

C-minus com operações matriciais

Leonardo Maffei da Silva*

2019

Resumo

Este documento atende os fins de documentação da segunda parte do projeto final da disciplina *Tradutores*, ministrada pela professora Dr.a Cláudia Nalon, no segundo semestre de 2019, na Universidade de Brasília. Tal artefato descreve um pouco da implementação do analisador léxico, dificuldades encontradas durante tal processo; a gramática da linguagem proposta na parte 1 deste trabalho, bem como as alterações nela realizada; políticas de tratamento de erros e arquivos de teste para o analisador produzido. ([ANôNIMO](#),).

Palavras-chave: C, linguagem, matriz, primitiva.

1 Introdução

Implementar-se-á, até a versão final deste artigo, um compilador para a linguagem proposta. Para sua realização, serão utilizados os conhecimentos adquiridos na disciplina *Tradutores*, ministrada pela professora [Cláudia Nalon](#).

2 Usuário característico

Destina-se ao estudante de álgebra linear, o qual pode usar a linguagem para, por exemplo, confirmar se sua resolução de um sistema linear encontra-se correta, tudo isso de maneira rápida, eficiente e *offline*.

3 Motivação

Durante a realização do curso de Cálculo Numérico, o grupo do autor notou a ausência dessa *feature* na linguagem C. Desse modo, foi necessária a simulação desse tipo de dados, à época implementada por meio de inúmeras funções. Se houvesse um tipo nativo para matriz bem como operações elementares sobre seus elementos, teria sido de grande auxílio à codificação dos diversos métodos numéricos requeridos pela disciplina.

*leoitu22hotmail.com@gmail.com. <<https://www.linkedin.com/in/leonardo-maffei-ti/>>

4 Gramática

A seguir, encontra-se a gramática da linguagem proposta:

$$\langle program \rangle ::= \langle decl-list \rangle$$
$$\langle decl-list \rangle ::= \langle decl-list \rangle \langle decl \rangle \mid \langle decl \rangle \mid \langle comment \rangle \mid \langle expr-stmt \rangle$$
$$\langle decl \rangle ::= \langle var-decl \rangle \mid \langle var-decl-init \rangle \mid \langle fun-decl \rangle$$
$$\langle comment \rangle ::= // \langle ASCII \rangle^* \mathbf{EOL}$$
$$\langle type-spec \rangle ::= \langle base-type-spec \rangle \mid \langle mat-spec \rangle$$
$$\langle mat-spec \rangle ::= \mathbf{mat} < \langle base-type-spec \rangle >$$
$$\langle base-type-spec \rangle ::= \mathbf{int} \mid \mathbf{float} \mid \mathbf{char}$$
$$\langle var-decl \rangle ::= \langle base-type-spec \rangle \langle var \rangle ; \mid \langle mat-spec \rangle \mathbf{ID} [\langle simple-expr \rangle][\langle simple-expr \rangle] ;$$
$$\langle var-decl-init \rangle ::= \langle base-type-spec \rangle \langle var \rangle = \langle expr \rangle ; \mid \langle mat-spec \rangle \mathbf{ID} [\langle simple-expr \rangle][\langle simple-expr \rangle] \\ = \langle seq-list \rangle ;$$
$$\langle fun-decl \rangle ::= \langle type-spec \rangle \langle ID \rangle (\langle params \rangle) \langle compound-stmt \rangle \mid \langle type-spec \rangle \langle ID \rangle (\langle params \rangle) ;$$
$$\langle params \rangle ::= \langle param-list \rangle \mid \epsilon$$
$$\langle param-list \rangle ::= \langle param-list \rangle , \langle param \rangle \mid \langle param \rangle$$
$$\langle param \rangle ::= \langle type-spec \rangle \langle ID \rangle \mid \langle base-type-spec \rangle \langle ID \rangle [] \mid \mathbf{mat} < \langle type-spec-base \rangle > \langle ID \rangle$$
$$\langle compound-stmt \rangle ::= \langle local-decls \rangle \langle stmt-list \rangle$$
$$\langle local-decls \rangle ::= \langle local-decls \rangle \langle var-decl \rangle \mid \epsilon$$
$$\langle stmt-list \rangle ::= \langle stmt-list \rangle \langle stmt \rangle \mid \epsilon$$
$$\langle stmt \rangle ::= \langle expr-stmt \rangle \mid \langle compound-stmt \rangle \mid \langle selection-stmt \rangle \\ \mid \langle while-stmt \rangle \mid \langle return-stmt \rangle$$
$$\langle expr-stmt \rangle ::= \langle expr \rangle ; \mid ;$$
$$\langle selection-stmt \rangle ::= \mathbf{if} (\langle expr \rangle) \langle stmt \rangle \mid \mathbf{if} (\langle expr \rangle) \langle stmt \rangle \mathbf{else} \langle stmt \rangle$$
$$\langle while-stmt \rangle ::= \mathbf{while} (\langle expr \rangle) \langle stmt \rangle$$
$$\langle return-stmt \rangle ::= \mathbf{return} ; \mid \mathbf{return} \langle expr \rangle ;$$
$$\langle expr \rangle ::= \langle var \rangle = \langle expr \rangle \mid \langle simple-expr \rangle \mid \langle int-seq \rangle \mid \langle float-seq \rangle \mid \langle int-seq-list \rangle \mid \langle float-seq-list \rangle$$
$$\langle seq-list \rangle ::= \langle int-seq-list \rangle \mid \langle float-seq-list \rangle$$
$$\langle int-seq \rangle ::= \langle simple-expr \rangle \mid \langle int-seq \rangle , \langle simple-expr \rangle$$

