INE5622 - Introdução a Compiladores

Analisador Léxico e Sintático

Entrega: até 18 de julho de 2024 (até 23:59h via Moodle)

Este trabalho 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**).

A avaliação será realizada através da análise léxica e sintática para algumas entradas.

Trabalharemos com uma linguagem denominada `LSI-2024-1`. Os *tokens* (os terminais da gramática) associados a essa linguagem estão disponíveis no fim deste texto. Se desejarem, os grupos poderão realizar *pequenas* modificações na linguagem. No entanto, qualquer modificação deverá ser detalhada no cabeçalho dos arquivos que definem os analisadores léxico e sintático.

A nota deste trabalho é $T = T_1 + T_2 + T_3$, onde T_1 está definida na seção 3, T_2 está definida na seção 4 e T_3 está definida na seção 5.

1 Tarefa AL

A tarefa **AL** consiste na implementação de um analisador léxico para a linguagem de programação dada. O analisador léxico precisa necessariamente ler caracter por caracter da entrada e deve ser baseado em diagramas de transição (autômatos com caracterização da saída).

Todos os integrantes dos grupos devem dominar qualquer questão relacionada à tarefa **AL**.

2 Tarefa AS

A tarefa **AS** consiste na implementação de um analisador sintático para uma linguagem de programação. A gramática associada (`LSI-2024-1`) **deve** estar em LL(1). O analisador sintático deve construir uma tabela de reconhecimento sintático uma única vez. Depois disso, o analisador deverá usar essa tabela para demonstrar (ou não) que um input dado pertence à linguagem gerada pela gramática (`LSI-2024-1`). Abaixo há algumas sugestões que podem ajudar a alcançar tal objetivo.

1. Remova recursão à esquerda de `LSI-2024-1`, se existir.
2. Fatore `LSI-2024-1` à esquerda, se ela não estiver fatorada.
3. Faça `LSI-2024-1` ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, caso necessário. Utilize o teorema visto em aula para demostrar que a gramática resultante está em LL(1).
4. Depois que `LSI-2024-1` estiver em LL(1), construa a tabela de reconhecimento sintático.

Mais uma vez, todos os integrantes dos grupos devem dominar qualquer questão relacionada à tarefa **AS**.

3 O que deve ser entregue?

A nota para esta entrega será T_1, sendo $(0 \leq T_1 \leq 3,0)$. A data para entregar o EP é até o dia 18 de julho de 2024 (até 23:59h via Moodle). Cada grupo deverá entregar um conjunto de arquivos com:

1. As definições dos analisadores léxico e sintático (pode ser um único arquivo ou vários arquivos);
2. Um programa escrito na linguagem `LSI-2024-1` (com pelo menos 50 linhas, sem erros léxicos e sem erros sintáticos);
3. Três programas escritos na linguagem `LSI-2024-1` (cada um com pelo menos 15 linhas), contendo erros léxicos e o resultado da captura desses erros pelo seu analisador;
4. Três programas escritos na linguagem `LSI-2024-1` (cada um com pelo menos 15 linhas), contendo erros sintáticos e o resultado da captura desses erros pelo seu analisador;
5. Um README com instruções para a execução apropriada de todos os programas desenvolvidos em um sistema operacional Linux.

4 O que será avaliado na análise léxica?

T_2 será a nota para a avaliação da análise léxica $(0 \leq T_2 \leq 3,0)$. Será observada a execução do analisador léxico lendo caracter por caracter do input e baseado em diagramas de transição. Serão inseridos erros léxicos na entrada que deverão ser capturados pelo analisador léxico.

5 O que será avaliado na análise sintática?

T_3 será a nota para a avaliação da execução da análise sintática $(0 \leq T_3 \leq 4,0)$. Será observada a construção da tabela de reconhecimento sintático (uma única vez), e a execução do analisador sintático para `LSI-2024-1` em LL(1). Serão inseridos erros sintáticos na entrada que deverão ser capturados pelo analisador sintático.

6 Sobre as execuções dos programas desenvolvidos

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

7 Itens que devem ser considerados

- A existência de sete programas de teste, conforme seção 3, para `LSI-2024-1` com extensão .lsi (caso algum dos sete testes esteja faltando, então $T = 0$);
- A execução correta dos programas entregues (se algum não executar corretamente, então $T = 0$);
- A existência de um README com instruções de execução (se ele não existir, então $T = 0$);
- A compilação dos programas desenvolvidos (se houver erros de compilação/ interpretação, então haverá descontos em T).

8 Sobre a entrada e a saída dos dados

Para cada execução, uma única entrada será dada: o caminho de um arquivo no formato `lsi` escrito na linguagem `LSI-2024-1`.
Exemplo de uma entrada: `/tmp/arvore-binaria-de-busca.lsi`.
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);
  - 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).

Outras observações importantes:

1. Os programas podem ser escritos em C (compatível com compilador gcc versão 13.2.0), C++ (compatível com compilador g++ versão 13.2.0), Java (compatível com compilador javac versão 21.0.3) ou Python 3 (compatível com versão 3.12.3) ou Rust (compatível com rustc versão 1.75.0) e deve ser compatível com Linux/Unix.
2. É importante que seu programa esteja escrito de maneira a destacar a estrutura do programa.
3. Cada arquivo entregue deve começar com um cabeçalho contendo pelo menos o nome de todos os integrantes do grupo.
4. Coloque comentários em pontos convenientes do programa, e faça uma saída clara.
5. O trabalho 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 `LSI-2024-1` já no formato convencional. 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 uma simplificação da gramática X++ de Delamaro (ver referência abaixo). Para o trabalho relacionado ao analisador léxico, precisamos destacar os *tokens*. Eles são precisamente os *símbolos terminais* de `LSI-2024-1` e que estão na cor azul. Os símbolos terminais não-triviais são somente `id` (identificadores de variáveis) e `num` (constantes decimais inteiras). Os *símbolos não-terminais* de `LSI-2024-1` estão em letra de forma.

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

Veja, a seguir, um exemplo de entrada (um código fonte) de acordo com a linguagem descrita pela gramática `LSI-2024-1`:

```
def func1(int A, int B)
{
  int C = A + B;
  int D = 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, id, (, int, id, ,, int, id, ), {, int, id, ..., return, ;, } ]`. A lista de tokens que deve ser elaborada pelo seu analisador léxico deverá ser similar a essa.

MAIN → STMT | FLIST | ε

FLIST → FDEF FLIST | FDEF

FDEF → **def id(**PARLIST**){**STMTLIST**}**

PARLIST → **int id,** PARLIST | **int id** | ε

STMT → **int id;**
| ATRIBST;
| PRINTST;
| RETURNST;
| IFSTMT
| **{**STMTLIST**}**
| **;**

ATRIBST → **id =** EXPR | **id =** FCALL

FCALL → **id(**PARLISTCALL**)**

PARLISTCALL → **id,** PARLISTCALL | **id** | ε

PRINTST → **print** EXPR

RETURNST → **return**

IFSTMT → **if(**EXPR**)** STMT **else** STMT
| **if(**EXPR**)** STMT

STMTLIST → STMT STMTLIST | STMT

EXPR → NUMEXPR **<** NUMEXPR
| NUMEXPR **>** NUMEXPR
| NUMEXPR **==** NUMEXPR
| NUMEXPR

NUMEXPR → NUMEXPR **+** TERM
| NUMEXPR **-** TERM
| TERM

TERM → TERM ***** FACTOR | FACTOR

FACTOR → **num** | **(**NUMEXPR**)** | **id**