# Tradutores - Analisador Léxico

Thiago Veras Machado[160146682]

Universidade de Brasília cic@unb.com

## 1 Introdução / Motivação

O projeto da disciplina Tradutores tem como principal objetivo estudar os aspectos teóricos relacionados à implementação de tradutores quanto à prática de sua implementação.

Nesse trabalho será implementado um tradutor para a linguagem C no qual utilizaremos como base o livro da disciplina [ALSU07].

Para efeito de simplificação a linguagem foi limitada a um conjunto básico de comandos: comando condicional, comando de repetição, tratamento de expressões aritméticas e chamada a sub-rotinas.

Durante a implementação do analisador sintático, a professora Cláudia Nalon solicitou a integração da estrutura de dados *set*. Juntamente com essa primitiva, novos métodos serão acrescentados (add, remove, in, is_set, exists e forall) nos quais executam tais operações em um *set* já declarado.

## 2 Descrição da análise léxica

C é uma linguagem de propósito geral, compilada, de alto nível, com sintaxe estruturada, imperativa e possui tipagem estática.Para o trabalho a linguagem conterá as seguintes estruturas básicas:

- Estrutura condicional: `if` e `else`.
- Estrutura de repetição: `for`.
- Tipos de dados: `int` (números inteiros), `float` (números no formato de ponto flutuante) `set` (estrutura de dados sobre conjuntos) e `elem` (tipo de dado polimórifoco).
- Definição e chamada de subrotinas com passagem de parâmetros.
- Operações aritméticas (`+`, `-`, `*`, `/`), lógicas básicas (`||`, `&&`, `!`) e relacionais (`==`, `>`, `<`, `!=`, `>=`, `<=`).

Além da utilização do [ALSU07], também foi utilizado o [Est], que consiste em uma ferramenta geradora de programas que reconhecem padrões léxicos em textos. A estrutura do código flex foi pensada e implementada com o intuito de facilitar a criação da tabela de simbolos futuramente.

## 3 Descrição dos arquivos de teste

Os arquivos de testes se encontram na pasta ***Input*** no qual possuem o prefixo *error_* representam os testes que possuem erros a partir de um análise léxica, análogo para os arquivos que possuem o prefixo *success_*.

Arquivos de error:

error_1.c

1. `ERROR line: 3 columns: 5 Undentified char: #`
2. `ERROR line: 5 columns: 5 Undentified char: #`

error_2.c

1. `ERROR line: 5 columns: 6 Undentified char: .`

## 4 Compilação e execução do programa.

Para facilitar a compilação e execução do programa, assim como rodar todos os testes e verificar se o comportamento foi o esperado, criei um script em *bash* que executa todas as etapas automaticamente para você.

A função do script é gerar o arquivo *yy.c* oriundo do *flex.l* e depois compilar esse arquivo *yy.c*, rodar todos os testes e comparar com a saída esperada. Para isso, basta rodar o comando:

```
bash run.sh
```

Caso queira roda individualmente cada teste com o intuito de ver a análise de cada token, basta rodar:

```
./a.out Input/NOME_DO_ARQUIVO.C
```

Ambiente utilizado para a criação do trabalho:

| SO | Windows 10 Enterprise 64-bit |
|---|---|
| Terminal | WSL |
| MEM | 16GB |

## Referências

[ALSU07] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, & Tools*. Pearson/Addison Wesley, 2nd edition, 2007.

[Est] Will Estes. Flex: Fast lexical analyzer generator. online; Acessado 24/02/2021.

[Gup13] Ajay Gupta. The syntax of c in backus-naur form, 2013. online; Acessado 24/02/2021.

# A Gramática

A gramática abaixo descreve a linguagem para qual o compilador será implementado [Gup13]. Adaptações foram feitas para se encaixar no escopo do trabalho.

