

编译原理Lab01: 词法分析

22281089 陈可致

- 编译原理Lab01: 词法分析
 - 22281089 陈可致
 - 一. 实验要求
 - 二. 开发环境
 - 三. 运行方式
 - 四. 项目概述
 - 五. 程序设计概述
 - 六. 程序设计
 - 七. 测试
 - 八. 心得体会
 - 附录

一. 实验要求

1. 实验项目

以下为正则文法所描述的C 语言子集单词符号的示例, 请补充单词符号:
++, --, >>, <<, +=, -=, *=, /=, &&(逻辑与), ||(逻辑或), !(逻辑非)等等, 给出补充后描述C 语言子集单词符号的正则文法, 设计并实现其词法分析程序.
以下文法为左线性正则文法:

1	<标识符>→字母 <标识符>字母 <标识符>d
2	<无符号整数>→数字 <无符号整数>数字
3	<单字符分界符>→+ - * ; , () { }
4	<双字符分界符>→<大于>= <小于>= <小于>> <感叹号>= <等于>= <斜竖>*
5	<小于>→< <等于>→= <大于>→> <斜竖> →/
6	<感叹号>→!

该语言的保留字: void、int、float、double、if、else、for、do、while 等等(也可补充).

2. 设计说明

- 可将该语言设计成大小写不敏感, 也可设计成大小写敏感, 用户定义的标识符最长不超过32 个字符
- 字母为a-z A-Z, 数字为0-9
- 可以对上述文法进行扩充和改造
- “/...../” 和 “//” (一行内)为程序的注释部分.

3. 设计要求

- 给出各单词符号的类别编码
- 词法分析程序应能发现输入串中的错误
- 词法分析作为单独一遍编写, 词法分析结果为二元式序列组成的中间文件
- 设计至少4 个测试用例(尽可能完备), 并给出测试结果.

二. 开发环境

- Ubuntu 24.04.1 LTS

- g++ (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0

三. 运行方式

1. 需要 g++ 编译器, 没有可以用以下命令安装

```
1  sudo apt update
2  sudo apt install g++
```

2. 对于每个实验, 都编写了 sh脚本 和 测试数据文件 用于测试项目, 只需要在项目文件夹目录下运行.sh文件即可进行测试

```
1  cd your_file_forder
2  bash go.sh
```

四. 项目概述

项目为一个词法分析程序, 用于识别C语言子集中的各类单词符号程序能够识别标识符、保留字、数字、单字符和双字符分界符、注释等, 并能处理词法错误, 生成二元式序列组成的中间文件该程序的主要功能包括:

- 识别标识符、保留字、数字、单字符和双字符分界符、注释等
- 处理词法错误
- 输出二元式序列

五. 程序设计概述

- token_type enum

```
1  enum token_type {
2      Reserved,
3      Identifier,
4      Number,
5      Single_char,
6      Comma,    // ,
7      Colon,    // :
8      Semi,     // ;
9      Bb_lft,   // {
10     Bb_rt,    // }
11     Sb_lft,   // (
12     Sb_rt,    // )
13     Plus,     // +
14     Minus,    // -
15     Mul,      // *
16     Div,      // /
17     Lt,       // <
18     Gt,       // >
19     Assign,   // =
20     Not,      // !
21     Double_char,
22     Le,       // <=
23     Ge,       // >=
24     Eq,       // ==
25     Ne,       // !=
26     Plus_assign, // +=
27     Minus_assign, // -=
28     Mul_assign,  // *=
29     Div_assign,  // /=
30     Self_inc,    // ++
31     Self_dec,    // --
32     Rsh,         // >>
33     Lsh,         // <<
34     And,         // &&
35     Or,          // ||
36     Comment,     // // or /*
```

```
37     Epsilon,          //  $\epsilon$ 
38     End,              // #
39     Error,
40     Non_terminal,
41     meion_cnt
42 };
```

