id</i> [] <i>Tipo id</i> , <i>ListaParametrosCont</i> <i>Tipo id</i> [] , <i>ListaParametrosCont</i>
<i>Bloco</i>	→ { <i>ListaDeclVar</i> <i>ListaComando</i> } { <i>ListaDeclVar</i> }
<i>ListaDeclVar</i>	→ ε <i>Tipo id DeclVar</i> ; <i>ListaDeclVar</i> <i>Tipo id</i> [intconst] <i>DeclVar</i> ; <i>ListaDeclVar</i>
<i>Tipo</i>	→ int car
<i>ListaComando</i>	→ <i>Comando</i> <i>Comando</i> <i>ListaComando</i>
<i>Comando</i>	→ ; <i>Expr</i> ; retorne <i>Expr</i> ; leia <i>LValueExpr</i> ; escreva <i>Expr</i> ; escreva “cadeiaCaracteres” ; novalinha ; se (<i>Expr</i>) entao <i>Comando</i> se (<i>Expr</i>) entao <i>Comando</i> senao <i>Comando</i> enquanto (<i>Expr</i>) execute <i>Comando</i> <i>Bloco</i>
<i>Expr</i>	→ <i>AssignExpr</i>
<i>AssignExpr</i>	→ <i>CondExpr</i> <i>LValueExpr</i> = <i>AssignExpr</i>
<i>CondExpr</i>	→ <i>OrExpr</i> <i>OrExpr</i> ? <i>Expr</i> : <i>CondExpr</i>
<i>OrExpr</i>	→ <i>OrExpr</i> ou <i>AndExpr</i> <i>AndExpr</i>

<i>AndExpr</i>	→	<i>AndExpr</i> e <i>EqExpr</i> <i>EqExpr</i>
<i>EqExpr</i>	→	<i>EqExpr</i> == <i>DesigExpr</i> <i>EqExpr</i> != <i>DesigExpr</i> <i>DesigExpr</i>
<i>DesigExpr</i>	→	<i>DesigExpr</i> < <i>AddExpr</i> <i>DesigExpr</i> > <i>AddExpr</i> <i>DesigExpr</i> >= <i>AddExpr</i> <i>DesigExpr</i> <= <i>AddExpr</i> <i>AddExpr</i>
<i>AddExpr</i>	→	<i>AddExpr</i> + <i>MulExpr</i> <i>AddExpr</i> - <i>MulExpr</i> <i>MulExpr</i>
<i>MulExpr</i>	→	<i>MulExpr</i> * <i>UnExpr</i> <i>MulExpr</i> / <i>UnExpr</i> <i>MulExpr</i> % <i>UnExpr</i> <i>UnExpr</i>
<i>UnExpr</i>	→	- <i>PrimExpr</i> ! <i>PrimExpr</i> <i>PrimExpr</i>
<i>LValueExpr</i>	→	id [<i>Expr</i>] id
<i>PrimExpr</i>	→	id (<i>ListExpr</i>) id () id [<i>Expr</i>] id carconst intconst (<i>Expr</i>)
<i>ListExpr</i>	→	<i>AssignExpr</i> <i>ListExpr</i> , <i>AssignExpr</i>

3 Analisador Léxico

A função que implementa o analisador léxico poderá ser gerada automaticamente, utilizando-se um gerador de analisadores léxicos, por exemplo, JFlex ou Flex. O JFlex gera um analisador escrito na linguagem Java e o Flex gera um analisador léxico escrito nas linguagens C ou C++. Pode ser utilizado também algum gerador de analisador léxico para a linguagem Python.

A função gerada deve ser capaz de reconhecer os *tokens* da gramática da linguagem *Cafe*-

zinho especificada na Seção 2. Os tokens correspondem aos símbolos e palavras que aparecem em negrito na gramática. As palavras reservadas de *Cafezinho* são: **programa**, **car**, **int**, **retorne**, **leia**, **escreva**, **novalinha**, **se**, **entao**, **senao**, **enquanto**, **execute**. No caso em que o *token* é uma constante *string* (**constString**), ou um identificador (**id**), a função deve gerar como lexema, o texto que forma a *string* ou o identificador (ex.: “x1”, “cont2”).

