

# INE5426 - Construção de Compiladores

## Analizador Léxico

**Entrega: 26 de fevereiro de 2021 (até 23:55h via Moodle)**

Este Exercício-Programa (EP) pode ser realizado por grupos (com 2 ou 3 integrantes). Cada grupo deverá executar uma tarefa:

- A construção de um analisador léxico para uma linguagem (**AL**).

Os resultados obtidos pelos grupos serão avaliados através da produção de um **relatório das atividades** desenvolvidas e através da compilação/interpretação/uso/execução do analisador léxico (tarefa **AL**).

Trabalharemos com a gramática denominada CC-2020-2 (disponível no fim deste texto e na Forma de Backus-Naur – BNF). Se desejarem, os grupos poderão realizar *pequenas* modificações na linguagem. No entanto, qualquer modificação deverá ser detalhada no relatório.

A nota deste EP é  $T_1 = \min\{(T_{1.1} + T_{1.2} + \epsilon), 10.0\}$ , onde  $T_{1.1}$  está definida na seção 4,  $T_{1.2}$  está definida na seção 5 e  $\epsilon \in [0 : 0,5]$  depende da participação de todos os integrantes de um grupo nas aulas de laboratório.

## 1 Tarefa AL

A tarefa **AL** consiste na implementação de um analisador léxico para a linguagem gerada por CC-2020-2. É permitido o uso de ferramentas que constroem um analisador léxico. No entanto, as tarefas que deverão ser descritas no relatório são:

1. identificação dos *tokens*;
2. produção das **definições regulares** para cada *token*;
3. construção dos **diagramas de transição** para cada *token*;
4. descrição de uma **tabela de símbolos** (como foi implementada, quais são os símbolos armazenados na tabela e quais são os atributos dos símbolos escolhidos para armazenar na tabela);
5. se não usou ferramenta, uma descrição da implementação do analisador léxico (Usou diagramas de transição? Quais? Quantos? Se não usou diagramas de transição, então o que foi usado?); e
6. se usou ferramenta, uma descrição detalhada da entrada exigida pela ferramenta e da saída dada por ela. É necessário haver exemplos pequenos da entrada e da saída gerada pela ferramenta com essa entrada.

Todos os integrantes dos grupos devem dominar qualquer questão relacionada a essa tarefa.

## 2 O que deve ser entregue?

A data para entregar o EP é dia 26 de fevereiro de 2021 (até 23:55h via Moodle). Cada grupo deverá entregar um conjunto de arquivos com:

1. um relatório com uma descrição das atividades realizadas (em PDF).
  - as respostas da tarefa **AL** devem ser descritas no relatório;
2. um conjunto de arquivos que definem o analisador léxico (pode ser um único arquivo);
3. três programas escritos na linguagem LCC-2020-2 gerada por CC-2020-2 (com pelo menos 100 linhas cada, sem erros léxicos);
4. um *Makefile* para compilação/interpretação/execução do analisador léxico;
5. um README com informações importantes para a execução apropriada de todos os programas desenvolvidos.

Orientações para construção de um *Makefile*: <https://www.gnu.org/software/make/manual/make.html>

## 3 Sobre as compilações/interpretações/execuções dos trabalhos

No momento da execução dos programas desenvolvidos por um grupo, a presença (virtual) de seus integrantes poderá ser necessária para a efetiva avaliação.

## 4 O que será avaliado no relatório?

Chamamos de  $T_{1,1}$  a nota para a avaliação dos relatórios.  $0 \leq T_{1,1} \leq 5$ . Se algum grupo não apresentar o relatório, então  $T_{1,1} = 0$ . A avaliação considerará a organização do texto e a qualidade da descrição das tarefas realizadas. A presença (virtual) dos integrantes de um grupo poderá ser necessária para a efetiva avaliação.

## 5 O que será avaliado na execução/uso do analisador léxico

Chamamos de  $T_{1,2}$  a nota para a avaliação da execução do analisador léxico.  $0 \leq T_{1,2} \leq 5$ . Abaixo listamos itens importantes com relação a essa avaliação.

- A existência de três programas para LCC-2020-2 com extensão .lcc e com pelo menos 100 linhas cada, sem erros léxicos (se não existir os três nas condições citadas, então  $T_{1,2} = 0$ );
- A existência de um *Makefile* (se algum não existir, então  $T_{1,2} = 0$ );
- A execução correta do *Makefile* (se algum não executar corretamente, então  $T_{1,2} = 0$ );
- A existência de um README (se não existir, então  $T_{1,2} = 0$ );
- A compilação/interpretação dos programas desenvolvidos (se houver erros de compilação/interpretação, então haverá descontos em  $T_{1,2}$ );
- A execução do seu programa em conjunto com uma ferramenta (se for o caso);

## 6 Sobre a entrada e a saída dos dados

Uma única entrada será dada: o caminho de um arquivo no formato `lcc` escrito na linguagem LCC-2020-2 derivada por CC-2020-2.

Exemplo de uma entrada: `/tmp/arvore-binaria-de-busca.lcc`.

As seguintes saídas são esperadas:

- para o analisador léxico:
  - se não houver erros léxicos → uma lista de *tokens* (na mesma ordem em que eles ocorrem no arquivo dado na entrada) e uma tabela de símbolos;
  - se houver erros léxicos → uma mensagem simples de erro léxico indicando a linha e a coluna do arquivo de entrada onde ele ocorre.