- token struct

```
1 struct takina {
2     token_type type;
3     string value;
4 };
```

- token solver

```
1 class lycoris {
2 public:
3     lycoris() {}
4     lycoris(const string &s) {}
5     meion build(const string &s) -> void {}
6     meion get_tokens() -> vector<takina> {}
7     meion get_tokens(const string &s) -> vector<takina> {}
8     meion get_token_type(const string &s) -> token_type {}
9     meion show_tokens() -> void {}
10    meion MeIoN_out0() -> void {}
11    meion MeIoN_out1() -> void {}
12 private:
13     string ar;
14     int pla;
15     meion go() -> void {}
16     meion meion_gc() -> char {}
17     meion get_string() -> takina {}
18     meion get_num() -> takina {}
19     meion get_single_char() -> takina {}
20     meion get_assign_operator() -> takina {}
21     meion get_double_operator() -> takina {}
22     meion get_comment() -> takina {}
23     meion get_token() -> takina {}
24 };
```

六. 程序设计

1. 项目结构

项目分为几个模块

- Lib: 头文件
 - 1. MeloN_H.hpp: 用到的标准库头文件. 使用的std容器, 宏定义
 - 2. MeloN_debug.hpp: 调试头文件, 用于格式化输出不定参数的变量信息, 标准运行环境下不会生效
 - 3. 0_token.hpp: 定义了token_type枚举、takina结构体以及相关的输出操作符
 - 4. 0_token_solver: 定义了lycoris类用于分析token, 提供了一个方法用于测试
- testcase 测试数据 | std
 - 1. 4组测试数据 (test0 - test3)
 - 2. 4组对应的标注输出 (std0 - std3)
- 测试程序
 - 1. test_token_solver.cpp: 定义了lycoris类, 并使用lycoris类进行词法分析测试
 - 2. go.sh: 用于测试项目的脚本
- 2. n_token_solver::lycoris类

1. 构造函数:

- lycoris(): 无参构造函数, 初始化一个空的字符串以供解析
- lycoris(const string &s): 带参构造函数, 直接传入一个字符串进行初始化

2. 公有方法:

- build(const string &s)
 - 功能: 设置或重置输入字符串
 - 用途: 为解析器指定新的待解析字符串
- get_tokens()
 - 功能: 从当前输入字符串中解析出所有标记
 - 返回值: 一个包含所有标记的vector
- get_tokens(const string &s)
 - 功能: 解析一个新的字符串并返回所有标记
 - 返回值: 一个标记向量
 - 用途: 用于一次性解析并获取标记
- get_token_type()
 - 功能: 从当前输入字符串中解析出所有标记
 - 返回值: 一个包含所有标记的vector
- show_tokens()
 - 功能: 打印所有标记及其详细信息到控制台
 - 用途: 调试和可视化解析结果
- MeloN_out0()
 - 功能: 以简单的格式输出所有标记的类型和值
 - 用途: 快速查看标记的解析结果
- MeloN_out1()
 - 功能: 输出所有标记, 显示标记类型的文字描述以及其值
 - 用途: 更直观地查看解析结果和标记类型

3. 私有方法

- go()
 - 功能: 跳过字符串中的空白字符
 - 用途: 辅助解析, 忽略无意义的空白符
- meion_gc()
 - 功能: 获取当前位置的字符并将位置前移
 - 返回值: 当前字符
 - 用途: 逐字符解析输入字符串
- get_token()
 - 功能: 获取输入字符串中的下一个标记。
 - 返回值: takina (一个标记的类型和值)。
 - 用途: 核心解析逻辑, 用于识别不同类型的标记。
- get_string(): 解析标识符或保留字
- get_num(): 解析数字字面量
- get_single_char(): 解析单字符标记 (如括号或运算符)
- get_assign_operator(): 解析赋值运算符 (如 <=)
- get_double_operator(): 解析双字符运算符 (如 ++)
- get_comment(): 解析注释 (如 // 或 /* */)

