Universidade Federal de Alagoas
Instituto de Computação
Ciência da Computação

Linguagem Ultima
Analisador Léxico

BELO, Bruno da Silva ROCHA, Wesley Marques

21 de agosto de 2016

Sumário

Sumário **i**

1 Códigos-fontes **1**

2 Olá Mundo **13**

3 Fibonacci **13**

4 Shellsort **15**

1 Códigos-fontes

Listing 1: 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 "lexer.h"
15 #include "token.h"
16 #include "util.h"
17 #include <algorithm>
18 #include <stdio.h>
19 #include <string>
20 #include <vector>
21
22 int main(int argc, char** argv) {
23   Expects(argc >= 2);
24
25   uc::Lexer lexer(argv[1]);
26   if (!lexer.is_ready()) {
27     fprintf(stderr, "Error while opening the source code!\n");
28     return -1;
29   }
30
31   auto t = lexer.nextToken();
32   std::vector<uc::Token> tokens;
33   while (!t.has_ended()) {
34     tokens.push_back(t);
35     t = lexer.nextToken();
36   }
37
```

```
38    auto s = uc::show_token_to_alcino(std::move(tokens));
39
40    printf("%s\n", s.c_str());
41
42    return 0;
43  }
```

Listing 2: token.h

```
1  /**
2      @file token.h
3      It have the types of the language as well functions associated with it, and
4      the Token class.
5
6      @brief It contains the types, and Tokens classes.
7      @author Bruno da Silva Belo
8  */
9  #ifndef TOKEN_H
10 #define TOKEN_H
11
12 #include <string>
13 #include <utility>
14
15 namespace uc {
16 enum class kind_t : uint8_t {
17    int_t  = 0x0,
18    int_l ,
19    float_t ,
20    float_l ,
21    string_t ,
22    string_l ,
23    vector_t ,
24    void_t ,
25    id_t ,
26    add_o,
27    mult_o,
28    inv_o ,
29    r_o ,
30    re_o ,
31    atr_o ,
32    and_o,
```

```
33    or_o,
34    neg_o,
35     if_c ,
36     else_c ,
37     for_c ,
38    while_c,
39    open_paren,
40    close_paren,
41    open_brace,
42     close_brace ,
43    semicolon,
44    colon,
45    quotation,
46    dot,
47    comma,
48    return_c,
49
50    fe,
51    error = 0xff
52  };
53
54  /**
55      @class Token
56      @brief The token class.
57  */
58  class Token {
59  public:
60    /**
61        @brief The default constructor.
62        @param _token The type of the token.
63        @param _lexval The name of the type.
64        @param _row The row that token is.
65        @param _col The column that token is.
66    */
67    Token(kind_t _token, std::string const& _lexval, unsigned _row, unsigned _col);
68    /**
69        @brief It can print the class.
70        @return std::string A string with the contents of the token.
71    */
72    std::string to_string() const;
73    /**
```

```
74        @brief It returns true if it is the final state token.
75        @return bool false if it has token to be read, true otherwise.
76     */
77    bool has_ended() const;
78
79    std::pair<unsigned, unsigned> get_position() const;
80
81    kind_t get_kind() const;
82
83  private:
84    std::string lexval;
85    unsigned row;
86    unsigned col;
87    kind_t token;
88  };
89
90  /**
91      @brief Returns a string with the type's name.
92      @param token The token that want it the type.
93      @return A string with the type's name.
94  */
95  std::string get_type(kind_t const token);
96  } // namespace uc
97
98  #endif /* TOKEN_H */
```

Listing 3: token.cpp

```
1   /**
2       @file token.cpp
3       @brief It contains the definitions of the functions on token.h
4
5       @author Bruno da Silva Belo
6    */
7
8   #include "token.h"
9
10  uc::Token::Token(kind_t _token, std::string const& _lexval, unsigned _row,
11                   unsigned _col)
12      : lexval(std::move(_lexval)), row(_row), col(_col), token(_token) {}
13
```

