Universidade Federal de Alagoas
Instituto de Computação
Ciência da Computação

Linguagem Ultima
Analisador Sintático

BELO, Bruno da Silva ROCHA, Wesley Marques

17 de outubro de 2016

Sumário

1 Códigos-fontes

Só foram adicionados os códigos-fontes criados depois do léxico, os códigos-fontes que não estão aqui sofreram nenhuma ou pouca modificação para poder se adequar

Listing 1: syntax.cpp

```
1  #include "syntax.h"
2  #include "token.h"
3  #include <array>
4  #include <cstdint>
5  #include <fstream>
6  #include <map>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <tuple>
10 #include <type_traits>
11 #include <utility>
12 #include <vector>
13
14 namespace {
15 std::map<uint8_t, std::map<uint8_t, uint8_t>> table;
16 std::array<std::pair<uc::not_term, std::string>, 31> const terminals = {
17     {std::make_pair(uc::not_term::Program, "Program"),
18      std::make_pair(uc::not_term::Program1, "Program1"),
19      std::make_pair(uc::not_term::Program2, "Program2"),
20      std::make_pair(uc::not_term::DecParamList, "DecParamList"),
21      std::make_pair(uc::not_term::DecParamList1, "DecParamList1"),
22      std::make_pair(uc::not_term::Statement, "Statement"),
23      std::make_pair(uc::not_term::Statement1, "Statement1"),
24      std::make_pair(uc::not_term::Statement2, "Statement2"),
25      std::make_pair(uc::not_term::Statement3, "Statement3"),
26      std::make_pair(uc::not_term::Else, "Else"),
27      std::make_pair(uc::not_term::AtrSt, "AtrSt"),
28      std::make_pair(uc::not_term::RValue, "RValue"),
29      std::make_pair(uc::not_term::RValue1, "RValue1"),
30      std::make_pair(uc::not_term::CallParamList, "CallParamList"),
31      std::make_pair(uc::not_term::CallParamList1, "CallParamList1"),
32      std::make_pair(uc::not_term::LogExp, "LogExp"),
33      std::make_pair(uc::not_term::LogExp1, "LogExp1"),
34      std::make_pair(uc::not_term::LogExp2, "LogExp2"),
```

```
35        std::make_pair(uc::not_term::RelExp, "RelExp"),
36        std::make_pair(uc::not_term::RelExp1, "RelExp1"),
37        std::make_pair(uc::not_term::RelExp2, "RelExp2"),
38        std::make_pair(uc::not_term::RelExp3, "RelExp3"),
39        std::make_pair(uc::not_term::AriExp, "AriExp"),
40        std::make_pair(uc::not_term::AriExp1, "AriExp1"),
41        std::make_pair(uc::not_term::AriExp2, "AriExp2"),
42        std::make_pair(uc::not_term::AriExp3, "AriExp3"),
43        std::make_pair(uc::not_term::AriExp4, "AriExp4"),
44        std::make_pair(uc::not_term::ParenAriExp, "ParenAriExp"),
45        std::make_pair(uc::not_term::Type, "Type"),
46        std::make_pair(uc::not_term::Opn, "Opn"),
47        std::make_pair(uc::not_term::Empty, "$")}};
48
49  struct type_stack {
50  public:
51    std::string type;
52  };
53
54  struct not_term_stack : public type_stack {
55  public:
56    uc::not_term nt;
57
58    not_term_stack(std::string const& _type, uc::not_term _nt) {
59      type = std::move(_type);
60      nt   = _nt;
61    }
62  };
63
64  struct term_stack : public type_stack {
65  public:
66    uc::kind_t t;
67
68    term_stack(std::string const& _type, uc::kind_t _t) {
69      type = std::move(_type);
70      t    = std::move(_t);
71    }
72  };
73
74  type_stack* create_type_stack(uc::not_term nt) {
75    return new not_term_stack("nt", nt);
```

```
76 }
77
78 type_stack* create_type_stack(uc::kind_t k) {
79   return new term_stack("t", std::move(k));
80 }
81
82 template <typename T> std::vector<type_stack*> foo(T const& t) {
83   return std::vector<type_stack*>(std::move(t));
84 }
85
86 template <std::size_t I = 0, typename Vec, typename Tup>
87 typename std::enable_if<I == std::tuple_size<Tup>::value,
88                         std::vector<type_stack*>>::type
89 add_to_vector(Vec& v, Tup& t) {
90   return v;
91 }
92
93 template <std::size_t I = 0, typename Vec, typename Tup>
94     typename std::enable_if <
95     I<std::tuple_size<Tup>::value, std::vector<type_stack*>>::type
96     add_to_vector(Vec& v, Tup& t) {
97   v.push_back(create_type_stack(std::get<I>(t)));
98   return add_to_vector<I + 1>(v, t);
99 }
100
101 template <typename... T> std::vector<type_stack*> vector_derivation(T... args) {
102   auto arg_tuple = std::make_tuple(args ...);
103
104   std::vector<type_stack*> v;
105   add_to_vector(v, arg_tuple);
106
107   return v;
108 }
109
110 std::array<std::vector<type_stack*>, 75> const derivation = {
111     {vector_derivation(uc::not_term::Type, uc::not_term::Program1),
112      vector_derivation(uc::kind_t::void_t, uc::not_term::Program2),
113      vector_derivation(
114          uc::kind_t::id_t, uc::kind_t::open_paren, uc::not_term::DecParamList,
115          uc::kind_t::close_paren, uc::kind_t::open_brace,
116          uc::not_term::Statement, uc::kind_t::return_c, uc::not_term::Opn,
```

```
117             uc::kind_t::semicolon, uc::kind_t:: close_brace , uc::not_term::Program),
118         vector_derivation (uc::kind_t::main_c, uc::kind_t::open_paren,
119                            uc::not_term::DecParamList, uc::kind_t::close_paren ,
120                            uc::kind_t::open_brace, uc::not_term::Statement,
121                            uc::kind_t::return_c , uc::not_term::Opn,
122                            uc::kind_t::semicolon, uc::kind_t:: close_brace ),
123         vector_derivation (uc::kind_t:: id_t , uc::kind_t::open_paren,
124                            uc::not_term::DecParamList, uc::kind_t::close_paren ,
125                            uc::kind_t::open_brace, uc::not_term::Statement,
126                            uc::kind_t:: close_brace , uc::not_term::Program),
127         vector_derivation (uc::kind_t::main_c, uc::kind_t::open_paren,
128                            uc::not_term::DecParamList, uc::kind_t::close_paren ,
129                            uc::kind_t::open_brace, uc::not_term::Statement,
130                            uc::kind_t:: close_brace ),
131         vector_derivation (uc::not_term::Type, uc::kind_t:: id_t ,
132                            uc::not_term::DecParamList1),
133         vector_derivation (uc::not_term::Empty),
134         vector_derivation (uc::kind_t::comma, uc::not_term::DecParamList),
135         vector_derivation (uc::not_term::Empty),
136         vector_derivation (uc::kind_t:: if_c , uc::kind_t::open_paren,
137                            uc::not_term::LogExp, uc::kind_t::close_paren ,
138                            uc::kind_t::open_brace, uc::not_term::Statement,
139                            uc::kind_t:: close_brace , uc::not_term::Else ),
140         vector_derivation (uc::kind_t::while_c , uc::kind_t::open_paren,
141                            uc::not_term::LogExp, uc::kind_t::close_paren ,
142                            uc::kind_t::open_brace, uc::not_term::Statement,
143                            uc::kind_t:: close_brace , uc::not_term::Statement),
144         vector_derivation (uc::kind_t:: for_c , uc::kind_t::open_paren,
145                            uc::not_term::AtrSt, uc::kind_t::semicolon,
146                            uc::not_term::Opn, uc::kind_t::semicolon,
147                            uc::not_term::Opn, uc::kind_t::close_paren ,
148                            uc::kind_t::open_brace, uc::not_term::Statement,
149                            uc::kind_t:: close_brace , uc::not_term::Statement),
150         vector_derivation (uc::kind_t:: int_t , uc::kind_t:: id_t ,
151                            uc::not_term::Statement1),
152         vector_derivation (uc::kind_t:: float_t , uc::kind_t:: id_t ,
153                            uc::not_term::Statement1),
154         vector_derivation (uc::kind_t:: string_t , uc::kind_t:: id_t ,
155                            uc::not_term::Statement1),
156         vector_derivation (uc::kind_t:: bool_t , uc::kind_t:: id_t ,
157                            uc::not_term::Statement1),
```

```
158     vector_derivation(uc::kind_t::vector_t, uc::not_term::Statement3),
159     vector_derivation(uc::kind_t::id_t, uc::not_term::Statement2),
160     vector_derivation(uc::not_term::Empty),
161     vector_derivation(uc::kind_t::semicolon, uc::not_term::Statement),
162     vector_derivation(uc::kind_t::atr_o, uc::not_term::RValue,
163                       uc::kind_t::semicolon, uc::not_term::Statement),
164     vector_derivation(uc::kind_t::atr_o, uc::not_term::RValue,
165                       uc::kind_t::semicolon, uc::not_term::Statement),
166     vector_derivation(uc::kind_t::open_paren, uc::not_term::CallParamList,
167                       uc::kind_t::close_paren, uc::kind_t::semicolon,