3. n_token_solver::test(): 测试词法分析 函数

1. 输入一段文本, 进行分析

七. 测试

• 测试用例

◦ test0:

```
1  int main() {
2      int a = 10;
3      float b = 20.5;
4      a++;
5      // This is a comment
6      /* This is a
7      multi-line comment */
8      a /= b;
9  }
```

◦ test1:

```
1  void foo() {
2      double x = 3.14;
3      int y = 42;
4      if (x > y) {
5          y += 1;
6      } else {
7          y -= 1;
8      }
9  }
```

◦ test2:

```
1  for (int i = 0; i < 10; i++) {
2      while (i < 5) {
3          i *= 2;
4      }
5      do {
6          i--;
7      } while (i > 0);
8  }
```

◦ test3:

```
1  int add(int x, int y) { return x + y; }
2
3  int main() {
4      int result = add(5, 3);
5      result <<= 1;
6      result >>= 1;
7      result &= 7;
8      result |= 2;
9      result ^= 4;
10     return result;
11 }
```

• std

◦ std0:

```
1  Ciallo~(∠ · ω< )∩★
2  token count: 24
3  type: reserved    value: int
4  type: identifier  value: main
5  type: (           value: (
6  type: )           value: )
```

```
7  type: {          value: {
8  type: reserved   value: int
9  type: identifier  value: a
10 type: =          value: =
11 type: number     value: 10
12 type: ;          value: ;
13 type: reserved   value: float
14 type: identifier  value: b
15 type: =          value: =
16 type: number     value: 20.5
17 type: ;          value: ;
18 type: identifier  value: a
19 type: ++         value: ++
20 type: ;          value: ;
21 type: identifier  value: a
22 type: /=         value: /=
23 type: identifier  value: b
24 type: ;          value: ;
25 type: }          value: }
26 type: end        value: #
```

o std1:

```
1  Ciallo~(∠ · w< )~★
2  token count: 36
3  type: reserved   value: void
4  type: identifier  value: foo
5  type: (          value: (
6  type: )          value: )
7  type: {          value: {
8  type: reserved   value: double
9  type: identifier  value: x
10 type: =          value: =
11 type: number     value: 3.14
12 type: ;          value: ;
13 type: reserved   value: int
14 type: identifier  value: y
15 type: =          value: =
16 type: number     value: 42
17 type: ;          value: ;
18 type: reserved   value: if
19 type: (          value: (
20 type: identifier  value: x
21 type: >          value: >
22 type: identifier  value: y
23 type: )          value: )
24 type: {          value: {
25 type: identifier  value: y
26 type: +=         value: +=
27 type: number     value: 1
28 type: ;          value: ;
29 type: }          value: }
30 type: reserved   value: else
31 type: {          value: {
32 type: identifier  value: y
33 type: -=         value: -=
34 type: number     value: 1
35 type: ;          value: ;
36 type: }          value: }
37 type: }          value: }
38 type: end        value: #
```

o std2:

```
1 Ciallo~( / · w< ) ^ ★
2 token count: 42
3 type: reserved    value: for
4 type: (           value: (
5 type: reserved    value: int
6 type: identifier  value: i
7 type: =           value: =
8 type: number      value: 0
9 type: ;           value: ;
10 type: identifier  value: i
11 type: <           value: <
12 type: number      value: 10
13 type: ;           value: ;
14 type: identifier  value: i
15 type: ++          value: ++
16 type: )           value: )
17 type: {           value: {
18 type: reserved    value: while
19 type: (           value: (
20 type: identifier  value: i
21 type: <           value: <
22 type: number      value: 5
23 type: )           value: )
24 type: {           value: {
25 type: identifier  value: i
26 type: *=          value: *=
27 type: number      value: 2
28 type: ;           value: ;
29 type: }           value: }
30 type: reserved    value: do
31 type: {           value: {
32 type: identifier  value: i
33 type: --          value: --
34 type: ;           value: ;
35 type: }           value: }
36 type: reserved    value: while
37 type: (           value: (
38 type: identifier  value: i
39 type: >           value: >
40 type: number      value: 0
41 type: )           value: )
42 type: ;           value: ;
43 type: }           value: }
44 type: end         value: #
```

