UNIVERSIDADE FEDERAL DE VIÇOSA – CAMPUS FLORESTAL

DANIEL FREITAS MARTINS – 2304

JOÃO ARTHUR GONÇALVES DO VALE – 3025

MARIA DALILA VIEIRA – 3030

NAIARA CRISTIANE DOS REIS DINIZ – 3005

Relatório referente ao Trabalho Prático 3 – Criação do analisador sintático e da Tabela de Símbolos para a linguagem Chameleon

Florestal

2020

DANIEL FREITAS MARTINS – 2304

JOÃO ARTHUR GONÇALVES DO VALE – 3025

MARIA DALILA VIEIRA – 3030

NAIARA CRISTIANE DOS REIS DINIZ – 3005

Relatório referente ao Trabalho Prático 3 – Criação do analisador sintático e da Tabela de Símbolos para a linguagem Chameleon

Documentação apresentada à disciplina CCF 441 – Compiladores do curso de Bacharelado em Ciência da Computação da Universidade Federal de Viçosa – *Campus* Florestal.

Orientador: Daniel Mendes Barbosa

Florestal

2020

SUMÁRIO

1 - Introdução

Neste trabalho criamos um analisador sintático usando LEX&YACC. Para isso, partimos da versão anterior implementada em LEX que imprimia os *tokens* encontrados na tela. Além disso, foi preciso fazer algumas alterações na gramática para mitigar erros de *reduce/reduce* e *shift/reduce* encontrados no processo de desenvolvimento dessa segunda etapa do compilador. Por fim, foi feita a tabela de símbolos que possui como entrada os *tokens* e dá acesso a demais informações sobre os símbolos como: endereço na memória, valor, tipo. Devido a limitações de tempo, nessa etapa não foi possível gerar uma nova versão do pré-processador, mas ainda há planos de melhorias na próxima etapa.

É importante ressaltar que o *script* para a geração do executável do compilador foi alterado. Com a existência dos novos arquivos responsáveis pelo analisador sintático e pela tabela de símbolos, a seguinte sequência de comandos pode ser utilizada para gerar o executável em questão:

1	**flex** lex.l
2	**yacc** translate.y **-d -v**
3	**gcc** symbol.h symbol.c hash-table.h hash-table.c y.tab.c lex.yy.c y.tab.h **-ll**

O parâmetro **-v** na segunda linha é opcional e serve apenas para imprimir informações adicionais em relação à geração do analisador sintático. Os três comandos podem ser colocados em um arquivo *script.sh*, separados por **&&**, para realizar a compilação de uma forma mais conveniente, sendo necessário apenas executar este *script*, se dadas as devidas permissões de execução. Para utilizar o executável do compilador gerado, use **./a.out arquivo_entrada**,

2 - A Linguagem Chameleon

A linguagem de programação Chameleon pretende oferecer um conjunto de comandos que seja interessante para programadores que desejam agilidade, simplicidade e espaço para customização. Sua construção foi baseada nas linguagens C, C++ e Python. Assim, temos uma linguagem imperativa procedural que segue o paradigma estrutural. A origem deste nome veio justamente desta característica de customização da linguagem com o uso de seu pré-processador. Em resumo, ela permite definir novas formas de escrita que são convertidas para a sintaxe padrão da linguagem. Essa ideia é similar ao uso de macros na linguagem C, mas com um propósito diferente. A logo da linguagem Chameleon pode ser visualizada na Fig. 2.1 a seguir. Maiores detalhes a respeito das alterações na gramática da linguagem, a criação do analisador sintático e da tabela de símbolos serão abordados nas próximas seções.

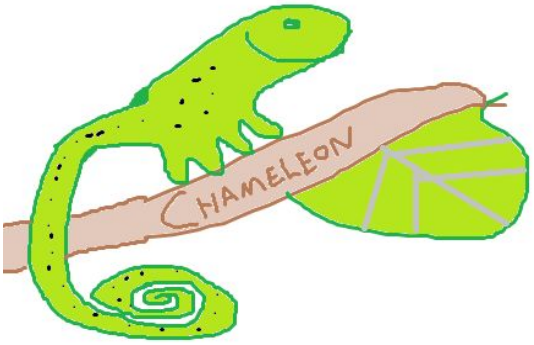

Figura 2.1: Logo da linguagem Chameleon. Fonte: Autores, 2020.

3 - Atualizações na Gramática

Para esta parte do trabalho foi preciso fazer algumas modificações consideráveis na gramática proposta na etapa anterior, objetivando a resolução de conflitos *shift/reduce* e *reduce/reduce*. A seguir são mostradas todas as produções enumeradas, de modo que as produções que precisaram ser alteradas são apresentadas duas vezes. Assim, primeiro é exibida a versão antiga que pode ter trechos marcados em vermelho, o que significa que essa parte foi removida da gramática. Logo em seguida, vem a versão modificada da produção, em que há trechos em azul que indicam partes que precisaram ser adicionadas. Sobretudo, foi preciso criar algumas novas produções, estas

também estão escritas em azul, posterior a produção cuja atualização resultou na criação delas. As novas produções geradas nessa etapa tem uma numeração referente a sua origem, por exemplo, se ela foi a primeira regra criada para corrigir conflitos na regra "16", ela deverá se chamar "16-a)".

1- *block* → **begin** *statement* **end**

1- *block* → **begin** *statement* **end** *block_continue*
| *task_command block_continue*
1-a) *block_continue* → **ε** */* palavra vazia */*
| *block*

2- *statement* → **ε**
| *command statement*
| *variable_declaration statement*
2- *statement* → **ε**
| *command statement*
| *variable_declaration statement*
| **begin** *statement* **end** *statement*

3- *command* → *variable* **attribution** *expression*
| *if_command*
| *for_command*
| *while_command*
| *farewell_command*
| **stop**
| *jumpto_command*
| *say_command*
| *listen_command*
| *task_command*
| *label*
3- *command* → *variable* **attribution** *expression*
| *variable* **attribution** word_*expression*
| *variable* **attribution** *task_call*
| *givingup*
| *if_command*
| *for_command*
| *while_command*
| *farewell_command*
| **stop**

| *jumpto_command*
| *say_command*
| *listen_command*
| *task_call*
| *label*

3-a) *givingup* →**GIVEUP**

4- *word_expression* → **word_value**
| *word_expression* **word_concat_operator** *expression*
| *expression* **word_concat_operator** *word_expression*

4- *word_expression* → **word_value**
| *word_expression* **word_concat_operator** *word_term_aux*

4-a) *word_term_aux* → *word_term*
| *word_term* **word_concat_operator** *word_term_aux*

4-b) *word_term* → **word_value**
| *expression*

5- *say_command* → **say** *expression*

5- *say_command* → **say** *expression*
| **say** *word_expression*

6- *listen_command* → **listen** *variable*

7- *if_command* → **if** *expression* **:** *statement* **endif**
| **if** *expression* **:** *statement* **elif** *expression* **:** *statement* **endelif**

8- *for_command* → **for** *expression* **,** *expression* **,** *expression* **:** *statement* **endfor**

9- *while_command* → **while** *expression* **:** *statement* **endwhile**

10- *farewell_command* → **farewell** *expression*

11- *task_command* → **task** *identifier task_parameters* **:** *expression* **endtask**

11- *task_command* → **task** *identifier task_parameters* **:** *command* **endtask**

12- *task_parameter* → *expression*
| *expression* **,** *task_parameter*

13- *task_parameters* → **ε**
| *task_parameter*

14- *task_call* → **task** *identifier* **(** *task_parameters* **)**

15- *jumpto_command* → **jumpto** *identifier*

16- *label* → *identifier* **;** *command*

17- *expression* → *add_expression*
| *div_expression*
| *pow_expression*
| *logic_expression*
| *rel_expression*
| **(** *expression* **)**
| **number**
| **real_number**
| *word_expression*
| *variable*
| *variable* **attribution** *expression*

17- *expression* → *math_expression*
| *variable* **attribution** *expression*

17-a) *math_expression* → *ex_aux_abre binary_operators math_expression*
| *unary_operators expression*
| *ex_aux_abre*

17-b) *ex_aux_abre* → *'(' math_expression ex_aux_fecha*
| *math_term*

17-c) *ex_aux_fecha* → ')'

17-d) *math_term* → NUMBER
| REAL_NUMBER
| *variable*

17-e) *binary_operators* → **add_operator**
| **neg_operator**

17-f) *unary_operators* → **add_operator**
| **div_operator**
| **pow_operator**
| **logic_operator**
| **rel_operator**

18- *variable_declaration* → *type* **identifier**
| *squad_declaration*
| *vector_declaration*

19- *variable_declarations* → *variable_declaration*

| *variable_declaration variable_declarations*

20- *vector_access* → *identifier* **[** *number* **]**
20- *vector_access* → *identifier* **[** *number* **]**
| *identifier* **[** *identifier* **]**

21- *squad_access* → **identifier squad_access_derreference identifier**
| *squad_access* **squad_access_derreference identifier**

22- *squad_declaration* → **squad** *identifier* **:** *variable_declarations* **endsquad**

23- *vector_declaration* → **vector identifier number**
| **vector identifier identifier**

24- *type* → **integer**
| **word**
| **real**

25- *variable* → **identifier**
| *vector_access*
| *squad_access*

26- *add_expression* → **add_operator** *expression*
| *expression* **add_operator** *expression*

27- *div_expression* → *expression* **div_operator** *expression*

28- *pow_expression* → *expression* **pow_operator** *expression*

29- *logic_expression* → *expression* **logic_operator** *expression*
| **neg_operator** *expression*

30- *rel_expression* → *expression* **rel_operator** *expression*

Assim, as produções de 26 a 30 foram removidas e englobadas nas produções "17-e)" e "17-f)", que derivam respectivamente *unary_operator e bynary_operator*. As operações unárias são definidas por terem apenas um operador, enquanto as binárias possuem dois.

A produção “1” foi alterada para permitir a definição de mais de um bloco em um arquivo, sendo que esses blocos não são aninhados. E também, *block* passou a derivar em *task* para permitir funções, as quais jamais podem ser definidas dentro de um escopo, evitando aninhamento. A produção “2” foi alterada para permitir a definição de blocos aninhados, ou seja, um bloco dentro do outro.

Com relação aos comandos, a produção “3” foi alterada, de modo que definição de função não deve ser um comando, e chamada de função passa a ser um comando, bem como atribuições de valores retornados por chamada de função, e ainda atribuição usando *word_expression.* Essa regra também passou a derivar em *givingup,* que é um comando que se assemelha ao *exit* da linguagem C, usado para desistir e encerrar a execução.

A regra “4” foi alterada, incluindo *word_term,* para passar a permitir conversão de operações ou valores numéricos (9 + 7 * 1) em cadeia de caracteres (“9 + 7 * 1”) por meio do operador de concatenação “++”. Isso foi criado no intuito de fazer uma saída/impressão em tela padronizada em cadeia de caracteres e que permita apresentar todos os tipos primitivos desejados. Caso queira concatenar o resultado de uma expressão a uma cadeia de caracteres, deve-se realizar as operações posteriormente atribuindo a uma variável. E esta é quem será usada para concatenação.

A regra “5” foi modificada para passar a derivar em “**say** *word_expression*”, porque *word_expression* deixou de ser incluída no conjunto das demais operações (atribuição e expressões matemáticas). Por conseguinte, como mencionado *word_expression* foi removida da produção “17”. Bem como, ela passou a derivar em *math_expression,* a qual engloba as expressões binárias e unárias, que usam os operadores que foram reunidos em *unary_operator e bynary_operator.*

Por fim, a produção “20” foi alterada para passar a permitir definição de tamanho de vetor por meio de identificadores. Antes era possível apenas valores inteiros dentro das chaves da definição de vetor.

4 - Modificações no Analisador Léxico

Para realizar a comunicação entre os analisadores léxico e sintático, o arquivo *lex.l* precisou sofrer modificações, como descritas a seguir. Inicialmente será abordada a modificação mais básica para a integração entre os módulos deste trabalho e por fim será abordada a parte que tem a ver com o uso da tabela de símbolos. O arquivo *lex.l* pode ser visualizado no Apêndice A deste documento.