168                       uc::not_term::Statement),
169     vector_derivation(uc::kind_t::id_t, uc::not_term::Statement1),
170     vector_derivation(uc::not_term::Type, uc::kind_t::id_t, uc::kind_t::colon,
171                       uc::not_term::Opn, uc::kind_t::semicolon,
172                       uc::not_term::Statement),
173     vector_derivation(uc::kind_t::else_c, uc::kind_t::open_brace,
174                       uc::not_term::Statement, uc::kind_t::close_brace,
175                       uc::not_term::Statement),
176     vector_derivation(uc::not_term::Statement),
177     vector_derivation(uc::not_term::Type, uc::kind_t::id_t, uc::kind_t::atr_o,
178                       uc::not_term::RValue),
179     vector_derivation(uc::kind_t::id_t, uc::kind_t::atr_o,
180                       uc::not_term::RValue1),
181     vector_derivation(uc::kind_t::id_t, uc::not_term::RValue1),
182     vector_derivation(uc::kind_t::neg_o, uc::not_term::RelExp,
183                       uc::not_term::LogExp2),
184     vector_derivation(uc::kind_t::inv_o, uc::not_term::ParenAriExp,
185                       uc::not_term::AriExp4, uc::not_term::AriExp3,
186                       uc::not_term::RelExp3, uc::not_term::RelExp2,
187                       uc::not_term::LogExp2),
188     vector_derivation(uc::kind_t::open_paren, uc::not_term::LogExp,
189                       uc::kind_t::close_paren, uc::not_term::AriExp4,
190                       uc::not_term::AriExp3, uc::not_term::RelExp3,
191                       uc::not_term::RelExp2, uc::not_term::LogExp2),
192     vector_derivation(uc::kind_t::int_l, uc::not_term::AriExp4,
193                       uc::not_term::AriExp3, uc::not_term::RelExp3,
194                       uc::not_term::RelExp2, uc::not_term::LogExp2),
195
196     vector_derivation(uc::kind_t::bool_l, uc::not_term::AriExp4,
197                       uc::not_term::AriExp3, uc::not_term::RelExp3,
198                       uc::not_term::RelExp2, uc::not_term::LogExp2),
```

```
199     vector_derivation(uc::kind_t:: string_l , uc::not_term::AriExp4,
200                       uc::not_term::AriExp3, uc::not_term::RelExp3,
201                       uc::not_term::RelExp2, uc::not_term::LogExp2),
202     vector_derivation(uc::kind_t::open_paren, uc::not_term::CallParamList,
203                       uc::kind_t::close_paren ),
204     vector_derivation(uc::not_term::AriExp4, uc::not_term::AriExp3,
205                       uc::not_term::RelExp3, uc::not_term::RelExp2,
206                       uc::not_term::LogExp2),
207     vector_derivation(uc::not_term::LogExp, uc::not_term::CallParamList1),
208     vector_derivation(uc::kind_t::comma, uc::not_term::CallParamList),
209     vector_derivation(uc::not_term::Empty),
210     vector_derivation(uc::not_term::LogExp1, uc::not_term::LogExp2),
211     vector_derivation(uc::kind_t::neg_o, uc::not_term::RelExp),
212     vector_derivation(uc::not_term::RelExp),
213     vector_derivation(uc::kind_t::and_o, uc::not_term::LogExp1,
214                       uc::not_term::LogExp2),
215     vector_derivation(uc::kind_t::or_o , uc::not_term::LogExp1,
216                       uc::not_term::LogExp2),
217     vector_derivation(uc::not_term::Empty),
218     vector_derivation(uc::not_term::RelExp1, uc::not_term::RelExp2),
219     vector_derivation(uc::not_term::AriExp, uc::not_term::RelExp3),
220     vector_derivation(uc::kind_t::re_o , uc::not_term::RelExp1),
221     vector_derivation(uc::not_term::Empty),
222     vector_derivation(uc::kind_t::r_o , uc::not_term::AriExp),
223     vector_derivation(uc::not_term::Empty),
224     vector_derivation(uc::not_term::AriExp1, uc::not_term::AriExp3),
225     vector_derivation(uc::not_term::AriExp2, uc::not_term::AriExp4),
226     vector_derivation(uc::kind_t::inv_o , uc::not_term::ParenAriExp),
227     vector_derivation(uc::not_term::ParenAriExp),
228     vector_derivation(uc::kind_t::add_o, uc::not_term::AriExp1,
229                       uc::not_term::AriExp3),
230     vector_derivation(uc::not_term::Empty),
231     vector_derivation(uc::kind_t::mult_o, uc::not_term::AriExp2,
232                       uc::not_term::AriExp4),
233     vector_derivation(uc::not_term::Empty),
234     vector_derivation(uc::kind_t::open_paren, uc::not_term::LogExp,
235                       uc::kind_t::close_paren ),
236     vector_derivation(uc::not_term::Opn), vector_derivation(uc::kind_t:: int_t ),
237     vector_derivation(uc::kind_t:: float_t ),
238     vector_derivation(uc::kind_t:: string_t ),
239     vector_derivation(uc::kind_t::bool_t ),
```

```
240         vector_derivation(uc::kind_t::vector_t),
241         vector_derivation(uc::kind_t::id_t), vector_derivation(uc::kind_t::int_l),
242         vector_derivation(uc::kind_t::bool_l),
243         vector_derivation(uc::kind_t::string_l),
244         vector_derivation(uc::kind_t::float_l),
245         vector_derivation(uc::kind_t::float_l, uc::not_term::AriExp4,
246                           uc::not_term::AriExp3, uc::not_term::RelExp3,
247                           uc::not_term::RelExp2, uc::not_term::LogExp2)}};
248 } // namespace
249
250 uc::Syntax::Syntax(std::string const& source, std::string const& conf_file)
251       : lexer(std::move(source)) {
252     if (!lexer.is_ready()) {
253       fprintf(stderr, "Error while opening the source code!\n");
254       abort();
255     }
256     configure(std::move(conf_file));
257 }
258
259 uc::actions uc::Syntax::process(uc::Token const& input) {
260     auto action                                  = actions::derive;
261     static std::vector<type_stack*> m_stack = {
262         create_type_stack(uc::not_term::Program)};
263
264     auto term = m_stack.back();
265     m_stack.pop_back();
266
267     if (term->type == "t") {
268       auto token = ((term_stack*)term);
269       if (token->t != input.get_kind()) {
270         printf("Token nao eh o esperado!\n");
271         abort();
272       }
273
274       return uc::actions::read_input;
275     }
276
277     std::string alcino_output =
278         uc::get_not_terminal(static_cast<not_term_stack*>(term)->nt) + " -> ";
279
280     auto nt = ((not_term_stack*)term)->nt;
```

```
281   if (nt == uc::not_term::Empty)
282     return action;
283
284   auto x = std::underlying_type<uc::not_term>::type(nt);
285   auto y = std::underlying_type<uc::kind_t>::type(input.get_kind());
286
287   auto search = table.find(x);
288   if (search == table.end()) {
289     printf("Nao-terminal_nao_encontrado!\n");
290     abort();
291   }
292
293   auto search1 = search->second.find(y);
294   if (search1 == search->second.end()) {
295     printf("Producao_inexistente!\n");
296     abort();
297   }
298
299   for (int i = derivation[search1->second].size() - 1; i >= 0; --i)
300     m_stack.push_back(derivation[search1->second][i]);
301
302   auto id = true;
303   for (auto& i : derivation[search1->second]) {
304     if (i->type == "nt")
305       alcino_output += uc::get_not_terminal(((not_term_stack*)i)->nt);
306     else {
307       alcino_output += "\"" + uc::get_type(((term_stack*)i)->t) + "\"";
308       if (id) {
309         id          = false;
310         auto tok = input.get_kind();
311         if (tok == uc::kind_t::add_o || tok == uc::kind_t::mult_o ||
312             tok == uc::kind_t::id_t  || tok == uc::kind_t::int_l  ||
313             tok == uc::kind_t::bool_l  || tok == uc::kind_t::string_l  ||
314             tok == uc::kind_t::float_l  || tok == uc::kind_t::r_o  ||
315             tok == uc::kind_t::re_o)
316           alcino_output += "(" + input.get_lexval() + ")";
317       }
318     }
319     alcino_output += "␣";
320   }
321
```

```
322     printf("%s\n", alcino_output.c_str());
323
324     return action;
325   }
326
327   void uc::Syntax::run() {
328     auto t = lexer.nextToken();
329     std::vector<uc::Token> tokens;
330     while (!t.has_ended()) {
331       auto action = process(t);
332       if (action == uc::actions::read_input)
333         t = lexer.nextToken();
334     }
335   }
336
337   void uc::Syntax::configure(std::string const& conf_file) const {
338     std::fstream conf(std::move(conf_file));
339
340     std::array<int, 3> terms;
341
342     while (conf.good()) {
343       conf >> terms[0] >> terms[1] >> terms[2];
344       table[terms[0]][terms[1]] = terms[2];
345     }
346   }
347
348   std::string uc::get_not_terminal(uc::not_term nt) {
349     return terminals[std::underlying_type<uc::not_term>::type(nt)].second;
350   }
```