⟨*start*⟩ ::= ⟨*program*⟩
⟨*program*⟩ ::= ⟨*function-definition*⟩
| ⟨*function-definition*⟩ ⟨*program*⟩
| ⟨*variables-declaration*⟩ ⟨*program*⟩
⟨*function-definition*⟩ ::= ⟨*function-declaration*⟩ ⟨*parameters*⟩ ⟨*function-body*⟩
⟨*function-declaration*⟩ ::= ⟨*type-identifier*⟩ ⟨*id*⟩
⟨*function-body*⟩ ::= '{' ⟨*statements*⟩ '}'
⟨*parameters*⟩ ::= '(' ⟨*parameters-list*⟩ ')'
⟨*parameters-list*⟩ ::= ⟨*parameter*⟩ ',' ⟨*parameters-list*⟩
| ⟨*parameter*⟩
⟨*parameter*⟩ ::= ⟨*type-identifier*⟩ ⟨*id*⟩
⟨*type-identifier*⟩ ::= ⟨*basic-type*⟩ | ⟨*elem*⟩ | ⟨*set*⟩
⟨*statements*⟩ ::= ⟨*statement*⟩ ⟨*statements*⟩
| ⟨*statement*⟩
| '`{`' ⟨*statements*⟩ '`}`'
⟨*statement*⟩ ::= ⟨*variables-declaration*⟩
| ⟨*return*⟩
| ⟨*conditional*⟩
| ⟨*for*⟩
| ⟨*expression_statement*⟩ '`;`'
| ⟨*io_statement*⟩
| ⟨*set_statement*⟩
⟨*set_statement*⟩ ::= '`is_set`' '`(`' ⟨*id*⟩ | ⟨*set_expression*⟩ '`)`';
| '`forall` `(`' ⟨*id*⟩ '`in`' ⟨*set_expression*⟩ '`)`' ⟨*statement*⟩
⟨*set_expression*⟩ ::= ⟨*set_operation_1*⟩ '`(`' '`in`' ⟨*expression*⟩ '`)`'
⟨*expression_statement*⟩ ::= ⟨*expression*⟩ '`;`' | '`;`'
⟨*expression*⟩ ::= ⟨*expression*⟩ = ⟨*expression*⟩ | ⟨*expression-1*⟩ | ⟨*set_expression_1*⟩
⟨*set_expression_1*⟩ ::= ⟨*expression-1*⟩ '`in`' ⟨*set_expression*⟩
⟨*set_operation_1*⟩ ::= '`add`' | '`remove`' | '`exists`' |
⟨*for*⟩ ::= for '`(`' ⟨*for_expression*⟩ '`)`' ⟨*statement*⟩
⟨*for_expression*⟩ ::= ⟨*expression*⟩? '`;`' ⟨*expression*⟩? '`;`' ⟨*expression*⟩?
⟨*io_statement*⟩ ::= read '`(`' ⟨*id*⟩ '`)`' '`;`'
| write'`(`' ⟨*string*⟩ | ⟨*expression*⟩ '`)`' '`;`'
| writeln'`(`' ⟨*string*⟩ | ⟨*expression*⟩'`)`' '`;`'
⟨*arguments_list*⟩ ::= ⟨*arguments_list*⟩ '`,`' ⟨*value*⟩
| ⟨*value*⟩
⟨*conditional*⟩ ::= '`if`' ⟨*conditional-expression*⟩ ⟨*statements*⟩
| '`if`' ⟨*conditional-expression*⟩ ⟨*statements*⟩ '`else`' ⟨*statements*⟩

⟨*conditional-expression*⟩ ::= '`(`' ⟨*expression*⟩ '`)`'
⟨*return*⟩ ::= '`return`' ⟨*value*⟩ '`;`'
| 'return :'
⟨*value*⟩ ::= ⟨*id*⟩
| ⟨*const*⟩
| ⟨*function_call*⟩
⟨*function_call*⟩ ::= ⟨*id*⟩ '`(`' ⟨*arguments-list*⟩ '`)`'
| ⟨*id*⟩ '`(`' '`)`'
⟨*variables-declaration*⟩ ::= ⟨*type-identifier*⟩ ⟨*identifiers_list*⟩ '`;`'
⟨*expression-1*⟩ ::= ⟨*expression-1*⟩ ⟨*operation_1*⟩ ⟨*expression-2*⟩
| ⟨*expression-2*⟩
⟨*expression-2*⟩ ::= ⟨*expression-2*⟩ ⟨*operation_2*⟩ ⟨*expression-3*⟩
| ⟨*expression-3*⟩
⟨*expression-3*⟩ ::= ⟨*value*⟩ ⟨*operation_3*⟩ ⟨*expression-3*⟩
| ⟨*value*⟩
| -⟨*value*⟩
⟨*const*⟩ ::= ⟨*integer*⟩ | ⟨*float*⟩ | ⟨*empty*⟩
⟨*integer*⟩ ::= ⟨*digit*⟩+
⟨*float*⟩ ::= ⟨*digit*⟩'.'⟨*digit*⟩
⟨*basic-type*⟩ ::= '`int`' | '`float`'
⟨*elem*⟩ ::= '`elem`'
⟨*set*⟩ ::= '`set`'
⟨*empty*⟩ ::= '`EMPTY`'
⟨*set_operation*⟩ ::= '`add`' | '`remove`' | '`is_set`' | '`exists`' | '`forall`'
⟨*string*⟩ ::= `\"[^\"]*\"|'[^']*`
⟨*digit*⟩ ::= `[0-9]`
⟨*id*⟩ ::= `[a-zA-Z_][_a-z0-9A-Z]*`
⟨*operation_1*⟩ ::= '`>`' | '`<`' | '`==`' | '`>=`' | '`<=`'
⟨*operation_2*⟩ ::= '`+`' | '`-`' | '`&&`' | '`||`'
⟨*operation_3*⟩ ::= '`*`' | '`/`'