Inicialmente, todos os *prints* de fluxo de execução do analisador léxico foram removidos, com exceção dos *prints* para informar os erros léxicos encontrados. No lugar, foram colocados os retornos que identificam os *tokens*, de acordo com as constantes geradas pelo YACC. A Fig. 4.1 mostra um trecho do arquivo *lex.l* ilustrando isso. As constantes **SAY**, **LISTEN** e **STOP** são automaticamente geradas pelo YACC, como será comentado na Seção 5. O caractere lido, como é o caso do *comma*, pode ser retornado também e corresponderá ao seu valor na tabela ASCII.

```
1  {say}      {return SAY;}
2  {listen}   {return LISTEN;}
3  {stop}     {return STOP;}
4  {comma}    {return yytext[0];} /* o caractere lido é retornado como token */
```

Figura 4.1 - Trecho de comandos com retorno de *tokens* no arquivo *lex.l*.

A Fig. 4.2 mostra a seção de declarações. Ela foi alterada para conseguir ter acesso tanto aos valores dos *tokens* presentes no arquivo *y.tab.h*, gerado pela compilação do arquivo de acordo com as especificações do YACC, quanto para acesso a recursos da tabela de símbolos (embutidos em *y.tab.h*) e outros recursos para facilidades de impressão, presentes no arquivo *custom_defines.h*.

```
1  %{
2  #include "custom_defines.h"
3  #include "y.tab.h" /* para poder usar as constantes dos tokens, por exemplo. */
4  #define PRINT_ERROR_EOF "-> Um ou mais erros foram encontrados. Corrija-os!\n"
5  void remover_espacos_e_print(int t);
6  #define _REAL 1
7  #define _NUMBER 2
8  int supress_errors_flag = 0;
9  int erro_encontrado = 0;
10 %}
```

Figura 4.2 - Seção de declarações de *lex.l*.

Outra alteração realizada foi a remoção da função *main* deste arquivo. Isso foi necessário uma vez que o arquivo *translate.y*, correspondente ao código para o analisador sintático, define este método. Por fim, para ser possível a utilização da tabela de símbolos (veja Seção 6), conforme solicitado neste trabalho, os tipos definidos de *symbol*, *symbol_table* e *type_aux* precisaram ser utilizados de modo que o analisador sintático reconhecesse os valores que estavam sendo passados para ele, em um desses tipos definidos pelo grupo. Isso entra em detalhes semânticos, uma vez que envolve a alteração de YYSTYPE do analisador sintático, por exemplo. No entanto, é importante ressaltar que as análises semânticas não foram elaboradas de fato, conforme especificação.

A Fig. 4.3 mostra as alterações realizadas para a identificação de identificadores e blocos. No caso de haver um casamento de padrão para um identificador, um símbolo correspondente é criado, correspondendo às informações daquele identificador. A função *createSymbol* é explicada em detalhes na Seção 6 mas, em linhas gerais, o tipo de dado retornado para o YACC após uma redução na gramática corresponde a um *symbol*, cujo lexema e tipo de *token* é retornado. O último parâmetro é passível de alteração, uma vez que o identificador pode corresponder ao identificador de uma função, por exemplo. Ele corresponde a uma classe de tipos de *tokens*, e tentaremos explorar isso na próxima fase.

Ainda na Fig. 4.3, ao se identificar o casamento de padrão correspondente ao início de um bloco, uma tabela de símbolos é criada. Ela corresponde a uma *HashTable* e é retornada para o YACC ao término da redução. Caso a tabela de símbolos atual seja NULL, isto é, ainda não existe uma criada, ela se torna a nova tabela criada e, da mesma forma, a referência para a primeira tabela de símbolos é atribuída, se esta for NULL.

```
1  /* outras regras (verifique Apêndice A) */
2  {identifier}          { createSymbol(&(yylval.symbol), yytext, IDENTIFIER, VALUE);
3                         return IDENTIFIER;}
4
5  {block_begin}         { yylval.symbol_table = ht_create(MAX_TAM_HASH);
6                          if(curr_symbol_table == NULL){
7                              curr_symbol_table = yylval.symbol_table;
```

```
8                                   if(first_symbol_table == NULL){
9                                       first_symbol_table = yylval.symbol_table;
10                                  }
11                              }
12                              return BLOCK_BEGIN;}
13  /* outras regras (verifique Apêndice A) */
```

Figura 4.3 - Trecho de *lex.l* que mostra alterações feitas para o uso da tabela de símbolos.

5 - O Analisador Sintático

O analisador sintático foi construído a partir da gramática modificada, conforme mencionado na Seção 3. Essas correções e adaptações da gramática criaram novas variáveis, bem como novas produções. As estratégias para resolução de conflitos *reduce/reduce* e *shift/reduce* consistiram em estabelecer um fluxo a ser seguido pela gramática de modo que ambiguidades em determinadas derivações fossem removidas. No entanto, na maioria das vezes a resolução de um conflito implicava na geração de outros conflitos. Desta forma, adotamos a estratégia de construir a gramática aos poucos de acordo com as especificações do YACC, tratando os conflitos à medida que foram aparecendo. Veja que, apesar dessa estratégia parecer boa, à medida que novos recursos eram adicionados à especificação da gramática como ela havia sido definida anteriormente, mais conflitos surgiam e mais difíceis eram de se resolver. Apesar de ter sido uma tarefa complicada, que demandou bastante tempo, o grupo acredita que este tempo gasto compensou para garantir a integridade da linguagem proposta.

O código correspondente ao analisador sintático está presente no Apêndice B deste documento. A Seção 5.1 irá descrever a respeito das produções e ações semânticas utilizadas.

5.1 - Produções e Ações Semânticas

A Fig. 5.1.1 mostra um trecho do código do analisador sintático. Ela mostra as produções de cabeças *expression*, *ex_aux_abre*, *ex_aux_fecha*, *math_expression*, *say_command* e *listen_command*. Observe que essas produções correspondem à gramática com as devidas correções, conforme mencionado anteriormente, e estão de

acordo com as especificações do YACC. Note que *tokens* como ‘=’, ‘(’ e ‘)’ são referenciados diretamente pelos seus valores da tabela ASCII, como comentado na Seção 4.

Os *prints* em cada uma das ações semânticas foram colocados para realizar a impressão da produção quando esta é finalizada/reduzida. Dessa maneira, conseguimos observar as reduções sendo feitas ao longo da execução. Note que as produções de *abre* e *fecha* parênteses precisaram ser divididas em três produções. Isso foi uma das modificações necessárias para a resolução de conflitos de *shift/reduce* e *reduce/reduce*. Nota-se, ainda, que a gramática em si ficou um pouco mais complexa de ser lida devido a essas adaptações da gramática. Porém, a gramática possui o mesmo comportamento daquela definida inicialmente no trabalho prático anterior, com as ressalvas levantadas na Seção 3.

```
1  /* Outras produções (veja Apêndice B) */
2  expression: math_expression        {PRINT(("expression -> [math_expression]\n"))}
3  | variable '=' expression          {PRINT(("expression -> [variable '=' expression]\n"))}
4  ;
5
6  ex_aux_abre: '(' math_expression ex_aux_fecha    {PRINT(("ex_aux_abre -> ['('
7  math_expression ex_aux_fecha]\n"))}
8  | math_term                        {PRINT(("ex_aux_abre -> [math_term]\n"))}
9  ;
10
11 ex_aux_fecha: ')'                  {PRINT(("ex_aux_fecha -> [')']\n"))}
12 ;
13
14 math_expression: ex_aux_abre binary_operators math_expression   {PRINT(("math_expression
15 -> [ex_aux_abre binary_operators math_expression]\n"))}
16 | unary_operators expression  {PRINT(("math_expression -> [unary_operators expression]\n"))}
17 | ex_aux_abre                 {PRINT(("math_expression -> [ex_aux_abre]\n"))}
18 ;
19 /* Outras produções (veja Apêndice B) */
20 say_command: SAY expression     {PRINT(("say_command -> [SAY expression]\n"))}
21 | SAY word_expression   {PRINT(("say_command -> [SAY word_expression]\n"))}
22 ;
23
24 listen_command: LISTEN variable {PRINT(("LISTEN -> [variable]\n"))}
25 ;
26 /* Outras produções (veja Apêndice B) */
```

Figura 5.1.1 - Trecho de *translate.y* para demonstrar algumas produções e ações semânticas.

O corpo básico do arquivo do YACC se assemelha muito com o do LEX. A Fig. 5.1.1 mostra um trecho correspondente à seção de definições. As linhas 2 a 5 correspondem às inclusões dos arquivos de cabeçalho padrões, sendo que a última inclusão possui *defines* para facilidades de impressão. A linha 7 indica o uso de uma variável externa, definida pelo LEX, para a contabilização do número de linhas de forma automática. A linha 8 indica o uso de uma variável externa, que possui o arquivo lido pelo programa e que é passado para o LEX. A linha 16 inclui uma rotina do BISON para informar ao YACC que o uso da tabela de símbolos é uma dependência que deve ser carregada primeiro. Isso foi necessário para realizar a compilação e vinculação dos módulos criados e é a forma recomendada de se fazer isso de acordo com a documentação [3].

Ainda na Fig. 5.1.1, a linha 20 define um novo tipo para o YYSTYPE, que equivale também ao tipo de *yylval* utilizado no LEX, como mostrado anteriormente na Fig. 4.3. Este *union* é similar ao *union* de C, e permite a utilização de tipos customizados, sendo que apenas um dos valores é ativo por vez. O uso de *type_aux* é apenas para facilitar as impressões nesta etapa do trabalho prático e talvez seja removido para a próxima etapa do trabalho prático. Dando sequência à explicação deste trecho, as linhas 26 e 27 são recursos do BISON e permitem impressões com maiores detalhes quando erros sintáticos são encontrados e são gerados pela própria ferramenta. As demais linhas, de 26 a 35 definem os *tokens*, conforme comentado na Seção 4. Tais *tokens* são referenciados por números inteiros a partir de 258 e estão definidos no arquivo *y.tab.h* gerado pelo YACC (veja Apêndice H). Das linhas 37 a 42, definimos as precedências de alguns *tokens* para resolver alguns problemas de *shift/reduce* e *reduce/reduce*, e a linha 44 explicita a variável de partida *block*.

```
1  %{
2     #include <stdio.h>
3     #include <stdlib.h>
4     #include <string.h>
5     #include "custom_defines.h"
6
7     extern int yylineno;
8     extern FILE* yyin;
9     int yylex();
10    int yyerror(const char*);
```

```
11      void raiseErrorVariableRedeclaration(char *lexem);
12      void raiseError(char *msg);
13
14      short flag_block_continue = 0;
15  %}
16      %code requires{
17          #include "hash-table.h"
18      }
19
20      %union{
21          Symbol *symbol;
22          hashtable_t *symbol_table;
23          char type_aux[20];
24      }
25
26      %define parse.error verbose
27      %define parse.lac full
28      %token WORD_VALUE
29      %token NUMBER REAL_NUMBER
30     /* Outras declarações de tokens */
31     /* Outras declarações de tokens */
32      %token IDENTIFIER
33      %token ADD_OPERATOR DIV_OPERATOR POW_OPERATOR LOGIC_OPERATOR
34      %token NEG_OPERATOR REL_OPERATOR WORD_CONCAT_OPERATOR
35      %token WHILE ENDWHILE
36
37      %left ADD_OPERATOR
38      %left DIV_OPERATOR
39      %left POW_OPERATOR
40      %left LOGIC_OPERATOR
41      %left REL_OPERATOR
42      %right '='
43
44      %start block
45  %%
```

Figura 5.1.2 - Trecho de *translate.y* para a seção de definições.

Para a utilização da tabela de símbolos (veja Seção 6) no YACC, o trecho de código da Fig. 5.1.2 mostra um exemplo de como a utilizamos. Nele estão presentes as ações semânticas referentes aos *tokens* **BLOCK_BEGIN** e **IDENTIFIER**.

Em relação às ações semânticas para o *token* **BLOCK_BEGIN**, aproveitamos o fato de que o YACC vai construindo uma pilha de execução e com isto conseguimos utilizar os valores retornados pelo LEX através da variável *yylval*. Neste contexto em específico para construção de uma tabela de símbolos para cada bloco, criamos uma estrutura que é discutida na Seção 7 que, em resumo, irá criar blocos irmãos e blocos filhos para os blocos, de modo que os pais conhecem os primogênitos e os filhos

conhecem os pais, bem como os irmãos mais velhos conhecem os irmãos mais novos. No trecho de código das linhas 2 a 14, se a *flag_block_continue* estiver ativa, isso significa que um bloco está sendo criado após o término de outro, indicando que este deve ser o irmão do bloco anterior. Neste sentido, o irmão recebe a tabela de símbolos criada pela chamada da função *ht_create* no LEX e é vinculado, como uma lista encadeada, a partir do irmão mais velho. A tabela de símbolos atual, *curr_symbol_table*, recebe então este novo bloco criado. A construção é análoga para os filhos e pode ser vista no Apêndice B, em uma das produções cuja cabeça é *statement*.