$$\begin{aligned}
\langle \textit{int-seq-list} \rangle &::= \langle \textit{int-seq} \rangle \mid \langle \textit{int-seq-list} \rangle, \langle \textit{int-seq} \rangle \mid \epsilon \\
\langle \textit{float-seq} \rangle &::= \langle \textit{simple-expr} \rangle \mid \langle \textit{float-seq} \rangle, \langle \textit{simple-expr} \rangle \\
\langle \textit{float-seq-list} \rangle &::= \langle \textit{float-seq} \rangle \mid \langle \textit{float-seq-list} \rangle, \langle \textit{float-seq} \rangle \mid \epsilon \\
\langle \textit{var} \rangle &::= \langle \textit{ID} \rangle \mid \langle \textit{ID} \rangle [\langle \textit{simple-expr} \rangle] \\
\langle \textit{simple-expr} \rangle &::= \langle \textit{add-expr} \rangle \langle \textit{relop} \rangle \langle \textit{add-expr} \rangle \mid \langle \textit{add-expr} \rangle \\
\langle \textit{relop} \rangle &::= <= \mid < \mid > \mid >= \mid == \mid != \\
\langle \textit{add-expr} \rangle &::= \langle \textit{add-expr} \rangle \langle \textit{addop} \rangle \langle \textit{term} \rangle \mid \langle \textit{term} \rangle \\
\langle \textit{addop} \rangle &::= + \mid - \\
\langle \textit{term} \rangle &::= \langle \textit{term} \rangle \langle \textit{mulop} \rangle \langle \textit{bin} \rangle \mid \langle \textit{bin} \rangle \\
\langle \textit{mulop} \rangle &::= * \mid / \mid @ \mid @@ \\
\langle \textit{bin} \rangle &::= \langle \textit{bin} \rangle \langle \textit{bin-logi} \rangle \langle \textit{unary} \rangle \mid \langle \textit{unary} \rangle \\
\langle \textit{bin-logi} \rangle &::= \&\& \\
&\quad \begin{array}{c} \mid \quad \parallel \\ \mid \quad \wedge \end{array} \\
\langle \textit{unary} \rangle &::= \langle \textit{unary-op} \rangle \langle \textit{factor} \rangle \mid \langle \textit{factor} \rangle \\
\langle \textit{unary-op} \rangle &::= ! \mid \& \mid - \\
\langle \textit{factor} \rangle &::= (\langle \textit{expr} \rangle) \mid \langle \textit{var} \rangle \mid \langle \textit{call} \rangle \mid \langle \textit{NUM} \rangle \\
\langle \textit{call} \rangle &::= \langle \textit{ID} \rangle (\langle \textit{args} \rangle) \\
\langle \textit{args} \rangle &::= \langle \textit{arg-list} \rangle \parallel \epsilon \\
\langle \textit{arg-list} \rangle &::= \langle \textit{arg-list} \rangle, \langle \textit{simple-expr} \rangle \mid \langle \textit{simple-expr} \rangle \\
\langle \textit{letter_} \rangle &::= \textit{a} \mid \dots \mid \textit{z} \mid \textit{A} \mid \dots \mid \textit{Z} \mid _ \\
\langle \textit{digit} \rangle &::= 0 \mid \dots \mid 9 \\
\langle \textit{ID} \rangle &::= \langle \textit{letter_} \rangle (\langle \textit{letter_} \rangle \mid \langle \textit{digit} \rangle)^* \\
\langle \textit{NUM} \rangle &::= \langle \textit{INT} \rangle \mid \langle \textit{FLOAT} \rangle \\
\langle \textit{INT} \rangle &::= \langle \textit{digit} \rangle^+ \\
\langle \textit{CHAR} \rangle &::= '\langle \textit{ASCII} \rangle' \\
\langle \textit{STRING} \rangle &::= <\textit{ASCII_}> \\
\langle \textit{FLOAT} \rangle &::= \langle \textit{digit} \rangle^+ . \mid \langle \textit{digit} \rangle^+
\end{aligned}$$

