Relatório do Compilador para a linguagem C.rveja Linguagens e Compiladores PCS-2056

Escola Politécnica da USP

PCS

Dezembro/2009

#hellweek

Leandro Cordeiro David
Wilson Faria

1) Introdução

Este documento apresenta a especificação, desenvolvimento e resultados obtidos de compilador que gera código para a maquina MVN. Este projeto foi feito no âmbito da disciplina PCS2056 - Linguagens e Compiladores. A implementação do compilador foi feita através da ferramenta Eclipse, utilizando a linguagem Java. Algumas outras ferramentas foram utilizadas para auxílio na documentação e projeto, como o SVN para controle de versão dos documentos e do código, hospedando no Google Code, e o JFlap para a geração dos autômatos.

2) Definição da linguagem

2.1) Dados Gerais

A linguagem **"C.rveja"** desenvolvida para esse projeto foi, a princípio, baseada na linguagem estruturada C. Várias adaptações foram feitas e a linugagem final é bastante diferente da linguagem C. Para simplificar o projeto, vários detalhes não foram implementados, como a operação de exponenciação, por exemplo, e algumas estruturas foram modificadas, como a função de retorno void, cujo nome foi mudado para "procedure".

2.2) Gramática Formal

A seguir apresentamos a descrição da linguagem, em notação de Wirth:

Gramática completa:

programa = "main" **escopoproc** { (**funcao**|**procedimento**)}.

escopoproc = "{" {(**declaracao**|**comando**)} "}".

escopofunc = "{" {(**declaracao**|**comando**)} "return" "identificador" "}".

funcao = "function" **tipo** "identificador" "(" [**tipo** "identificador" {",”**tipo** "identificador"}]")”**escopofunc.**

procedimento ="procedure" "identificador" "(" [**tipo** "identificador" {",”**tipo** "identificador"}]")”**escopoproc.**

tipo = ”int” **indicemultiplo** |”float”**indicemultiplo** |”string”**indicemultiplo** |”boolean”**indicemultiplo** |**"identificador" indicemultiplo**.

indicemultiplo ={"["("inteiro"|"identificador"**)**"]"}.

declaracao = **tipo** "identificador" ";"|"struct" "identificador" "{" {**tipo** "identificador" ";" } "}".

comando = **atribuicao** | **condicional** | **iterativo** | **entrada** | **saida** | **selecao**| **chamadaprocedimento**";".
atribuicao = "identificador" **indicemultiplo** "=" **expressao** ";".
condicional = "if" "("**expbooleana**")" **escopo** ["else" **escopo**].
iterativo = "while" "("**expbooleana**")" **escopo**.
entrada = "input" "identificador" **indicemultiplo** ";".
saida = "output" "identificador" **indicemultiplo** ";".

expressao = expbooleana|**exparitmetica|expstring|exprelacional|chamadafuncao.**

expbooleana = **termobool** {**oplogico termobool**}.
termobool = "identificador" **indicemultiplo |truefalse|**"("**expbooleana**")"**|**"!"
"("**expbooleana**")".

exprelacional = "identificador" **indicemultiplo |termorel oprel termorel.**

termorel = "numero"|"identificador" **indicemultiplo** .

oprel = "<"|">"|"<="|">="|"=="|"!=".
truefalse = "true"| "false".
oplogico = "|"|"&"|"!="|"==".

exparitmetica =["-"]**termoarit** {("+"|"-")**termoarit**}.
termoarit = **fator**{("/"|"*")**fator**}.

fator="identificador" **indicemultiplo** |"numero"|"("**exparitmetica**")".

expstring = stringidentificador{"+"**stringidentificador**}.
stringidentificador = "string" |"identificador"**.**
chamadafuncao= "callfunc" "identificador" "("[("numero"|"string"|"identificador"
indicemultiplo){","("numero"|"string"|"identificador" **indicemultiplo**)}]")"";".
chamadaprocedimento = "callproc" "identificador" "("[("numero"|"string"|"identificador"
indicemultiplo){","("numero"|"string"|"identificador" **indicemultiplo**)}]")"";".

E abaixo uma lista com os terminais da linguagem
Lista de Terminais:
"

1. "main"
2. "struct"
3. "declare"
4. "identificador"
5. "true"
6. "false"
7. "("
8. "!"
9. "["
10. "|"
11. "&"
12. "!="
13. "=="
14. "inteiro"
15. ")"
16. "]"
17. "int"
18. "float"
19. "string"
20. "boolean"
21. "{"
22. "}"
23. "procedure"
24. ","
25. "numero"
26. "<"
27. ">"
28. "<="
29. ">="
30. "if"

31. "while"
32. "input"
33. "output"
34. "callproc"
35. "callfunc"
36. "="
37. "else"
38. "-"
39. "/"
40. "*"
41. "+"
42. "return"
43. "last"

2.3) Exemplo de Código

Segue um exemplo de um código que realiza "bubblesort" na linguagem:

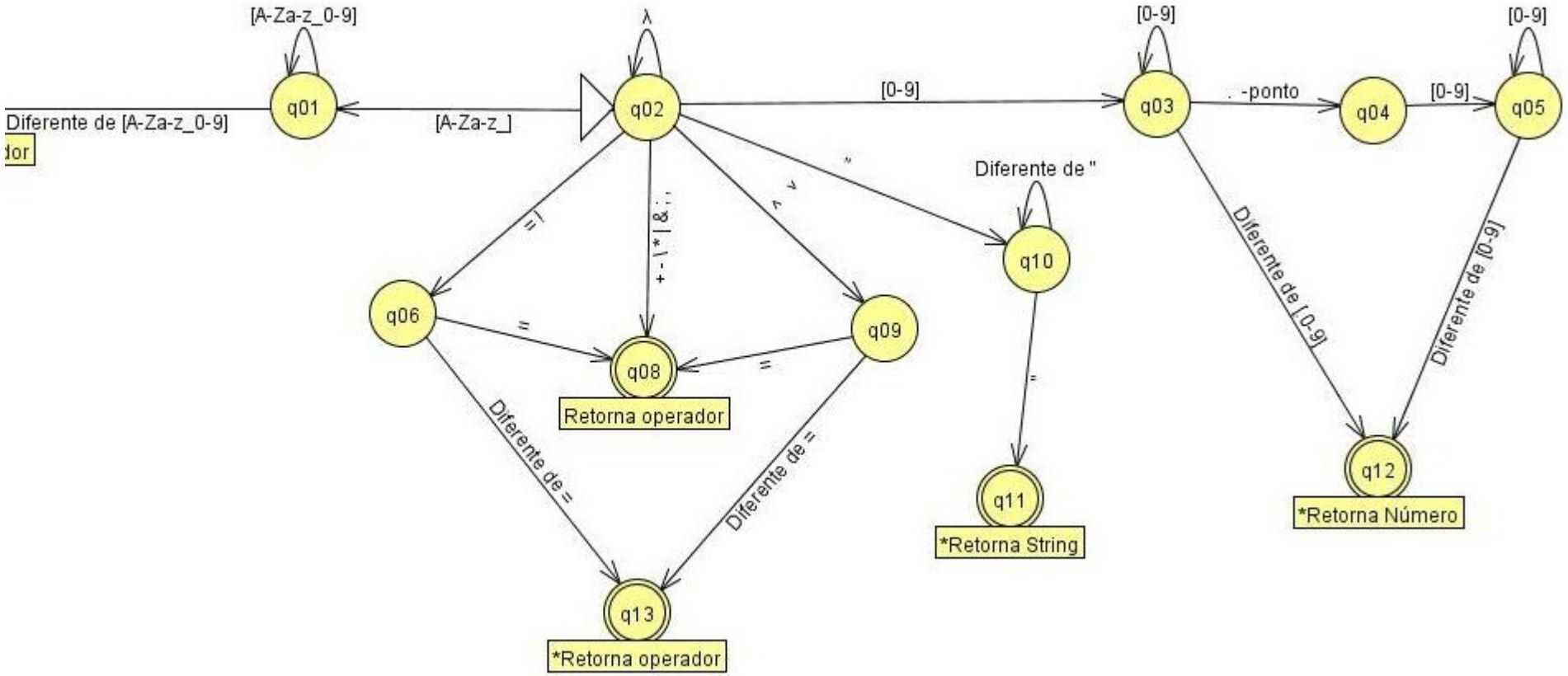
```
main{
   int[10] vetor;
   int i;
   i=0;
   while(i<=10){
      vetor[i] = mix(i);
      i=i+1;
   }
   vetor = bubble(vetor,10);
   cospe(vetor);
}

function int[] mix(int semente){
   int valor;
   if(semente == 0){ valor=  9 }
   if(semente == 1{ valor=  3 }
   if(semente == 2{ valor=  8 }
   if(semente == 3{ valor=  1 }
   if(semente == 4{ valor=  2 }
   if(semente == 5{ valor=  7 }
   if(semente == 6{ valor=  4 }
   if(semente == 7{ valor=  5 }
   if(semente == 8{ valor=  0 }
   if(semente == 9{ input valor }
   }
   return valor;
}

function int[] bubble( int v[], int qtd )
{
    int i;
    int j;
    int j2;
    int aux;
    int k ;
    k = qtd - 1 ;
    i=0;
    while(i<=qtd)
    {
```

```
        j=0;
        while(j<k)
        {
            j2=j+1;
            if(v[j] > v[j2])
            {
                aux = v[j];
                v[j] = v[j2];
                v[j2]=aux;
             }
        j=j+1;
        }
    i=i+1;
    }
    return v;
}
```

```
procedure cospe(int[] vetor, int tamanho){
   int i;
   while (i<tamanho){
      output vetor[i];
      i=i+1;
   }
}
```