Em relação às ações semânticas para o token **IDENTIFIER**, note a produção cuja cabeça é *variable_declaration*. Veja que o tipo dos elementos retornados pelo LEX agora são do tipo *symbol* e seus lexemas e atributos, quando for o caso, podem ser acessados. Inicialmente há a verificação se o identificador já foi utilizado. Em caso positivo, um erro é impresso através da função *raiseErrorVariableRedeclaration* e, em caso negativo, o símbolo é recebido e armazenado na tabela de símbolos ativa naquele momento. Essa foi uma etapa que já conseguimos adiantar, uma vez que o uso da *hash* permite que isso seja feito de forma mais facilitada, bastando procurar se o lexema já foi definido em algum momento, por exemplo.

```
1  /* Outras produções (veja Apêndice B) */
2  block: BLOCK_BEGIN {
3     if(flag_block_continue){
4        flag_block_continue = 0;
5        if(curr_symbol_table->brother_hash == NULL){
6           hashtable_t *brother_symbol_table = $<symbol_table>1;
7           curr_symbol_table->brother_hash = brother_symbol_table;
8           curr_symbol_table = brother_symbol_table;
9        } else{
10          raiseError("O Brother e nao nulo!\n");
11       }
12
13    }
14 } statement BLOCK_END block_continue {PRINT(("block -> [BLOCK_BEGIN statement
15 BLOCK_END block_continue]\n"))}
16    | task_command block_continue                {PRINT(("block -> [task_command
17 block_continue]\n"))}
18 ;
19
20 variable_declaration: type IDENTIFIER       {PRINT(("variable_declaration -> [type
21 IDENTIFIER]\n"))
22    if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
23       $<symbol>2->data->v.word = strdup($<type_aux>1);
```

```
24          ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
25      } else{
26          raiseErrorVariableRedeclaration($<symbol>2->lexem);
27      }
28  }
29  | squad_declaration   {PRINT(("variable_declaration -> [squad_declaration]\n"))}
30  | vector_declaration  {PRINT(("variable_declaration -> [vector_declaration]\n"))}
31  ;
32  /* Outras produções (veja Apêndice B) */
```

Figura 5.1.3 - Trecho de *translate.y* que ilustra o uso da tabela de símbolos.

6 - Tabela de Símbolos

Como não foi implementado o analisador semântico ainda, a tabela de símbolos é utilizada apenas para guardar os identificadores já utilizados no código até um determinado ponto do processamento do código na compilação. Assim, quando ocorre a primeira aparição de um determinado identificador, ele é inserido na tabela de símbolos. Devido a composição da gramática, os únicos *tokens* que precisam estar na tabela são os identificadores. Feito isso, caso ocorra alguma redefinição desse tipo em um dado escopo, deverá ser informado erro.

A tabela é composta por: nome do *token,* o tipo do identificador e um endereço de memória. Esse é um endereço falso que deverá ser mapeado para a memória em etapas posteriores de adequação à uma arquitetura. Para definir o nosso endereço falso utilizamos a própria chave de hash para todo valor/identificador.

Algo que ainda precisa ser feito e pensado são os marcadores de pontos de programa, necessários para construções de desvio. De modo que ao desviar o fluxo de execução é preciso ter salvo em algum lugar para qual ponto deverá retornar ao fim de uma função, por exemplo, ou onde está o *label*.

É importante destacar que não inicializamos a tabela de símbolos com as palavras reservadas porque o próprio LEX&YACC já barra o uso delas como identificadores por todas elas serem *tokens*. O arquivo *y.tab.h*, presente no Apêndice H, define os *tokens* como números, e portanto ficam registrados para o uso de acordo como parte integrante da tabela de símbolos.

A estrutura da tabela de símbolos se encontra nos Apêndices C, D, E e F. Os Apêndices C e D consistem dos códigos para as estruturas de símbolos e os Apêndices

E e F consistem dos códigos para a estrutura da tabela em si, como uma tabela *hash*. Acreditamos que essas estruturas ainda sofrerão alterações para a próxima etapa deste trabalho, porém, a base da tabela de símbolos já foi construída.

6.1 - Estrutura Symbol

A estrutura básica de um símbolo é mostrada na Fig. 6.1.1. Note que há uma separação entre classes de tipos através do enum *SymbolType*. Isso poderá ser útil para a próxima etapa deste trabalho, mas ainda estamos avaliando se será necessário. Definimos as estruturas de *Squad*, *Data* e *Symbol*. Para Vector, note que o encadeamento em *Symbol* pode ser utilizado para este fim, mas precisamos elaborar melhor ainda como seria a estrutura Vector, bem como revisar a estrutura de Squad. Veja que funções foram criadas para auxiliar o processo de criação e destruição de símbolos (se necessário). Em particular, a função *createSymbol* espera como parâmetros o endereço para uma referência de um símbolo, seu lexema, tipo de *token* e classe de tipo. É importante ressaltar que durante a criação do símbolo pelo LEX, essa classe de tipo deve mudar para a correspondente na fase de análise semântica. Os valores dos dados deverão então posteriormente serem atribuídos de acordo com cada tipo de dado e no contexto em que estiverem inseridos.

```
1  enum SymbolType {FUNCTION, VALUE, AGGREGATED};
2
3  typedef void* (Function) (void* p, ...);
4
5  typedef struct Squad{
6     struct Symbol *internal_variables;
7  } Squad;
8
9  typedef struct Data{
10        enum SymbolType type;
11        union{
12           int integer;
13           double real;
14           char *word;
15           Squad squad;
16           Function *Function;
17        } v;
18 } Data;
19
20 typedef struct Symbol{
```

```
21      char *lexem;
22      int token_type;
23      Data *data;
24  } Symbol;
25
26  void createData(Data **data, enum SymbolType type);
27  void _createSymbol(Symbol **symbol, char *lexem, int token_type, Data *data);
28  void createSymbol(Symbol **symbol, char *lexem, int token_type, enum SymbolType type);
29  void destroySymbol(Symbol **symbol);
30  void destroyData(Data **data);
31  void destroySquad(Squad **squad);
```

Figura 6.1.1 - Trecho de *symbol.h*.

6.2 - Estrutura HashTable

Para a construção da tabela de símbolos, o grupo optou por utilizar uma tabela *hash*. O código referente à tabela *hash* foi adaptado de [2], para se adequar aos nossos tipos de dados, bem como para termos o comportamento esperado no processo de compilação. Para esta etapa, implementamos também a função para a impressão do *hash* e o mesmo pode ser encontrado no Apêndice F deste documento. Ela consiste de uma função recursiva que imprime primeiro os blocos filhos, e depois os blocos irmãos.

Das modificações do código original, podemos destacar a modificação de todas as estruturas da Fig. 6.2.1. Em *entry_s*, foram adicionados o *fake_memory_address*, e o valor do tipo *Symbol*; em *hashtable_s* foram adicionadas as referências *previous_hash*, *child_hash* e *brother_hash*; no escopo geral, foram adicionadas duas referências globais, *first_symbol_table* e *curr_symbol_table*, bem como a modificação dos protótipos e corpos de algumas das funções para a adequação ao nosso compilador.

Em se tratando de escopos de programação, as mudanças feitas em *hashtable_s* permitem que as tabelas de símbolos conheçam as tabelas de símbolos dos escopos mais internos (filhos) e conheçam também as tabelas de símbolos de escopos distintos (irmãos). Em outras palavras, todos os escopos são ligados entre si da seguinte forma: a tabela pai conhece seu filho primogênito (*child_hash*), enquanto que este conhece seu irmão (*brother_hash*), se houver; e todos os filhos conhecem seus pais (*previous_hash*). Essa construção é interessante pois permite que qualquer escopo consiga consultar qualquer outro escopo no programa. É possível pensar em algumas construções

interessantes com este recurso, como o compartilhamento de informações em diferentes escopos, mas não serão considerados neste trabalho.

Note que há uma macro para *MAX_TAM_HASH*, definindo-o como o valor 500. Isso significa que, em um bloco, o número máximo de símbolos na tabela de símbolos é de 500 símbolos. No entanto, essa constante não está presa à implementação; ela é usada apenas se for de interesse o seu uso para padronização. Poderemos implementar futuramente também estratégias para o aumento da tabela *hash*, caso esgote seu limite, por exemplo.

```
1  // https://gist.github.com/tonious/1377667/d9e4f51f05992f79455756836c9371942d0f0cee
2
3  // Outros include
4  #include "symbol.h"
5
6  #define MAX_TAM_HASH 500
7
8  struct entry_s {
9      int fake_memory_address;
10     char *key;
11     Symbol *value;
12     struct entry_s *next;
13 };
14
15 typedef struct entry_s entry_t;
16 typedef struct hashtable_s hashtable_t;
17
18 struct hashtable_s {
19     int size;
20     struct entry_s **table;
21     hashtable_t *previous_hash;
22     hashtable_t *child_hash;
23     hashtable_t *brother_hash;
24 };
25
26 hashtable_t *first_symbol_table; // global
27 hashtable_t *curr_symbol_table; // global
28
29 hashtable_t *ht_create( int size );
30 int ht_hash( hashtable_t *hashtable, char *key ) ;
31 entry_t *ht_newpair( char *key, Symbol *value ) ;
32 void ht_set( hashtable_t *hashtable, char *key, Symbol *value ) ;
33 Symbol *ht_get( hashtable_t *hashtable, char *key ) ;
34 void ht_print( hashtable_t *hashtable );
```

Figura 6.2.1 - Trecho de *hash-table.h*.

7 - Diagrama de Escopo

Ainda nesta etapa, foi um objetivo definir como seriam feitos os escopos do programa. Por serem uma parte fundamental para a utilização das tabelas de símbolos, as quais deverão remeter cada uma a um escopo. Assim, ao não encontrar a referência de um *token*/identificador na tabela referente ao escopo em que ele se encontra, deve-se fazer uma busca recursiva às tabelas dos escopos mais externos.

A definição de escopos de Chameleon foi baseada na linguagem ADA, e permite definições de escopo aninhadas por meio dos marcadores de início e fim de bloco *begin* e *end*. No entanto, optamos por não permitir definições de funções aninhadas por ser uma questão mais complexa, devido a correspondência que precisa ser feita entre parâmetros reais e parâmetros formais.

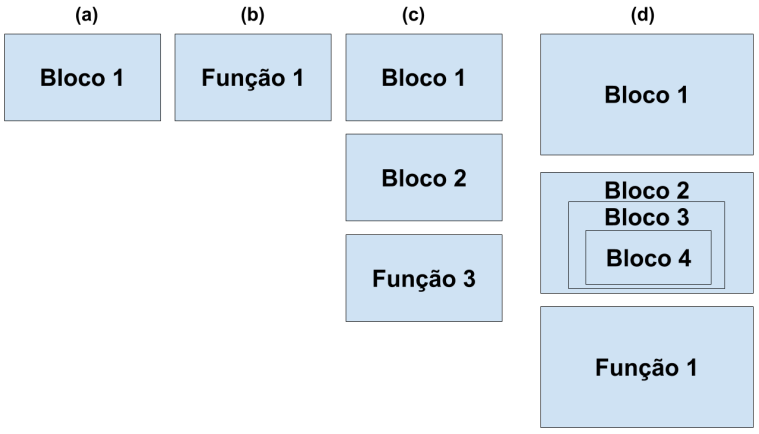

Figura 7.1 - Diagrama de blocos.

O diagrama a seguir apresenta os possíveis usos de escopo que Chameleon permite, de modo que é possível um ou mais blocos e funções definidos sem aninhamento. Ademais, é possibilitado também composições de blocos aninhados, com uma ou mais funções, porém funções jamais podem ser definidas dentro de outros tipos de blocos. A Fig. 7.2 mostra um exemplo válido de código em Chameleon que apresenta algumas das possíveis organizações de blocos válidas. Cada bloco é legendado com uma cor e um nome de B1 até B8.