Listing 2: syntax.h

```
1   #ifndef ULTIMA_SYNTAX_H
2   #define ULTIMA_SYNTAX_H
3
4   #include "lexer.h"
5   #include <string>
6
7   namespace uc {
8   enum class not_term : size_t {
9     Program,
```

```
10    Program1,
11    Program2,
12    DecParamList,
13    DecParamList1,
14    Statement,
15    Statement1,
16    Statement2,
17    Statement3,
18    Else,
19    AtrSt,
20    RValue,
21    RValue1,
22    CallParamList,
23    CallParamList1,
24    LogExp,
25    LogExp1,
26    LogExp2,
27    RelExp,
28    RelExp1,
29    RelExp2,
30    RelExp3,
31    AriExp,
32    AriExp1,
33    AriExp2,
34    AriExp3,
35    AriExp4,
36    ParenAriExp,
37    Type,
38    Opn,
39    Empty
40  };
41
42  enum class actions { read_input, derive };
43
44  class Token;
45
46  class Syntax {
47  public:
48    Syntax(std::string const& source, std::string const& table);
49    actions process(Token const& input);
50    void run();
```

```
51
52 private:
53   Lexer lexer;
54   void configure(std::string const& conf_file) const;
55 };
56
57 std::string get_not_terminal(not_term nt);
58 } // namespace uc
59
60 #endif /* ULTIMA_SYNTAX_H */
```

Listing 3: main.cpp

```
1  /**
2     @mainpage Ultima Compiler
3     @brief A compiler to pass in compiler course.
4
5     The front-end of the compiler, verifying if the parameters is right and the
6     format of the file is usable.
7     Always use clang-format.
8     @author Bruno da Silva Belo
9     @see https://github.com/isocpp/CppCoreGuidelines
10    @see http://llvm.org/docs/tutorial/LangImpl1.html
11    @see http://clang.llvm.org/docs/ClangFormat.html
12 */
13 #include "gsl/gsl_assert.h"
14 #include "syntax.h"
15 #include <string>
16
17 namespace {
18 const std::string conf_file = "../src/syntax.conf";
19 }
20
21 int main(int argc, char** argv) {
22   Expects(argc >= 2);
23
24   uc::Syntax syn(argv[1], conf_file );
25   syn.run();
26
27   return 0;
28 }
```

Este arquivo é a tabela do analisador. Cada linha representa uma derivação da tabela, onde o primeiro número é o não-terminal, o segundo é o terminal e o terceiro é a derivação em si, pórem o número da derivação do arquivo é -1 o tabela, por conta que o array começa do 0 e não do 1, assim, por exemplo, a D3 é o 2 no arquivo.

Listing 4: syntax.conf

```
1  0 7 1
2  0 0 0
3  0 2 0
4  0 4 0
5  0 8 0
6  0 6 0
7  1 10 2
8  1 34 3
9  1 2 5
10 2 10 4
11 2 34 5
12 3 25 7
13 3 0 6
14 3 2 6
15 3 4 6
16 3 8 6
17 3 6 6
18 4 25 9
19 4 32 8
20 5 10 18
21 5 33 19
22 5 27 19
23 5 20 10
24 5 23 11
25 5 22 12
26 5 0 13
27 5 2 14
28 5 4 15
29 5 8 16
30 5 6 17
31 6 28 20
32 6 16 21
33 7 24 23
34 7 16 22
```

```
35  8 10 24
36  8 0 25
37  8 2 25
38  8 4 25
39  8 6 25
40  8 8 25
41  9 10 27
42  9 33 27
43  9 27 27
44  9 20 27
45  9 23 27
46  9 22 27
47  9 0 27
48  9 2 27
49  9 4 27
50  9 6 27
51  9 8 27
52  9 21 26
53  10 10 29
54  10 0 28
55  10 2 28
56  10 4 28
57  10 6 28
58  10 8 28
59  11 10 30
60  11 24 33
61  11 19 31
62  11 13 32
63  11 1 34
64  11 3 74
65  11 9 35
66  11 5 36
67  12 24 37
68  12 28 38
69  12 17 38
70  12 18 38
71  12 15 38
72  12 14 38
73  12 11 38
74  12 12 38
75  13 10 39
```

```
117  19 13 49
118  19 1 49
119  19 3 49
120  19 9 49
121  19 5 49
122  20 25 51
123  20 28 51
124  20 32 51
125  20 17 51
126  20 18 51
127  20 15 50
128  21 25 53
129  21 28 53
130  21 32 53
131  21 17 53
132  21 18 53
133  21 15 53
134  21 14 52
135  22 10 54
136  22 24 54
137  22 19 54
138  22 13 54
139  22 1 54
140  22 3 54
141  22 9 54
142  22 5 54
143  23 10 55
144  23 24 55
145  23 19 55
146  23 13 55
147  23 1 55
148  23 3 55
149  23 9 55
150  23 5 55
151  24 10 57
152  24 24 56
153  24 19 57
154  24 13 57
155  24 1 57
156  24 3 57
157  24 9 57
```