3) Análise Léxica

A análise léxica tem a função de percorrer cada caracter do código e verificar se a sequência de palavras no mesmo correspondem aos requisitos da linguagem. Cada partícula do código é chamada de "tokem", e a função do léxico é encontrar todos os tokens, separá-los por tipo, e listá-los. O analisador léxico do compilador da linguagem C.rveja foi construido baseado em uma tabela de transições gerada a partir do autômato base, e, ao alcançar estados finais do mesmo, o léxico gera a lista tokens de acordo com seus tipos. Foram definidos inúmeros tipos de tokens, que vão desde tipos básicos, como "int", "string", etc, incluindo também palavras reservadas, símbolos especiais, operadores, e delimitadores de escopo. O analisador léxico possui tipos determinados de saída: Número, String, Operador e Identificador. Ele também detecta inconsistências básicas de codificação, apenas em nível de caracteres errados ou tokens em lugares indevidos. O autômato utilizado na análise léxica do compilador para a C.rveja segue abaixo:


[A-Za-z_0-9]
λ
[0-9]
[0-9]
Diferente de [A-Za-z_0-9]
dor
q01
[A-Za-z_]
q02
[0-9]
q03
. -ponto
q04
[0-9]
q05
=
+ - * | & : ;
^ v
"
Diferente de "
q10
Diferente de [0-9]
Diferente de [0-9]
q06
=
q08
Retorna operador
q09
=
=
Diferente de =
Diferente de =
q11
*Retorna String
q12
*Retorna Número
q13
*Retorna operador

Tipos de Token gerados
O analisador léxico contém uma classe TipoToken do tipo Enum que lista os seguintes possívos tipos de Token

1. ID
2. NUMERO
3. STRING
4. PR_TRUE
5. PR_FALSE
6. PR_INT
7. PR_FLOAT
8. PR_STRING
9. PR_BOOLEAN
10. PR_PROCEDURE
11. PR_CALLPROC
12. PR_FUNCTION
13. PR_CALLFUNC
14. PR_IF
15. PR_ELSE
16. PR_WHILE
17. PR_INPUT
18. PR_OUTPUT
19. PR_RETURN
20. PARENTESE_ABRE
21. PARENTESE_FECHA
22. NOT
23. COLCHETE_ABRE
24. COLCHETE_FECHA
25. CHAVE_ABRE
26. CHAVE_FECHA
27. OR
28. AND
29. DIFFERENT
30. EQUAL_COMPARISON
31. DIVIDE
32. MULTIPLY
33. PONTOEVIRGULA
34. VIRGULA
35. ATRIB
36. MAIOR_IGUAL
37. MENOR_IGUAL
38. MAIOR
39. MENOR
40. MINUS
41. PLUS
42. QUEBRADO
43. PR_MAIN
44. PR_STRUCT
45. PR_DECLARE
46. LAST

4) Análise Sintática

O analisador sintático preocupa-se em reconhecer os tokens gerados, verificando se estão de acordo com a gramática. É no analisador sintático, portanto, que as submáquinas são geradas, sendo que cada submáquina chamará uma ação semântica posteriormente.

4.1) Gramática Simplificada

A partir da descriçãocompleta da linguagem em wirth, foi gerada uma nova descrição simplificada em notação de wirth. Esta descrição simplificada reduz a quantidade de submáquinas. para a linguagem C.rveja.

programa = "main" "{" {(declaracao|comando)} "}" { (funcao|procedimento)}.
funcao = "function" ("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"|"identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} |"identificador" {"["("inteiro"|"identificador")"]"}) "identificador" "(" [("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"|"identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} |"identificador" {"["("inteiro"|"identificador")"]"}) "identificador" {"," ("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"|"identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} |"identificador" {"["("inteiro"|"identificador")"]"}) "identificador"}]")" "{" {(declaracao|comando)} "return" "identificador" ";" "}" .
procedimento ="procedure" "identificador" "(" [("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"|"identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} |"identificador" {"["("inteiro"|"identificador")"]"}) "identificador" { "," ("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"|"identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} |"identificador" {"["("inteiro"|"identificador")"]"}) "identificador"}]")" "{" {(declaracao|comando)} "}".
declaracao = (("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"|"identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} | "declare" "identificador" {"["("inteiro"|"identificador")"]"}) "identificador" ";"|"struct" "identificador" "{" { ("int" {"["("inteiro"|"identificador")"]"} |"float" {"["("inteiro"|"identificador")"]"} |"string" {"["("inteiro"| "identificador")"]"} |"boolean" {"["("inteiro"|"identificador")"]"} |"identificador" {"["("inteiro"|"identificador")"]"}) "identificador" ";" } "}").
comando = ("identificador" {"["("inteiro"|"identificador")"]"} "=" expressao ";" | "if" "("expbooleana")" "{" {(declaracao|comando)} "}" ["else" "{" {(declaracao|comando)} "}"] | "while" "("expbooleana")" "{" {(declaracao|comando)} "}" | "input" "identificador" {"["("inteiro"|"identificador")"]"} ";" | "output" "identificador" {"["("inteiro"|"identificador")"]"} ";" | "callproc" "identificador" "("[("numero"|"string"|"identificador" {"["("inteiro"|"identificador")"]"}) {","("numero"|"string"|"identificador" {"["("inteiro"|"identificador")"]"})}]")" ";").
expressao = (expbooleana|exparitmetica|expstring|exprelacional| "callfunc" "identificador" "("[("numero"|"string"|"identificador" {"["("inteiro"|"identificador")"]"}) {","("numero"|"string"|"identificador" {"["("inteiro"|"identificador")"]"})}]")" ";").
expbooleana = ("identificador" {"["("inteiro"|"identificador")"]"} | ("true"| "false") |"("expbooleana")"|"!" "("expbooleana")") { ("|"|"&"|"!="|"==") ("identificador" {"["("inteiro"|"identificador")"]"} | ("true"| "false") |"("expbooleana")"|"!" "("expbooleana")") }.
exprelacional = ("identificador" {"["("inteiro"|"identificador")"]"} | ("numero"|"identificador" {"["("inteiro"|"identificador")"]"}) ("<"|">"|"<="|">="|"=="|"!=") ("numero"|"identificador" {"["("inteiro"|"identificador")"]"})).
exparitmetica =("-"|"+") ("identificador" {"["("inteiro"|"identificador")"]"} |"numero"|"("exparitmetica")") {("/"|"*") ("identificador" {"["("inteiro"|"identificador")"]"} |"numero"|"("exparitmetica")") } {("+"|"-") ("identificador" {"["("inteiro"|"identificador")"]"} |"numero"|"("exparitmetica")") {("/"|"*") ("identificador" {"["("inteiro"|"identificador")"]"} |"numero"|"("exparitmetica")") } }.
expstring = "tipo_string" {"+" ("tipo_string" |"identificador")}.