o std3:

```
1 Ciallo~( / · w< ) ^ ★
2 token count: 58
3 type: reserved    value: int
4 type: identifier  value: add
5 type: (           value: (
6 type: reserved    value: int
7 type: identifier  value: x
8 type: ,           value: ,
9 type: reserved    value: int
10 type: identifier  value: y
11 type: )           value: )
12 type: {           value: {
13 type: identifier  value: return
14 type: identifier  value: x
```

```
15 type: +          value: +
16 type: identifier  value: y
17 type: ;           value: ;
18 type: }           value: }
19 type: reserved    value: int
20 type: identifier  value: main
21 type: (           value: (
22 type: )           value: )
23 type: {           value: {
24 type: reserved    value: int
25 type: identifier  value: result
26 type: =           value: =
27 type: identifier  value: add
28 type: (           value: (
29 type: number      value: 5
30 type: ,           value: ,
31 type: number      value: 3
32 type: )           value: )
33 type: ;           value: ;
34 type: identifier  value: result
35 type: <<          value: <<
36 type: =           value: =
37 type: number      value: 1
38 type: ;           value: ;
39 type: identifier  value: result
40 type: >>          value: >>
41 type: =           value: =
42 type: number      value: 1
43 type: ;           value: ;
44 type: identifier  value: result
45 type: double char value: &=
46 type: number      value: 7
47 type: ;           value: ;
48 type: identifier  value: result
49 type: double char value: |=
50 type: number      value: 2
51 type: ;           value: ;
52 type: identifier  value: result
53 type: double char value: ^=
54 type: number      value: 4
55 type: ;           value: ;
56 type: identifier  value: return
57 type: identifier  value: result
58 type: ;           value: ;
59 type: }           value: }
60 type: end         value: #
```

• 测试结果

- 结果正确 要看程序输出的话可以把sh脚本中删除输出文件的语句注释

```
1  bash go.sh
2  test: 1
3  accept
4  test: 2
5  accept
6  test: 3
7  accept
8  test: 4
9  accept
```

八. 心得体会

大模拟写写写 zzz

整体实现很像算法竞赛中的 fread 快速读入, 借鉴了相应的思想和实现 见[我的个人快读模板链接\(github\)](#), 对各种token的分割分别做了处理, 在通用的get_token()中调用各种token对应的方法.

附录

1. MeloN_H.hpp

```
1  #pragma once
2  #include <algorithm>
3  #include <array>
4  #include <bitset>
5  #include <cassert>
6  #include <cctype>
7  #include <chrono>
8  #include <cmath>
9  #include <cstring>
10 #include <ctime>
11 #include <fstream>
12 #include <functional>
13 #include <iomanip>
14 #include <iostream>
15 #include <limits>
16 #include <map>
17 #include <queue>
18 #include <random>
19 #include <ranges>
20 #include <set>
21 #include <stack>
22 #include <string>
23 #include <tuple>
24 #include <unordered_map>
25 #include <unordered_set>
26
27 using std::array, std::bitset, std::deque, std::greater, std::less, std::map,
28       std::multiset, std::pair, std::priority_queue, std::set, std::stack,
29       std::string, std::vector, std::tuple, std::function;
30
31 using NAME = void;      using uint = unsigned;   using ll = long long;      using ull = unsigned long long;
32 using ld = long double; using i128 = __int128_t; using u128 = __uint128_t; using f128 = __float128;
33
34 #define meion      auto
35 #define iroha      return
36
```