158 24 5 57
159 25 25 59
160 25 28 59
161 25 32 59
162 25 17 59
163 25 18 59
164 25 15 59
165 25 14 59
166 25 11 58
167 26 25 61
168 26 28 61
169 26 32 61
170 26 17 61
171 26 18 61
172 26 15 61
173 26 14 61
174 26 11 61
175 26 12 60
176 27 10 63
177 27 19 64
178 27 1 63
179 27 3 63
180 27 9 63
181 27 5 63
182 28 0 64
183 28 2 65
184 28 4 66
185 28 6 68
186 28 8 67
187 29 10 69
188 29 1 70
189 29 3 73
190 29 9 71
191 29 5 72

2 Resultados

$ representa a produção vazia

2.1 Olá Mundo

Listing 5: Resultado de Hello World

```
1  Program −> Type Program1
2  Type −> "int_t"
3  Program1 −> "main_c" "open_paren" DecParamList "close_paren" "open_brace" Statement "return
4  DecParamList −> $
5  Statement −> "id_t"(outputString) Statement2
6  Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
7  CallParamList −> LogExp CallParamList1
8  LogExp −> LogExp1 LogExp2
9  LogExp1 −> RelExp
10 RelExp −> RelExp1 RelExp2
11 RelExp1 −> AriExp RelExp3
12 AriExp −> AriExp1 AriExp3
13 AriExp1 −> AriExp2 AriExp4
14 AriExp2 −> ParenAriExp
15 ParenAriExp −> Opn
16 Opn −> "string_l"("Hello_World_!")
17 AriExp4 −> $
18 AriExp3 −> $
19 RelExp3 −> $
20 RelExp2 −> $
21 LogExp2 −> $
22 CallParamList1 −> $
23 Statement −> $
24 Opn −> "int_l"(0)
```

2.2 Fibonacci

Listing 6: Resultado de Fibonacci

```
1  Program −> Type Program1
2  Type −> "int_t"
3  Program1 −> "id_t"(fibonacci) "open_paren" DecParamList "close_paren" "open_brace" Statement
4  DecParamList −> Type "id_t" DecParamList1
5  Type −> "int_t"
6  DecParamList1 −> $
7  Statement −> "int_t" "id_t" Statement1
8  Statement1 −> "atr_o" RValue "semicolon" Statement
```

```
 9  RValue −> "int_l"(0) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
10  AriExp4 −> $
11  AriExp3 −> $
12  RelExp3 −> $
13  RelExp2 −> $
14  LogExp2 −> $
15  Statement −> "int_t" "id_t" Statement1
16  Statement1 −> "atr_o" RValue "semicolon" Statement
17  RValue −> "int_l"(1) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
18  AriExp4 −> $
19  AriExp3 −> $
20  RelExp3 −> $
21  RelExp2 −> $
22  LogExp2 −> $
23  Statement −> "int_t" "id_t" Statement1
24  Statement1 −> "atr_o" RValue "semicolon" Statement
25  RValue −> "int_l"(0) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
26  AriExp4 −> $
27  AriExp3 −> $
28  RelExp3 −> $
29  RelExp2 −> $
30  LogExp2 −> $
31  Statement −> "if_c" "open_paren" LogExp "close_paren" "open_brace" Statement "close_b
32  LogExp −> LogExp1 LogExp2
33  LogExp1 −> RelExp
34  RelExp −> RelExp1 RelExp2
35  RelExp1 −> AriExp RelExp3
36  AriExp −> AriExp1 AriExp3
37  AriExp1 −> AriExp2 AriExp4
38  AriExp2 −> ParenAriExp
39  ParenAriExp −> Opn
40  Opn −> "id_t"(n)
41  AriExp4 −> $
42  AriExp3 −> $
43  RelExp3 −> "r_o"(<) AriExp
44  AriExp −> AriExp1 AriExp3
45  AriExp1 −> AriExp2 AriExp4
46  AriExp2 −> ParenAriExp
47  ParenAriExp −> Opn
48  Opn −> "int_l"(0)
49  AriExp4 −> $
```

50 AriExp3 −> $
51 RelExp2 −> $
52 LogExp2 −> $
53 Statement −> "id_t"(fi) Statement2
54 Statement2 −> "atr_o" RValue "semicolon" Statement
55 RValue −> "int_l"(0) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
56 AriExp4 −> $
57 AriExp3 −> $
58 RelExp3 −> $
59 RelExp2 −> $
60 LogExp2 −> $
61 Statement −> $
62 Else −> Statement
63 Statement −> "id_t"(outputInt) Statement2
64 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
65 CallParamList −> LogExp CallParamList1
66 LogExp −> LogExp1 LogExp2
67 LogExp1 −> RelExp
68 RelExp −> RelExp1 RelExp2
69 RelExp1 −> AriExp RelExp3
70 AriExp −> AriExp1 AriExp3
71 AriExp1 −> AriExp2 AriExp4
72 AriExp2 −> ParenAriExp
73 ParenAriExp −> Opn
74 Opn −> "int_l"(0)
75 AriExp4 −> $
76 AriExp3 −> $
77 RelExp3 −> $
78 RelExp2 −> $
79 LogExp2 −> $
80 CallParamList1 −> $
81 Statement −> "id_t"(outputString) Statement2
82 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
83 CallParamList −> LogExp CallParamList1
84 LogExp −> LogExp1 LogExp2
85 LogExp1 −> RelExp
86 RelExp −> RelExp1 RelExp2
87 RelExp1 −> AriExp RelExp3
88 AriExp −> AriExp1 AriExp3
89 AriExp1 −> AriExp2 AriExp4
90 AriExp2 −> ParenAriExp

```
 91 ParenAriExp −> Opn
 92 Opn −> "string_l"(",␣")
 93 AriExp4 −> $
 94 AriExp3 −> $
 95 RelExp3 −> $
 96 RelExp2 −> $
 97 LogExp2 −> $
 98 CallParamList1 −> $
 99 Statement −> "id_t"(outputInt) Statement2
100 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
101 CallParamList −> LogExp CallParamList1
102 LogExp −> LogExp1 LogExp2
103 LogExp1 −> RelExp
104 RelExp −> RelExp1 RelExp2
105 RelExp1 −> AriExp RelExp3
106 AriExp −> AriExp1 AriExp3
107 AriExp1 −> AriExp2 AriExp4
108 AriExp2 −> ParenAriExp
109 ParenAriExp −> Opn
110 Opn −> "int_l"(1)
111 AriExp4 −> $
112 AriExp3 −> $
113 RelExp3 −> $
114 RelExp2 −> $
115 LogExp2 −> $
116 CallParamList1 −> $
117 Statement −> "if_c" "open_paren" LogExp "close_paren" "open_brace" Statement "close_b
118 LogExp −> LogExp1 LogExp2
119 LogExp1 −> RelExp
120 RelExp −> RelExp1 RelExp2
121 RelExp1 −> AriExp RelExp3
122 AriExp −> AriExp1 AriExp3
123 AriExp1 −> AriExp2 AriExp4
124 AriExp2 −> ParenAriExp
125 ParenAriExp −> Opn
126 Opn −> "id_t"(n)
127 AriExp4 −> $
128 AriExp3 −> $
129 RelExp3 −> $
130 RelExp2 −> "re_o"(==) RelExp1
131 RelExp1 −> AriExp RelExp3
```

