# C-minus com operações matriciais

Leonardo Maffei da Silva*

2019

## Resumo

Neste documento encontra-se a especificação da gramática de uma linguagem que estende uma versão mínima do C (conhecida por *C-minus*). A nova linguagem proposta apresenta como primitivo o tipo matriz, bem como a realização das operações aritméticas básicas e outras operações mais complexas tais como potenciação e escalonamento, sendo a última realizada por função da linguagem. Grande parte da gramática foi diretamente tirada de (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 exmeplo, confirmar se sua resolução de um sisema 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 houvese 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:

⟨*program*⟩ ::= ⟨*declaration-list*⟩

⟨*declaration-list*⟩ ::= ⟨*declaration-list*⟩ ⟨*declaration*⟩ | ⟨*declaration*⟩ | ⟨*comment*⟩

⟨*comment*⟩ :: // ⟨*ASCII*⟩* **EOL**

⟨*declaration*⟩ ::= ⟨*var-declaration*⟩ | ⟨*fun-declaration*⟩

⟨*var-declaration*⟩ ::= ⟨*base-type-specifier*⟩ ID ; | ⟨*base-type-specifier*⟩ ID [⟨*expression*⟩];
| ⟨*mat-specifier*⟩ ID [⟨*expression*⟩][⟨*expression*⟩] ;

⟨*type-specifier*⟩ ::= ⟨*base-type-specifier*⟩ | ⟨*mat-specifier*⟩

⟨*base-type-specifier*⟩ ::= **int** | **float**

⟨*mat-specifier*⟩ ::= **mat** <⟨*base-type-specifier*⟩ >

⟨*fun-declaration*⟩ ::= ⟨*type-specifier*⟩ ⟨*ID*⟩ ( ⟨*params*⟩ ) ⟨*compound-stmt*⟩ | ⟨*mat-specifier*⟩ ⟨*ID*⟩ ( ⟨*params*⟩ ) ⟨*compound-stmt*⟩

⟨*params*⟩ ::= ⟨*param-list*⟩ | $\epsilon$

⟨*param-list*⟩ ::= ⟨*param-list*⟩ , ⟨*param*⟩ | ⟨*param*⟩

⟨*param*⟩ ::= ⟨*type-specifier*⟩ ⟨*ID*⟩ | ⟨*type-specifier*⟩ ⟨*ID*⟩ [ ] | mat<⟨*type-specifier-base*⟩> ⟨*ID*⟩

⟨*compound-stmt*⟩ ::= ⟨*local-declarations*⟩ ⟨*statement-list*⟩

⟨*local-declarations*⟩ ::= ⟨*local-declarations*⟩ ⟨*var-declaration*⟩ | $\epsilon$

⟨*statement-list*⟩ ::= ⟨*statement-list*⟩ ⟨*statement*⟩ | $\epsilon$

⟨*statement*⟩ ::= ⟨*expression-stmt*⟩ | ⟨*compound-stmt*⟩ | ⟨*selection-stmt*⟩
| ⟨*iteration-stmt*⟩ | ⟨*return-stmt*⟩

⟨*expression-stmt*⟩ ::= ⟨*expression*⟩ ; | ;

⟨*selection-stmt*⟩ ::= **if** ( ⟨*expression*⟩ ) ⟨*statement*⟩ | **if** ( ⟨*expression*⟩ ) ⟨*statement*⟩ **else** ⟨*statement*⟩

⟨*iteration-stmt*⟩ ::= **while** ( ⟨*expression*⟩ ) ⟨*statement*⟩

⟨*return-stmt*⟩ ::= **return** ; | **return** ⟨*expression*⟩ ;

⟨*expression*⟩ ::= ⟨*var*⟩ = ⟨*expression*⟩ | ⟨*simple-expression*⟩ | ⟨*int-seq*⟩ | ⟨*float-seq*⟩ | ⟨*int-nested-seq*⟩ | ⟨*float-nested-seq*⟩

⟨*int-seq*⟩ ::= ⟨*simple-expression*⟩ | ⟨*int-seq*⟩, ⟨*simple-expression*⟩

⟨*float-seq*⟩ ::= ⟨*simple-expression*⟩ | ⟨*float-seq*⟩, ⟨*expression*⟩

⟨*int-nested-seq*⟩ ::= ⟨*int-seq*⟩ | ⟨*int-nested-seq*⟩, ⟨*int-seq*⟩ | $\epsilon$

⟨*float-nested-seq*⟩ ::= ⟨*float-seq*⟩ | ⟨*float-nested-seq*⟩, ⟨*float-seq*⟩ | $\epsilon$