Um identificador em *Cafezinho* deve iniciar com letra ou “_”, seguido de zero ou mais letras, dígitos ou “_”.

O analisador léxico deverá também processar e descartar comentários. Os comentários podem ocupar mais de uma linha do programa fonte. Um comentário em *Cafezinho* inicia com o par de símbolos “/*” e termina com o par de símbolos “*/”. Um erro deve ser reportado, caso um comentário não termine.

O analisador léxico deve reportar, através de mensagens, erros léxicos encontrados no arquivo de entrada. São exemplos de erros léxicos: caracteres inválidos na linguagem, comentários que não terminam, cadeia de caracteres que não terminam ou que ocupam mais de uma linha no arquivo de entrada.

Caso o arquivo contenha um erro léxico, o programa deverá imprimir uma linha contendo exatamente o seguinte: **ERRO:**, seguido por um espaço em branco e, por uma das seguintes mensagens de erro: **CARACTERE INVÁLIDO** ou **COMENTÁRIO NAO TERMINA** ou **CADEIA DE CARACTERES OCUPA MAIS DE UMA LINHA**. Após a mensagem de erro o programa deve imprimir, na mesma linha, o número da linha do programa fonte onde o erro foi encontrado.

4 Analisador Sintático

A função do compilador que implementa o analisador sintático pode ser gerada automaticamente utilizando-se um gerador de analisadores sintáticos ou *parsers*. Neste trabalho poderá ser utilizado um dos seguintes geradores: BYACC/J ou o Bison. Ambos utilizam o método LALR para a geração do analisador sintático. Poderá ser utilizado também um gerador de analisador sintático para a linguagem Python. O trabalho de implementação do analisador sintático consiste na preparação do arquivo de entrada para o gerador de analisador sintático e na adaptação da função analisador sintático gerada para que possa funcionar utilizando a função analisador léxico obtida no trabalho especificado na Seção 3.

O programa principal que chama a função analisador sintático deve receber o nome do arquivo a ser compilado como parâmetro de entrada. Especificamente, dado que o programa executável do analisador sintático tenha o nome *cafezinho*, e supondo que o arquivo de entrada seja *teste.z*, deve ser possível executar a análise sintática do arquivo através do seguinte comando em uma *shell* do Linux: *./cafezinho teste.z*

Deve ser implementado o corpo da função *yyperror()*, de tal modo que erros sintáticos detectados pela função *yyparse()* sejam emitidos na tela do computador, juntamente com o número da linha onde o erro foi detectado. A mensagem de erro deve iniciar com a palavra **ERRO:** seguida por um espaço.

5 Informações Sobre a Implementação e a Entrega

O trabalho é individual e deve ser entregue até o dia 12/11/2023 via tarefa criada no sistema Moodle. Devem ser entregues:

- O código de entrada para o gerador de analisador léxico utilizado (arquivo com extensão “.l”).
- O código de entrada para o gerador de analisador sintático utilizado (arquivo com extensão “.y”).
- O *Makefile* contendo:
 - Comandos de execução do gerador de analisador léxico (Flex ou JFlex) para conversão do arquivo de entrada do gerador de analisador léxico em programas em C ou Java.
 - Comandos de execução do gerador de analisador sintático (YACC/J ou Bison) para conversão do arquivo de entrada do gerador de analisador sintático.
 - Comandos de compilação e link-edição para compilar e ligar o programa principal com os códigos dos analisadores léxicos e sintáticos gerados.

Os itens acima devem estar agrupados em um arquivo do tipo *tar* compactado com o utilitário *gzip* e submetidos como uma tarefa a ser criada no ambiente Moodle da disciplina. Os códigos gerados devem estar preparados para executarem no sistema operacional Linux (ambiente onde os trabalhos serão avaliados). O programa executável deve ter o nome *cafezinho*.

Importante:

Cópias idênticas ou modificadas dos códigos ou de partes dos códigos são consideradas plágios. **Plágio é crime.** O aluno que cometer plágio em seu trabalho receberá nota zero no mesmo.