```
14 std::string uc::Token::to_string() const {
15   std::string out = "[";
16   out += std::to_string(row) + ", " + std::to_string(col) + ", " +
17          get_type(token) + ", " + lexval + "]";
18
19   return out;
20 }
21
22 bool uc::Token::has_ended() const { return token == kind_t::fe; }
23
24 std::pair<unsigned, unsigned> uc::Token::get_position() const {
25   return std::pair<unsigned, unsigned>(row, col);
26 }
27
28 uc::kind_t uc::Token::get_kind() const { return token; }
29
30 std::string uc::get_type(kind_t const token) {
31   switch (token) {
32   case kind_t::id_t:
33     return std::string("id_t");
34   case kind_t::int_t:
35     return std::string("int_t");
36   case kind_t::int_l:
37     return std::string("int_l");
38   case kind_t::float_t:
39     return std::string("float_t");
40   case kind_t::float_l:
41     return std::string("float_l");
42   case kind_t::string_t:
43     return std::string("string_t");
44   case kind_t::string_l:
45     return std::string("string_l");
46   case kind_t::vector_t:
47     return std::string("vector_t");
48   case kind_t::void_t:
49     return std::string("void_t");
50   case kind_t::add_o:
51     return std::string("add_o");
52   case kind_t::mult_o:
53     return std::string("mult_o");
54   case kind_t::inv_o:
```

```
55      return std::string("inv_o");
56    case kind_t::r_o:
57      return std::string("r_o");
58    case kind_t::re_o:
59      return std::string("re_o");
60    case kind_t::atr_o:
61      return std::string("atr_o");
62    case kind_t::and_o:
63      return std::string("and_o");
64    case kind_t::or_o:
65      return std::string("or_o");
66    case kind_t::neg_o:
67      return std::string("neg_o");
68    case kind_t:: if_c :
69      return std::string(" if_c ");
70    case kind_t:: else_c :
71      return std::string(" else_c ");
72    case kind_t::while_c:
73      return std::string("while_c");
74    case kind_t:: for_c :
75      return std::string(" for_c ");
76    case kind_t::open_paren:
77      return std::string("open_paren");
78    case kind_t::close_paren:
79      return std::string("close_paren");
80    case kind_t::open_brace:
81      return std::string("open_brace");
82    case kind_t:: close_brace :
83      return std::string("close_brace");
84    case kind_t::semicolon:
85      return std::string("semicolon");
86    case kind_t::colon:
87      return std::string("colon");
88    case kind_t::quotation:
89      return std::string("quotation");
90    case kind_t::dot:
91      return std::string("dot");
92    case kind_t::comma:
93      return std::string("comma");
94    case kind_t::return_c:
95      return std::string("return_c");
```

```
96    case kind_t::fe:
97      return std::string("File's_End");
98    case kind_t::error:
99      return std::string("error");
100   }
101 }
```

Listing 4: lexer.h

```
1  #ifndef LEXER_H
2  #define LEXER_H
3
4  #include "token.h"
5  #include <fstream>
6  #include <string>
7
8  namespace uc {
9  class Lexer {
10 public:
11   explicit Lexer(std::string const& source);
12   bool is_ready() const;
13   Token nextToken();
14
15 private:
16   std::fstream code;
17   unsigned row = 1;
18   unsigned col = 0;
19
20   Token get_token(kind_t token, std::string const& lexval) const;
21 };
22 } // namespace uc
23
24 #endif /* LEXER_H */
```

Listing 5: lexer.cpp

```
1  #include "lexer.h"
2  #include <algorithm>
3  #include <cctype>
4  #include <stdio.h>
5
6  namespace {
```

```
7  constexpr char COMMENT = '#';
8  }
9
10 uc::Lexer::Lexer(std::string const& source) : code(std::move(source)) {}
11
12 bool uc::Lexer::is_ready() const { return code.is_open(); }
13
14 uc::Token uc::Lexer::nextToken() {
15   static auto next_character = [this]() {
16     ++col;
17     return static_cast<char>(code.get());
18   };
19
20   static char last_char = next_character();
21
22   // Ignored caracteres
23   while (isspace(last_char)) {
24     if (last_char == '\n') {
25       ++row;
26       col = 0;
27     }
28     last_char = next_character();
29   }
30
31   // Names
32   if (isalpha(last_char) || last_char == '_') {
33     std::string lexval = std::string(1, last_char);
34     last_char          = next_character();
35
36     while (isalnum(last_char) || last_char == '_') {
37       lexval += last_char;
38       last_char = next_character();
39     }
40
41     if (lexval == "int")
42       return get_token(uc::kind_t::int_t, std::move(lexval));
43     if (lexval == "float")
44       return get_token(uc::kind_t::float_t, std::move(lexval));
45     if (lexval == "string")
46       return get_token(uc::kind_t::string_t, std::move(lexval));
47     if (lexval == "void")
```