4.2) Submáquinas Geradas - Ações semânticas

A seguir, são apresentadas as descrições dos autômatos para cada submáquina, com as respectivas marcações de estado.

Programa:

Marcação de estados:
programa = 0 *"main" 1 "{" 2 { 3 (3* ***declaracao*** *5* | *3* ***comando*** *6) 4 } 3 "}"* 7 { 8 (8 **funcao** 10 | 8 **procedimento** 11) 9 } 8 .

Autômato gerado

initial: 0
final: 3
0:(0, "main") -> 1 Ação Semântica:
1:(1, "{") -> 2 Ação Semântica:Gera início do código MVN - Inicia novo escopo
2:(2, declaracao) -> 2 Ação Semântica:
3:(2, comando) -> 2 Ação Semântica:
4:(2, "}") -> 3 Ação Semântica: Finaliza um escopo
5:(3, funcao) -> 3 Ação Semântica:
6:(3, procedimento) -> 3 Ação Semântica:

Automato:

Funcao:

Marcação de estados

funcao = 0 *"function" 1 (1 "int" 3 { 4 "[" 5 (5 "inteiro" 7 | 5 "identificador" 8) 6 "]" 9 } 4 | 1 "float" 10 { 11 "[" 12 (12 "inteiro" 14 | 12 "identificador" 15) 13 "]" 16 } 11 | 1 "string" 17 { 18 "[" 19 (19 "inteiro" 21 | 19 "identificador" 22) 20 "]" 23 } 18 | 1 "boolean" 24 { 25 "[" 26 (26 "inteiro" 28 | 26 "identificador" 29) 27 "]" 30 } 25 | 1 "identificador" 31 { 32 "[" 33 (33 "inteiro" 35 | 33 "identificador" 36) 34 "]" 37 } 32) 2 "identificador" 38 "(" 39 [39 (39 "int" 42 { 43 "[" 44 (44 "inteiro" 46 | 44 "identificador" 47) 45 "]" 48 } 43 | 39 "float" 49 { 50 "[" 51 (51 "inteiro" 53 | 51 "identificador" 54) 52 "]" 55 } 50 | 39 "string" 56 { 57 "[" 58 (58 "inteiro" 60 | 58 "identificador" 61) 59 "]" 62 } 57 | 39 "boolean" 63 { 64 "[" 65 (65 "inteiro" 67 | 65 "identificador" 68) 66 "]" 69 } 64 | 39 "identificador" 70 { 71 "[" 72 (72 "inteiro" 74 | 72 "identificador" 75) 73 "]" 76 } 71) 41 "identificador" 77 { 78 "," 79 (79 "int" 81 { 82 "[" 83 (83 "inteiro" 85 | 83 "identificador" 86) 84 "]" 87 } 82 | 79 "float" 88 { 89 "["*

90 (90 "inteiro" 92 | 90 "identificador" 93) 91 "]" 94 } 89 | 79 "string" 95 { 96 "[" 97 (97 "inteiro" 99 | 97 "identificador" 100) 98 "]" 101 } 96 | 79 "boolean" 102 { 103 "[" 104 (104 "inteiro" 106 | 104 "identificador" 107) 105 "]" 108 } 103 | 79 "identificador" 109 { 110 "[" 111 (111 "inteiro" 113 | 111 "identificador" 114) 112 "]" 115 } 110) 80 "identificador" 116 } 78] 40 ")" 117 "{" 118 { 119 (119 ***declaracao*** *121 | 119* ***comando*** *122) 120 } 119 "return" 123 "identificador" 124 ";" 125 "}"* 126 .

Autômato gerado:

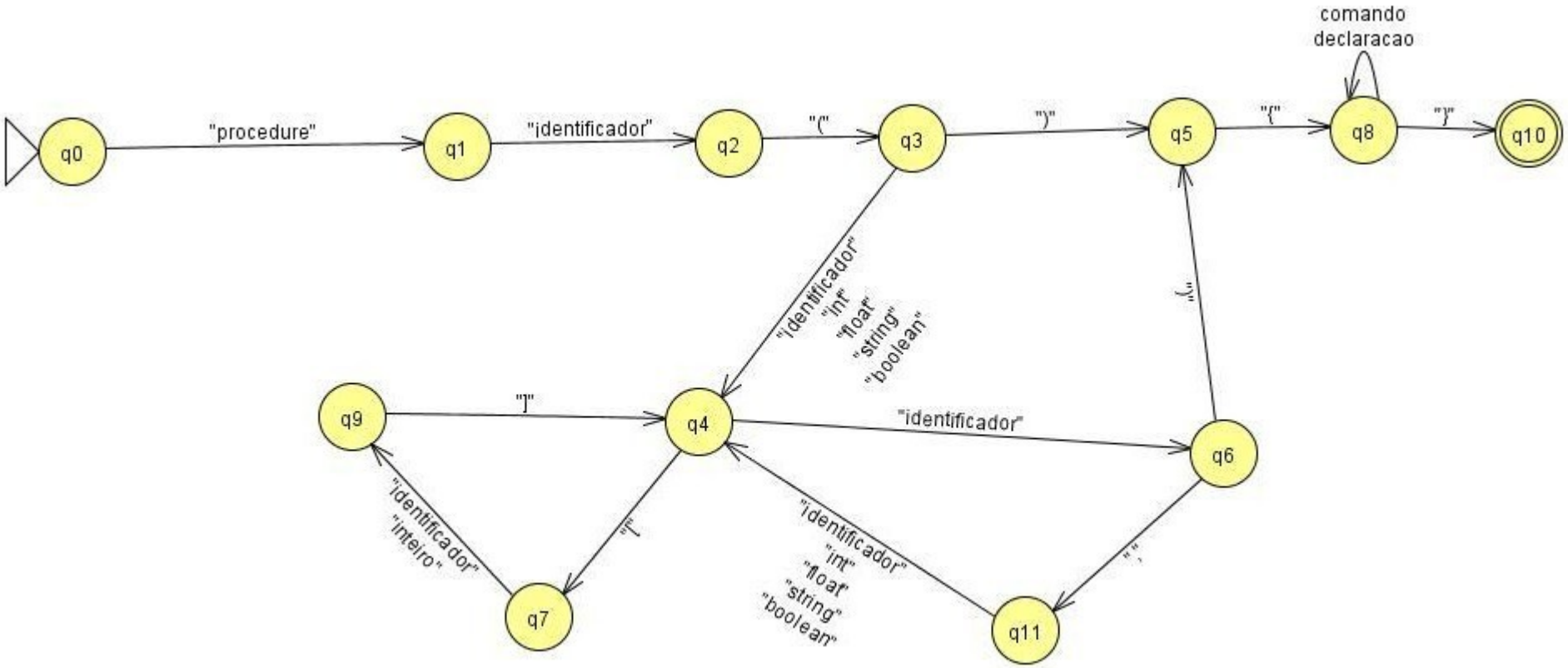
initial: 0
final: 13
0:(0, "function") -> 1 Ação Semântica:
1:(1, "int") -> 2 Ação Semântica: empilha token
2:(1, "identificador") -> 2 Ação Semântica: Declaracao de funcao qu eretorna um homegeneio previamente definido como um struct- verifica no escopo a existencia de um símbolo para este identificador com um descritor do tipo struct. Se não houver, gera um erro de tipo não declarado
3:(1, "float") -> 2 Ação Semântica: empilha tokem - cria um vetor de índices vazio e o empilha
4:(1, "string") -> 2 Ação Semântica:empilha tokem - cria um vetor de índices vazio e o empilha
5:(1, "boolean") -> 2 Ação Semântica:empilha tokem - cria um vetor de índices vazio e o empilha
6:(2, "[") -> 3 Ação Semântica: ignora (o colchete só é importante para o reconhecimento sintático, semanticamente nao tem significado)
7:(2, "identificador") -> 4 Ação Semântica:verifica se já existe um símbolo para este identificador na tabela de simbolos do escopo atual, se não houver cria um simbolo com descritor de funcao com retorno do tipo empilhado na pilha semantica, o descritor deve indicar se o tipo retornado é vetor ou nao baseado no tamanho do vetor de indices empilhado. Se já existir um simbolo para este identificador, retorna um erro de identificador de função já declarado. Esta linguagem não permite funções de mesmo nome. Este descritor criado deve permanecer empilhado para a verificação de tipo após a palavra return no final do escopo da função.
8:(3, "inteiro") -> 6 Ação Semântica: Adiciona o inteiro ao um vetor de índices empilhado.
9:(3, "identificador") -> 6 Ação Semântica: Busca o descritor deste simbolo na tabela de simbolos e o adiciona no vetor de indices.
10:(4, "(") -> 5 Ação Semântica:inicializa uma lista de descritores de argumentos
11:(5, "int") -> 7 Ação Semântica:empilha token
12:(5, "identificador") -> 7 Ação Semântica: idem à ação 2
13:(5, "float") -> 7 Ação Semântica: idem à ação 3
14:(5, "string") -> 7 Ação Semântica: idem à ação 4
15:(5, "boolean") -> 7 Ação Semântica: idem à ação 5
16:(5, ")") -> 8 Ação Semântica:Gera código de declaração de função
17:(6, "]") -> 2 Ação Semântica:
18:(7, "[") -> 14 Ação Semântica:
19:(7, "identificador") -> 15 Ação Semântica: idem à ação 7
20:(8, "{") -> 9 Ação Semântica: Inicia um novo escopo
21:(9, declaracao) -> 9 Ação Semântica:
22:(9, comando) -> 9 Ação Semântica:
23:(9, "return") -> 10 Ação Semântica:
24:(10, "identificador") -> 11 Ação Semântica: Verifica se há um símbolo criado para este identificador e checa se o tipo é o mesmo do descritor empilhado que representa o tipo de retorno da função
25:(11, ";") -> 12 Ação Semântica: Esse ";" é extremamente desnecessário. Mas o código fica feio sem ele. Programadores odeiam ; mas não vivem sem ele =)
26:(12, "}") -> 13 Ação Semântica: Encerra escopo

27:(14, "inteiro") -> 17 Ação Semântica:
28:(14, "identificador") -> 17 Ação Semântica:
29:(15, ",") -> 16 Ação Semântica:
30:(15, ")") -> 8 Ação Semântica:
31:(16, "int") -> 7 Ação Semântica:
32:(16, "identificador") -> 7 Ação Semântica:
33:(16, "float") -> 7 Ação Semântica:
34:(16, "string") -> 7 Ação Semântica:
35:(16, "boolean") -> 7 Ação Semântica:
36:(17, "]") -> 7 Ação Semântica:

Procedimento:

Marcação de estados
procedimento = 0 *"procedure" 1 "identificador" 2 "(" 3 [3 (3 "int" 6 { 7 "[" 8 (8 "inteiro" 10 | 8 "identificador" 11) 9 "]" 12 } 7 | 3 "float" 13 { 14 "[" 15 (15 "inteiro" 17 | 15 "identificador" 18) 16 "]" 19 } 14 | 3 "string" 20 { 21 "[" 22 (22 "inteiro" 24 | 22 "identificador" 25) 23 "]" 26 } 21 | 3 "boolean" 27 { 28 "[" 29 (29 "inteiro" 31 | 29 "identificador" 32) 30 "]" 33 } 28 | 3 "identificador" 34 { 35 "[" 36 (36 "inteiro" 38 | 36 "identificador" 39) 37 "]" 40 } 35) 5 "identificador" 41 { 42 "," 43 (43 "int" 45 { 46 "[" 47 (47 "inteiro" 49 | 47 "identificador" 50) 48 "]" 51 } 46 | 43 "float" 52 { 53 "[" 54 (54 "inteiro" 56 | 54 "identificador" 57) 55 "]" 58 } 53 | 43 "string" 59 { 60 "[" 61 (61 "inteiro" 63 | 61 "identificador" 64) 62 "]" 65 } 60 | 43 "boolean" 66 { 67 "[" 68 (68 "inteiro" 70 | 68 "identificador" 71) 69 "]" 72 } 67 | 43 "identificador" 73 { 74 "[" 75 (75 "inteiro" 77 | 75 "identificador" 78) 76 "]" 79 } 74) 44 "identificador" 80 } 42] 4 ")" 81 "{" 82 { 83 (83* ***declaracao*** *85 |* *83* ***comando*** *86) 84 } 83 "}"* 87 .

initial: 0
final: 10
0:(0, "procedure") -> 1 Ação Semântica:
1:(1, "identificador") -> 2 Ação Semântica:
2:(2, "(") -> 3 Ação Semântica:
3:(3, "identificador") -> 4 Ação Semântica:
4:(3, "int") -> 4 Ação Semântica:
5:(3, "float") -> 4 Ação Semântica:
6:(3, "string") -> 4 Ação Semântica:
7:(3, "boolean") -> 4 Ação Semântica:
8:(3, ")") -> 5 Ação Semântica:
9:(4, "identificador") -> 6 Ação Semântica:
10:(4, "[") -> 7 Ação Semântica:
11:(5, "{") -> 8 Ação Semântica:
12:(6, ",") -> 11 Ação Semântica:
13:(6, ")") -> 5 Ação Semântica:
14:(7, "identificador") -> 9 Ação Semântica:
15:(7, "inteiro") -> 9 Ação Semântica:
16:(8, declaracao) -> 8 Ação Semântica:
17:(8, comando) -> 8 Ação Semântica:
18:(8, "}") -> 10 Ação Semântica:
19:(9, "]") -> 4 Ação Semântica:
20:(11, "identificador") -> 4 Ação Semântica:
21:(11, "int") -> 4 Ação Semântica:
22:(11, "float") -> 4 Ação Semântica:
23:(11, "string") -> 4 Ação Semântica:
24:(11, "boolean") -> 4 Ação Semântica:

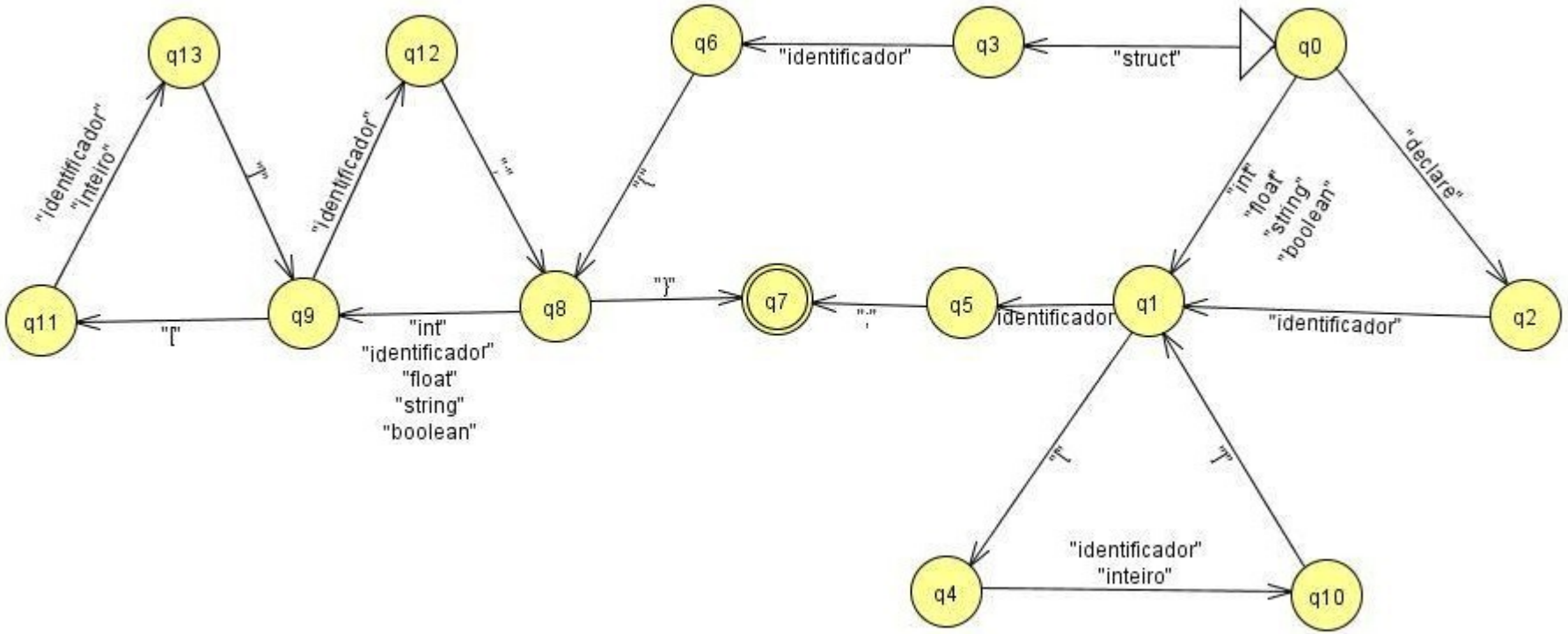

comando
declaracao
q0
"procedure"
q1
"identificador"
q2
"("
q3
")"
q5
"{"
q8
"}"
q10
"identificador"
"int"
"float"
"string"
"boolean"
";"
q9
"]"
q4
"identificador"
q6
"identificador"
"inteiro"
"["
"identificador"
"int"
"float"
"string"
"boolean"
","
q7
q11

Declaracao:

Marcação de estados

declaracao = 0 (0 (0 *"int" 3 { 4 "[" 5 (5 "inteiro" 7 | 5 "identificador" 8) 6 "]" 9 } 4 | 0 "float" 10 { 11 "[" 12 (12 "inteiro" 14 | 12 "identificador" 15) 13 "]" 16 } 11 | 0 "string" 17 { 18 "[" 19 (19 "inteiro" 21 | 19 "identificador" 22) 20 "]" 23 } 18 | 0 "boolean" 24 { 25 "[" 26 (26 "inteiro" 28 | 26 "identificador" 29) 27 "]" 30 } 25 | 0 "declare" 31 "identificador" 32 { 33 "[" 34 (34 "inteiro" 36 | 34 "identificador" 37) 35 "]" 38 } 33) 2 "identificador" 39 ";" 40 | 0 "struct" 41 "identificador" 42 "{" 43 { 44 (44 "int" 46 { 47 "[" 48 (48 "inteiro" 50 | 48 "identificador" 51) 49 "]" 52 } 47 | 44 "float" 53 { 54 "[" 55 (55 "inteiro" 57 | 55 "identificador" 58) 56 "]" 59 } 54 | 44 "string" 60 { 61 "[" 62 (62 "inteiro" 64 | 62 "identificador" 65) 63 "]" 66 } 61 | 44 "boolean" 67 { 68 "[" 69 (69 "inteiro" 71 | 69 "identificador" 72) 70 "]" 73 } 68 | 44 "identificador" 74 { 75 "[" 76 (76 "inteiro" 78 | 76 "identificador" 79) 77 "]" 80 } 75) 45 "identificador" 81 ";" 82 } 44 "}"* 83) 1 .

initial: 0
final: 7
0:(0, "int") -> 1 Ação Semântica:
1:(0, "float") -> 1 Ação Semântica:
2:(0, "string") -> 1 Ação Semântica:
3:(0, "boolean") -> 1 Ação Semântica:
4:(0, "declare") -> 2 Ação Semântica:
5:(0, "struct") -> 3 Ação Semântica:
6:(1, "[") -> 4 Ação Semântica:
7:(1, "identificador") -> 5 Ação Semântica:
8:(2, "identificador") -> 1 Ação Semântica:
9:(3, "identificador") -> 6 Ação Semântica:
10:(4, "inteiro") -> 10 Ação Semântica:
11:(4, "identificador") -> 10 Ação Semântica:
12:(5, ";") -> 7 Ação Semântica:
13:(6, "{") -> 8 Ação Semântica:
14:(8, "int") -> 9 Ação Semântica:
15:(8, "identificador") -> 9 Ação Semântica:
16:(8, "float") -> 9 Ação Semântica:
17:(8, "string") -> 9 Ação Semântica:
18:(8, "boolean") -> 9 Ação Semântica:
19:(8, "}") -> 7 Ação Semântica:
20:(9, "[") -> 11 Ação Semântica:
21:(9, "identificador") -> 12 Ação Semântica:
22:(10, "]") -> 1 Ação Semântica:
23:(11, "inteiro") -> 13 Ação Semântica:
24:(11, "identificador") -> 13 Ação Semântica:
25:(12, ";") -> 8 Ação Semântica:
26:(13, "]") -> 9 Ação Semântica:


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
"struct"
"identificador"
"declare"
"int"
"float"
"string"
"boolean"
"identificador"
"identificador"
"{"
"}"
"int"
"identificador"
"float"
"string"
"boolean"
"identificador"
"inteiro"
"identificador"
"inteiro"

Comando:

comando = 0 (0 *"identificador" 2 { 3 "[" 4 (4 "inteiro" 6 | 4 "identificador" 7) 5 "]" 8 } 3 "=" 9* ***expressao*** *10 ";" 11 | 0 "if" 12 "(" 13* ***expbooleana*** *14 ")" 15 "{" 16 { 17 (17* ***declaracao*** *19 | 17* ***comando*** *20) 18 } 17 "}" 21 [21 "else" 23 "{" 24 { 25 (25* ***declaracao*** *27 | 25* ***comando*** *28) 26 } 25 "}" 29] 22 | 0 "while" 30 "(" 31* ***expbooleana*** *32 ")" 33 "{" 34 { 35 (35* ***declaracao*** *37 | 35* ***comando*** *38) 36 } 35 "}" 39 | 0 "input" 40 "identificador" 41 { 42 "[" 43 (43 "inteiro" 45 | 43 "identificador" 46) 44 "]" 47 } 42 ";" 48 | 0 "output" 49 "identificador" 50 { 51 "[" 52 (52 "inteiro" 54 | 52 "identificador" 55) 53 "]" 56 } 51 ";" 57 | 0 "callproc" 58 "identificador" 59 "(" 60 [60 (60 "numero" 63 | 60 "string" 64 | 60 "identificador" 65 { 66 "[" 67 (67 "inteiro" 69 | 67 "identificador" 70) 68 "]" 71 } 66) 62 { 72 "," 73 (73 "numero" 75 | 73 "string" 76 | 73 "identificador" 77 { 78 "[" 79 (79 "inteiro" 81 | 79 "identificador" 82) 80 "]" 83 } 78) 74 } 72] 61 ")" 84 ";"* 85) 1 .

initial: 0
final: 11, 22
0:(0, "identificador") -> 1 Ação Semântica:
1:(0, "if") -> 2 Ação Semântica:
2:(0, "while") -> 3 Ação Semântica:
3:(0, "input") -> 4 Ação Semântica:
4:(0, "output") -> 4 Ação Semântica:
5:(0, "callproc") -> 5 Ação Semântica:
6:(1, "[") -> 6 Ação Semântica:
7:(1, "=") -> 7 Ação Semântica:
8:(2, "(") -> 17 Ação Semântica:
9:(3, "(") -> 26 Ação Semântica:
10:(4, "identificador") -> 25 Ação Semântica:
11:(5, "identificador") -> 8 Ação Semântica:
12:(6, "identificador") -> 15 Ação Semântica:
13:(6, "inteiro") -> 15 Ação Semântica:
14:(7, expressao) -> 9 Ação Semântica:
15:(8, "(") -> 10 Ação Semântica:
16:(9, ";") -> 11 Ação Semântica:
17:(10, "identificador") -> 12 Ação Semântica:
18:(10, ")") -> 9 Ação Semântica:
19:(10, "numero") -> 13 Ação Semântica:
20:(10, "string") -> 13 Ação Semântica:
21:(12, "[") -> 16 Ação Semântica:
22:(12, ")") -> 9 Ação Semântica:
23:(12, ",") -> 14 Ação Semântica:
24:(13, ")") -> 9 Ação Semântica:
25:(13, ",") -> 14 Ação Semântica:
26:(14, "identificador") -> 12 Ação Semântica:
27:(14, "numero") -> 13 Ação Semântica:
28:(14, "string") -> 13 Ação Semântica:
29:(15, "]") -> 1 Ação Semântica:
30:(16, "identificador") -> 18 Ação Semântica:
31:(16, "inteiro") -> 18 Ação Semântica:
32:(17, expbooleana) -> 19 Ação Semântica:
33:(18, "]") -> 12 Ação Semântica:
34:(19, ")") -> 20 Ação Semântica:
35:(20, "{") -> 21 Ação Semântica:
36:(21, declaracao) -> 21 Ação Semântica:
37:(21, comando) -> 21 Ação Semântica:
38:(21, "}") -> 22 Ação Semântica:
39:(22, "else") -> 23 Ação Semântica:

40:(23, "{") -> 24 Ação Semântica:
41:(24, declaracao) -> 24 Ação Semântica:
42:(24, comando) -> 24 Ação Semântica:
43:(24, "}") -> 11 Ação Semântica:
44:(25, "[") -> 27 Ação Semântica:
45:(25, ";") -> 11 Ação Semântica:
46:(26, expbooleana) -> 28 Ação Semântica:
47:(27, "identificador") -> 29 Ação Semântica:
48:(27, "inteiro") -> 29 Ação Semântica:
49:(28, ")") -> 23 Ação Semântica:
50:(29, "]") -> 25 Ação Semântica:

Expressao:

expressao = 0 (0 **expbooleana** 2 | 0 **exparitmetica** 3 | 0 **expstring** 4 | 0 **exprelacional** 5 | 0 *"callfunc" 6 "identificador" 7 "(" 8 [8 (8 "numero" 11 | 8 "string" 12 | 8 "identificador" 13 { 14 "[" 15 (15 "inteiro" 17 | 15 "identificador" 18) 16 "]" 19 } 14) 10 { 20 "," 21 (21 "numero" 23 | 21 "string" 24 | 21 "identificador" 25 { 26 "[" 27 (27 "inteiro" 29 | 27 "identificador" 30) 28 "]" 31 } 26) 22 } 20] 9 ")" 32 ";"* 33) 1 .

initial: 0
final: 1
0:(0, expbooleana) -> 1 Ação Semântica:
1:(0, exparitmetica) -> 1 Ação Semântica:
2:(0, expstring) -> 1 Ação Semântica:
3:(0, exprelacional) -> 1 Ação Semântica:
4:(0, "callfunc") -> 2 Ação Semântica:
5:(2, "identificador") -> 3 Ação Semântica:
6:(3, "(") -> 4 Ação Semântica:
7:(4, "identificador") -> 5 Ação Semântica:
8:(4, "numero") -> 6 Ação Semântica:
9:(4, "string") -> 6 Ação Semântica:
10:(4, ")") -> 7 Ação Semântica:
12:(5, "[") -> 8 Ação Semântica:
13:(5, ",") -> 9 Ação Semântica:
14:(5, ")") -> 7 Ação Semântica:
15:(6, ",") -> 9 Ação Semântica:
16:(6, ")") -> 7 Ação Semântica:
17:(7, ";") -> 1 Ação Semântica:
18:(8, "identificador") -> 10 Ação Semântica:
19:(8, "inteiro") -> 10 Ação Semântica:
20:(9, "identificador") -> 5 Ação Semântica:
21:(9, "numero") -> 6 Ação Semântica:
22:(9, "string") -> 6 Ação Semântica:
23:(10, "]") -> 5 Ação Semântica:

ExpBooleana:

expbooleana = 0 (0 *"identificador" 2 { 3 "[" 4 (4 "inteiro" 6 | 4 "identificador" 7) 5 "]" 8 } 3 | 0 (0 "true" 10 | 0 "false" 11) 9 | 0 "(" 12* ***expbooleana*** *13 ")" 14 | 0 "!" 15 "(" 16* ***expbooleana*** *17 ")" 18) 1 { 19 (19 "|" 21 | 19 "&" 22 | 19 "!=" 23 | 19 "==" 24) 20 (20 "identificador" 26 { 27 "[" 28 (28 "inteiro" 30 | 28 "identificador" 31) 29 "]" 32 } 27 | 20 (20 "true" 34 | 20 "false" 35) 33 | 20 "(" 36* ***expbooleana*** *37 ")" 38 | 20 "!" 39 "(" 40* ***expbooleana*** *41 ")"* 42) 25 } 19 .