2. MeloN_debug.hpp

```
1  #pragma once
2
3  template <class T, size_t size = std::tuple_size<T>::value>
4  std::string to_debug(T, std::string s = "")
5  {
6      requires(not std::ranges::range<T>);
7      std::string to_debug(meion x)
8      {
9          requires requires(std::ostream& os) { os << x; }
10         iroha static_cast<std::ostringstream>(std::ostringstream() << x).str();
11     }
12     std::string to_debug(std::ranges::range meion x, std::string s = "")
13     {
14         requires(not std::is_same_v<decltype(x), std::string>)
15         {
16             for (meion xi : x) {
17                 s += ", " + to_debug(xi);
18             }
19         }
20     }
21 }
```

```

16     }
17     iroha "[" + s.substr(s.empty() ? 0 : 2) + "];"
18 }
19 template <class T, size_t size>
20 std::string to_debug(T x, std::string s)
21     requires(not std::ranges::range<T>)
22 {
23     [&<size_t... I>(std::index_sequence<I...>) {
24         ((s += ", " + to_debug(std::get<I>(x))), ...);
25     }(std::make_index_sequence<size>());
26     iroha "(" + s.substr(s.empty() ? 0 : 2) + "));"
27 }
28 #ifdef MeIoN
29 #define debug(...) std::cout << "Ciallo~(∠ · ω< ) ^ ★ " << "(" #__VA_ARGS__ " ) = " << to_debug(std::tuple<__VA_ARGS__>()) << "\n";
30 #else
31 #define debug(...) void(0)
32 #endif

```

3.0 token.hpp

```

1  #pragma once
2  enum token_type {
3      Reserved,
4      Identifier,
5      Number,
6      Single_char,
7      Comma,    // ,
8      Colon,    // :
9      Semi,     // ;
10     Bb_lft,    // {
11     Bb_rt,     // }
12     Sb_lft,    // (
13     Sb_rt,     // )
14     Plus,      // +
15     Minus,     // -
16     Mul,       // *
17     Div,       // /
18     Lt,        // <
19     Gt,        // >
20     Assign,    // =
21     Not,       // !
22     Double_char,
23     Le,        // <=
24     Ge,        // >=
25     Eq,        // ==
26     Ne,        // !=
27     Plus_assign, // +=
28     Minus_assign, // -=
29     Mul_assign,  // *=
30     Div_assign,  // /=
31     Self_inc,    // ++
32     Self_dec,    // --
33     Rsh,         // >>
34     Lsh,         // <<
35     And,         // &&
36     Or,          // ||
37     Comment,     // // or /*
38     Epsilon,     // ε
39     End,         // #
40     Error,
41     Non_terminal,
42     meion_cnt

```

```

43 };
44
45 const std::string type_to_s[meion_cnt] = {
46     "reserved", "identifier", "number", "single_char", ",", ":",
47     ";", "{", "}", "(", ")", "+",
48     "_", "*", "/", "<", ">", "=",
49     "!", "double char", "<=", ">=", "==", "!=",
50     "+=", "-=", "*=", "/=", "++", "--",
51     ">>", "<<", "and", "or", "comment", "ε",
52     "end", "error", "non terminal"};
53
54 const std::unordered_set<std::string> reserved_words = {
55     "void", "int", "float", "double", "if", "else", "for", "do", "while"};
56
57
58 // token
59 struct takina {
60     token_type type;
61     string value;
62 };
63
64 std::ostream &operator<<(std::ostream &os, const takina &x) {
65     os << type_to_s[x.type] << ' ' << x.value;
66     iroha os;
67 }
68
69 void show(const takina &x) {
70     std::cout << "type: " << std::setw(12) << std::left << type_to_s[x.type]
71         << "value: " << std::setw(10) << std::left << x.value
72         << std::endl;
73 }
74

