Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

SCC0217 – Linguagens de Programação e Compiladores

Trabalho 3 – Analisador Semântico para LALG

Elias Italiano Rodrigues – 7987251
Vinicius Katata Biondo – 6783972

São Carlos, 03 de julho de 2015

Sumário

1 Introdução

Este trabalho implementa um analisador semântico com tratamento de erros para a linguagem de programação `LALG` utilizando as ferramentas `flex` e `bison`. Foram seguidas as instruções dadas em sala de aula assim como consultadas em manual [1] e em livro [2].

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`, as ferramentas `flex` e `bison`, assim como o utilitário `make` em sistema operacional `Linux`.

2.2 Execução

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

```
./main
```

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

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

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

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

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

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

```
make run
```

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

2.3 Exemplo de Execução

Arquivo `./test/sem/error_varios1.lalg` – programa fictício com vários erros semânticos:

```
 1. program test1;
 2.
 3.     { OK: declaracao de constantes }
 4.     const a = 10;
 5.     const b = 11;
 6.     const k = 238.11;
 7.
 8.     { ERRO: constante ja declarada }
 9.     const a = 74.2;
10.
11.     { OK: declaracao de variaveis}
12.     var x, y: real;
13.     var c, i, j: integer;
14.     var opcao1, opcao2: char;
15.
16.     { ERRO: variavel ja declarada }
17.     var x: integer;
18.
19.     { OK: declaracao de procedimento }
20.     procedure my_proc(x: integer; y, z: real);
21.         { OK: declaracao de variaveis }
22.         var i, j: integer;
23.     begin
24.         j := 7;
25.         for i:=1 to 5 do
26.         begin
27.             { ERRO: variavel 'c' nao declarada neste escopo }
28.             c := 5 * i + j * x + y + z;
29.         end;
30.
31.     end;
32.
33.     { ERRO: procedimento ja foi declarado }
34.     procedure my_proc(x: integer; y, z: real);
35.     begin
36.     end;
37.
38. begin
39.     { OK: divisao entre inteiros }
40.     c := 439 / 2;
41.
42.     { ERRO: tipo incompativel real <- integer }
43.     x := a;
44.     x := 25;
45.
46.
```

```
47.     { ERROS: divisao entre nao-inteiros (mas faz atribuicao) }
48.     y := 439.1 / 2;
49.     y := 500.0 / 2.0;
50.
51.     { OK: atribuicoes sem problemas }
52.     opcao1 := 'A';
53.     opcao2 := '\n';
54.     c := a + b - 10;
55.     x := k * 2.0;
56.     x := y;
57.     x := 12.57 * 6.3;
58.
59.     { ERRO: tipo incompativel integer <- real }
60.     c := x;
61.     c := 9.21;
62.
63.     { OK: comandos sem problemas }
64.     read(x, y);
65.     write(x, y);
66.
67.     { OK: atribuicoes sem problemas }
68.     c := 9;
69.     i := c + a + b;
70.     j := i * 2;
71.
72.     { ERRO: variavel nao declarada }
73.     t := x;
74.
75.     { OK: chamada sem problemas }
76.     my_proc(c; x; y);
77.
78.     my_proc;       { ERRO: argumentos insuficientes }
79.     my_proc(c);    { ERRO: argumentos insuficientes }
80.     my_proc(x);    { ERRO: argumentos insuficientes }
81.     my_proc(c; y); { ERRO: argumentos insuficientes }
82.     my_proc(c; x; y; j); { ERRO: argumentos demais }
83.     my_proc(x; y; j);    { ERRO: tipos incompativies }
84.
85.     { ERRO: procedimento nao declarado }
86.     other_proc(x; y);
87.
88.     { ERROS: comandos com variaveis de tipos diferentes }
89.     read(x, c);
90.     write(x, c);
91. end.
```

Comando:

```
./main < ./test/sem/error_varios1.lalg
```

Saída:

```
[ 9,19]: constant 'a' has already been declared
[17,19]: variable 'x' has already been declared
[28,39]: undeclared variable 'c'
[36,8 ]: procedure 'my_proc' has already been declared
[43,11]: incompatible type for variable 'x'
[44,12]: incompatible type for variable 'x'
[48,19]: division with non-integer numerator
[49,18]: division with non-integer denominator
[49,21]: division with non-integer numerator
[60,11]: incompatible type for variable 'c'
[61,14]: incompatible type for variable 'c'
[73,11]: undeclared variable 't'
[78,12]: insufficient number of arguments for procedure 'my_proc'
[79,14]: insufficient number of arguments for procedure 'my_proc'
[80,14]: insufficient number of arguments for procedure 'my_proc'
[81,17]: insufficient number of arguments for procedure 'my_proc'
[82,23]: too much arguments for procedure 'my_proc'
[83,20]: incompatible type in argument 1 for 'my_proc' procedure
[83,20]: incompatible type in argument 3 for 'my_proc' procedure
[86,20]: undeclared procedure 'other_proc'
[89,14]: read/write command with different variable types
[90,15]: read/write command with different variable types
```

Onde `[i,j]` indica linha `i` na coluna `j`.

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

3 Organização dos Arquivos

O diretório do trabalho está organizado da seguinte maneira:

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

`./test` : diretório com exemplos de programa `LALG` para testes.
`|-- ./lex` : exemplos léxicos.
`|-- ./sem` : exemplos semânticos.
`|-- ./sin` : exemplos sintáticos.

`LALG` : definição da linguagem `LALG`.

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

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

`README` : arquivo com instruções.

`lalg.l` : programa `Lex` para a linguagem `LALG`.

`lalg.y` : programa em `Bison` para a linguagem `LALG`.

`lex.yy.c` : programa `C` gerado pelo `flex`.

`y.tab.c` : programa principal `C` gerado pelo `bison`.

`y.tab.h` : cabeçalho `C` gerado pelo `bison`.

`main` : o programa principal a ser executado para fazer a análise semântica.

4 Decisões de Projeto

4.1 `lalg.y`

Para armazenar os símbolos – variáveis, constantes e procedimentos –, foram utilizadas tabelas *hash* da biblioteca `<search.h>`. Uma tabela *hash* é utilizada por escopo, ou seja, ao necessitar de um novo escopo, uma nova tabela é criada para armazenar os símbolos desse novo escopo.

Para simplificar a codificação, definiu-se estruturas e uniões de dados convenientes, como por exemplo, `struct data_s` que guarda o tipo e o valor de um dado; e o valor de um dado é representado pela `union val_u` que pode assumir qualquer um dos possíveis valores do LALG.

A detecção dos erros semânticos consistiu em manipular as tabelas *hash* e estruturas de dados auxiliares como listas temporárias de variáveis, parâmetros e argumentos.

4.2 `lalg.l`

No parte léxica foi apenas necessário alterar a os retornos dos valores dos *tokens*, que agora utilizam a `struct data_s`.

5 Conclusão

O trabalho desenvolvido cumpre a especificação dada. Foi possível aprender mais sobre a ferramenta `bison` e concluir o analisador semântico de `LALG` que usou o analisador sintático e léxico implementados nos trabalhos anteriores.

Referências

[1] Bison 3.0.4
<http://www.gnu.org/software/bison/manual/html_node/index.html>
Acesso em: 6 de maio de 2015

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