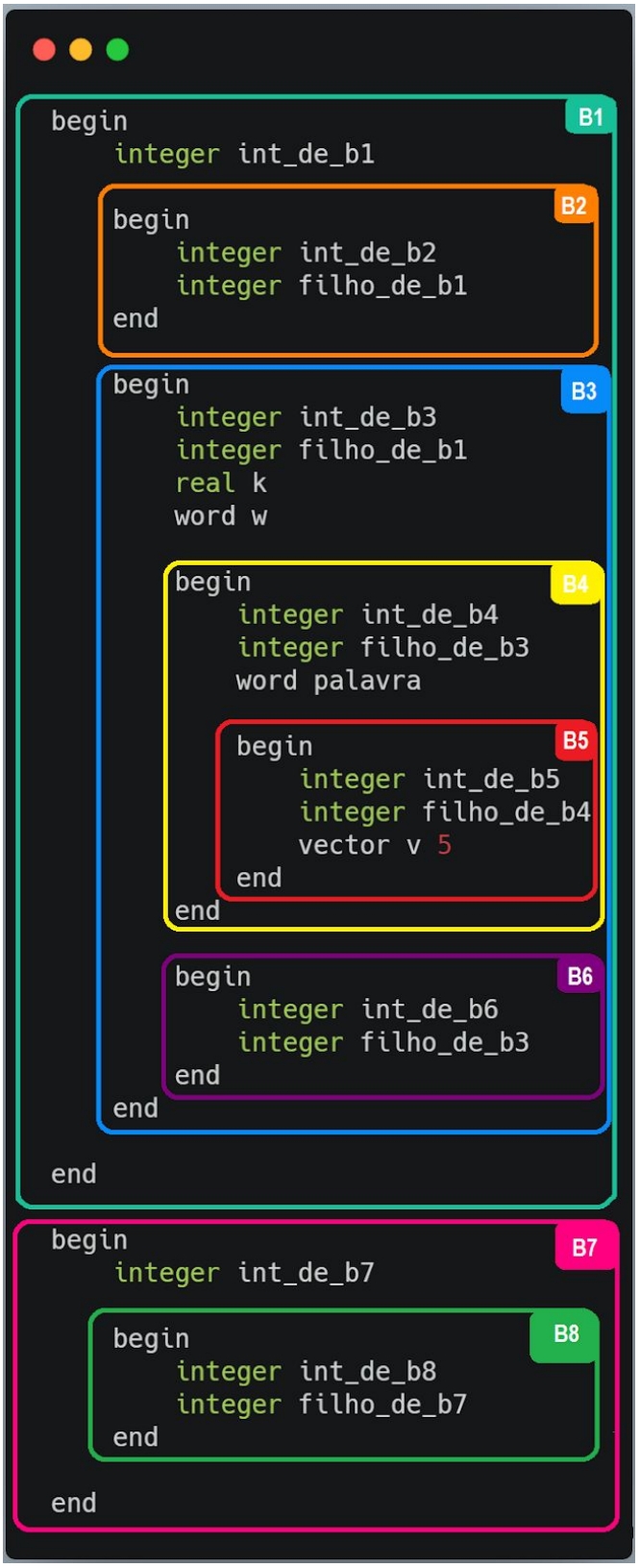

Figura 7.2 - Diagrama de blocos para a entrada 8.1.3 da Seção 8.

8 - Programas, Testes e Resultados

Esta seção apresenta programas sintaticamente válidos e inválidos para a linguagem Chameleon. Junto de cada programa, as saídas dos analisadores léxico (quando for o caso) e sintáticos são mostradas.

Para cada saída, é possível de se observar os seguintes itens: o código lido, com as linhas numeradas; as derivações realizadas após as reduções pelo analisador sintático e por fim as tabelas de símbolos geradas.

8.1 - Entradas lexicalmente e sintaticamente válidas

Entrada 8.1.1 - Cálculo de Fibonacci

```
---------- Readed Code ----------

1 begin
2     integer N
3     N = 20
4     vector v N
5     integer f1
6     integer f2
7     f1 = 0
8     f2 = 1
9     integer i
10      integer temp
11     // calculando Fib(N) \\
12     for i = 0, i < N, i = i+1:
13         v[i] = f1
14         temp = fib2
15         fib2 = fib1 + fib2
16         fib = temp
17     endfor
18     say "O resultado de Fib(" ++ N ++ ") = " ++ fib2 ++ "\n"
19 end

type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
vector_declaration -> [VECTOR IDENTIFIER IDENTIFIER]
```

```
variable_declaration -> [vector_declaration]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
expression -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
expression -> [variable '=' expression]
vector_access -> [IDENTIFIER '[' IDENTIFIER ']']
variable -> [vector_access]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
```

```
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
for_command -> [FOR expression ',' expression ',' expression ',' ':' statement ENDFOR]
command -> [for_command]
word_term -> [WORD_VALUE]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
word_term -> [expression]
word_term -> [WORD_VALUE]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
word_term -> [expression]
word_term -> [WORD_VALUE]
word_term_aux -> [word_term]
word_term_aux -> [word_term WORD_CONCAT_OPERATOR word_term]
word_term_aux -> [word_term WORD_CONCAT_OPERATOR word_term]
word_term_aux -> [word_term WORD_CONCAT_OPERATOR word_term]
word_expression -> [word_term WORD_CONCAT_OPERATOR expression]
say_command -> [SAY word_expression]
command -> [say_command]
statement -> []
```

```
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [command statement]
statement -> [variable_declaration statement]
block_continue -> []
block -> [BLOCK_BEGIN statement BLOCK_END block_continue]

------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

f1                      integer                         5
v                       vector                        277
f2                      integer                       301
temp                    integer                       417
i                       integer                       429
N                       integer                       437

--------------------------------------------------------------------------------------------
```

Entrada 8.1.2 - Uso de *jumpto* e *squad*

```
-------------- Readed Code --------------

1 // programa que incrementa uma variavel ate 5 usando jumpto \\
2 begin
3     integer i
4     i = 0
5
6     begin
7         squad pessoa :
8             word nome
9             integer idade
10        endsquad
11        listen pessoa->nome
12        say "Bem vindo " ++ pessoa->nome
13    end
14
15    RETORNAR: // label de retorno \\
16    i = i + 1
17    if i < 5:
18        jumpto RETORNAR
19    endif
20
```

```
21     say i // deve imprimir 5 \\
22 end

type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
type -> [WORD]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable_declarations -> [variable_declaration]
variable_declarations -> [variable_declaration variable_declarations]
squad_declaration -> [SQUAD IDENTIFIER ':' variable_declarations ENDSQUAD]
variable_declaration -> [squad_declaration]
squad_access -> [IDENTIFIER '->' IDENTIFIER ]
variable -> [squad_access]
LISTEN -> [variable]
command -> [listen_command]
word_term -> [WORD_VALUE]
squad_access -> [IDENTIFIER '->' IDENTIFIER ]
variable -> [squad_access]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
word_term -> [expression]
word_term_aux -> [word_term]
word_expression -> [word_term WORD_CONCAT_OPERATOR expression]
say_command -> [SAY word_expression]
command -> [say_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
label -> [IDENTIFIER ':' command]
command -> [label]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
```

```
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
jumpto_command -> [JUMPTO IDENTIFIER]
command -> [jumpto_command]
statement -> []
statement -> [command statement]
if_command -> [IF expression ':' statement ENDIF]
command -> [if_command]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
say_command -> [SAY expression]
command -> [say_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [command statement]
statement -> [variable_declaration statement]
block_continue -> []
block -> [BLOCK_BEGIN statement BLOCK_END block_continue]

---------------------------------- SYMBOL TABLE ----------------------------------

Lexem                   Data Type               Memory Address (fake)

i                         integer                       429
RETORNAR                  label                         481

------------------------------------------------------------------------------------


---------------------------------- SYMBOL TABLE ----------------------------------

Lexem                   Data Type               Memory Address (fake)

idade                    integer                        77
nome                      word                         105
pessoa                    squad                        277

------------------------------------------------------------------------------------
```

Entrada 8.1.3 - Código com vários blocos internos e externos para visualização de tabelas de símbolos definidas por blocos

```
---------- Readed Code ----------

1 begin
2    integer int_de_b1
3
4    begin
5       integer int_de_b2
6       integer filho_de_b1
7    end
8
9    begin
10       integer int_de_b3
11       integer filho_de_b1
12       real k
13       word w
14
15       begin
16          integer int_de_b4
17          integer filho_de_b3
18          word palavra
19
20          begin
21             integer int_de_b5
22             integer filho_de_b4
23             vector v 5
24          end
25       end
26
27       begin
28          integer int_de_b6
29          integer filho_de_b3
30       end
31    end
32
33 end
34
35 begin
36    integer int_de_b7
37
38    begin
39       integer int_de_b8
40       integer filho_de_b7
41    end
42
43 end

type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
```

```
statement -> []
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [REAL]
variable_declaration -> [type IDENTIFIER]
type -> [WORD]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [WORD]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
vector_declaration -> [VECTOR IDENTIFIER NUMBER]
variable_declaration -> [vector_declaration]
statement -> []
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> []
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
statement -> []
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> []
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> []
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [variable_declaration statement]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
```

```
statement -> []
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> []
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [variable_declaration statement]
block_continue -> []
block -> [BLOCK_BEGIN statement BLOCK_END block_continue]
block_continue -> [block]
block -> [BLOCK_BEGIN statement BLOCK_END block_continue]

------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

int_de_b1                integer                        257

--------------------------------------------------------------------------------------


------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

int_de_b2                integer                         53
filho_de_b1              integer                        257

--------------------------------------------------------------------------------------


------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

k                         real                          21
w                         word                          73
filho_de_b1              integer                        257
int_de_b3                integer                        349

--------------------------------------------------------------------------------------


------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

int_de_b4                integer                        145
palavra                  word                           225
filho_de_b3               integer                        349

--------------------------------------------------------------------------------------


------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)
```

```
v                       vector                  49
filho_de_b4             integer                 145
int_de_b5               integer                 441

----------------------------------------------------------------------------------------

---------------------------------- SYMBOL TABLE ----------------------------------

Lexem                   Data Type               Memory Address (fake)

int_de_b6               integer                 237
filho_de_b3             integer                 349

----------------------------------------------------------------------------------------

---------------------------------- SYMBOL TABLE ----------------------------------

Lexem                   Data Type               Memory Address (fake)

int_de_b7               integer                 33

----------------------------------------------------------------------------------------

---------------------------------- SYMBOL TABLE ----------------------------------

Lexem                   Data Type               Memory Address (fake)

filho_de_b7             integer                 33
int_de_b8               integer                 329

----------------------------------------------------------------------------------------
```

Observe a entrada 8.1.3. As tabelas de símbolos impressas correspondem às tabelas de cada um dos blocos da Fig. 7.2. Pela ordem de exibição das tabelas de símbolo (blocos filhos primeiro, blocos irmãos depois), observe que o encadeamento dessas tabelas de símbolo estão de acordo com o que deve ser.