```
48          return get_token(uc::kind_t::void_t, std::move(lexval));
49        if (lexval == "while")
50          return get_token(uc::kind_t::while_c, std::move(lexval));
51        if (lexval == "if")
52          return get_token(uc::kind_t::if_c , std::move(lexval));
53        if (lexval == "else")
54          return get_token(uc::kind_t::else_c , std::move(lexval));
55        if (lexval == "for")
56          return get_token(uc::kind_t::for_c , std::move(lexval));
57        if (lexval == "return")
58          return get_token(uc::kind_t::return_c, std::move(lexval));
59
60        return get_token(uc::kind_t::id_t , std::move(lexval));
61      }
62
63      // Numbers
64      if ( isdigit (last_char )) {
65        std::string lexval = std::string(1, last_char );
66        while (isdigit(last_char = next_character()))
67          lexval += last_char;
68
69        if (last_char == '.') {
70          lexval += last_char;
71          while (isdigit(last_char = next_character()))
72            lexval += last_char;
73
74          return get_token(uc::kind_t::float_l , std::move(lexval));
75        }
76
77        return get_token(uc::kind_t::int_l , std::move(lexval));
78      }
79
80      // strings
81      if (last_char == '"') {
82        std::string lexval = std::string(1, last_char );
83        while ((last_char = next_character()) != '"') {
84          if (last_char == EOF) {
85            return get_token(uc::kind_t::quotation, std::move(lexval));
86          }
87
88          if (last_char != '\n') {
```

```
89            lexval += last_char;
90            continue;
91          }
92
93          auto token = get_token(uc::kind_t::error, std::move(lexval));
94          ++row;
95          col        = 0;
96          last_char = next_character();
97          return token;
98        }
99
100       lexval += last_char;
101       last_char = next_character();
102       return get_token(uc::kind_t::string_l, std::move(lexval));
103     }
104
105     // = or ==
106     if (last_char == '=') {
107       std::string lexval = std::string(1, last_char);
108       last_char          = next_character();
109
110       if (last_char == '=') {
111         lexval += last_char;
112         last_char = next_character();
113       }
114
115       return get_token(uc::kind_t::atr_o, std::move(lexval));
116     }
117
118     // < ,<=, > or >=
119     if ((last_char == '<') || (last_char == '>')) {
120       std::string lexval = std::string(1, last_char);
121       last_char          = next_character();
122
123       if (last_char == '=') {
124         lexval += last_char;
125         last_char = next_character();
126       }
127
128       return get_token(uc::kind_t::r_o, std::move(lexval));
129     }
```

```
130
131   // Comentaries
132   if (last_char == COMMENT) {
133     do {
134       last_char = next_character();
135     } while (last_char != EOF && last_char != '\n');
136
137     if (last_char != EOF)
138       return nextToken();
139   }
140
141   // EOF
142   if (last_char == EOF)
143     return get_token(uc::kind_t::fe, "");
144
145   auto one_char                = last_char;
146   last_char                    = next_character();
147   static auto one_char_token = [this](char& character) {
148     if (character == '+' || character == '-')
149       return get_token(uc::kind_t::add_o, std::string(1, character));
150     if (character == '*' || character == '/' || character == '%')
151       return get_token(uc::kind_t::mult_o, std::string(1, character));
152     if (character == '\302')
153       return get_token(uc::kind_t::neg_o, std::string(1, '\302'));
154     if (character == '~')
155       return get_token(uc::kind_t::inv_o, std::string(1, character));
156     if (character == '{')
157       return get_token(uc::kind_t::open_brace, std::string(1, character));
158     if (character == '}')
159       return get_token(uc::kind_t::close_brace, std::string(1, character));
160     if (character == '(')
161       return get_token(uc::kind_t::open_paren, std::string(1, character));
162     if (character == ')')
163       return get_token(uc::kind_t::close_paren, std::string(1, character));
164     if (character == ';')
165       return get_token(uc::kind_t::semicolon, std::string(1, character));
166     if (character == ':')
167       return get_token(uc::kind_t::colon, std::string(1, character));
168     if (character == ',')
169       return get_token(uc::kind_t::comma, std::string(1, character));
170     if (character == '|')
```

