Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

SCC0217 – Linguagens de Programação e Compiladores

# Trabalho 1 – Analisador Léxico para LALG

Elias Italiano Rodrigues – 7987251
Vinicius Katata Biondo – 6783972

São Carlos, 26 de abril de 2015

# Sumário

# 1 Introdução

Este trabalho implementa um analisador léxico para a linguagem de programação `LALG` utilizando a ferramenta `flex`. Foram seguidas as instruções dadas em sala de aula assim como consultadas em tutoriais na Web [1] [2] e em livro [3].

# 2 Como Usar

## 2.1 Compilação

O trabalho entregue, como requisitado, já foi previamente compilado (`Linux`), não havendo necessidade de executar esse passo. Porém, caso queira ou precise compilar novamente, basta estar dentro do diretório do trabalho e executar:

```
make
```

É necessário ter instalado o compilador `gcc`, a ferramenta `flex`, assim como o utilitário `make` em sistema operacional `Linux`.

## 2.2 Execução

Para executar o trabalho, basta estar dentro de seu diretório e executar:

```
./bin/main
```

Dessa maneira, o programa `LALG` será lido da entrada padrão `stdin`.

Para executá-lo sobre um arquivo, basta redirecionar a entrada:

```
./bin/main < meu-programa.lalg
```

No diretório `./test` encontram-se alguns exemplos de programa em `LALG` para testar. Por exemplo:

```
./bin/main < ./test/programa1.lalg
```

Opcionalmente, para rodar para todos os programas `.lalg` de `./test`, execute:

```
make run
```

As saídas serão escritas em arquivos com sufixo `_out` na própria pasta `./test`.

## 2.3 Exemplo de Execução

Comando:

```
./bin/main < ./test/programa1.lalg
```

Saída:

```
 2: program - program
 2:   nome1 - IDENT (identificador)
 2:       ; - SEMICOLON (ponto-virgula)
 3:     var - var
 3:       a - IDENT (identificador)
 3:       , - COMMA (virgula)
 3:      a1 - IDENT (identificador)
 3:       , - COMMA (virgula)
 3:       b - IDENT (identificador)
 3:       : - COLON (dois-pontos)
 3: integer - integer
 3:       ; - SEMICOLON (ponto-virgula)
 4:   begin - begin
 5:    read - read
 5:       ( - OPAR (abre parenteses)
 5:       a - IDENT (identificador)
 5:       ) - CPAR (fecha parenteses)
 5:       ; - SEMICOLON (ponto-virgula)
 6:      a1 - IDENT (identificador)
 6:      := - ATR (atribuicao)
 6:       a - IDENT (identificador)
 6:       * - MULT (multiplicacao)
 6:       2 - INTEGER (numero inteiro)
 6:       ; - SEMICOLON (ponto-virgula)
 8:   while - while
 8:       ( - OPAR (abre parenteses)
 8:      a1 - IDENT (identificador)
 8:       > - GR (maior)
 8:       0 - INTEGER (numero inteiro)
 8:       ) - CPAR (fecha parenteses)
 8:      do - do
 9:   begin - begin
10:   write - write
10:       ( - OPAR (abre parenteses)
10:      a1 - IDENT (identificador)
10:       ) - CPAR (fecha parenteses)
10:       ; - SEMICOLON (ponto-virgula)
11:      a1 - IDENT (identificador)
11:      := - ATR (atribuicao)
11:      a1 - IDENT (identificador)
11:       - - MINUS (subtracao)
11:       1 - INTEGER (numero inteiro)
11:       ; - SEMICOLON (ponto-virgula)
12:     end - end
12:       ; - SEMICOLON (ponto-virgula)
14:     for - for
14:       b - IDENT (identificador)
14:      := - ATR (atribuicao)
14:       1 - INTEGER (numero inteiro)
14:      to - to
14:      10 - INTEGER (numero inteiro)
14:      do - do
15:   begin - begin
16:       b - IDENT (identificador)
16:      := - ATR (atribuicao)
16:       b - IDENT (identificador)
16:       + - PLUS (adicao)
16:       2 - INTEGER (numero inteiro)
16:       ; - SEMICOLON (ponto-virgula)
17:       a - IDENT (identificador)
17:      := - ATR (atribuicao)
17:       a - IDENT (identificador)
17:       - - MINUS (subtracao)
17:       1 - INTEGER (numero inteiro)
17:       ; - SEMICOLON (ponto-virgula)
18:     end - end
18:       ; - SEMICOLON (ponto-virgula)
20:      if - if
20:       a - IDENT (identificador)
20:      <> - DIFFERENT (diferente)
20:       b - IDENT (identificador)
20:    then - then
21:   write - write
21:       ( - OPAR (abre parenteses)
21:       a - IDENT (identificador)
21:       ) - CPAR (fecha parenteses)
21:       ; - SEMICOLON (ponto-virgula)
22:     end - end
22:       . - DOT (ponto)
```

Mais exemplos estão disponíves no diretório `./test`.

# 3 Organização dos Arquivos

O diretório do trabalho está organizado da seguinte maneira:

`./bin` : diretório para arquivos binários executáveis.
  `|-- main` : o programa principal a ser executado para fazer a análise léxica.

`./doc` : diretório dos arquivos LaTeX fonte deste relatório.

`./src` : diretório com os códigos-fonte do programa.
  `|-- lalg.h` : definições dos *tokens* da linguagem `LALG` e erros.
  `|-- lalg.l` : programa `Lex` para a linguagem `LALG`.
  `|-- lalg.c` : programa `C` gerado pelo `flex`.
  `|-- main.c` : programa principal que usa o `lalg.c`.