Em relação à versão pregressa passado, note-se as principais mudanças:

- adição dos operadores *@* e *@@*, sendo respectivamente o operador de multiplicação e potenciação de matrizes.
- adição do tipo de dado *char*, para possibilitar uma capacidade de comunicação com o usuário superior à que havia antes, a qual proporcionava apenas tipos de dados numéricos.
- adição de *strings*, tendo em vista a interação com o usuário (semelhante à adição do *char*). Seu destaque fica por conta da simplicidade em se mostrar mensagens "de uma só vez" quando comparado a imprimir um caractere por comando. Limitação: o usuário apenas pode trabalhar com *strings literais*, e não existe o tipo de dados *string*.
- removeu-se a palavra reservada **void** da linguagem. Desse modo, todas as funções passam a retornar algum valor e não é mais possível a declaração de funções cujo único conteúdo entre os delimitadores de parâmetros era esta palavra reservada. Onde havia ocorrência daquela foi deixada a cadeia vazia. **void**. A vantagem é a simplificação da gramática bem como remoção de um tipo de função "não essencial" ao usuário.
- sutil alteração na regra *<param>*, tal que agora é gramaticalmente inválido a declaração de um vetor de matrizes (erro semântico, mas que em versão anterior da gramática era sintaticamente válido).
- Além das mudanças acima citadas, foram realizadas algumas renomeações de regras afim de representação mais compacta da gramática e desse modo requerindo menos esforço visual para perceber todas as suas regras bem como a relação entre elas. As mudanças de nome realizadas foram:
 - *declaration* → *decl*
 - *specifier* → *spec*
 - *declarations* → *decls*
 - *statement* → *stmt*
 - *iteration* → *while*
 - *int-nested-seq* → *int-seq-list*
 - *float-nested-seq* → *int-seq-list*
 - *expression* → *expr*
- por fim, há a alteração do lado direito da regra *<arg-list>*, substituindo *<expression>* por *<simple-expr>*, visto que de fato não faz sentido permitir operações de atribuição na passagem de parâmetros para funções.

5 Semântica

A semântica da linguagem é quase semelhante à da linguagem C: declarações de variáveis (a menos das do tipo **mat**), funções e expressões têm semântica similar. Sendo esta linguagem uma extensão de um subconjunto da linguagem *C*, a principal

diferença está no tipo de dados **mat** (abreviação de *matrix*). Esse tipo de dados é similar aos *arrays* em C, porém limitado do ponto de vista da composição pois não é possível a criação de matrizes aninhadas, nem de matrizes de vetores. Entretanto, é possível a realização das quatro operações aritméticas básicas diretamente com matrizes, bem como a realização de potenciação de matrizes de forma *rápida* e algumas operações sobre elas, como resolução de sistemas lineares e escalonamento. Multiplicação e potenciação de matrizes são respectivamente expressas pelos novos operadores **@** e **@@**.

6 Exemplo de programa na linguagem

A seguir, trechos de código pertencente à nova linguagem.

```
1 int main() {
2 float a = 10.1;
3 float c = 10.;
4 float d = .1;
5 float b = .29;
6
7 mat<int> m[3][3] = {{1, 0, 0}, {0 ,1, 0}, {0, 0, 1}};
8 scan(a);
9 } 'a'; '\n'; '\r'; '\\';
```

```
1 float main;
2 {1,2,3};
3 "Pode ir, tudo bem..."
4 /* Lucero mto bom */
5 print('h');
6 print('e');
7 print('l');
8 print('l');
```

7 Exemplo de programa não pertencentes à linguagem

```
1 " string sem dim!
2
3 int main() $$ {
4 ;;;
5 }
```

```
1 float chr(void);
2 '\\\\';
3 /*
4 int main() {
5
6 }
7 print("e agora, joseh?")
8 Obs: a exemplo do tratamento proporcionado pelo gcc,
9 nao eh feito nenhuma tentativa de recuperacao para
10 erros de comentarios sem fechamento ;)
```

8 Implementação

O presente analisador léxico tem como dependências, além do programa *flex*, os respectivos arquivos fonte, além de seus respectivos cabeçalhos (com exceção do último, por tratar do código fonte utilizado pelo *flex* para geração do arquivo **lex.yy.c**):