⟨*var*⟩ ::= ⟨*ID*⟩ | ⟨*ID*⟩ [ ⟨*expression*⟩ ]

⟨*simple-expression*⟩ ::= ⟨*additive-expression*⟩ ⟨*relop*⟩ ⟨*additive-expression*⟩ | ⟨*additive-expression*⟩

⟨*relop*⟩ ::= <= | < | > | >= | == | !=

⟨*additive-expression*⟩ ::= ⟨*additive-expression*⟩ ⟨*addop*⟩ ⟨*term*⟩ | ⟨*term*⟩

⟨*addop*⟩ ::= + | -

⟨*term*⟩ ::= ⟨*term*⟩ ⟨*mulop*⟩ ⟨*bin*⟩ | ⟨*bin*⟩

⟨*mulop*⟩ ::= * | /

⟨*bin*⟩ ::= ⟨*bin*⟩ ⟨*bin-logi*⟩ ⟨*unary*⟩ | ⟨*unary*⟩

⟨*bin-logi*⟩ ::= &&
| ||
| ^

⟨*unary*⟩ ::= ⟨*unary-op*⟩ ⟨*factor*⟩ | ⟨*factor*⟩

⟨*unary-op*⟩ ::= ! | &

⟨*factor*⟩ ::= ( ⟨*expression*⟩ ) | ⟨*var*⟩ | ⟨*call*⟩ | ⟨*NUM*⟩

⟨*call*⟩ ::= ⟨*ID*⟩ ( ⟨*args*⟩ )

⟨*args*⟩ ::= ⟨*arg-list*⟩ || ⟨*empty*⟩

⟨*arg-list*⟩ ::= ⟨*arg-list*⟩ , ⟨*expression*⟩ | ⟨*expression*⟩

⟨***letter_***⟩ ::= a | ... | z | A | ... | Z | _

⟨***digit***⟩ ::= 0 | ... | 9

⟨***ID***⟩ ::= ⟨*letter_*⟩ (⟨*letter_*⟩ | ⟨*digit*⟩)*

⟨***NUM***⟩ ::= ⟨*INT*⟩ | ⟨*FLOAT*⟩

⟨***INT***⟩ ::= ⟨*digit*⟩$^+$

⟨***FLOAT***⟩ ::= ⟨*digit*⟩$^+$. | ⟨*digit*⟩$^+$

Os *tokens* da linguagem são: '**,**', '**.**', '>=', '>', '<', '<=', '==', '**!=**', '**int**', '**float**', '**while**', '**return**', '**if**', '**else**', '**mat**', '**EOL**', '+', '**-**', '*', '/', '**&&**', '^', '||', '**!**', '**&**', '**;**', '**(**', '**)**', '[', ']', '**,** ' e '//', sendo este último um caso especial por delimitar comentários de uma linha. *EOL* simboliza o final de uma linha qualquer, e as seguintes palavras são reservadas e portanto não podem ser utilizadas como identificadores: **else**, **float**, **if**, **int**, **mat** e **return**. A regra de produção *ASCII* produz qualquer símbolo contido na tabela *ascii*. Por fim, não constam na gramática, mas as funções *scan* e *print* fazem parte da linguagem, sendo responsáveis pelas operações de leitura e impressão na saída padrão do ambiente de execução do programa (usualmente, o console).

## 5 Semântica

A semântica da linguagem é quase idêntica à da linguagem C: declarações de variável (a menos das do tipo **mat**), funções e expressões têm semântica similar. Sendo esta linguagem uma estensão de um subconjunto da linguagem *C*, a principal diferença está no tipo de dados **mat** (abreviação de *matrix*, "matriz"em inglês). 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. Entretando, 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. Tais operações mais complexas estarão presentes por meio de funções padrão da linguagem.

## 6 Exemplo de programa na linguagem

A seguir, trechos de código pertencente à nova linguagem.

```
int foo(void);
mat<int> matriz[3][4];  // Criacao de um sistema
matrix = {
  {1, 2, 3, 4},
  {4, 3 ,2, 1},
  {-1, -2, -3, -4},
  {45, 87, 9, 18}
};

int main(){
  int x;
  mat<float> casted = matriz + matriz;
  scan(x);
  print(x);
}
```

## 7 Agradecimentos

Agradecimentos ao autor anônimo 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 4.

CROCRO. Disponível em: <https://tex.stackexchange.com/questions/348651/c-code-to-add-in-the-document>. Citado na página 4.

MUNN, A. Disponível em: <https://tex.stackexchange.com/questions/24886/which-package-can-be-used-to-write-bnf-grammars/39751>. Citado na página 4.