`./test` : diretório com exemplos de programa `LALG`.

`Makefile` : arquivo para automizar compilação e execução usando o utilitário `make`.

`LALG` : definição da linguagem `LALG`.

`RELATORIO.pdf` : este relatório PDF compilado a partir de `./doc`.

`README` : arquivo com instruções.

# 4 Decisões de Projeto

## 4.1 *Tokens*

Para catalogar os *tokens*, foi decido o formato <`token, simbolo_token`>. Os símbolos foram definidos conforme mostra a Tabela 1.

## 4.2 Palavras Reservadas

Para catalogar as palavras reservadas, foi decido o formato <`palavra_reservada, palavra_reservada`>. Segue a lista de palavras reservadas definidas:

`begin`, `char`, `const`, `do`, `else`, `end`, `for`, `function`, `if`, `integer`, `procedure`, `program`, `read`, `real`, `repeat`, `then`, `to`, `until`, `var`, `while`, `write`

A tabela de palavras reservadas foi descrita diretamente no código-fonte (*hard-coded*) para melhor desempenho e em ordem alfabética. Mais informações estão documentadas no arquivo `./src/main.c`.

Para conferir se um *token* casado como `IDENT` é ou não uma palavra reservada, foi criada uma função que faz **busca binária** sobre a tabela de palavras reservadas. Devido à pequena quantidade de palavras reservadas, 21, esse método é satisfatório em sua eficiência que é $O(\log_2 21)$ no pior caso.

| *Token* | Símbolo | Descrição |
|---|---|---|
| : | COLON | dois-pontos |
| ; | SEMICOLON | ponto-e-vírgula |
| . | DOT | ponto |
| , | COMMA | vírgula |
| ( | OPAR | abre parênteses |
| ) | CPAR | fecha parênteses |
| := | ATR | atribuição |
| >= | GOE | maior ou igual |
| <= | LOE | menor ou igual |
| <> | DIFFERENT | diferente |
| = | EQUAL | igual |
| > | GR | maior |
| < | LE | menor |
| + | PLUS | adição |
| - | MINUS | subtração |
| * | MULT | multiplicação |
| / | DIV | divisão |
| <ident> | IDENT | identificador |
| <numero_int> | INTEGER | número inteiro |
| <numero_real> | REAL | número real |
| <char> | CHAR | caractere entre ' escapado ou não com \ |
| <reservado> | RESERVED | qualquer palavra reservada |
| <desconhecido> | UNKNOWN | um token desconhecido |

Tabela 1: Descrição dos símbolos adotados para os *tokens*. Observação: os *tokens* <reservado> e <desconhecido> não fazem parte da gramática LALG em si, mas foram definidos apenas para documentação, pois foram usados na implementação.

### 4.3 Erros Léxicos

Os códigos para os possíveis erros léxicos foram definidos como mostra a Tabela 2. Também foram definidos comprimentos máximos para alguns *tokens* listados na Tabela 3.

Usou-se **código** C para os erros referentes a comprimento e **expressões regulares** no próprio Lex ./src/lalg.l para casar os erros referentes a má formação.

Para cada *token* lido pela função que faz a análise léxica, ele é impresso na saída padrão stdout juntamente com informações descritivas dos erros ocorridos.

## 5 Conclusão

O trabalho desenvolvido cumpre a especificação dada. Foi possível aprender mais sobre a ferramenta flex e concluir o analisador léxico de LALG que servirá como base para o próximo trabalho.

| Código | Descrição |
|---|---|
| SUCCESS | nenhum erro |
| ERR_BAD_IDENT | idenfiticador mal formado |
| ERR_MAX_IDENT | idenfiticador muito grande |
| ERR_BAD_INTEGER | numero inteiro mal formado |
| ERR_MAX_INTEGER | numero inteiro muito grande |
| ERR_BAD_REAL | numero real mal formado |
| ERR_MAX_REAL | numero real muito grande |
| ERR_BAD_CHAR | caractere mal formado |
| ERR_MAX_CHAR | caractere muito grande |
| ERR_CHAR_EMPTY | caractere vazio |
| ERR_CHAR_BREAK | caractere nao inline |
| ERR_CHAR_OPEN | caractere nao fechado |
| ERR_COMMENT_BREAK | comentario nao inline |
| ERR_COMMENT_OPEN | comentraio nao fechado |
| ERR_UNKNOWN | token desconhecido |

Tabela 2: Descrição dos códigos de erro adotados.

| Código | Valor | *Token* |
|---|---|---|
| MAX_LENGTH_IDENT | 13 | <ident> |
| MAX_LENGTH_INTEGER | 13 | <numero_int> |
| MAX_LENGTH_REAL | 13 | <numero_real> |

Tabela 3: Comprimentos máximos definidos para alguns *tokens*.

# Referências


[1] Lexical Analysis With Flex, for Flex 2.5.37
<http://flex.sourceforge.net/manual/>
Acesso em: 1 de abril de 2015

[2] Part 01: Tutorial on lex/yacc
<https://www.youtube.com/watch?v=54bo1qaHAfk>
Acesso em: 1 de abril de 2015

Part 02: Tutorial on lex/yacc.
<https://www.youtube.com/watch?v=__-wUHG2rfM>
Acesso em: 1 de abril de 2015

[3] LEVINE, John. flex & bison. United States of America: O'Reilly, 2009.