Entrada 8.1.4 - Várias operações

```
---------- Readed Code ----------

1 begin // B0 \\
2     integer a
3     real b
4     real c
```

```
5     word maria
6     maria = "dalila"
7
8     listen a
9     listen b
10     c = ((a + b) * b) ^ 5
11     c = 1 / (c - 1) + (- 2)
12
13    lab: say c
14
15     while(1 == 1):
16        stop // quebrando o loop \\
17     endwhile
18
19     if b == c:
20        say "sim"
21     elif b != 5:
22        say "quase"
23        b = c
24        jumpto lab
25     endelif
26
27     for b = 0, b > 5, b = b + 1:
28        a = a + 2
29     endfor
30
31     begin // B1 \\
32        integer a
33        integer b
34        a = 5
35        b = 3
36        begin // B2 \\
37           integer b
38           b = 8
39           b = a * b
40           say "B2: " ++ b ++ "\n"
41        end
42        b = a * b
43        say "B1: " ++ b ++ "\n"
44     end
45 end
46
47 task somaum a:
48    farewell (a + 1)
49 endtask

type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [REAL]
variable_declaration -> [type IDENTIFIER]
type -> [REAL]
variable_declaration -> [type IDENTIFIER]
type -> [WORD]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
word_expression -> [WORD_VALUE]
```

```
command -> [variable '=' word_expression]
variable -> [IDENTIFIER]
LISTEN -> [variable]
command -> [listen_command]
variable -> [IDENTIFIER]
LISTEN -> [variable]
command -> [listen_command]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
binary_operators -> [DIV_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
binary_operators -> [POW_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
binary_operators -> [DIV_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
binary_operators -> [ADD_OPERATOR]
unary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
math_expression -> [unary_operators expression]
ex_aux_fecha -> [')']
```

```
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
say_command -> [SAY expression]
command -> [say_command]
label -> [IDENTIFIER ':' command]
command -> [label]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [STOP]
statement -> []
statement -> [command statement]
while_command -> [WHILE expression ':' statement ENDWHILE]
command -> [while_command]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
word_expression -> [WORD_VALUE]
say_command -> [SAY word_expression]
command -> [say_command]
statement -> []
statement -> [command statement]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
word_expression -> [WORD_VALUE]
```

```
say_command -> [SAY word_expression]
command -> [say_command]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
jumpto_command -> [JUMPTO IDENTIFIER]
command -> [jumpto_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
if_command -> [IF expression ':' statement ELIF expression ':' statement ENDELIF]
command -> [if_command]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
expression -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
expression -> [variable '=' expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
statement -> []
statement -> [command statement]
```

```
for_command -> [FOR expression ',' expression ',' expression ',' ':' statement ENDFOR]
command -> [for_command]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [DIV_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
word_term -> [WORD_VALUE]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
word_term -> [expression]
word_term -> [WORD_VALUE]
word_term_aux -> [word_term]
word_term_aux -> [word_term WORD_CONCAT_OPERATOR word_term]
word_expression -> [word_term WORD_CONCAT_OPERATOR expression]
say_command -> [SAY word_expression]
command -> [say_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
```

```
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [DIV_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
word_term -> [WORD_VALUE]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
word_term -> [expression]
word_term -> [WORD_VALUE]
word_term_aux -> [word_term]
word_term_aux -> [word_term WORD_CONCAT_OPERATOR word_term]
word_expression -> [word_term WORD_CONCAT_OPERATOR expression]
say_command -> [SAY word_expression]
command -> [say_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> []
statement -> [BLOCK_BEGIN statement BLOCK_END statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
task_parameter -> [expression]
task_parameter -> [task_parameter]
variable -> [IDENTIFIER]
```

```
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
farewell_command -> [FAREWELL expression]
command -> [farewell_command]
statement -> []
statement -> [command statement]
task_command -> [TASK IDENTIFIER task_parameters ':' ENDTASK]
block_continue -> []
block -> [task_command block_continue]
block_continue -> [block]
block -> [BLOCK_BEGIN statement BLOCK_END block_continue]

------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

a                        integer                        61
maria                   word                           117
c                        real                           153
b                        real                           357
lab                      jumpto                         477

--------------------------------------------------------------------------------------------


------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

a                        integer                        61
b                        integer                        357

--------------------------------------------------------------------------------------------


------------------------------------ SYMBOL TABLE ------------------------------------

Lexem                   Data Type               Memory Address (fake)

b                         integer                        357

--------------------------------------------------------------------------------------------
```

8.2 - Entradas lexicalmente inválidas, porém sintaticamente válidas

Entrada 8.2.1 - Soma de dois números - Erros léxicos

```
---------- Readed Code ----------

1 begin
2     real g
3     g = 5.3
4     integer $a
5     real $b
6
7     $a = 2
8     listen $b;
9     $b = $a + $b; // soma 2 ao numero lido \\
10     say $b;
11 end
12

type -> [REAL]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [REAL_NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
type -> [INTEGER]
---------- Erro encontrado na linha 4 ----------
-> $
int_code_s0: 36
variable_declaration -> [type IDENTIFIER]
type -> [REAL]
---------- Erro encontrado na linha 5 ----------
-> $
int_code_s0: 36
variable_declaration -> [type IDENTIFIER]
---------- Erro encontrado na linha 7 ----------
-> $
int_code_s0: 36
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
---------- Erro encontrado na linha 8 ----------
-> $
int_code_s0: 36
---------- Erro encontrado na linha 9 ----------
-> $
```

```
int_code_s0: 36
variable -> [IDENTIFIER]
LISTEN -> [variable]
command -> [listen_command]
variable -> [IDENTIFIER]
---------- Erro encontrado na linha 9 ----------
-> $
int_code_s0: 36
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
---------- Erro encontrado na linha 9 ----------
-> $
int_code_s0: 36
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
---------- Erro encontrado na linha 10 ----------
-> $
int_code_s0: 36
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
say_command -> [SAY expression]
command -> [say_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
statement -> [command statement]
statement -> [variable_declaration statement]
-> Um ou mais erros foram encontrados. Corrija-os!
```

Entrada 8.2.2 - Soma de dois números - Erros léxicos - Supressão de erros léxicos

```
-------------- Readed Code --------------

1 SUPRESS_ERRORS
2 begin
3     integer $a
```

```
4     real $b
5
6     $a = 2
7     listen $b
8     $b = $a + $b // soma 2 ao numero lido \\
9     say $b
10 end

type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
type -> [REAL]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
LISTEN -> [variable]
command -> [listen_command]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
command -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
say_command -> [SAY expression]
command -> [say_command]
statement -> []
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [command statement]
statement -> [variable_declaration statement]
statement -> [variable_declaration statement]
block_continue -> []
block -> [BLOCK_BEGIN statement BLOCK_END block_continue]

------------------------------------ SYMBOL TABLE ----------------------------------

Lexem                    Data Type                Memory Address (fake)

a                          integer                          61
b                            real                          357
```

```
-------------------------------------------------------------------------------------------
```

Observe que a entrada 8.2.2 é a mesma da entrada 8.2.1 com a inclusão da primeira linha do programa como SUPRESS_ERRORS. O código, mesmo com erros léxicos, foi para o analisador sintático ignorando tais erros léxicos. Apesar disso poder gerar problemas em relação à semântica do código, fica a cargo do programador utilizar este recurso ou não.

8.3 - Entradas lexicalmente válidas, porém sintaticamente inválidas

Essa seção permite explorar uso de tokens válidos porém com ordens que não são aceitas pela gramática, como blocos abertos que não são finalizados. Note na última linha que o analisador sintático encerra ao encontrar o erro informado o número da linha é o *token* inesperado nesse ponto do código.

Entrada 8.3.1 - Parênteses faltando em uma soma

```
---------- Readed Code ----------

1 begin
2     integer i
3     i = (15 + (84 + (100)) // falta um fecha parenteses \\
4     say "Nao chego aqui.. falta parenteses"
5 end

type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
Linha: 4 ---> syntax error, unexpected SAY <---
```

Entrada 8.3.2 - While sem fim!!

```
---------- Readed Code ----------

1 begin
2     for i=0, i<50, i=i+1:
3         integer k
4         k = 0
5         while(1==1):
6             k = k + 1
7         // while sem fim! \\
8     endfor
9 end

variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
expression -> [variable '=' expression]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
expression -> [math_expression]
expression -> [variable '=' expression]
type -> [INTEGER]
variable_declaration -> [type IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
command -> [variable '=' expression]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
binary_operators -> [REL_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
```

```
math_expression -> [ex_aux_abre]
math_expression -> [ex_aux_abre binary_operators math_expression]
ex_aux_fecha -> [')']
ex_aux_abre -> ['(' math_expression ex_aux_fecha]
math_expression -> [ex_aux_abre]
expression -> [math_expression]
variable -> [IDENTIFIER]
variable -> [IDENTIFIER]
math_term -> [variable]
ex_aux_abre -> [math_term]
binary_operators -> [ADD_OPERATOR]
math_term -> [NUMBER]
ex_aux_abre -> [math_term]
Linha: 8 ---> syntax error, unexpected ENDFOR <---
```

9 - Considerações Finais

Neste trabalho foram apresentadas as correções e adaptações da gramática elaborada para a linguagem Chameleon, alterações no arquivo do gerador de analisador léxico, bem como foram apresentadas também o arquivo para o analisador sintático e os módulos referentes à tabela de símbolos. Além disso, foi possível aplicar conceitos da teoria de compiladores às tecnologias LEX e YACC, bem como, aprender funcionalidades dessas ferramentas importantes para a construção de compiladores de forma menos manual, os geradores de analisadores. Nesta etapa em específico, foram feitas correções da gramática conforme o YACC apresentava conflitos de *shift/reduce* e *reduce/reduce.* Ademais, foram adicionados fragmentos de código nas produções para mostrar o comportamento das derivações e o uso da tabela de símbolos.

Em um passo posterior, foi criada a estrutura da tabela de símbolos, a qual é definida como hash em nossa implementação. Por enquanto a funcionalidade dessa tabela é apenas receber os identificadores no momento em que são definidos.

Por fim, foi verificado que os conceitos estudados, referentes ao funcionamento de um analisador sintático e suas responsabilidades, auxiliaram o grupo a alcançar êxito na implementação. Também é possível afirmar que esta etapa é fundamental na criação de compilador, visto que muitos erros foram descobertos ao longo da execução deste trabalho, os quais não haviam sido notados com o uso apenas do analisador léxico.

Referências Bibliográficas

[1] AHO, A.V.; LAM, M.S.; SETHI, R.; ULLMAN, J.D. Compiladores: Princípios, técnicas e ferramentas. Segunda Edição. Pearson Addison-Wesley, 2008.

[2] Código base para nossa implementação de hash, pode ser encontrado no github: https://gist.github.com/tonious/1377667/d9e4f51f05992f79455756836c9371942d0f0cee

[3] %code Summary. Disponível em: <https://www.gnu.org/software/bison/manual/html_node/_0025code-Summary.html>

Apêndice A - Código *lex.l* para a linguagem Chameleon

```
%{
#include "custom_defines.h"
#include "y.tab.h" /* para poder usar as constantes dos tokens, por exemplo. */

#define PRINT_ERROR_EOF "-> Um ou mais erros foram encontrados. Corrija-os!\n"

void remover_espacos_e_print(int t);
#define _REAL 1
#define _NUMBER 2

int supress_errors_flag = 0;
int erro_encontrado = 0;
%}
/* condicao exclusiva (bloqueia as demais regras) */
%x comment_condition
%x word_condition
/* condicao que e ativa mas mantem as demais ativadas tambem */
%s supress_errors_condition
/* Permitir a contabilizacao de linhas */
%option   yylineno

supress_errors            "SUPRESS_ERRORS"

/* captura uma ocorrencia de espaco em branco, tabulacao ou quebra de linha*/
delim                     [ \t\n\r]
/* ign (ignorador) ira ignorar um ou mais delim*/
ign                       {delim}+
letter                    [A-Za-z]
digit                     [0-9]
underline                   _
word_value                  (\\.|[^"\\])*
number                    {ign}*({digit}{ign}*)+
type                      "integer"|"word"|"real"

squad_declaration           "squad"
vector_declaration          "vector"

end_squad                   "endsquad"
block_begin                 "begin"
block_end                   "end"

for                       "for"
end_for                     "endfor"
while                     "while"
end_while                   "endwhile"
if                        "if"
end_if                    "endif"
elif                      "elif"
end_elif                    "endelif"
task                      "task"
end_task                    "endtask"

jumpto                    "jumpto"
farewell                  "farewell"
say                       "say"
listen                    "listen"
stop                      "stop"
comma                       ","
open_parenthesis            "("
close_parenthesis           ")"

identifier_complement       ({digit}|{letter}|{underline})*
identifier                ({letter}|{underline}{letter}){identifier_complement}