```
171         return get_token(uc::kind_t::or_o, std::string(1, character));
172       if (character == '&')
173         return get_token(uc::kind_t::and_o, std::string(1, character));
174
175       return get_token(uc::kind_t::error, std::string(1, character));
176     };
177
178   return one_char_token(one_char);
179 }
180
181 uc::Token uc::Lexer::get_token(uc::kind_t token,
182                                std::string const& lexval) const {
183   unsigned size = lexval.size();
184   return uc::Token(token, std::move(lexval), row, col - size);
185 }
```

Listing 6: util.h

```
1  #ifndef UTIL_H
2  #define UTIL_H
3
4  #include <string>
5  #include <vector>
6
7  namespace uc {
8  class Token;
9  std::string show_token_to_alcino(std::vector<Token> const& container);
10 }
11
12 #endif /* UTIL_H */
```

Listing 7: util.cpp

```
1  #include "util.h"
2  #include "token.h"
3  #include <map>
4  #include <utility>
5
6  std::string uc::show_token_to_alcino(std::vector<Token> const& container) {
7    std::map<unsigned, std::map<unsigned, std::string>> m_map;
8
9    for (auto& token : container) {
```

```
10      auto position                                = token.get_position ();
11      m_map[position.first ][ position .second] = token.to_string ();
12    }
13
14    std :: string  out = "";
15    for (auto& i : m_map) {
16      for (auto& j : i.second) {
17        out += j.second + " ";
18      }
19      out += "\n";
20    }
21
22    return out;
23  }
```

2 Olá Mundo

```
[1, 1, int_t, int] [1, 5, id_t, main] [1, 9, open_paren, (] [1,
    10, close_paren, )] [1, 12, open_brace, {]
  [2, 3, id_t, outputString] [2, 15, open_paren, (] [2, 16,
      string_l, "Hello World !"] [2, 31, close_paren, )] [2, 32,
      semicolon, ;]
  [4, 3, return_c, return] [4, 10, int_l, 0] [4, 11, semicolon,
      ;]
  [5, 1, close_brace, }]
```

3 Fibonacci

```
[1, 1, int_t, int] [1, 5, id_t, fibonacci] [1, 14, open_paren,
    (] [1, 15, int_t, int] [1, 19, id_t, n] [1, 20, close_paren,
    )] [1, 22, open_brace, {]
  [2, 3, int_t, int] [2, 7, id_t, f1] [2, 10, atr_o, =] [2, 12,
      int_l, 0] [2, 13, semicolon, ;]
  [3, 3, int_t, int] [3, 7, id_t, f2] [3, 10, atr_o, =] [3, 12,
      int_l, 1] [3, 13, semicolon, ;]
  [4, 3, int_t, int] [4, 7, id_t, fi] [4, 10, atr_o, =] [4, 12,
      int_l, 0] [4, 13, semicolon, ;]
  [6, 3, if_c, if] [6, 6, open_paren, (] [6, 7, id_t, n] [6, 9,
      r_o, <] [6, 11, int_l, 0] [6, 12, close_paren, )] [6, 14,
      open_brace, {]
```