132 AriExp −> AriExp1 AriExp3
133 AriExp1 −> AriExp2 AriExp4
134 AriExp2 −> ParenAriExp
135 ParenAriExp −> Opn
136 Opn −> "int_l"(0)
137 AriExp4 −> $
138 AriExp3 −> $
139 RelExp3 −> $
140 LogExp2 −> "or_o" LogExp1 LogExp2
141 LogExp1 −> RelExp
142 RelExp −> RelExp1 RelExp2
143 RelExp1 −> AriExp RelExp3
144 AriExp −> AriExp1 AriExp3
145 AriExp1 −> AriExp2 AriExp4
146 AriExp2 −> ParenAriExp
147 ParenAriExp −> Opn
148 Opn −> "id_t"(n)
149 AriExp4 −> $
150 AriExp3 −> $
151 RelExp3 −> $
152 RelExp2 −> "re_o"(==) RelExp1
153 RelExp1 −> AriExp RelExp3
154 AriExp −> AriExp1 AriExp3
155 AriExp1 −> AriExp2 AriExp4
156 AriExp2 −> ParenAriExp
157 ParenAriExp −> Opn
158 Opn −> "int_l"(1)
159 AriExp4 −> $
160 AriExp3 −> $
161 RelExp3 −> $
162 LogExp2 −> $
163 Statement −> "id_t"(fi) Statement2
164 Statement2 −> "atr_o" RValue "semicolon" Statement
165 RValue −> "id_t"(n) RValue1
166 RValue1 −> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
167 AriExp4 −> $
168 AriExp3 −> $
169 RelExp3 −> $
170 RelExp2 −> $
171 LogExp2 −> $
172 Statement −> $

```
173 Else -> Statement
174 Statement -> "while_c" "open_paren" LogExp "close_paren" "open_brace" Statement "clo
175 LogExp -> LogExp1 LogExp2
176 LogExp1 -> RelExp
177 RelExp -> RelExp1 RelExp2
178 RelExp1 -> AriExp RelExp3
179 AriExp -> AriExp1 AriExp3
180 AriExp1 -> AriExp2 AriExp4
181 AriExp2 -> ParenAriExp
182 ParenAriExp -> Opn
183 Opn -> "id_t"(fi)
184 AriExp4 -> $
185 AriExp3 -> $
186 RelExp3 -> "r_o"(<) AriExp
187 AriExp -> AriExp1 AriExp3
188 AriExp1 -> AriExp2 AriExp4
189 AriExp2 -> ParenAriExp
190 ParenAriExp -> Opn
191 Opn -> "id_t"(n)
192 AriExp4 -> $
193 AriExp3 -> $
194 RelExp2 -> $
195 LogExp2 -> $
196 Statement -> "id_t"(fi) Statement2
197 Statement2 -> "atr_o" RValue "semicolon" Statement
198 RValue -> "id_t"(f1) RValue1
199 RValue1 -> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
200 AriExp4 -> $
201 AriExp3 -> "add_o"(+) AriExp1 AriExp3
202 AriExp1 -> AriExp2 AriExp4
203 AriExp2 -> ParenAriExp
204 ParenAriExp -> Opn
205 Opn -> "id_t"(f2)
206 AriExp4 -> $
207 AriExp3 -> $
208 RelExp3 -> $
209 RelExp2 -> $
210 LogExp2 -> $
211 Statement -> "id_t"(f1) Statement2
212 Statement2 -> "atr_o" RValue "semicolon" Statement
213 RValue -> "id_t"(f2) RValue1
```

```
214 RValue1 -> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
215 AriExp4 -> $
216 AriExp3 -> $
217 RelExp3 -> $
218 RelExp2 -> $
219 LogExp2 -> $
220 Statement -> "id_t"(f2) Statement2
221 Statement2 -> "atr_o" RValue "semicolon" Statement
222 RValue -> "id_t"(fi) RValue1
223 RValue1 -> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
224 AriExp4 -> $
225 AriExp3 -> $
226 RelExp3 -> $
227 RelExp2 -> $
228 LogExp2 -> $
229 Statement -> "id_t"(outputString) Statement2
230 Statement2 -> "open_paren" CallParamList "close_paren" "semicolon" Statement
231 CallParamList -> LogExp CallParamList1
232 LogExp -> LogExp1 LogExp2
233 LogExp1 -> RelExp
234 RelExp -> RelExp1 RelExp2
235 RelExp1 -> AriExp RelExp3
236 AriExp -> AriExp1 AriExp3
237 AriExp1 -> AriExp2 AriExp4
238 AriExp2 -> ParenAriExp
239 ParenAriExp -> Opn
240 Opn -> "string_l"(",␣")
241 AriExp4 -> $
242 AriExp3 -> $
243 RelExp3 -> $
244 RelExp2 -> $
245 LogExp2 -> $
246 CallParamList1 -> $
247 Statement -> "id_t"(outputInt) Statement2
248 Statement2 -> "open_paren" CallParamList "close_paren" "semicolon" Statement
249 CallParamList -> LogExp CallParamList1
250 LogExp -> LogExp1 LogExp2
251 LogExp1 -> RelExp
252 RelExp -> RelExp1 RelExp2
253 RelExp1 -> AriExp RelExp3
254 AriExp -> AriExp1 AriExp3
```

255 AriExp1 −> AriExp2 AriExp4
256 AriExp2 −> ParenAriExp
257 ParenAriExp −> Opn
258 Opn −> "id_t"(fi)
259 AriExp4 −> $
260 AriExp3 −> $
261 RelExp3 −> $
262 RelExp2 −> $
263 LogExp2 −> $
264 CallParamList1 −> $
265 Statement −> $
266 Statement −> $
267 Opn −> "id_t"(fi)
268 Program −> Type Program1
269 Type −> "int_t"
270 Program1 −> "main_c" "open_paren" DecParamList "close_paren" "open_brace" Statemen
271 DecParamList −> $
272 Statement −> "int_t" "id_t" Statement1
273 Statement1 −> "semicolon" Statement
274 Statement −> "id_t"(inputInt) Statement2
275 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
276 CallParamList −> LogExp CallParamList1
277 LogExp −> LogExp1 LogExp2
278 LogExp1 −> RelExp
279 RelExp −> RelExp1 RelExp2
280 RelExp1 −> AriExp RelExp3
281 AriExp −> AriExp1 AriExp3
282 AriExp1 −> AriExp2 AriExp4
283 AriExp2 −> ParenAriExp
284 ParenAriExp −> Opn
285 Opn −> "id_t"(n)
286 AriExp4 −> $
287 AriExp3 −> $
288 RelExp3 −> $
289 RelExp2 −> $
290 LogExp2 −> $
291 CallParamList1 −> $
292 Statement −> "int_t" "id_t" Statement1
293 Statement1 −> "atr_o" RValue "semicolon" Statement
294 RValue −> "id_t"(fibonacci) RValue1
295 RValue1 −> "open_paren" CallParamList "close_paren"

```
296 CallParamList −> LogExp CallParamList1
297 LogExp −> LogExp1 LogExp2
298 LogExp1 −> RelExp
299 RelExp −> RelExp1 RelExp2
300 RelExp1 −> AriExp RelExp3
301 AriExp −> AriExp1 AriExp3
302 AriExp1 −> AriExp2 AriExp4
303 AriExp2 −> ParenAriExp
304 ParenAriExp −> Opn
305 Opn −> "id_t"(n)
306 AriExp4 −> $
307 AriExp3 −> $
308 RelExp3 −> $
309 RelExp2 −> $
310 LogExp2 −> $
311 CallParamList1 −> $
312 Statement −> $
313 Opn −> "int_l"(0)
```