- Array.c
- Colorfy.c
- SymTable.c
- leo.l

Todos os arquivos estão disponíveis publicamente [neste link](#), bem como o *makefile* utilizado para compilação do analisador. Contudo, **recomenda-se fortemente** a execução das seguintes instruções, na ordem em que aparecem, para garantia da correta geração do analisador:

Listing 1 – bash version

```
make clean
flex leo.l
make
./lexico <caminho-para-arquivo>
```

As instruções além do clássico *make* fazem-se necessárias pois não foi conseguido pelo autor a automação do processo de transpilação do código em *leo.l* para *lex.yy.c* seguido da compilação e ligação entre os arquivos necessários à geração do analisador de forma automática *com a garantia de nova geração do **lex.yy.c** sempre que fosse dado o comando **make***. Não foi necessária nenhuma modificação direta do arquivo **lex.yy.c** em momento algum do desenvolvimento.

8.1 Funcionamento

Após gerado o analisador léxico seguindo os passos descritos no início desta seção, sua utilização é bastante simples: execute o programa passando como argumento o caminho para o arquivo que deve ser aberto e processado pelo léxico. O programa então lê sequencialmente o arquivo caractere a caractere e vai exibindo os *tokens* presentes no arquivo apontado à medida que os encontra, bem como trata os erros descritos na subseção logo abaixo. A saída é exibida na saída padrão (usualmente, um console) e de forma *colorida*. Entende-se que isso facilita a visualização por parte dos humanod; contudo, isso deve ser adaptado na próxima fase para que seja gerado um arquivo estruturado contendo todos os tokens identificados, o qual será utilizado pelo analisador sintático.

8.2 Tratamento de Erros

O analisador léxico desenvolvido é capaz de detectar os seguintes erros:

1. *string* sem fechamento
2. comentários em bloco sem fechamento
3. caracteres não pertencentes à linguagem (individualmente)

O erro 1 é tratado considerando que o usuário termina a *string* ao final da linha, visto que não são permitidas *strings* multi-linhas, e continuando a análise como se não houvesse erro. O próximo erro (2) não é exatamente tratado; a abordagem utilizada é a mesma do compilador gcc 7.7.0: é emitido um aviso ao usuário informando-lhe linha e coluna onde

se inicia o comentário não finalizado. Por fim, o último erro (3) é tratado informando ao usuário as ocorrências desses caracteres, porém sem entrar em modo de pânico.

8.3 Dificuldades Encontradas

Uma das dificuldades foi sem dúvida a familiarização com a ferramenta *flex*, a qual demonstrou-se muito competente porém não tão intuitiva, bem como alguns trechos de seu [manual](#). Além disso, surgiram dúvidas a respeito do que seriam erros léxicos, como por exemplo qual o escopo dessa categoria de erros.

Além das dúvidas acima, foram encontrados obstáculos na automação da geração do analisador por meio da ferramenta *make*, questões sobre como deveria ser exibida a sequência de *tokens* (quais possíveis metadados deveriam ser exibidos?) e algumas pontualidades acerca de recursão na gramática.

Em termos de codificação, além do problema recém-citado, não foi possível implementar a tabela de símbolos a tempo da entrega (ocorreram problemas de vazamento de memória e alocação equivocada). Embora esta não seja fundamental para a exibição dos tokens lidos em sequência, sua presença certamente enriqueceria o presente trabalho.

8.4 Arquivos de teste

Os arquivos de teste encontram-se disponíveis [neste link](#). Os arquivos desprovidos de erros léxicos têm seus nomes iniciados pela letra 'c', ao passo que os demais iniciam com a letra 'e'.

9 Agradecimentos

Agradecimentos ao responsável por ([ANôNIMO](#),), sem o qual este trabalho teria sido muito mais custoso, bem como ao usuário [CroCro](#), o qual possibilitou conforme ([CROCRO](#),) rápida customização do código-exemplo da nova linguagem. Por fim, o pacote utilizado para redação da gramática foi sugerido pelo usuário [AlanMunn](#), conforme ([MUNN](#),).

Referências

ANôNIMO. *C-syntax*: Bnf grammar for c-minus. Disponível em: <http://www.csci-snc.com/ExamplesX/C-Syntax.pdf>. Citado 2 vezes nas páginas 1 e 7.

CROCRO. Disponível em: <https://tex.stackexchange.com/questions/348651/c-code-to-add-in-the-document>. Citado na página 7.

MUNN, A. Disponível em: <https://tex.stackexchange.com/questions/24886/which-package-can-be-used-to-write-bnf-grammars/39751>. Citado na página 7.