```
    [7, 5, return_c, return] [7, 12, int_l, 0] [7, 13, semicolon,
        ;]
    [8, 3, close_brace, }]
  [10, 3, id_t, outputInt] [10, 12, open_paren, (] [10, 13, int_l
      , 0] [10, 14, close_paren, )] [10, 15, semicolon, ;]
  [11, 3, id_t, outputString] [11, 15, open_paren, (] [11, 16,
      string_l, ", "] [11, 20, close_paren, )] [11, 21, semicolon
      , ;]
  [12, 3, id_t, outputInt] [12, 12, open_paren, (] [12, 13, int_l
      , 1] [12, 14, close_paren, )] [12, 15, semicolon, ;]
  [14, 3, if_c, if] [14, 6, open_paren, (] [14, 7, id_t, n] [14,
      9, atr_o, ==] [14, 12, int_l, 0] [14, 14, or_o, |] [14, 16,
       id_t, n] [14, 18, atr_o, ==] [14, 21, int_l, 1] [14, 22,
      close_paren, )] [14, 24, open_brace, {]
    [15, 5, return_c, return] [15, 12, int_l, 1] [15, 13,
        semicolon, ;]
    [16, 3, close_brace, }]
  [18, 3, while_c, while] [18, 9, open_paren, (] [18, 10, id_t,
      fi] [18, 13, r_o, <] [18, 15, id_t, n] [18, 16, close_paren
      , )] [18, 18, open_brace, {]
    [19, 5, id_t, fi] [19, 8, atr_o, =] [19, 10, id_t, f1] [19,
        13, add_o, +] [19, 15, id_t, f2] [19, 17, semicolon, ;]
    [20, 5, id_t, f1] [20, 8, atr_o, =] [20, 10, id_t, f2] [20,
        12, semicolon, ;]
    [21, 5, id_t, f2] [21, 8, atr_o, =] [21, 10, id_t, fi] [21,
        12, semicolon, ;]
    [22, 5, id_t, outputString] [22, 17, open_paren, (] [22, 18,
        string_l, ", "] [22, 22, close_paren, )] [22, 23,
        semicolon, ;]
    [23, 5, id_t, outputInt] [23, 14, open_paren, (] [23, 15,
        id_t, fi] [23, 17, close_paren, )] [23, 18, semicolon, ;]
    [24, 3, close_brace, }]
  [26, 3, return_c, return] [26, 10, id_t, fi] [26, 12, semicolon
      , ;]
  [27, 1, close_brace, }]
[29, 1, int_t, int] [29, 5, id_t, main] [29, 9, open_paren, (]
    [29, 10, close_paren, )] [29, 12, open_brace, {]
  [30, 3, int_t, int] [30, 7, id_t, n] [30, 8, semicolon, ;]
  [31, 3, id_t, inputInt] [31, 11, open_paren, (] [31, 12, id_t,
      n] [31, 13, close_paren, )] [31, 14, semicolon, ;]
  [33, 3, int_t, int] [33, 7, id_t, fib] [33, 11, atr_o, =] [33,
      13, id_t, fibonacci] [33, 22, open_paren, (] [33, 23, id_t,
       n] [33, 24, close_paren, )] [33, 25, semicolon, ;]
```

```
[35, 3, return_c, return] [35, 10, int_l, 0] [35, 11, semicolon
    , ;]
[36, 1, close_brace, }]
```

4 Shellsort

```
[1, 1, void_t, void] [1, 6, id_t, shellsort] [1, 15, open_paren,
    (] [1, 16, id_t, vector] [1, 23, int_t, int] [1, 27, id_t,
   vet] [1, 30, comma, ,] [1, 32, int_t, int] [1, 36, id_t, size
   ] [1, 40, close_paren, )] [1, 42, open_brace, {]
 [2, 3, int_t, int] [2, 7, id_t, value] [2, 12, semicolon, ;]
 [3, 3, int_t, int] [3, 7, id_t, gap] [3, 11, atr_o, =] [3, 13,
    int_l, 1] [3, 14, semicolon, ;]
 [4, 3, while_c, while] [4, 9, open_paren, (] [4, 10, id_t, gap]
     [4, 14, r_o, <] [4, 16, id_t, size] [4, 20, close_paren, )
    ] [4, 22, open_brace, {]
  [5, 5, id_t, gap] [5, 9, atr_o, =] [5, 11, int_l, 3] [5, 13,
     mult_o, *] [5, 15, id_t, gap] [5, 19, add_o, +] [5, 21,
     int_l, 1] [5, 22, semicolon, ;]
  [6, 3, close_brace, }]
 [8, 3, while_c, while] [8, 9, open_paren, (] [8, 10, id_t, gap]
     [8, 14, r_o, >] [8, 16, int_l, 1] [8, 17, close_paren, )]
    [8, 19, open_brace, {]
  [9, 5, id_t, gap] [9, 9, atr_o, =] [9, 11, id_t, gap] [9, 15,
      mult_o, /] [9, 17, int_l, 3] [9, 18, semicolon, ;]
  [10, 5, for_c, for] [10, 9, open_paren, (] [10, 10, int_t,
     int] [10, 14, id_t, i] [10, 16, atr_o, =] [10, 18, id_t,
     gap] [10, 21, semicolon, ;] [10, 23, id_t, i] [10, 25,
     r_o, <] [10, 27, id_t, size] [10, 31, semicolon, ;] [10,
     33, id_t, i] [10, 35, atr_o, =] [10, 37, id_t, i] [10,
     39, add_o, +] [10, 41, int_l, 1] [10, 42, close_paren, )]
      [10, 44, open_brace, {]
    [11, 7, id_t, value] [11, 13, atr_o, =] [11, 15, id_t,
       getValue] [11, 23, open_paren, (] [11, 24, id_t, vet]
       [11, 27, comma, ,] [11, 29, id_t, i] [11, 30,
       close_paren, )] [11, 31, semicolon, ;]
    [12, 7, int_t, int] [12, 11, id_t, j] [12, 13, atr_o, =]
       [12, 15, id_t, i] [12, 17, add_o, -] [12, 19, id_t, gap
       ] [12, 22, semicolon, ;]
    [14, 7, while_c, while] [14, 13, open_paren, (] [14, 14,
       id_t, j] [14, 16, r_o, >=] [14, 19, int_l, 0] [14, 21,
       and_o, &] [14, 23, id_t, value] [14, 29, r_o, <] [14,
       31, id_t, getValue] [14, 39, open_paren, (] [14, 40,
```