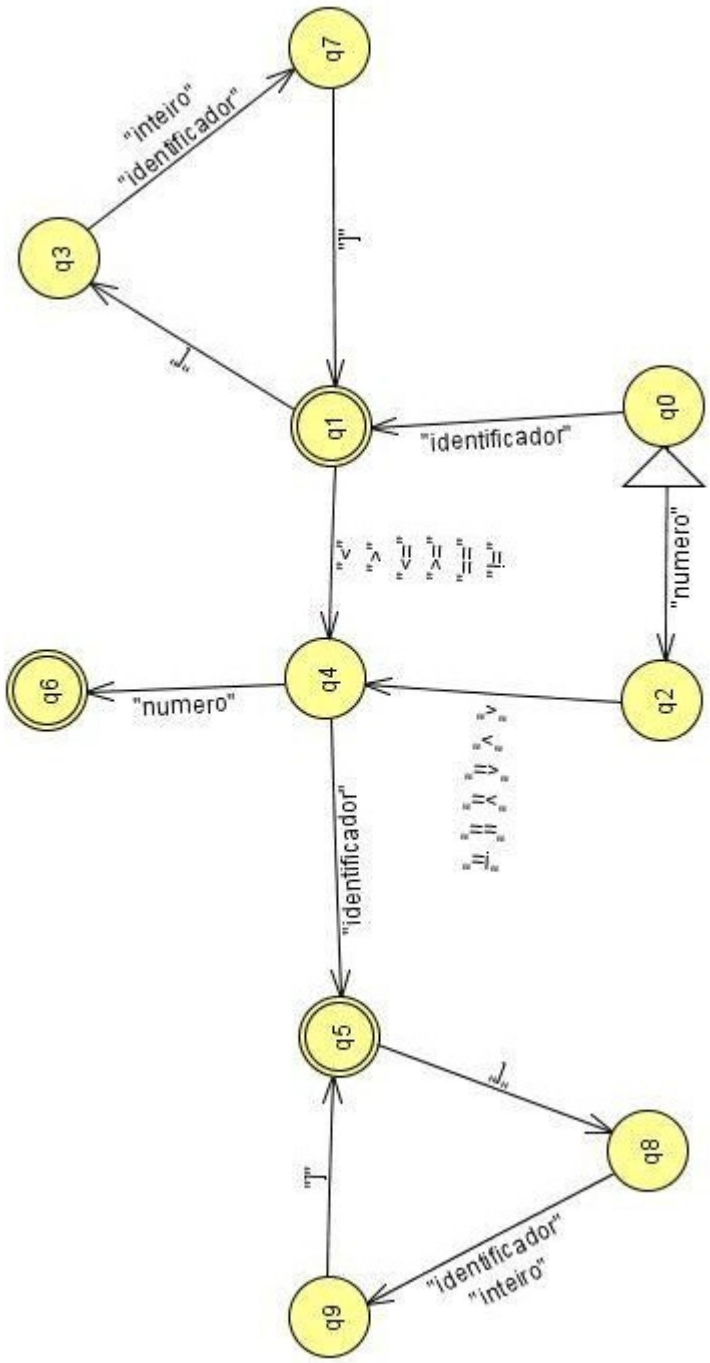
initial: 0
final: 1, 2
0:(0, "identificador") -> 1 Ação Semântica:
1:(0, "true") -> 2 Ação Semântica:

2:(0, "false") -> 2 Ação Semântica:
3:(0, "(") -> 3 Ação Semântica:
4:(0, "!") -> 4 Ação Semântica:
5:(1, "[") -> 5 Ação Semântica:
6:(1, "|") -> 0 Ação Semântica:
7:(1, "&") -> 0 Ação Semântica:
8:(1, "!=") -> 0 Ação Semântica:
9:(1, "==") -> 0 Ação Semântica:
100:(2, "|") -> 0 Ação Semântica:
1:(2, "&") -> 0 Ação Semântica:
2:(2, "!=") -> 0 Ação Semântica:
3:(2, "==") -> 0 Ação Semântica:
4:(3, expbooleana) -> 6 Ação Semântica:
5:(4, "(") -> 3 Ação Semântica:
6:(5, "identificador") -> 7 Ação Semântica:
7:(5, "inteiro") -> 7 Ação Semântica:
8:(6, ")") -> 2 Ação Semântica:
9:(7, "]") -> 1 Ação Semântica:

exprelacional:

exprelacional = 0 (0 *"identificador" 2 { 3 "[" 4 (4 "inteiro" 6 | 4 "identificador" 7) 5 "]" 8 } 3 | 0 (0 "numero" 10 | 0 "identificador" 11 { 12 "[" 13 (13 "inteiro" 15 | 13 "identificador" 16) 14 "]" 17 } 12) 9 (9 "<" 19 | 9 ">" 20 | 9 "<=" 21 | 9 ">=" 22 | 9 "==" 23 | 9 "!=" 24) 18 (18 "numero" 26 | 18 "identificador" 27 { 28 "[" 29 (29 "inteiro" 31 | 29 "identificador" 32) 30 "]"* 33 } 28) 25) 1 .

initial: 0
final: 1, 5, 6
0:(0, "identificador") -> 1 Ação Semântica:
1:(0, "numero") -> 2 Ação Semântica:
2:(1, "[") -> 3 Ação Semântica:
3:(1, "<") -> 4 Ação Semântica:
4:(1, ">") -> 4 Ação Semântica:
5:(1, "<=") -> 4 Ação Semântica:
6:(1, ">=") -> 4 Ação Semântica:
7:(1, "==") -> 4 Ação Semântica:
8:(1, "!=") -> 4 Ação Semântica:
9:(2, "<") -> 4 Ação Semântica:
10:(2, ">") -> 4 Ação Semântica:
11:(2, "<=") -> 4 Ação Semântica:
12:(2, ">=") -> 4 Ação Semântica:
13:(2, "==") -> 4 Ação Semântica:
14:(2, "!=") -> 4 Ação Semântica:
15:(3, "identificador") -> 7 Ação Semântica:
16:(3, "inteiro") -> 7 Ação Semântica:
17:(4, "identificador") -> 5 Ação Semântica:
18:(4, "numero") -> 6 Ação Semântica:
19:(5, "[") -> 8 Ação Semântica:
20:(7, "]") -> 1 Ação Semântica:
21:(8, "identificador") -> 9 Ação Semântica:
22:(8, "inteiro") -> 9 Ação Semântica:
23:(9, "]") -> 5 Ação Semântica:
24: Ação Semântica:



exparitmetica:

exparitmetica = 0 (0 *"-" 2 | 0 "+" 3) 1 (1 "identificador" 5 { 6 "[" 7 (7 "inteiro" 9 | 7 "identificador" 10) 8 "]" 11 } 6 | 1 "numero" 12 | 1 "(" 13* ***exparitmetica*** *14 ")" 15) 4 { 16 (16 "/" 18 | 16 "*" 19) 17 (17 "identificador" 21 { 22 "[" 23 (23 "inteiro" 25 | 23 "identificador" 26) 24 "]" 27 } 22 | 17 "numero" 28 | 17 "(" 29* ***exparitmetica*** *30 ")" 31) 20 } 16 { 32 (32 "+" 34 | 32 "-" 35) 33 (33 "identificador" 37 { 38 "[" 39 (39 "inteiro" 41 | 39 "identificador" 42) 40 "]" 43 } 38 | 33 "numero" 44 | 33 "(" 45* ***exparitmetica*** *46 ")" 47) 36 { 48 (48 "/" 50 | 48 "*" 51) 49 (49 "identificador" 53 { 54 "[" 55 (55 "inteiro" 57 | 55 "identificador" 58) 56 "]" 59 } 54 | 49 "numero" 60 | 49 "(" 61* ***exparitmetica*** *62 ")"* 63) 52 } 48 } 32 .