### Observações importantes:

1. Os programas podem ser escritos em C (compatível com compilador gcc versão 7.5.0), C++ (compatível com compilador g++ versão 7.5.0), Java (compatível com compilador javac versão 11.0.10), Python 3 (compatível com versão 3.6.9) ou Rust (compatível com rustc versão 1.47.0) e deve ser compatível com Linux/Unix.
2. Se for desenvolver em Python 3, então especifique (no *Makefile* principalmente) qual é a versão que está usando. Prepare seu *Makefile* considerando a versão usada.
3. Exercícios-Programas atrasados **não** serão aceitos.
4. Programas com *warning* na compilação terão diminuição da nota.
5. É importante que seu programa esteja escrito de maneira a destacar a estrutura do programa.
6. O relatório e o programa devem começar com um cabeçalho contendo pelo menos o nome de todos os integrantes do grupo.
7. Coloque comentários em pontos convenientes do programa, e faça uma saída clara.
8. A entrega do Exercício-Programa deverá ser feita no Moodle.
9. O Exercício-Programa é individual por grupo. Não copie o programa de outro grupo, não empreste o seu programa para outro grupo, e tome cuidado para que não copiem seu programa sem a sua permissão. Todos os programas envolvidos em cópias terão nota  $T_1$  igual a ZERO.

Bom trabalho!

Abaixo encontra-se a gramática CC-2020-2 na forma BNF. Ela é fortemente baseada na gramática X++ de Delamaro (veja bibliografia no plano de ensino). Os símbolos terminais de CC-2020-2 estão na cor amarela. Os terminais não-triviais são somente *ident*, *int\_constant*, *float\_constant* e *string\_constant*. Os símbolos não-terminais de CC-2020-2 estão em letra de forma. Os demais símbolos (na cor azul) são símbolos da notação BNF. Consulte o livro de Delamaro para mais informações sobre a notação BNF (seção 2.3 - página 12).

Livro do Delamaro: <http://conteudo.icmc.usp.br/pessoas/delamaro/SlidesCompiladores/CompiladoresFinal.pdf>

<i>PROGRAM</i>	→ ( <i>STATEMENT</i>   <i>FUNCLIST</i> )?
<i>FUNCLIST</i>	→ <i>FUNCDEF</i> <i>FUNCLIST</i>   <i>FUNCDEF</i>
<i>FUNCDEF</i>	→ <i>def ident</i> ( <i>PARAMLIST</i> ){ <i>STATELIST</i> }
<i>PARAMLIST</i>	→ (( <i>int</i>   <i>float</i>   <i>string</i> ) <i>ident</i> , <i>PARAMLIST</i>   ( <i>int</i>   <i>float</i>   <i>string</i> ) <i>ident</i> )?
<i>STATEMENT</i>	→ ( <i>VARDECL</i> ;   <i>ATRIBSTAT</i> ;   <i>PRINTSTAT</i> ;   <i>READSTAT</i> ;   <i>RETURNSTAT</i> ;   <i>IFSTAT</i>   <i>FORSTAT</i>   { <i>STATELIST</i> }   <i>break</i> ;   ; )
<i>VARDECL</i>	→ ( <i>int</i>   <i>float</i>   <i>string</i> ) <i>ident</i> ([ <i>int_constant</i> ])*
<i>ATRIBSTAT</i>	→ <i>LVALUE</i> = ( <i>EXPRESSION</i>   <i>ALLOCEXPRESSION</i>   <i>FUNCCALL</i> )
<i>FUNCCALL</i>	→ <i>ident</i> ( <i>PARAMLISTCALL</i> )
<i>PARAMLISTCALL</i>	→ ( <i>ident</i> , <i>PARAMLISTCALL</i>   <i>ident</i> )?
<i>PRINTSTAT</i>	→ <i>print</i> <i>EXPRESSION</i>
<i>READSTAT</i>	→ <i>read</i> <i>LVALUE</i>
<i>RETURNSTAT</i>	→ <i>return</i>
<i>IFSTAT</i>	→ <i>if</i> ( <i>EXPRESSION</i> ) <i>STATEMENT</i> ( <i>else</i> <i>STATEMENT</i> )?
<i>FORSTAT</i>	→ <i>for</i> ( <i>ATRIBSTAT</i> ; <i>EXPRESSION</i> ; <i>ATRIBSTAT</i> ) <i>STATEMENT</i>
<i>STATELIST</i>	→ <i>STATEMENT</i> ( <i>STATELIST</i> )?
<i>ALLOCEXPRESSION</i>	→ <i>new</i> ( <i>int</i>   <i>float</i>   <i>string</i> ) ([ <i>NUMEXPRESSION</i> ])+
<i>EXPRESSION</i>	→ <i>NUMEXPRESSION</i> (( <   >   <=   >=   ==   != ) <i>NUMEXPRESSION</i> )?
<i>NUMEXPRESSION</i>	→ <i>TERM</i> ((+   -) <i>TERM</i> )*
<i>TERM</i>	→ <i>UNARYEXPR</i> (( *   \   %) <i>UNARYEXPR</i> )*
<i>UNARYEXPR</i>	→ ((+   -)? <i>FACTOR</i>
<i>FACTOR</i>	→ ( <i>int_constant</i>   <i>float_constant</i>   <i>string_constant</i>   <i>null</i>     <i>LVALUE</i>   ( <i>NUMEXPRESSION</i> ))
<i>LVALUE</i>	→ <i>ident</i> ( [ <i>NUMEXPRESSION</i> ] )*