INE5426 - Construção de Compiladores

Analisador Léxico e Analisador Sintático

Entrega: 08 de outubro de 2019 (até 23:55h via Moodle)

Este Exercício-Programa (EP) pode ser realizado por grupos (com no mínimo 2 e no máximo 4 integrantes). Cada grupo deverá executar duas tarefas.

- A construção de um analisador léxico para uma linguagem (**AL**); e
- A construção de um analisador sintático para uma linguagem (**AS**).

Os resultados obtidos pelos grupos serão avaliados através da apresentação de um **relatório das atividades** desenvolvidas e através da compilação/interpretação/uso/execução do analisador léxico (tarefa **AL**) e do analisador sintático (tarefa **AS1**).

Trabalharemos com a linguagem denominada `CC-2019-2` (disponível no fim deste texto e na Forma de Backus-Naur - BNF). 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 5, $T_{1.2}$ está definida na seção 6 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 `CC-2019-2`. É permitido o uso de ferramentas que constrõem um analisador léxico. No entanto, as tarefas que deverão ser descritas no relatório são:

1. identificação dos *tokens*;
2. construção dos **diagramas de transição** para cada *token*;
3. 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
4. se usou ferramenta, uma descrição da entrada exigida pela ferramenta e da saída dada por ela.

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

2 Tarefa AS

A tarefa **AS** consiste na construção de um analisador sintático preditivo para uma gramática em LL(1) e na execução das seguintes questões:

1. `CC-2019-2` está na forma BNF. Coloque-a na forma *convencional* de gramática. Chame tal gramática de `CCC-2019-2`.

2. `CCC-2019-2` possui recursão à esquerda? Justifique detalhadamente sua resposta. Se ela tiver recursão à esquerda, então remova tal recursão.

3. `CCC-2019-2` está fatorada à esquerda? Justifique detalhadamente sua resposta. Se ela não tiver fatorada à esquerda, então fatore.

4. Faça `CCC-2019-2` ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, se achar necessário. Depois disso, mostre que `CCC-2019-2` está em LL(1) (você pode usar o Teorema ou a tabela de reconhecimento sintático vistos em sala de aula).

5. se não usou ferramenta, uma descrição da implementação do analisador sintático (Usou uma tabela de reconhecimento sintático para gramáticas em LL(1)? Se não, então o que foi usado?); e

6. se usou ferramenta, uma descrição da entrada exigida pela ferramenta e da saída dada por ela.

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

3 O que deve ser entregue?

A data para entregar o EP é dia 08 de outubro (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 das tarefas **AL** e **AS** devem ser descritas no relatório.
2. um conjunto de arquivos que definem o analisador léxico (pode ser um único arquivo);
3. um conjunto de arquivos que definem o analisador sintático (pode ser um único arquivo);
4. três programas escritos na linguagem gerada por `CCC-2019-2` (com pelo menos 100 linhas cada, sem erros léxicos e sem erros sintáticos);
5. um *Makefile* para compilação/interpretação/execução do analisador léxico;
6. um *Makefile* para compilação/interpretação/execução do analisador sintático; e
7. 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`

4 Sobre as compilações/interpretações/execuções dos trabalhos

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

5 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.

6 O que será avaliado na execução/uso do analisador léxico e sintático

Chamamos de $T_{1.2}$ a nota para a avaliação da execução do analisador léxico e sintático. $0 \leq T_{1.2} \leq 5$. Abaixo listamos itens importantes com relação a essa avaliação.

- A existência de três programas para `CCC-2019-2` com extensão .ccc e com pelo menos 100 linhas cada, sem erros léxicos e sem erros sintáticos (se não existir os três nas condições citadas, então $T_{1.2} = 0$);
- A existência de *Makefiles* (se algum não existir, então $T_{1.2} = 0$);
- A execução correta dos *Makefiles* (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);

7 Sobre a entrada e a saída dos dados

Uma única entrada será dada: o caminho de um arquivo no formato `ccc` escrito na linguagem derivada por `CCC-2019-2`.
Exemplo de uma entrada: `/tmp/arvore-binaria-de-busca.ccc`.
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.
- para o analisador sintático:
  - se não houver erros sintáticos → uma mensagem de sucesso
  - se houver erros sintáticos → uma mensagem de insucesso indicando qual é a entrada na tabela de reconhecimento sintático que está vazia (qual é a forma sentencial α, qual é o símbolo não-terminal mais à esquerda de α e qual é o *token* da entrada).

Observações importantes:

1. Os programas podem ser escritos em C (compatível com compilador gcc versão 7.4.0), C++ (compatível com compilador g++ versão 7.4.0), Java (compativel com compilador javac versão 11.0.4) ou Python (compatível com versão 3.6.8), e deve ser compatível com Linux/Unix.

2. Se for desenvolver em Python, 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-2019-2` na forma BNF. Ela é fortemente baseada na gramática X++ de Delamaro (veja bibliografia no plano de ensino). Os símbolos terminais de `CC-2019-2` estão na cor amarela. Os terminais não-triviais são somente *int_contant*, *float_constant* e *string_constant*. Os símbolos não-terminais de `CC-2019-2` 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`

$$
\begin{array}{lcl}
PROGRAM & \rightarrow & (STATEMENT)? \\
STATEMENT & \rightarrow & (VARDECL; \mid \\
 & & ATRIBSTAT; \mid \\
 & & PRINTSTAT; \mid \\
 & & READSTAT; \mid \\
 & & RETURNSTAT; \mid \\
 & & IFSTAT; \mid \\
 & & FORSTAT; \mid \\
 & & \{STATELIST\} \mid \\
 & & break\ ; \mid \\
 & & ;) \\
 & & \\
VARDECL & \rightarrow & (\ int \mid float \mid string\ )\ ident\ ([int])^* \\
ATRIBSTAT & \rightarrow & LVALUE = (\ EXPRESSION \mid ALLOCEXPRESSION) \\
PRINTSTAT & \rightarrow & print\ EXPRESSION \\
READSTAT & \rightarrow & read\ LVALUE \\
RETURNSTAT & \rightarrow & return \\
IFSTAT & \rightarrow & if(\ EXPRESSION\ )\ STATEMENT\ (else\ STATEMENT)? \\
FORSTAT & \rightarrow & for(ATRIBSTAT;\ NUMEXPRESSION;\ ATRIBSTAT) \\
 & & STATEMENT \\
STATELIST & \rightarrow & STATEMENT\ (STATELIST)? \\
ALLOCEXPRESSION & \rightarrow & new\ (int \mid float \mid string)\ ([\ EXPRESSION\ ])^+ \\
EXPRESSION & \rightarrow & NUMEXPRESSION((< \mid > \mid <= \mid >= \mid == \mid !=) \\
 & & NUMEXPRESSION)? \\
NUMEXPRESSION & \rightarrow & TERM\ ((+ \mid -)\ TERM)^* \\
TERM & \rightarrow & UNARYEXPR((* \mid \backslash \mid \%)\ UNARYEXPR)^* \\
UNARYEXPR & \rightarrow & ((+ \mid -))?\ FACTOR \\
FACTOR & \rightarrow & (int\_contant \mid float\_constant \mid string\_constant \mid null \mid \\
 & & \mid LVALUE \mid (EXPRESSION)) \\
LVALUE & \rightarrow & ident(\ [EXPRESSION]\ )^*
\end{array}
$$