initial: 0
final: 1, 5, 6
0:(0, "identificador") -> 1 Ação Semântica:
1:(0, "numero") -> 2 Ação Semântica:
2:(1, "[") -> 3 Ação Semântica:
3:(1, "<") -> 4 Ação Semântica:
4:(1, ">") -> 4 Ação Semântica:
5:(1, "<=") -> 4 Ação Semântica:
6:(1, ">=") -> 4 Ação Semântica:
7:(1, "==") -> 4 Ação Semântica:
8:(1, "!=") -> 4 Ação Semântica:
9:(2, "<") -> 4 Ação Semântica:
10:(2, ">") -> 4 Ação Semântica:
11:(2, "<=") -> 4 Ação Semântica:
12:(2, ">=") -> 4 Ação Semântica:
13:(2, "==") -> 4 Ação Semântica:
14:(2, "!=") -> 4 Ação Semântica:
15:(3, "identificador") -> 7 Ação Semântica:
16:(3, "inteiro") -> 7 Ação Semântica:
17:(4, "identificador") -> 5 Ação Semântica:
18:(4, "numero") -> 6 Ação Semântica:
19:(5, "[") -> 8 Ação Semântica:
20:(7, "]") -> 1 Ação Semântica:
21:(8, "identificador") -> 9 Ação Semântica:
22:(8, "inteiro") -> 9 Ação Semântica:
23:(9, "]") -> 5 Ação Semântica:

expstring:

expstring = 0 *"tipo_string" 1 { 2 "+" 3 (3 "tipo_string" 5* | *3 "identificador"* 6) 4 } 2 .

initial: 0
final: 1
0:(0, "tipo_string") -> 1 Ação Semântica:
1:(1, "+") -> 2 Ação Semântica:
2:(2, "tipo_string") -> 1 Ação Semântica:
3:(2, "identificador") -> 1 Ação Semântica:

5) Análise Semântica

O analisador semântico é responsável pelo controle de escopo, controle de dados nas tabelas de símbolos e geração de código para a MVN.
O analisador semântico foi desenvolvido como um *switch case* para cada submáquina com um *case* para cada transição do automato, no qual é inserido o respectivo código de ação semântica, caso esta exista.

A seguir temos um trecho do código das ações semânticas referentes às transições do autômato de pilha estruturado da submáquina programa.

```
switch (tipoMaquina) {
    case PROGRAMA:
        switch (transicaoSemantica) {
        case 0://(0, "main") -> 1
            pilhaSemantico.push(token);

            break;
        case 1://(1, "{") -> 2
            pilhaSemantico.push(token);
            geraCodigo_inicial();
            contadorEscopo++;
            escopo = new Escopo(escopo, contadorEscopo);
            elSemantico.pilhaSemantico = pilhaSemantico;
            elSemantico.escopo = escopo;
            break;
        case 2://(2, declaracao) -> 2
            break;
        case 3://(2, comando) -> 2
            break;
        case 4://(2, "}") -> 3
            escopo = escopo.escopoPai;
            analisaToken(pilhaSemantico.pop_Token(),TipoToken.CHAVE_ABRE);
            analisaToken(pilhaSemantico.pop_Token(),TipoToken.PR_MAIN);
            break;
```

6) Detecção de Erros

Cada estágio da compilação(Léxico, Sintático,Semântico) possui uma lista de erros que é atualizada toda vez que um erro é detectado. Quando um erro é grave , uma exceção é gerada e tratada no nível mais alto do compilador, que gera uma mensagem de erro e exibe a lista de erros gerada. Caso um erro não seja grave, a estratégia adotada é a de tentar continuar a compilação com o próximo token, no caso de erro Sintático ou Semântico, ou , iniciaro reconhecimento de um novo token no caso de erro Léxico.

Através de uma classe enum, foram definidos tipos padronizados de erro, que possibilitam um maior nível de detalhe e inspeção por parte do programador. Os tipos de erro, e respectivas mensagems exibidas ao programador, definidos na linguagem são:

LEXICO -"Erro Lexico"
LEXICO_ARQUIVO_EM_BRACO-"Erro Léxico - Arquivo em branco"
LEXICO_ERRO_DE_ES-"Erro Léxico - Erro de E/S"
LEXICO_ERRO_DE_LEITURA_DE_ARQUIVO-"Erro na leitura de caracter do arquivo"
SINTATICO-"Erro sintático"
SINTATICO_FIMDEPILHA_ANTES_DO_FIM_DOS_TOKENS-"Erro Sintático - Pilha Sintática esvaziada antes do fim dos tokens"
SINTATICO_FIMDETOKENS_ANTES_DO_FIM_DA_PILHA-"Erro Sintático - Fim de tokens sem esvaziar a pilha sintática - Esqueceu de algo no fim do código?Parece meio incompleto."
SINTATICO_ESTADO_SINTATICO_INESPERADO-"Erro Sintático - Estado sintático inesperado atingido"
SEMANTICO_TRANS_SEMANTICA_INESPERADA-"Erro Semântico - Transicão semantica inesperada"
SEMANTICO-"Erro Semantico"
SEMANTICO_REDECLARACAO_DE_VARIAVEL-"Erro Semantico - Variavel já foi declarada no escopo"

SEMANTICO_VARIAVEL_NAO_DECLARADA-"Erro Semantico - Variável Não Declarada"
SEMANTICO_TIPOS_DE_DADOS_INCOMPATIVEIS-"Erro Semantico - Tipo de dados incompativeis"

7) Geração de código

A geração de código é feita em chamadas executadas dentro do semantico, dentro das respectivas açoes semanticas. Para a geracao de codigo da area de dados, uma lista de variaveis é mandtida durante toda a compilaçao e, ao término desta, o código que reserva os espaços de memória para estas variáveis é gerado.

8) Considerações Finais

Devido a limitações de tempo, nem todas as funcionalidades previstas para o compilador foram conlcuídas, sendo necessária um foco maior nas funções básicas. Todavia, as funções implementadas foram testadas e não apresentam graves problemas. Para códigos simples e dentro dos parâmetros da linguagem, o compilador gera o código MVN sem apresentar erros.

O que foi implementado:

- Analisador léxico completo com geração de lista de erros
- Analisador sintático completo com geração de lista de erros
- Analisador semântico incompleto por falta de tempo. As operações aceitas semanticamente dão:
    - Declaração de variáveis do tipo int
    - Declaração de variáveis do tipo Boolean
    - Declaração e vetores unidimensionais e multidimensionais de int e boolean
    - Atribuição de variaaveis e vetores do tipo int (ex. a = b ; a= +3; a=-7;)