2.3 Shellsort

Listing 7: Resultado de Shellsort

```
1  Program −> "void_t" Program2
2  Program2 −> "id_t"(shellsort) "open_paren" DecParamList "close_paren" "open_brace" Statement
3  DecParamList −> Type "id_t" DecParamList1
4  Type −> "vector_t"
5  DecParamList1 −> "comma" DecParamList
6  DecParamList −> Type "id_t" DecParamList1
7  Type −> "int_t"
8  DecParamList1 −> $
9  Statement −> "int_t" "id_t" Statement1
10 Statement1 −> "semicolon" Statement
11 Statement −> "int_t" "id_t" Statement1
12 Statement1 −> "atr_o" RValue "semicolon" Statement
13 RValue −> "int_l"(1) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
14 AriExp4 −> $
15 AriExp3 −> $
16 RelExp3 −> $
17 RelExp2 −> $
```

18 LogExp2 −> $
19 Statement −> ”while_c” ”open_paren” LogExp ”close_paren” ”open_brace” Statement ”clo
20 LogExp −> LogExp1 LogExp2
21 LogExp1 −> RelExp
22 RelExp −> RelExp1 RelExp2
23 RelExp1 −> AriExp RelExp3
24 AriExp −> AriExp1 AriExp3
25 AriExp1 −> AriExp2 AriExp4
26 AriExp2 −> ParenAriExp
27 ParenAriExp −> Opn
28 Opn −> ”id_t”(gap)
29 AriExp4 −> $
30 AriExp3 −> $
31 RelExp3 −> ”r_o”(<) AriExp
32 AriExp −> AriExp1 AriExp3
33 AriExp1 −> AriExp2 AriExp4
34 AriExp2 −> ParenAriExp
35 ParenAriExp −> Opn
36 Opn −> ”id_t”(size)
37 AriExp4 −> $
38 AriExp3 −> $
39 RelExp2 −> $
40 LogExp2 −> $
41 Statement −> ”id_t”(gap) Statement2
42 Statement2 −> ”atr_o” RValue ”semicolon” Statement
43 RValue −> ”int_l”(3) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
44 AriExp4 −> ”mult_o”(∗) AriExp2 AriExp4
45 AriExp2 −> ParenAriExp
46 ParenAriExp −> Opn
47 Opn −> ”id_t”(gap)
48 AriExp4 −> $
49 AriExp3 −> ”add_o”(+) AriExp1 AriExp3
50 AriExp1 −> AriExp2 AriExp4
51 AriExp2 −> ParenAriExp
52 ParenAriExp −> Opn
53 Opn −> ”int_l”(1)
54 AriExp4 −> $
55 AriExp3 −> $
56 RelExp3 −> $
57 RelExp2 −> $
58 LogExp2 −> $

```
59 Statement −> $
60 Statement −> "while_c" "open_paren" LogExp "close_paren" "open_brace" Statement "close_brace"
61 LogExp −> LogExp1 LogExp2
62 LogExp1 −> RelExp
63 RelExp −> RelExp1 RelExp2
64 RelExp1 −> AriExp RelExp3
65 AriExp −> AriExp1 AriExp3
66 AriExp1 −> AriExp2 AriExp4
67 AriExp2 −> ParenAriExp
68 ParenAriExp −> Opn
69 Opn −> "id_t"(gap)
70 AriExp4 −> $
71 AriExp3 −> $
72 RelExp3 −> "r_o"(>) AriExp
73 AriExp −> AriExp1 AriExp3
74 AriExp1 −> AriExp2 AriExp4
75 AriExp2 −> ParenAriExp
76 ParenAriExp −> Opn
77 Opn −> "int_l"(1)
78 AriExp4 −> $
79 AriExp3 −> $
80 RelExp2 −> $
81 LogExp2 −> $
82 Statement −> "id_t"(gap) Statement2
83 Statement2 −> "atr_o" RValue "semicolon" Statement
84 RValue −> "id_t"(gap) RValue1
85 RValue1 −> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
86 AriExp4 −> "mult_o"(/) AriExp2 AriExp4
87 AriExp2 −> ParenAriExp
88 ParenAriExp −> Opn
89 Opn −> "int_l"(3)
90 AriExp4 −> $
91 AriExp3 −> $
92 RelExp3 −> $
93 RelExp2 −> $
94 LogExp2 −> $
95 Statement −> "for_c" "open_paren" AtrSt "semicolon" Opn "semicolon" Opn "close_paren" "open_
96 AtrSt −> Type "id_t" "atr_o" RValue
97 Type −> "int_t"
98 RValue −> "id_t"(gap) RValue1
99 RValue1 −> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
```

100 AriExp4 −> $
101 AriExp3 −> $
102 RelExp3 −> $
103 RelExp2 −> $
104 LogExp2 −> $
105 Opn −> "int_l"(10)
106 Opn −> "int_l"(1)
107 Statement −> "id_t"(value) Statement2
108 Statement2 −> "atr_o" RValue "semicolon" Statement
109 RValue −> "id_t"(getValue) RValue1
110 RValue1 −> "open_paren" CallParamList "close_paren"
111 CallParamList −> LogExp CallParamList1
112 LogExp −> LogExp1 LogExp2
113 LogExp1 −> RelExp
114 RelExp −> RelExp1 RelExp2
115 RelExp1 −> AriExp RelExp3
116 AriExp −> AriExp1 AriExp3
117 AriExp1 −> AriExp2 AriExp4
118 AriExp2 −> ParenAriExp
119 ParenAriExp −> Opn
120 Opn −> "id_t"(vet)
121 AriExp4 −> $
122 AriExp3 −> $
123 RelExp3 −> $
124 RelExp2 −> $
125 LogExp2 −> $
126 CallParamList1 −> "comma" CallParamList
127 CallParamList −> LogExp CallParamList1
128 LogExp −> LogExp1 LogExp2
129 LogExp1 −> RelExp
130 RelExp −> RelExp1 RelExp2
131 RelExp1 −> AriExp RelExp3
132 AriExp −> AriExp1 AriExp3
133 AriExp1 −> AriExp2 AriExp4
134 AriExp2 −> ParenAriExp
135 ParenAriExp −> Opn
136 Opn −> "id_t"(i)
137 AriExp4 −> $
138 AriExp3 −> $
139 RelExp3 −> $
140 RelExp2 −> $