```

```
65  vector_access_start        "["
66  vector_access_end          "]"
67
68  squad_access_derreference   "->"
69  separator                  ":"
70  real_number                {ign}*({digit}{ign}*)+{ign}*(\.{ign}*({digit}{ign}*)+)
71  word_concat_operator       "++"
72  add_operator               [+-]
73  div_operator               [/*\%]
74  pow_operator               [\^]
75  rel_operator               "=="|"!="|">="|"<="|[><]
76  logic_operator             "and"|"or"|"!"
77  attribution                "="
78  creditos                   "??creditos??"
79
80  %%
81  {supress_errors}           { BEGIN(supress_errors_condition); supress_errors_flag = 1; }
82  "giveup"                   {return GIVEUP;}
83  "//"                      BEGIN(comment_condition);
84  {ign}                      {}
85
86  <comment_condition>.|\n     {}
87  <comment_condition>"\\\\"   { BEGIN(INITIAL); if(supress_errors_flag) BEGIN(supress_errors_condition); }
88
89  "\""                       BEGIN(word_condition);
90  <word_condition>{word_value}       {return WORD_VALUE;}
91  <word_condition>"\""               { BEGIN(INITIAL);  if(supress_errors_flag) BEGIN(supress_errors_condition); }
92
93  {creditos}                 {PRINT(("Feito por:\n%s\n",
94                             "Daniel Freitas Martins - 2304\n"
95                             "João Arthur Gonçalves do Vale - 3025\n"
96                             "Maria Dalila Vieira - 3030\n"
97                             "Naiara Cristiane dos Reis Diniz - 3005"))}
98
99  {type}                     {
100                            if(strcmp("integer", yytext) == 0)
101                               return INTEGER;
102                            if(strcmp("real", yytext) == 0)
103                               return REAL;
104                            return WORD;
105                         }
106 {squad_declaration}     {strcpy(yylval.type_aux, "squad"); return SQUAD;}
107 {vector_declaration}    {strcpy(yylval.type_aux, "vector"); return VECTOR;}
108
109 {end_squad}               {return ENDSQUAD;}
110 {block_begin}             {
111                           yylval.symbol_table = ht_create(MAX_TAM_HASH);
112                           if(curr_symbol_table == NULL){
113                              curr_symbol_table = yylval.symbol_table;
114                              if(first_symbol_table == NULL){
115                                 first_symbol_table = yylval.symbol_table;
116                              }
117                           }
118                           return BLOCK_BEGIN;}
119 {block_end}               {return BLOCK_END;}
120
121 {for}                     {return FOR;}
122 {end_for}                 {return ENDFOR;}
123 {while}                   {return WHILE;}
124 {end_while}               {return ENDWHILE;}
125 {if}                     {return IF;}
126 {end_if}                  {return ENDIF;}
127 {elif}                    {return ELIF;}
128 {end_elif}                {return ENDELIF;}
129 {task}                    {strcpy(yylval.type_aux, "task"); return TASK;}
130 {end_task}                {return ENDTASK;}
131
132 {jumpto}                  {return JUMPTO;}
133 {farewell}                {return FAREWELL;}
```

```
{say}                {return SAY;}
{listen}             {return LISTEN;}
{stop}               {return STOP;}

{comma}              {return yytext[0];}
{open_parenthesis}   {return yytext[0];}
{close_parenthesis}  {return yytext[0];}

{vector_access_start}        {return yytext[0];}
{vector_access_end}          {return yytext[0];}


{squad_access_derreference}    {return SQUAD_ACCESS_DERREFERENCE;}
{separator}                  {return yytext[0];}
{word_concat_operator}         {return WORD_CONCAT_OPERATOR;}
{add_operator}               {return ADD_OPERATOR;}
{div_operator}               {return DIV_OPERATOR;}
{pow_operator}               {return POW_OPERATOR;}
{rel_operator}               {return REL_OPERATOR;}
{logic_operator}             {
                           if(yytext[0] == '!')
                              return NEG_OPERATOR;
                           return LOGIC_OPERATOR;
                         }
{attribution}                {return yytext[0];}

{real_number}                {remover_espacos_e_print(_REAL); return REAL_NUMBER;}
{number}                     {remover_espacos_e_print(_NUMBER); return NUMBER;}

{identifier}                 { createSymbol(&(yylval.symbol), yytext, IDENTIFIER, VALUE);
                           // (yylval.symbol)->data->v.integer = atoi(yytext);
                           return IDENTIFIER;}

<supress_errors_condition>.    {}
";"+                       {} /* ignorando ponto e virgula */
.                          {PRINT((PRINT_ERROR "-> %s\nint_code_s0: %d\n", yylineno, yytext, yytext[0])) erro_encontrado = 1;}
/* Ignorar o que nao foi definido */
<<EOF>>                      {
            if(erro_encontrado){
               PRINT((PRINT_ERROR_EOF))
               exit(1);
            }
            return 0;
         }

%%
void remover_espacos_e_print(int t){
   char* s; /* tera a nova string sem os espacos em branco */
   int i, j, tam_yytext = strlen(yytext);
   s = (char*) malloc(tam_yytext*sizeof(char));
   j = 0;
   for(i = 0; i < tam_yytext; i++){
      if(yytext[i] == ' ' || yytext[i] == '\n' || yytext[i] == '\t')
         continue;
      s[j++] = yytext[i];
   }
   s[j] = '\0';

   strcpy(yytext, s);

   switch(t){
      case _REAL:
         //PRINT((PRINT_REAL_NUMBER PRINT_LEXEME, yylineno, s))
      break;
      case _NUMBER:
         //PRINT((PRINT_NUMBER PRINT_LEXEME, yylineno, s))
      break;
   }
   free(s);
```

```
203 }
204 int yywrap(){ return 1; } /* se EOF for encontrado, encerre. */
```