```
          id_t, vet] [14, 43, comma, ,] [14, 45, id_t, j] [14,
          46, close_paren, )] [14, 47, close_paren, )] [14, 49,
          open_brace, {]
        [15, 9, id_t, setValue] [15, 17, open_paren, (] [15, 18,
           id_t, vet] [15, 21, comma, ,] [15, 23, id_t, j] [15,
           25, add_o, +] [15, 27, id_t, gap] [15, 30, comma, ,]
           [15, 32, id_t, getValue] [15, 40, open_paren, (] [15,
           41, id_t, vet] [15, 44, comma, ,] [15, 46, id_t, j]
           [15, 47, close_paren, )] [15, 48, close_paren, )] [15,
            49, semicolon, ;]
        [16, 9, id_t, j] [16, 11, atr_o, =] [16, 13, id_t, j]
           [16, 15, add_o, -] [16, 17, id_t, gap] [16, 20,
           semicolon, ;]
        [17, 7, close_brace, }]
      [19, 7, id_t, setValue] [19, 15, open_paren, (] [19, 16,
         id_t, vet] [19, 19, comma, ,] [19, 21, id_t, j] [19,
         23, add_o, +] [19, 25, id_t, gap] [19, 28, comma, ,]
         [19, 30, id_t, value] [19, 35, close_paren, )] [19, 36,
          semicolon, ;]
      [20, 5, close_brace, }]
    [21, 3, close_brace, }]
  [22, 1, close_brace, }]
[24, 1, int_t, int] [24, 5, id_t, main] [24, 9, open_paren, (]
    [24, 10, close_paren, )] [24, 12, open_brace, {]
  [25, 3, int_t, int] [25, 7, id_t, size] [25, 11, semicolon, ;]
  [26, 3, id_t, inputInt] [26, 11, open_paren, (] [26, 12, id_t,
     size] [26, 16, close_paren, )] [26, 17, semicolon, ;]
  [28, 3, id_t, vector] [28, 10, int_t, int] [28, 14, id_t, vet]
     [28, 18, colon, :] [28, 20, id_t, size] [28, 24, semicolon,
      ;]
  [29, 3, for_c, for] [29, 7, open_paren, (] [29, 8, int_t, int]
     [29, 12, id_t, i] [29, 14, atr_o, =] [29, 16, int_l, 0]
     [29, 17, semicolon, ;] [29, 19, id_t, size] [29, 23,
     semicolon, ;] [29, 25, int_l, 1] [29, 26, close_paren, )]
     [29, 28, open_brace, {]
    [30, 5, int_t, int] [30, 9, id_t, x] [30, 10, semicolon, ;]
    [31, 5, id_t, inputInt] [31, 13, open_paren, (] [31, 14, id_t
       , x] [31, 15, close_paren, )] [31, 16, semicolon, ;]
    [32, 5, id_t, addInt] [32, 11, open_paren, (] [32, 12, id_t,
       vet] [32, 15, comma, ,] [32, 17, id_t, x] [32, 18,
       close_paren, )] [32, 19, semicolon, ;]
    [33, 3, close_brace, }]
  [35, 3, id_t, shellsort] [35, 12, open_paren, (] [35, 13, id_t,
```

```
    vet] [35, 16, comma, ,] [35, 18, id_t, size] [35, 22,
   close_paren, )] [35, 23, semicolon, ;]
[38, 3, return_c, return] [38, 10, int_l, 0] [38, 11, semicolon
   , ;]
[39, 1, close_brace, }]
```