INE5622 - Introdução a Compiladores

Analisador Léxico e Sintático

Entrega: 11 de março de 2022 (até 23:55h via Moodle)

Este Exercício-Programa (EP) pode ser realizado por grupos (com até 4 integrantes). Cada grupo deverá executar as seguintes tarefas:

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

Os resultados obtidos pelos grupos serão avaliados através da produção de um **relatório das atividades** desenvolvidas e através da compilação/interpretação/uso/execução dos analisadores léxico e sintático (tarefas **AL** e **AS**).

Trabalharemos com uma linguagem denominada `LCC-2021-2`. Os *tokens* associados a essa linguagem estão disponível no fim deste texto. Se desejarem, os grupos poderão realizar *pequenas* modificações na linguagem. No entanto, qualquer modificação deverá ser detalhada no relatório.

A nota deste EP é $T = T_1 + T_2$, onde T_1 está definida na seção 5 e T_1 está definida na seção 6.

1 Tarefa AL

A tarefa **AL** consiste na implementação de um analisador léxico para uma linguagem de programação. Mais detalhes dessa linguagem serão dados no próximo trabalho. É permitido o uso de ferramentas que constrõem um analisador léxico. No entanto, as tarefas que deverão ser descritas no relatório são:

1. produção das **definições regulares** para cada *token*;
2. construção dos **diagramas de transição** para cada *token*;
3. descrição de uma **tabela de símbolos** (a tabela deverá guardar tokens *identificadores* seguido por pelo menos um atributo do token guardado na tabela);
4. 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
5. 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 Tarefa AS

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

1. `CC-2021-2` está na forma BNF. Coloque-a na forma *convencional* de gramática. Chame tal gramática de `LCC-2021-2`.

2. A sua `LCC-2021-2` possui recursão à esquerda? Justifique detalhadamente sua resposta. Se ela tiver recursão à esquerda, então remova tal recursão.

3. A sua `LCC-2021-2` está fatorada à esquerda? Justifique detalhadamente sua resposta. Se ela não estiver fatorada à esquerda, então fatore.

4. Faça `LCC-2021-2` ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, se achar necessário. Depois disso, mostre que `LCC-2021-2` está em LL(1) (você pode usar o Teorema ou a tabela de reconhecimento sintático vistos em videoaula).

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.

Mais uma vez, 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 11 de março de 2022 (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;
   - as respostas da tarefa **AS** devem ser descritas no relatório.

2. um conjunto de arquivos que definem os analisadores léxico e sintático (pode ser um único arquivo ou vários arquivos);

3. três programas escritos na linguagem gerada por `LCC-2021-2` (com pelo menos 100 linhas cada, sem erros léxicos e sem erros sintáticos);

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`

4 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.

5 O que será avaliado no relatório?

Chamamos de T_1 a nota para a avaliação dos relatórios. $0 \leq T_1 \leq 5$. Se algum grupo não apresentar o relatório, então $T_1 = 0$. 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.

6 O que será avaliado na execução/uso dos analisadores léxico e sintático

Chamamos de T_2 a nota para a avaliação da execução dos analisadores léxico e sintático. $0 \leq T_2 \leq 5$. Abaixo listamos itens importantes com relação a essa avaliação.

- A existência de três programas para `LCC-2021-2` com extensão .lcc e com pelo menos 100 linhas cada, sem erros léxicos e sem erros sintáticos (se não existir os três nas condições citadas, então $T_2 = 0$);
- A existência de um *Makefile* (se algum não existir, então $T_2 = 0$);
- A execução correta do *Makefile* (se algum não executar corretamente, então $T_2 = 0$);
- A existência de um README (se não existir, então $T_2 = 0$);
- A compilação/interpretação dos programas desenvolvidos (se houver erros de compilação/ interpretação, então haverá descontos em T_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 `lcc` escrito na linguagem `LCC-2021-2` derivada por `CC-2021-2`.
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.
- 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 9.3.0), C++ (compatível com compilador g++ versão 9.3.0), Java (compatível com compilador javac versão 11.0.2, Python 3 (compatível com versão 3.8.10) ou Rust (compatível com versão 1.53.0).

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 igual a ZERO.

Bom trabalho!

Na próxima página você encontrará uma gramática `CC-2021-2` na forma BNF. Assim como as expressões regulares, as gramáticas também descrevem linguagens. No entanto, as gramáticas são mais poderosas pois podem descrever linguagens que expressões regulares não podem. A gramática abaixo é fortemente baseada na gramática X++ de Delamaro. Para o trabalho relacionado ao analisador léxico, precisamos destacar os *tokens*. Eles são precisamente os *símbolos terminais* de `CC-2021-2` e que estão na cor amarela. Os símbolos terminais não-triviais são somente *ident*, *int_constant*, *float_constant* e *string_constant*. Os *símbolos não-terminais* de `CC-2021-2` 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`

Em seguida apresentamos uma entrada (um código fonte) da linguagem descrita pela gramática `LCC-2021-2`.

```
def func1(int A, int B)
{
  int SM[2];
  SM[0] = A + B;
  SM[1] = B * C;
  return;
}

def principal()
{
  int C;
  int D;
  int R;
  C = 4;
  D = 5;
  R = func1(C, D);
  return;
}
```

Para este exemplo, a lista de tokens é [def, ident, (, int, ident, ,, int, ident,), {, int, ident, ..., return, ;, }]. A lista de tokens que deve ser elaborada pelo seu analisador léxico deverá ser similar a essa. A tabela de símbolos deverá conter uma entrada para cada token ident com pelo menos um atributo (por exemplo, a linha onde ocorre tal token – `SM` ocorre nas linhas 3, 4 e 5).

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