Apêndice B - Código *translate.y* para a linguagem Chameleon

```
%{
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include "custom_defines.h"

   extern int yylineno;
   extern FILE* yyin;
   int yylex();
   int yyerror(const char*);
   void raiseErrorVariableRedeclaration(char *lexem);
   void raiseError(char *msg);

   short flag_block_continue = 0;
%}
   %code requires{
      #include "hash-table.h"
   }

   %union{
      Symbol *symbol;
      hashtable_t *symbol_table;
      char type_aux[20];
   }

   %define parse.error verbose
   %define parse.lac full
   %token WORD_VALUE
   %token NUMBER REAL_NUMBER
   %token SQUAD_ACCESS_DERREFERENCE
   %token INTEGER WORD REAL
   %token SQUAD ENDSQUAD
   %token VECTOR
   %token IDENTIFIER
   %token ADD_OPERATOR DIV_OPERATOR POW_OPERATOR LOGIC_OPERATOR
   %token NEG_OPERATOR REL_OPERATOR WORD_CONCAT_OPERATOR
   %token GIVEUP
   %token BLOCK_BEGIN BLOCK_END
   %token SAY LISTEN
   %token IF ENDIF ELIF ENDELIF
   %token FOR ENDFOR
   %token WHILE ENDWHILE
   %token TASK ENDTASK
   %token FAREWELL
   %token JUMPTO
   %token STOP

   %left ADD_OPERATOR
   %left DIV_OPERATOR
   %left POW_OPERATOR
   %left LOGIC_OPERATOR
   %left REL_OPERATOR
   %right '='

   %start block
%%

   block: BLOCK_BEGIN {
      if(flag_block_continue){
         flag_block_continue = 0;
         if(curr_symbol_table->brother_hash == NULL){
            hashtable_t *brother_symbol_table = $<symbol_table>1;
            curr_symbol_table->brother_hash = brother_symbol_table;
            curr_symbol_table = brother_symbol_table;
```

```
65          } else{
66             raiseError("O Brother e nao nulo!\n");
67          }
68
69       }
70    } statement BLOCK_END block_continue {PRINT(("block -> [BLOCK_BEGIN statement BLOCK_END
71 block_continue]\n"))}
72       | task_command block_continue                {PRINT(("block -> [task_command block_continue]\n"))}
73    ;
74
75    block_continue: /* palavra vazia */     {PRINT(("block_continue -> []\n"))}
76             | { flag_block_continue = 1; } block                {PRINT(("block_continue -> [block]\n"))}
77    ;
78
79    statement: /* palavra vazia */             {PRINT(("statement -> []\n"))}
80          | command statement                {PRINT(("statement -> [command statement]\n"))}
81          | variable_declaration statement    {PRINT(("statement -> [variable_declaration statement]\n"))}
82          | BLOCK_BEGIN {
83                if(curr_symbol_table->child_hash != NULL){
84                   while(curr_symbol_table->child_hash != NULL){
85                      curr_symbol_table = curr_symbol_table->child_hash;
86                   }
87                   hashtable_t *brother_symbol_table = $<symbol_table>1;
88                   curr_symbol_table->brother_hash = brother_symbol_table;
89                   brother_symbol_table->previous_hash = curr_symbol_table->previous_hash;
90                   curr_symbol_table = brother_symbol_table;
91                } else{
92                   hashtable_t *child_symbol_table = $<symbol_table>1;
93                   child_symbol_table->previous_hash = curr_symbol_table;
94                   curr_symbol_table->child_hash = child_symbol_table;
95                   curr_symbol_table = child_symbol_table;
96                }
97             } statement BLOCK_END {
98                if(curr_symbol_table->previous_hash != NULL)
99                   curr_symbol_table = curr_symbol_table->previous_hash;
100            } statement {PRINT(("statement -> [BLOCK_BEGIN statement BLOCK_END statement]\n"))}
101   ;
102
103   command: variable '=' expression       {PRINT(("command -> [variable '=' expression]\n"))}
104         | variable '=' word_expression  {PRINT(("command -> [variable '=' word_expression]\n"))}
105         | variable '=' task_call        {PRINT(("command -> [variable '=' task_call]\n"))}
106         | givingup                    {PRINT(("command -> [givingup]\n"))}
107         | if_command                    {PRINT(("command -> [if_command]\n"))}
108         | for_command                   {PRINT(("command -> [for_command]\n"))}
109         | while_command                 {PRINT(("command -> [while_command]\n"))}
110         | farewell_command              {PRINT(("command -> [farewell_command]\n"))}
111         | STOP                        {PRINT(("command -> [STOP]\n"))}
112         | jumpto_command                {PRINT(("command -> [jumpto_command]\n"))}
113         | say_command                   {PRINT(("command -> [say_command]\n"))}
114         | listen_command                {PRINT(("command -> [listen_command]\n"))}
115         | task_call                   {PRINT(("command -> [task_call]\n"))}
116         | label                       {PRINT(("command -> [label]\n"))}
117   ;
118
119   say_command: SAY expression     {PRINT(("say_command -> [SAY expression]\n"))}
120         | SAY word_expression   {PRINT(("say_command -> [SAY word_expression]\n"))}
121   ;
122
123   listen_command: LISTEN variable {PRINT(("LISTEN -> [variable]\n"))}
124   ;
125
126   if_command: IF expression ':' statement ENDIF                    {PRINT(("if_command -> [IF expression ':' statement
127 ENDIF]\n"))}
128            | IF expression ':' statement ELIF expression ':' statement ENDELIF  {PRINT(("if_command -> [IF expression ':'
129 statement ELIF expression ':' statement ENDELIF]\n"))}
130   ;
131
132   for_command: FOR expression ',' expression ',' expression ':' statement ENDFOR {PRINT(("for_command -> [FOR
133 expression ',' expression ',' expression ',' ':' statement ENDFOR]\n"))}
```

```
134     ;
135
136     while_command: WHILE expression ':' statement ENDWHILE  {PRINT(("while_command -> [WHILE expression ':'
137 statement ENDWHILE]\n"))}
138     ;
139
140     farewell_command: FAREWELL expression   {PRINT(("farewell_command -> [FAREWELL expression]\n"))}
141     ;
142
143     task_command: TASK IDENTIFIER task_parameters ':' statement ENDTASK {PRINT(("task_command -> [TASK
144 IDENTIFIER task_parameters ':' ENDTASK]\n"))
145                 /*
146                 if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
147                     $<symbol>2->data->v.word = strdup($<type_aux>1);
148                     ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
149                 } else{
150                     char msg[50];
151                     strcpy(msg, "Funcao ");
152                     strcat(msg, $<symbol>2->lexem);
153                     strcat(msg, " ja declarada");
154                     raiseError(msg);
155                 }*/
156             }
157     ;
158
159     task_parameter: expression                  {PRINT(("task_parameter -> [expression]\n"))}
160             | expression ',' task_parameter     {PRINT(("task_parameter -> [expression ',' task_parameter]\n"))}
161     ;
162
163     task_parameters: /* palavra vazia */    {PRINT(("task_parameter -> []\n"))}
164                 | task_parameter        {PRINT(("task_parameter -> [task_parameter]\n"))}
165     ;
166
167     task_call: TASK IDENTIFIER '(' task_parameters ')'   {PRINT(("task_call -> [TASK IDENTIFIER '(' task_parameter ')']\n"))
168                 if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
169                     $<symbol>2->data->v.word = strdup($<type_aux>1);
170                     ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
171                 }
172             }
173     ;
174
175     jumpto_command: JUMPTO IDENTIFIER   {PRINT(("jumpto_command -> [JUMPTO IDENTIFIER]\n"))
176                     if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
177                         $<symbol>2->data->v.word = strdup("jumpto");
178                         ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
179                     }
180                 }
181     ;
182
183     label: IDENTIFIER ':' command       {PRINT(("label -> [IDENTIFIER ':' command]\n"))
184                     if(ht_get(curr_symbol_table, $<symbol>1->lexem) == NULL){
185                         $<symbol>1->data->v.word = strdup("label");
186                         ht_set(curr_symbol_table, $<symbol>1->lexem, $<symbol>1);
187                     }
188                 }
189     ;
190
191     expression: math_expression         {PRINT(("expression -> [math_expression]\n"))}
192             | variable '=' expression   {PRINT(("expression -> [variable '=' expression]\n"))}
193     ;
194
195     ex_aux_abre: '(' math_expression ex_aux_fecha    {PRINT(("ex_aux_abre -> ['(' math_expression ex_aux_fecha]\n"))}
196         | math_term                         {PRINT(("ex_aux_abre -> [math_term]\n"))}
197     ;
198
199     ex_aux_fecha: ')'                       {PRINT(("ex_aux_fecha -> [')']\n"))}
200     ;
201
202     math_expression: ex_aux_abre binary_operators math_expression   {PRINT(("math_expression -> [ex_aux_abre
```

```
203 binary_operators math_expression]\n"))}
204       | unary_operators expression                    {PRINT(("math_expression -> [unary_operators expression]\n"))}
205       | ex_aux_abre                                   {PRINT(("math_expression -> [ex_aux_abre]\n"))}
206    ;
207
208    variable_declarations: variable_declaration                    {PRINT(("variable_declarations -> [variable_declaration]\n"))}
209                 | variable_declaration variable_declarations {PRINT(("variable_declarations -> [variable_declaration
210 variable_declarations]\n"))}
211    ;
212       /* {PRINT(("Squad encontrado %d\n", yylval))} guardando o yylval pra nao esquecermos */
213    variable_declaration: type IDENTIFIER       {PRINT(("variable_declaration -> [type IDENTIFIER]\n"))
214                         if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
215                            $<symbol>2->data->v.word = strdup($<type_aux>1);
216                            ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
217                         } else{
218                            raiseErrorVariableRedeclaration($<symbol>2->lexem);
219                         }
220                       }
221                     | squad_declaration   {PRINT(("variable_declaration -> [squad_declaration]\n"))}
222                     | vector_declaration  {PRINT(("variable_declaration -> [vector_declaration]\n"))}
223    ;
224
225    vector_access: IDENTIFIER '[' NUMBER ']'       {PRINT(("vector_access -> [IDENTIFIER '[' NUMBER ']']\n"))
226                                          /* PRINT((PRINT_ERROR "-> %s\nint_code_s0\n", yylineno, $<symbol>3->lexem))*/}
227              | IDENTIFIER '[' IDENTIFIER ']'   {PRINT(("vector_access -> [IDENTIFIER '[' IDENTIFIER ']']\n"))}
228    ;
229
230    squad_access: IDENTIFIER SQUAD_ACCESS_DERREFERENCE IDENTIFIER       {PRINT(("squad_access ->
231 [IDENTIFIER '->' IDENTIFIER ]\n"))}
232              |  squad_access SQUAD_ACCESS_DERREFERENCE IDENTIFIER   {PRINT(("squad_access -> [IDENTIFIER '->'
233 IDENTIFIER ]\n"))}
234    ;
235
236    squad_declaration: SQUAD IDENTIFIER ':' variable_declarations ENDSQUAD {PRINT(("squad_declaration -> [SQUAD
237 IDENTIFIER ':' variable_declarations ENDSQUAD]\n"))
238                    if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
239                       $<symbol>2->data->v.word = strdup($<type_aux>1);
240                       ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
241                    } else{
242                       raiseErrorVariableRedeclaration($<symbol>2->lexem);
243                    }
244    }
245    ;
246
247    vector_declaration: VECTOR IDENTIFIER NUMBER     {PRINT(("vector_declaration -> [VECTOR IDENTIFIER
248 NUMBER]\n"))
249                       if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
250                          $<symbol>2->data->v.word = strdup($<type_aux>1);
251                          ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
252                       } else{
253                          raiseErrorVariableRedeclaration($<symbol>2->lexem);
254                       }
255                     }
256                | VECTOR IDENTIFIER IDENTIFIER   {PRINT(("vector_declaration -> [VECTOR IDENTIFIER
257 IDENTIFIER]\n"))
258                       if(ht_get(curr_symbol_table, $<symbol>2->lexem) == NULL){
259                          $<symbol>2->data->v.word = strdup($<type_aux>1);
260                          ht_set(curr_symbol_table, $<symbol>2->lexem, $<symbol>2);
261                       } else{
262                          raiseErrorVariableRedeclaration($<symbol>2->lexem);
263                       }
264                  }
265    ;
266
267    type: INTEGER   {PRINT(("type -> [INTEGER]\n"))
268                    strcpy($<type_aux>$, "integer");
269             }
270       | REAL     {PRINT(("type -> [REAL]\n"))
271                    strcpy($<type_aux>$, "real");
```

```
272             }
273       | WORD      {PRINT(("type -> [WORD]\n"))
274                   strcpy($<type_aux>$, "word");
275             }
276    ;
277
278    unary_operators: ADD_OPERATOR   {PRINT(("unary_operators -> [ADD_OPERATOR]\n"))}
279                   | NEG_OPERATOR  {PRINT(("unary_operators -> [NEG_OPERATOR]\n"))}
280    ;
281
282    binary_operators: ADD_OPERATOR   {PRINT(("binary_operators -> [ADD_OPERATOR]\n"))}
283                   | DIV_OPERATOR   {PRINT(("binary_operators -> [DIV_OPERATOR]\n"))}
284                   | POW_OPERATOR   {PRINT(("binary_operators -> [POW_OPERATOR]\n"))}
285                   | LOGIC_OPERATOR {PRINT(("binary_operators -> [LOGIC_OPERATOR]\n"))}
286                   | REL_OPERATOR   {PRINT(("binary_operators -> [REL_OPERATOR]\n"))}
287    ;
288
289    word_term: WORD_VALUE       {PRINT(("word_term -> [WORD_VALUE]\n"))}
290             | expression    {PRINT(("word_term -> [expression]\n"))}
291    ;
292
293    word_term_aux: word_term                          {PRINT(("word_term_aux -> [word_term]\n"))}
294          | word_term WORD_CONCAT_OPERATOR word_term_aux  {PRINT(("word_term_aux -> [word_term
295 WORD_CONCAT_OPERATOR word_term]\n"))}
296    ;
297
298    word_expression: WORD_VALUE                                {PRINT(("word_expression -> [WORD_VALUE]\n"))}
299                   | word_term WORD_CONCAT_OPERATOR word_term_aux  {PRINT(("word_expression -> [word_term
300 WORD_CONCAT_OPERATOR expression]\n"))}
301    ;
302
303    variable: IDENTIFIER    {PRINT(("variable -> [IDENTIFIER]\n"))}
304          | vector_access {PRINT(("variable -> [vector_access]\n"))}
305          | squad_access  {PRINT(("variable -> [squad_access]\n"))}
306    ;
307
308    math_term: NUMBER   {PRINT(("math_term -> [NUMBER]\n"))}
309       | REAL_NUMBER   {PRINT(("math_term -> [REAL_NUMBER]\n"))}
310       | variable      {PRINT(("math_term -> [variable]\n"))} /* ATENCAO A ESTA VARIAVEL AQUI */
311    ;
312
313
314    givingup: GIVEUP     {PRINT(("givingup -> [GIVEUP]\nADEUS :)\n"))
315                         exit(0);}
316    ;
317
318 %%
319    void raiseError(char *msg){
320       printf("Error: %s\n", msg);
321    }
322
323    void raiseErrorVariableRedeclaration(char *lexem){
324       char error_msg[50];
325       char *lexem_copy = strdup(lexem);
326       lexem_copy[strlen(lexem_copy)-4]='\0';
327       sprintf(error_msg, "A variavel %s ja foi declarada!", lexem_copy);
328       free(lexem_copy);
329       yyerror(error_msg);
330    }
331
332    int yyerror(const char *s){
333       fprintf(stderr, "Linha: %d ---> %s <---\n\n", yylineno, s);
334       exit(1);
335    }
336    int main(int argc, char *argv[]){
337       first_symbol_table = NULL;
338       curr_symbol_table = NULL;
339       //printf("Numero de parametros: %d\n", argc);
340       if(argc == 2){
```

```
341         //FILE *f;
342         if(!(yyin = fopen(argv[1], "r"))){
343             PRINT(("Nao foi possivel abrir o arquivo!\n"))
344             return 0;
345         }
346         char c;
347         printf("-------------- Readed Code --------------\n\n");
348         printf("1 ");
349         int l = 2;
350         while((c = fgetc(yyin)) != EOF){
351             printf("%c", c);
352             if(c == '\n'){
353                 printf("%d ", l++);
354             }
355         }
356         printf("\n\n");
357         rewind(yyin);
358     }
359
360     int r = yyparse();
361     ht_print(first_symbol_table);
362     return r;
363 }
```

Apêndice C - *symbol.h*

```
1   #ifndef ESTRUTURA_TIPOS_INCLUDED
2   #define ESTRUTURA_TIPOS_INCLUDED
3
4   #ifdef __cplusplus
5   extern "C" {
6   #endif
7
8      #include <stdlib.h>
9      #include <stdio.h>
10     #include <limits.h>
11     #include <string.h>
12
13     //#define TAM_LEXEMA 255 // colocar?
14
15     enum SymbolType {FUNCTION, VALUE, AGGREGATED};
16
17     typedef void* (Function) (void* p, ...);
18
19     typedef struct Squad{
20        struct Symbol *internal_variables; // TODO: Arrumar
21     } Squad;
22
23     typedef struct Data{
24        enum SymbolType type;
25        union{
26           int integer;
27           double real;
28           char *word;
29           Squad squad;
30           Function *Function;
31        } v;
32     } Data;
33
34     typedef struct Symbol{
35        char *lexem;
36        int token_type;
37        Data *data;
38     } Symbol;
39
40     void createData(Data **data, enum SymbolType type);
41     void _createSymbol(Symbol **symbol, char *lexem, int token_type, Data *data);
42     void createSymbol(Symbol **symbol, char *lexem, int token_type, enum SymbolType type);
43     void destroySymbol(Symbol **symbol);
44     void destroyData(Data **data);
45     void destroySquad(Squad **squad);
46
47  #ifdef __cplusplus
48  }
49  #endif
50
51  #endif /* ESTRUTURA_TIPOS_INCLUDED  */
```

Apêndice D - *symbol.c*

```
1   #include "symbol.h"
2
3   void createData(Data **data, enum SymbolType type){
4       (*data) = (Data*) malloc(sizeof(Data));
5       (*data)->type = type;
6   }
7
8   void _createSymbol(Symbol **symbol, char *lexem, int token_type, Data *data){
9       (*symbol) = (Symbol*) malloc(sizeof(Symbol));
10      (*symbol)->lexem = strdup(lexem);
11      strcat((*symbol)->lexem, "_key");
12      (*symbol)->token_type = token_type;
13      (*symbol)->data = data;
14  }
15
16  void createSymbol(Symbol **symbol, char *lexem, int token_type, enum SymbolType type){
17
18      Data *d;
19      createData(&d, type);
20      _createSymbol(symbol, lexem,token_type, d);
21  }
22
23  void destroySymbol(Symbol **symbol){
24      if(symbol != NULL && *symbol != NULL){
25          Symbol *removedor = (*symbol); // TODO: melhorar isso, dando free nas coisas internas
26          free(removedor);
27      }
28  }
29
30  void destroyData(Data **data){
31      if(data != NULL && *data != NULL){
32          Data *removedor = (*data);
33          free(removedor);
34      }
35  }
36
37  void destroySquad(Squad **squad){
38      // TODO
39  }
```