141 LogExp2 −> $
142 CallParamList1 −> $
143 Statement −> "int_t" "id_t" Statement1
144 Statement1 −> "atr_o" RValue "semicolon" Statement
145 RValue −> "id_t"(i) RValue1
146 RValue1 −> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
147 AriExp4 −> $
148 AriExp3 −> "add_o"(−) AriExp1 AriExp3
149 AriExp1 −> AriExp2 AriExp4
150 AriExp2 −> ParenAriExp
151 ParenAriExp −> Opn
152 Opn −> "id_t"(gap)
153 AriExp4 −> $
154 AriExp3 −> $
155 RelExp3 −> $
156 RelExp2 −> $
157 LogExp2 −> $
158 Statement −> "int_t" "id_t" Statement1
159 Statement1 −> "atr_o" RValue "semicolon" Statement
160 RValue −> "id_t"(getValue) RValue1
161 RValue1 −> "open_paren" CallParamList "close_paren"
162 CallParamList −> LogExp CallParamList1
163 LogExp −> LogExp1 LogExp2
164 LogExp1 −> RelExp
165 RelExp −> RelExp1 RelExp2
166 RelExp1 −> AriExp RelExp3
167 AriExp −> AriExp1 AriExp3
168 AriExp1 −> AriExp2 AriExp4
169 AriExp2 −> ParenAriExp
170 ParenAriExp −> Opn
171 Opn −> "id_t"(vet)
172 AriExp4 −> $
173 AriExp3 −> $
174 RelExp3 −> $
175 RelExp2 −> $
176 LogExp2 −> $
177 CallParamList1 −> "comma" CallParamList
178 CallParamList −> LogExp CallParamList1
179 LogExp −> LogExp1 LogExp2
180 LogExp1 −> RelExp
181 RelExp −> RelExp1 RelExp2

182 RelExp1 −> AriExp RelExp3
183 AriExp −> AriExp1 AriExp3
184 AriExp1 −> AriExp2 AriExp4
185 AriExp2 −> ParenAriExp
186 ParenAriExp −> Opn
187 Opn −> "id_t"(j)
188 AriExp4 −> $
189 AriExp3 −> $
190 RelExp3 −> $
191 RelExp2 −> $
192 LogExp2 −> $
193 CallParamList1 −> $
194 Statement −> "while_c" "open_paren" LogExp "close_paren" "open_brace" Statement "clo
195 LogExp −> LogExp1 LogExp2
196 LogExp1 −> RelExp
197 RelExp −> RelExp1 RelExp2
198 RelExp1 −> AriExp RelExp3
199 AriExp −> AriExp1 AriExp3
200 AriExp1 −> AriExp2 AriExp4
201 AriExp2 −> ParenAriExp
202 ParenAriExp −> Opn
203 Opn −> "id_t"(j)
204 AriExp4 −> $
205 AriExp3 −> $
206 RelExp3 −> "r_o"(>=) AriExp
207 AriExp −> AriExp1 AriExp3
208 AriExp1 −> AriExp2 AriExp4
209 AriExp2 −> ParenAriExp
210 ParenAriExp −> Opn
211 Opn −> "int_l"(0)
212 AriExp4 −> $
213 AriExp3 −> $
214 RelExp2 −> $
215 LogExp2 −> "and_o" LogExp1 LogExp2
216 LogExp1 −> RelExp
217 RelExp −> RelExp1 RelExp2
218 RelExp1 −> AriExp RelExp3
219 AriExp −> AriExp1 AriExp3
220 AriExp1 −> AriExp2 AriExp4
221 AriExp2 −> ParenAriExp
222 ParenAriExp −> Opn

223 Opn −> "id_t"(value)
224 AriExp4 −> $
225 AriExp3 −> $
226 RelExp3 −> "r_o"(<) AriExp
227 AriExp −> AriExp1 AriExp3
228 AriExp1 −> AriExp2 AriExp4
229 AriExp2 −> ParenAriExp
230 ParenAriExp −> Opn
231 Opn −> "id_t"(x)
232 AriExp4 −> $
233 AriExp3 −> $
234 RelExp2 −> $
235 LogExp2 −> $
236 Statement −> "id_t"(setValue) Statement2
237 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
238 CallParamList −> LogExp CallParamList1
239 LogExp −> LogExp1 LogExp2
240 LogExp1 −> RelExp
241 RelExp −> RelExp1 RelExp2
242 RelExp1 −> AriExp RelExp3
243 AriExp −> AriExp1 AriExp3
244 AriExp1 −> AriExp2 AriExp4
245 AriExp2 −> ParenAriExp
246 ParenAriExp −> Opn
247 Opn −> "id_t"(vet)
248 AriExp4 −> $
249 AriExp3 −> $
250 RelExp3 −> $
251 RelExp2 −> $
252 LogExp2 −> $
253 CallParamList1 −> "comma" CallParamList
254 CallParamList −> LogExp CallParamList1
255 LogExp −> LogExp1 LogExp2
256 LogExp1 −> RelExp
257 RelExp −> RelExp1 RelExp2
258 RelExp1 −> AriExp RelExp3
259 AriExp −> AriExp1 AriExp3
260 AriExp1 −> AriExp2 AriExp4
261 AriExp2 −> ParenAriExp
262 ParenAriExp −> Opn
263 Opn −> "id_t"(j)

264 AriExp4 −> $
265 AriExp3 −> ”add_o”(+) AriExp1 AriExp3
266 AriExp1 −> AriExp2 AriExp4
267 AriExp2 −> ParenAriExp
268 ParenAriExp −> Opn
269 Opn −> ”id_t”(gap)
270 AriExp4 −> $
271 AriExp3 −> $
272 RelExp3 −> $
273 RelExp2 −> $
274 LogExp2 −> $
275 CallParamList1 −> ”comma” CallParamList
276 CallParamList −> LogExp CallParamList1
277 LogExp −> LogExp1 LogExp2
278 LogExp1 −> RelExp
279 RelExp −> RelExp1 RelExp2
280 RelExp1 −> AriExp RelExp3
281 AriExp −> AriExp1 AriExp3
282 AriExp1 −> AriExp2 AriExp4
283 AriExp2 −> ParenAriExp
284 ParenAriExp −> Opn
285 Opn −> ”id_t”(x)
286 AriExp4 −> $
287 AriExp3 −> $
288 RelExp3 −> $
289 RelExp2 −> $
290 LogExp2 −> $
291 CallParamList1 −> $
292 Statement −> ”id_t”(j) Statement2
293 Statement2 −> ”atr_o” RValue ”semicolon” Statement
294 RValue −> ”id_t”(j) RValue1
295 RValue1 −> AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
296 AriExp4 −> $
297 AriExp3 −> ”add_o”(−) AriExp1 AriExp3
298 AriExp1 −> AriExp2 AriExp4
299 AriExp2 −> ParenAriExp
300 ParenAriExp −> Opn
301 Opn −> ”id_t”(gap)
302 AriExp4 −> $
303 AriExp3 −> $
304 RelExp3 −> $

305 RelExp2 −> $
306 LogExp2 −> $
307 Statement −> $
308 Statement −> "id_t"(setValue) Statement2
309 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
310 CallParamList −> LogExp CallParamList1
311 LogExp −> LogExp1 LogExp2
312 LogExp1 −> RelExp
313 RelExp −> RelExp1 RelExp2
314 RelExp1 −> AriExp RelExp3
315 AriExp −> AriExp1 AriExp3
316 AriExp1 −> AriExp2 AriExp4
317 AriExp2 −> ParenAriExp
318 ParenAriExp −> Opn
319 Opn −> "id_t"(vet)
320 AriExp4 −> $
321 AriExp3 −> $
322 RelExp3 −> $
323 RelExp2 −> $
324 LogExp2 −> $
325 CallParamList1 −> "comma" CallParamList
326 CallParamList −> LogExp CallParamList1
327 LogExp −> LogExp1 LogExp2
328 LogExp1 −> RelExp
329 RelExp −> RelExp1 RelExp2
330 RelExp1 −> AriExp RelExp3
331 AriExp −> AriExp1 AriExp3
332 AriExp1 −> AriExp2 AriExp4
333 AriExp2 −> ParenAriExp
334 ParenAriExp −> Opn
335 Opn −> "id_t"(j)
336 AriExp4 −> $
337 AriExp3 −> "add_o"(+) AriExp1 AriExp3
338 AriExp1 −> AriExp2 AriExp4
339 AriExp2 −> ParenAriExp
340 ParenAriExp −> Opn
341 Opn −> "id_t"(gap)
342 AriExp4 −> $
343 AriExp3 −> $
344 RelExp3 −> $
345 RelExp2 −> $