```

4.0 token_solver

```

1  #pragma once
2  #include "MeIoN_H.hpp"
3  #include "MeIoN_debug.hpp"
4  #include "0_token.hpp"
5  // slover
6  namespace n_token_solver {
7      class lycoris {
8      public:
9          lycoris() { build(""); }
10         lycoris(const string &s) { build(s); }
11         meion build(const string &s) -> void {
12             ar = s;
13         }
14
15         meion get_tokens() -> vector<takina> {
16             pla = 0;
17             vector<takina> ret;
18             while (true) {
19                 meion [type, value] = get_token();
20                 if (type == Error) {
21                     std::cerr << "Zako~ coder, " << value << std::endl;
22                 } else if (type == Comment) {
23                     continue;
24                 } else {
25                     ret.emplace_back(type, value);
26                 }
27                 if (type == End) {

```

```
28         break;
29     }
30 }
31 iroha ret;
32 }
33
34 meion get_tokens(const string &s) -> vector<takina> {
35     build(s);
36     iroha get_tokens();
37 }
38
39 meion get_token_type(const string &s) -> token_type {
40     if (s.empty()) {
41         iroha Epsilon;
42     } else if (s[0] == '#') {
43         iroha End;
44     }
45     pla = 0;
46     build(s);
47     iroha get_token().type;
48 }
49
50 meion show_tokens() -> void {
51     vector tokens = get_tokens();
52     std::cout << "Ciallo~(∠ · ω< )∩★ " << std::endl;
53     std::cout << "token count: " << tokens.size() << std::endl;
54     for (const meion &x : tokens) {
55         show(x);
56     }
57 }
58
59 meion MeIoN_out0() -> void {
60     for (const meion &[type, value] : get_tokens()) {
61         std::cout << type << ' ' << value << '\n';
62     }
63 }
64 meion MeIoN_out1() -> void {
65     for (const meion &[type, value] : get_tokens()) {
66         std::cout << type_to_s[type] << ' ' << value << '\n';
67     }
68 }
69
70 private:
71     string ar;
72     int pla;
73
74     meion go() -> void {
75         while (pla < ar.length() and isspace(ar[pla])) {
76             ++pla;
77         }
78     }
79
80     meion meion_gc() -> char {
81         iroha ar[pla++];
82     }
83     meion get_string() -> takina {
84         string value;
85         while (pla < ar.length() and (std::isalnum(ar[pla]) or ar[pla] == '_')) {
86             value += meion_gc();
87         }
88
89         if (reserved_words.contains(value)) {
```

```
90         iroha {Reserved, value};
91     }
92     iroha {Identifier, value};
93 }
94 meion get_num() -> takina {
95     string value;
96     bool RE = false;
97     while (pla < ar.length() and std::isdigit(ar[pla])) {
98         value += meion_gc();
99     }
100    if (pla < ar.length() and ar[pla] == '.') {
101        RE = true;
102        value += meion_gc();
103        while (pla < ar.length() and std::isdigit(ar[pla])) {
104            value += meion_gc();
105        }
106    }
107
108    if (RE and value[0] == '.' or value.back() == '.') {
109        iroha {Error, value};
110    }
111    iroha {Number, value};
112 }
113 meion get_single_char() -> takina {
114     meion op = meion_gc();
115     switch (op) {
116         case ',': iroha {Comma, string{op}};
117         case ':': iroha {Colon, string{op}};
118         case ';': iroha {Semi, string{op}};
119         case '{': iroha {Bb_lft, string{op}};
120         case '}': iroha {Bb_rt, string{op}};
121         case '(': iroha {Sb_lft, string{op}};
122         case ')': iroha {Sb_rt, string{op}};
123         case '+': iroha {Plus, string{op}};
124         case '-': iroha {Minus, string{op}};
125         case '*': iroha {Mul, string{op}};
126         case '/': iroha {Div, string{op}};
127         case '<': iroha {Lt, string{op}};
128         case '>': iroha {Gt, string{op}};
129         case '=': iroha {Assign, string{op}};
130         case '!': iroha {Not, string{op}};
131     }
132     iroha {Single_char, string{op}};
133 }