Apêndice E - *hash-table.h*

```
1   // https://gist.github.com/tonious/1377667/d9e4f51f05992f79455756836c9371942d0f0cee
2
3   #include <stdlib.h>
4   #include <stdio.h>
5   #include <limits.h>
6   #include <string.h>
7   #include "symbol.h"
8
9   #define MAX_TAM_HASH 500
10
11  struct entry_s {
12      int fake_memory_address;
13      char *key;
14      Symbol *value;
15      struct entry_s *next;
16  };
17
18  typedef struct entry_s entry_t;
19  typedef struct hashtable_s hashtable_t;
20
21  struct hashtable_s {
22      int size;
23      struct entry_s **table;
24      hashtable_t *previous_hash;
25      hashtable_t *child_hash;
26      hashtable_t *brother_hash;
27  };
28
29  hashtable_t *first_symbol_table; // global
30  hashtable_t *curr_symbol_table; // global
31
32  hashtable_t *ht_create( int size );
33  int ht_hash( hashtable_t *hashtable, char *key ) ;
34  entry_t *ht_newpair( char *key, Symbol *value ) ;
35  void ht_set( hashtable_t *hashtable, char *key, Symbol *value ) ;
36  Symbol *ht_get( hashtable_t *hashtable, char *key ) ;
37  void ht_print( hashtable_t *hashtable );
```

Apêndice F - *hash-table.c*

```
1   #include "hash-table.h"
2
3   // Create a new hashtable.
4   hashtable_t *ht_create( int size ) {
5
6       hashtable_t *hashtable = NULL;
7       int i;
8
9       if( size < 1 ) return NULL;
10
11      // Allocate the table itself.
12      if( ( hashtable = malloc( sizeof( hashtable_t ) ) ) == NULL ) {
13          return NULL;
14      }
15
16      // Allocate pointers to the head nodes.
17      if( ( hashtable->table = malloc( sizeof( entry_t * ) * size ) ) == NULL ) {
18          return NULL;
19      }
20      for( i = 0; i < size; i++ ) {
21          hashtable->table[i] == NULL;
22      }
23
24      hashtable->size = size;
25      hashtable->previous_hash = NULL;
26      hashtable->brother_hash = NULL;
27      hashtable->child_hash = NULL;
28
29      return hashtable;
30  }
31
32  // Hash a string for a particular hash table.
33  int ht_hash( hashtable_t *hashtable, char *key ) {
34
35      unsigned long int hashval;
36      int i = 0;
37      //printf("Imakey %s  %d\n", key, strlen( key ));
38      // Convert our string to an integer
39      while( hashval < ULONG_MAX && i < strlen( key ) ) {
40          hashval = hashval << 8;
41          hashval += key[ i ];
42          i++;
43      }
44      //printf("HASHVAL: %ul    %d      %d\n", hashval, hashtable->size, hashval % hashtable->size);
45      return hashval % hashtable->size;
46  }
47
48  // Create a key-value pair.
49  entry_t *ht_newpair( char *key, Symbol *value ) {
50      entry_t *newpair;
51
52      if( ( newpair = malloc( sizeof( entry_t ) ) ) == NULL ) {
53          return NULL;
54      }
55
56      if( ( newpair->key = strdup( key ) ) == NULL ) {
57          return NULL;
58      }
59
60      if( ( newpair->value = value ) == NULL ) {
61          return NULL;
62      }
63
64      newpair->next = NULL;
```

```
65
66          return newpair;
67  }
68
69  // Insert a key-value pair into a hash table.
70  void ht_set( hashtable_t *hashtable, char *key, Symbol *value ) {
71          int bin = 0;
72          entry_t *newpair = NULL;
73          entry_t *next = NULL;
74          entry_t *last = NULL;
75
76          bin = ht_hash( hashtable, key );
77
78          next = hashtable->table[ bin ];
79
80          while( next != NULL && next->key != NULL && strcmp( key, next->key ) > 0 ) {
81                  last = next;
82                  next = next->next;
83          }
84
85          // There's already a pair.  Let's replace that string.
86          if( next != NULL && next->key != NULL && strcmp( key, next->key ) == 0 ) {
87
88                  //free( next->value );
89                  //printf("TTTTTT\n");
90                  destroySymbol(&(next->value));
91                  next->value = value;
92
93          // Nope, could't find it.  Time to grow a pair.
94          } else {
95                  //printf("YYYYYYYYY %d\n", bin);
96                  newpair = ht_newpair( key, value );
97                  newpair->fake_memory_address = bin;
98
99                  // We're at the start of the linked list in this bin.
100                 if( next == hashtable->table[ bin ] ) {
101                         newpair->next = next;
102                         hashtable->table[ bin ] = newpair;
103
104                 // We're at the end of the linked list in this bin.
105                 } else if ( next == NULL ) {
106                         last->next = newpair;
107
108                 // We're in the middle of the list.
109                 } else  {
110                         newpair->next = next;
111                         last->next = newpair;
112                 }
113         }
114 }
115
116 // Retrieve a key-value pair from a hash table.
117 Symbol *ht_get( hashtable_t *hashtable, char *key ) {
118         int bin = 0;
119         entry_t *pair;
120         //printf("CHAVE: %s\n", key);
121         bin = ht_hash( hashtable, key );
122         //printf("%d\n", bin);
123         // Step through the bin, looking for our value.
124         pair = hashtable->table[ bin ];
125         while( pair != NULL && pair->key != NULL && strcmp( key, pair->key ) > 0 ) {
126                 pair = pair->next;
127         }
128
129         // Did we actually find anything?
130         if( pair == NULL || pair->key == NULL || strcmp( key, pair->key ) != 0 ) {
131                 /*printf("AAAAAAA\n");
132                 if(pair == NULL) printf("ee\n");
133                 else if(pair->key == NULL) printf("ff\n");
```

```
134             else if(strcmp( key, pair->key ) != 0) printf("ggg\n");*/
135
136             return NULL;
137
138         } else {
139             return pair->value;
140         }
141
142 }
143
144 void ht_print( hashtable_t *hashtable ){
145         if(hashtable == NULL){
146             return;
147         }
148         hashtable_t *st = hashtable;
149         int i = 0;
150         entry_t *pair;
151         printf("\n----------------------------------- SYMBOL TABLE -----------------------------------\n\n");
152         int t_max = 43;
153         int t = t_max-strlen("Lexem");
154         printf("Lexem%*s%*s\n\n", t <= 0 ? 1 : t, "Data Type", t_max, "Memory Address (fake)");
155         for(i = 0; i < st->size; i++){
156             pair = st->table[i];
157             if(pair != NULL){
158                 //char *key = pair->key;
159                 Symbol *s = pair->value;
160                 char *lexem_copy = strdup(s->lexem);
161                 lexem_copy[strlen(lexem_copy)-4]='\0';
162                 int t = t_max-strlen(lexem_copy);
163                 printf("%s%*s%*d\n", lexem_copy, t <= 0 ? 1 : t, s->data->v.word, t_max, pair->fake_memory_address);
164                 free(lexem_copy);
165             }
166         }
167         printf("\n-------------------------------------------------------------------------------------------\n\n");
168
169         if(hashtable->child_hash != NULL){ // imprime os filhos primeiro
170             ht_print(hashtable->child_hash);
171         }
172         if(hashtable->brother_hash != NULL){
173             ht_print(hashtable->brother_hash);
174         }
175 }
```

Apêndice G - *custom_defines.h*

```
1  #ifndef CUSTOM_DEFINES_H_HEADER_
2  #define CUSTOM_DEFINES_H_HEADER_
3
4  #define PRINT_ERROR "---------- Erro encontrado na linha %d ----------\n"
5  #define PRINT(args) printf args ;
6
7  #endif
```

Apêndice H - *y.tab.h* (gerado pelo YACC)

```
1   /* A Bison parser, made by GNU Bison 3.5.1.  */
2
3   /* Bison interface for Yacc-like parsers in C
4
5     Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2020 Free Software Foundation,
6     Inc.
7
8     This program is free software: you can redistribute it and/or modify
9     it under the terms of the GNU General Public License as published by
10    the Free Software Foundation, either version 3 of the License, or
11    (at your option) any later version.
12
13    This program is distributed in the hope that it will be useful,
14    but WITHOUT ANY WARRANTY; without even the implied warranty of
15    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
16    GNU General Public License for more details.
17
18    You should have received a copy of the GNU General Public License
19    along with this program.  If not, see <http://www.gnu.org/licenses/>.  */
20
21  /* As a special exception, you may create a larger work that contains
22    part or all of the Bison parser skeleton and distribute that work
23    under terms of your choice, so long as that work isn't itself a
24    parser generator using the skeleton or a modified version thereof
25    as a parser skeleton.  Alternatively, if you modify or redistribute
26    the parser skeleton itself, you may (at your option) remove this
27    special exception, which will cause the skeleton and the resulting
28    Bison output files to be licensed under the GNU General Public
29    License without this special exception.
30
31    This special exception was added by the Free Software Foundation in
32    version 2.2 of Bison.  */
33
34  /* Undocumented macros, especially those whose name start with YY_,
35    are private implementation details.  Do not rely on them.  */
36
37  #ifndef YY_YY_Y_TAB_H_INCLUDED
38  # define YY_YY_Y_TAB_H_INCLUDED
39  /* Debug traces.  */
40  #ifndef YYDEBUG
41  # define YYDEBUG 0
42  #endif
43  #if YYDEBUG
44  extern int yydebug;
45  #endif
46  /* "%code requires" blocks.  */
47  #line 16 "translate.y"
48
49        #include "hash-table.h"
50
51
52  #line 53 "y.tab.h"
53
54  /* Token type.  */
55  #ifndef YYTOKENTYPE
56  # define YYTOKENTYPE
57    enum yytokentype
58    {
59      WORD_VALUE = 258,
60      NUMBER = 259,
61      REAL_NUMBER = 260,
62      SQUAD_ACCESS_DERREFERENCE = 261,
63      INTEGER = 262,
64      WORD = 263,
```

```
65      REAL = 264,
66      SQUAD = 265,
67      ENDSQUAD = 266,
68      VECTOR = 267,
69      IDENTIFIER = 268,
70      ADD_OPERATOR = 269,
71      DIV_OPERATOR = 270,
72      POW_OPERATOR = 271,
73      LOGIC_OPERATOR = 272,
74      NEG_OPERATOR = 273,
75      REL_OPERATOR = 274,
76      WORD_CONCAT_OPERATOR = 275,
77      GIVEUP = 276,
78      BLOCK_BEGIN = 277,
79      BLOCK_END = 278,
80      SAY = 279,
81      LISTEN = 280,
82      IF = 281,
83      ENDIF = 282,
84      ELIF = 283,
85      ENDELIF = 284,
86      FOR = 285,
87      ENDFOR = 286,
88      WHILE = 287,
89      ENDWHILE = 288,
90      TASK = 289,
91      ENDTASK = 290,
92      FAREWELL = 291,
93      JUMPTO = 292,
94      STOP = 293
95    };
96  #endif
97  /* Tokens.  */
98  #define WORD_VALUE 258
99  #define NUMBER 259
100 #define REAL_NUMBER 260
101 #define SQUAD_ACCESS_DERREFERENCE 261
102 #define INTEGER 262
103 #define WORD 263
104 #define REAL 264
105 #define SQUAD 265
106 #define ENDSQUAD 266
107 #define VECTOR 267
108 #define IDENTIFIER 268
109 #define ADD_OPERATOR 269
110 #define DIV_OPERATOR 270
111 #define POW_OPERATOR 271
112 #define LOGIC_OPERATOR 272
113 #define NEG_OPERATOR 273
114 #define REL_OPERATOR 274
115 #define WORD_CONCAT_OPERATOR 275
116 #define GIVEUP 276
117 #define BLOCK_BEGIN 277
118 #define BLOCK_END 278
119 #define SAY 279
120 #define LISTEN 280
121 #define IF 281
122 #define ENDIF 282
123 #define ELIF 283
124 #define ENDELIF 284
125 #define FOR 285
126 #define ENDFOR 286
127 #define WHILE 287
128 #define ENDWHILE 288
129 #define TASK 289
130 #define ENDTASK 290
131 #define FAREWELL 291
132 #define JUMPTO 292
133 #define STOP 293
```

```
134
135 /* Value type.  */
136 #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
137 union YYSTYPE
138 {
139 #line 20 "translate.y"
140
141         Symbol *symbol;
142         hashtable_t *symbol_table;
143         char type_aux[20];
144
145
146 #line 147 "y.tab.h"
147
148 };
149 typedef union YYSTYPE YYSTYPE;
150 # define YYSTYPE_IS_TRIVIAL 1
151 # define YYSTYPE_IS_DECLARED 1
152 #endif
153
154
155 extern YYSTYPE yylval;
156
157 int yyparse (void);
158
159 #endif /* !YY_YY_Y_TAB_H_INCLUDED  */
```