346 LogExp2 −> $
347 CallParamList1 −> ”comma” CallParamList
348 CallParamList −> LogExp CallParamList1
349 LogExp −> LogExp1 LogExp2
350 LogExp1 −> RelExp
351 RelExp −> RelExp1 RelExp2
352 RelExp1 −> AriExp RelExp3
353 AriExp −> AriExp1 AriExp3
354 AriExp1 −> AriExp2 AriExp4
355 AriExp2 −> ParenAriExp
356 ParenAriExp −> Opn
357 Opn −> ”id_t”(value)
358 AriExp4 −> $
359 AriExp3 −> $
360 RelExp3 −> $
361 RelExp2 −> $
362 LogExp2 −> $
363 CallParamList1 −> $
364 Statement −> $
365 Statement −> $
366 Statement −> $
367 Program −> Type Program1
368 Type −> ”int_t”
369 Program1 −> ”main_c” ”open_paren” DecParamList ”close_paren” ”open_brace” Statemen
370 DecParamList −> $
371 Statement −> ”int_t” ”id_t” Statement1
372 Statement1 −> ”semicolon” Statement
373 Statement −> ”id_t”(inputInt) Statement2
374 Statement2 −> ”open_paren” CallParamList ”close_paren” ”semicolon” Statement
375 CallParamList −> LogExp CallParamList1
376 LogExp −> LogExp1 LogExp2
377 LogExp1 −> RelExp
378 RelExp −> RelExp1 RelExp2
379 RelExp1 −> AriExp RelExp3
380 AriExp −> AriExp1 AriExp3
381 AriExp1 −> AriExp2 AriExp4
382 AriExp2 −> ParenAriExp
383 ParenAriExp −> Opn
384 Opn −> ”id_t”(size)
385 AriExp4 −> $
386 AriExp3 −> $

387 RelExp3 −> $
388 RelExp2 −> $
389 LogExp2 −> $
390 CallParamList1 −> $
391 Statement −> ”vector_t” Statement3
392 Statement3 −> Type ”id_t” ”colon” Opn ”semicolon” Statement
393 Type −> ”int_t”
394 Opn −> ”id_t”(size)
395 Statement −> ”for_c” ”open_paren” AtrSt ”semicolon” Opn ”semicolon” Opn ”close_paren” ”open_
396 AtrSt −> Type ”id_t” ”atr_o” RValue
397 Type −> ”int_t”
398 RValue −> ”int_l”(0) AriExp4 AriExp3 RelExp3 RelExp2 LogExp2
399 AriExp4 −> $
400 AriExp3 −> $
401 RelExp3 −> $
402 RelExp2 −> $
403 LogExp2 −> $
404 Opn −> ”id_t”(size)
405 Opn −> ”int_l”(1)
406 Statement −> ”int_t” ”id_t” Statement1
407 Statement1 −> ”semicolon” Statement
408 Statement −> ”id_t”(inputInt) Statement2
409 Statement2 −> ”open_paren” CallParamList ”close_paren” ”semicolon” Statement
410 CallParamList −> LogExp CallParamList1
411 LogExp −> LogExp1 LogExp2
412 LogExp1 −> RelExp
413 RelExp −> RelExp1 RelExp2
414 RelExp1 −> AriExp RelExp3
415 AriExp −> AriExp1 AriExp3
416 AriExp1 −> AriExp2 AriExp4
417 AriExp2 −> ParenAriExp
418 ParenAriExp −> Opn
419 Opn −> ”id_t”(x)
420 AriExp4 −> $
421 AriExp3 −> $
422 RelExp3 −> $
423 RelExp2 −> $
424 LogExp2 −> $
425 CallParamList1 −> $
426 Statement −> ”id_t”(addInt) Statement2
427 Statement2 −> ”open_paren” CallParamList ”close_paren” ”semicolon” Statement

428 CallParamList −> LogExp CallParamList1
429 LogExp −> LogExp1 LogExp2
430 LogExp1 −> RelExp
431 RelExp −> RelExp1 RelExp2
432 RelExp1 −> AriExp RelExp3
433 AriExp −> AriExp1 AriExp3
434 AriExp1 −> AriExp2 AriExp4
435 AriExp2 −> ParenAriExp
436 ParenAriExp −> Opn
437 Opn −> "id_t"(vet)
438 AriExp4 −> $
439 AriExp3 −> $
440 RelExp3 −> $
441 RelExp2 −> $
442 LogExp2 −> $
443 CallParamList1 −> "comma" CallParamList
444 CallParamList −> LogExp CallParamList1
445 LogExp −> LogExp1 LogExp2
446 LogExp1 −> RelExp
447 RelExp −> RelExp1 RelExp2
448 RelExp1 −> AriExp RelExp3
449 AriExp −> AriExp1 AriExp3
450 AriExp1 −> AriExp2 AriExp4
451 AriExp2 −> ParenAriExp
452 ParenAriExp −> Opn
453 Opn −> "id_t"(x)
454 AriExp4 −> $
455 AriExp3 −> $
456 RelExp3 −> $
457 RelExp2 −> $
458 LogExp2 −> $
459 CallParamList1 −> $
460 Statement −> $
461 Statement −> "id_t"(shellsort) Statement2
462 Statement2 −> "open_paren" CallParamList "close_paren" "semicolon" Statement
463 CallParamList −> LogExp CallParamList1
464 LogExp −> LogExp1 LogExp2
465 LogExp1 −> RelExp
466 RelExp −> RelExp1 RelExp2
467 RelExp1 −> AriExp RelExp3
468 AriExp −> AriExp1 AriExp3

```
469 AriExp1 -> AriExp2 AriExp4
470 AriExp2 -> ParenAriExp
471 ParenAriExp -> Opn
472 Opn -> "id_t"(vet)
473 AriExp4 -> $
474 AriExp3 -> $
475 RelExp3 -> $
476 RelExp2 -> $
477 LogExp2 -> $
478 CallParamList1 -> "comma" CallParamList
479 CallParamList -> LogExp CallParamList1
480 LogExp -> LogExp1 LogExp2
481 LogExp1 -> RelExp
482 RelExp -> RelExp1 RelExp2
483 RelExp1 -> AriExp RelExp3
484 AriExp -> AriExp1 AriExp3
485 AriExp1 -> AriExp2 AriExp4
486 AriExp2 -> ParenAriExp
487 ParenAriExp -> Opn
488 Opn -> "id_t"(size)
489 AriExp4 -> $
490 AriExp3 -> $
491 RelExp3 -> $
492 RelExp2 -> $
493 LogExp2 -> $
494 CallParamList1 -> $
495 Statement -> $
496 Opn -> "int_l"(0)
```

Apêndice

Pelo fato da gramática está mal formulada, alguns erros foram encontrado nela, como:

```
1   int foo(vector int a:5) {
2     int x = 4;
3     if (x < foo2(1)) {
4       return 2;
5     }
6
```

```
7       return 0;
8   }
```

Dado o exemplo acima, há 3 coisas que na gramática ficou devendo.

1. Em foo(vector int a:5), a gramática não consegue identificar o “vector int a:5”, ela está identificando só consegue com “foo(vector a)”.

2. O retorno dentro do “if” não é possível na gramática, pois só se foi considerado retorno de fim de função, logo não há uma produção que suporta retornos no meio da função.

3. Em “if(x ¡ foo2(1))”, a gramática não suporta chamada de funções dentro de expressões.