134 meion get_assign_operator() -> takina {
135     if (pla + 1 < ar.length()) {
136         meion l = meion_gc(), r = meion_gc();
137         if (r != '=') {
138             iroha {Error, "inv assign operator"};
139         }
140         switch (l) {
141             case '<': iroha {Le, string{l, r}};
142             case '>': iroha {Ge, string{l, r}};
143             case '=': iroha {Eq, string{l, r}};
144             case '!': iroha {Ne, string{l, r}};
145             case '+': iroha {Plus_assign, string{l, r}};
146             case '-': iroha {Minus_assign, string{l, r}};
147             case '*': iroha {Mul_assign, string{l, r}};
148             case '/': iroha {Div_assign, string{l, r}};
149         }
150         iroha {Double_char, string{l, r}};
151     }
```

```

152         iroha {Error, "inv assign operator"};
153     }
154     meion get_double_operator() -> takina {
155         if (pla + 1 < ar.length()) {
156             meion l = meion_gc(), r = meion_gc();
157             if (l != r) {
158                 iroha {Error, "inv double operator"};
159             }
160             switch (l) {
161                 case '+': iroha {Self_inc, string{l, r}};
162                 case '-': iroha {Self_dec, string{l, r}};
163                 case '<': iroha {Lsh, string{l, r}};
164                 case '>': iroha {Rsh, string{l, r}};
165                 case '&': iroha {And, string{l, r}};
166                 case '|': iroha {Or, string{l, r}};
167             }
168             iroha {Double_char, string{l, r}};
169         }
170         iroha {Error, string{meion_gc()}};
171     }
172     meion get_comment() -> takina {
173         if (ar[pla + 1] == '/') {
174             while (meion_gc() != '\n') ;
175             iroha {Comment, "//"};
176         } else if (ar[pla + 1] == '*') {
177             char l = 'm', r = 'l';
178             while (pla < ar.length() and not(l == '*' and r == '/')) {
179                 l = r;
180                 r = meion_gc();
181             }
182             iroha {Comment, "/*...*/"};
183         }
184         iroha {Error, "inv comment"};
185     }
186     meion get_token() -> takina {
187         go();
188         if (pla + 1 > ar.length()) iroha {End, "#"};
189         char go = ar[pla];
190         if ((go < 'z' + 1 and go > 'a' - 1) or
191             (go < 'Z' + 1 and go > 'A' - 1)) {
192             iroha get_string();
193         } else if (go < '9' + 1 and go > '0' - 1) {
194             iroha get_num();
195         } else if (go == '/') {
196             if (pla + 1 < ar.length()) {
197                 if (ar[pla + 1] == '=') {
198                     iroha get_assign_operator();
199                 } else if (ar[pla + 1] == '/' or ar[pla + 1] == '*') {
200                     iroha get_comment();
201                 }
202             }
203             iroha get_single_char();
204         } else if (go == '*' or go == '%' or go == '!' or go == '^' or
205             go == '^' or go == '~') {
206             if (pla + 1 < ar.length() and ar[pla + 1] == '=') {
207                 iroha get_assign_operator();
208             }
209             iroha get_single_char();
210         } else if (go == '+' or go == '-' or go == '>' or go == '<' or go == '=' or go == '&' or go ==
211             if (pla + 1 < ar.length()) {
212                 if (ar[pla + 1] == '=') {
213                     iroha get_assign_operator();

```

```
214         } else if (ar[pla + 1] == go) {
215             iroha get_double_operator();
216         }
217     }
218     iroha get_single_char();
219 }
220 if (ispunct(go)) {
221     iroha get_single_char();
222 }
223 iroha {Error, string{meion_gc()}};
224 }
225 };
226
227 meion test() -> void {
228     string s, t;
229     while (std::getline(std::cin, s)) {
230         t += s;
231         t += '\n';
232     }
233     lycoris chisato(t);
234     chisato.show_tokens();
235 }
236 } // namespace n_token_solver
```
