BJTU编译原理lab1实验报告

编译原理Lab01: 词法分析

22281089 陈可致

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:

```
<标识符>→字母 | <标识符>字母 | <标识符>d
<无符号整数>→数字 | <无符号整数>数字
<单字符分界符>→+ | － | * | ; | , | ( | ) | { | }
<双字符分界符>→<大于>= | <小于>= | <小于>> | <感叹号>= | <等于>= | <斜竖>*
<小于>→< <等于>→= <大于>→> <斜竖> →/
<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

```
sudo apt update
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

```
cd your_file_forder
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

```
enum token_type {
    Reserved,
    Identifier,
    Number,
    Single_char,
    Comma,   // ,
    Colon,   // :
    Semi,    // ;
    Bb_lft,  // {
    Bb_rt,   // }
    Sb_lft,  // (
    Sb_rt,   // )
    Plus,    // +
    Minus,   // -
    Mul,     // *
    Div,     // /
    Lt,      // <
    Gt,      // >
    Assign,  // =
    Not,     // !
    Double_char,
    Le,           // <=
    Ge,           // >=
    Eq,           // ==
    Ne,           // !=
    Plus_assign,  // +=
    Minus_assign, // -=
    Mul_assign,   // *=
    Div_assign,   // /=
    Self_inc,     // ++
    Self_dec,     // --
    Rsh,          // >>
    Lsh,          // <<
    And,          // &&
    Or,           // ||
    Comment,      // // or /*
```

```
    Epsilon,       // ε
    End,           // #
    Error,
    Non_terminal,
    meion_cnt
};
```

- token struct

```
struct takina {
    token_type type;
    string value;
};
```

- token solver

```
class lycoris {
   public:
    lycoris() {}
    lycoris(const string &s) {}
    meion build(const string &s) -> void {}
    meion get_tokens() -> vector<takina> {}
    meion get_tokens(const string &s) -> vector<takina> {}
    meion get_token_type(const string &s) -> token_type {}
    meion show_tokens() -> void {}
    meion MeIoN_out0() -> void {}
    meion MeIoN_out1() -> void {}
   private:
    string ar;
    int pla;
    meion go() -> void {}
    meion meion_gc() -> char {}
    meion get_string() -> takina {}
    meion get_num() -> takina {}
    meion get_single_char() -> takina {}
    meion get_assign_operator() -> takina {}
    meion get_double_operator() -> takina {}
    meion get_comment() -> takina {}
    meion get_token() -> takina {}
};
```

六. 程序设计

1. 项目结构

   项目分为几个模块

   - Lib: 头文件
     1. MeIoN_H.hpp: 用到的标准库头文件. 使用的stl容器, 宏定义
     2. MeIoN_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:

```
int main() {
    int a = 10;
    float b = 20.5;
    a++;
    // This is a comment
    /* This is a
    multi-line comment */
    a /= b;
}
```

  - test1:

```
void foo() {
    double x = 3.14;
    int y = 42;
    if (x > y) {
        y += 1;
    } else {
        y -= 1;
    }
}
```

  - test2:

```
for (int i = 0; i < 10; i++) {
    while (i < 5) {
        i *= 2;
    }
    do {
        i--;
    } while (i > 0);
}
```

  - test3:

```
int add(int x, int y) { return x + y; }

int main() {
    int result = add(5, 3);
    result <<= 1;
    result >>= 1;
    result &= 7;
    result |= 2;
    result ^= 4;
    return result;
}
```

- **std**
  - std0:

```
Ciallo～(∠·ω< )⌒★
token count: 24
type: reserved    value: int
type: identifier  value: main
type: (           value: (
type: )           value: )
```

```
type: {           value: {
type: reserved    value: int
type: identifier  value: a
type: =           value: =
type: number      value: 10
type: ;           value: ;
type: reserved    value: float
type: identifier  value: b
type: =           value: =
type: number      value: 20.5
type: ;           value: ;
type: identifier  value: a
type: ++          value: ++
type: ;           value: ;
type: identifier  value: a
type: /=          value: /=
type: identifier  value: b
type: ;           value: ;
type: }           value: }
type: end         value: #
```

- std1:

```
Ciallo～(∠·ω< )⌒★
token count: 36
type: reserved    value: void
type: identifier  value: foo
type: (           value: (
type: )           value: )
type: {           value: {
type: reserved    value: double
type: identifier  value: x
type: =           value: =
type: number      value: 3.14
type: ;           value: ;
type: reserved    value: int
type: identifier  value: y
type: =           value: =
type: number      value: 42
type: ;           value: ;
type: reserved    value: if
type: (           value: (
type: identifier  value: x
type: >           value: >
type: identifier  value: y
type: )           value: )
type: {           value: {
type: identifier  value: y
type: +=          value: +=
type: number      value: 1
type: ;           value: ;
type: }           value: }
type: reserved    value: else
type: {           value: {
type: identifier  value: y
type: -=          value: -=
type: number      value: 1
type: ;           value: ;
type: }           value: }
type: }           value: }
type: end         value: #
```

- std2:

```
Ciallo～(∠·ω< )⌒★
token count: 42
type: reserved    value: for
type: (           value: (
type: reserved    value: int
type: identifier  value: i
type: =           value: =
type: number      value: 0
type: ;           value: ;
type: identifier  value: i
type: <           value: <
type: number      value: 10
type: ;           value: ;
type: identifier  value: i
type: ++          value: ++
type: )           value: )
type: {           value: {
type: reserved    value: while
type: (           value: (
type: identifier  value: i
type: <           value: <
type: number      value: 5
type: )           value: )
type: {           value: {
type: identifier  value: i
type: *=          value: *=
type: number      value: 2
type: ;           value: ;
type: }           value: }
type: reserved    value: do
type: {           value: {
type: identifier  value: i
type: --          value: --
type: ;           value: ;
type: }           value: }
type: reserved    value: while
type: (           value: (
type: identifier  value: i
type: >           value: >
type: number      value: 0
type: )           value: )
type: ;           value: ;
type: }           value: }
type: end         value: #
```

- std3:

```
Ciallo～(∠·ω< )⌒★
token count: 58
type: reserved    value: int
type: identifier  value: add
type: (           value: (
type: reserved    value: int
type: identifier  value: x
type: ,           value: ,
type: reserved    value: int
type: identifier  value: y
type: )           value: )
type: {           value: {
type: identifier  value: return
type: identifier  value: x
```

```
type: +           value: +
type: identifier  value: y
type: ;           value: ;
type: }           value: }
type: reserved    value: int
type: identifier  value: main
type: (           value: (
type: )           value: )
type: {           value: {
type: reserved    value: int
type: identifier  value: result
type: =           value: =
type: identifier  value: add
type: (           value: (
type: number      value: 5
type: ,           value: ,
type: number      value: 3
type: )           value: )
type: ;           value: ;
type: identifier  value: result
type: <<          value: <<
type: =           value: =
type: number      value: 1
type: ;           value: ;
type: identifier  value: result
type: >>          value: >>
type: =           value: =
type: number      value: 1
type: ;           value: ;
type: identifier  value: result
type: double char value: &=
type: number      value: 7
type: ;           value: ;
type: identifier  value: result
type: double char value: |=
type: number      value: 2
type: ;           value: ;
type: identifier  value: result
type: double char value: ^=
type: number      value: 4
type: ;           value: ;
type: identifier  value: return
type: identifier  value: result
type: ;           value: ;
type: }           value: }
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~~

```
bash go.sh
test: 1
accept
test: 2
accept
test: 3
accept
test: 4
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

```cpp
#pragma once
#include <algorithm>
#include <array>
#include <bitset>
#include <cassert>
#include <cctype>
#include <chrono>
#include <cmath>
#include <cstring>
#include <ctime>
#include <fstream>
#include <functional>
#include <iomanip>
#include <iostream>
#include <limits>
#include <map>
#include <queue>
#include <random>
#include <ranges>
#include <set>
#include <stack>
#include <string>
#include <tuple>
#include <unordered_map>
#include <unordered_set>

using   std::array, std::bitset, std::deque, std::greater, std::less, std::map,
        std::multiset, std::pair, std::priority_queue, std::set, std::stack,
        std::string, std::vector, std::tuple, std::function;

using NAME = void;       using uint = unsigned;   using ll = long long;      using ull = unsigned long lon
using ld = long double;  using i128 = __int128_t; using u128 = __uint128_t;  using f128 = __float128;

#define meion     auto
#define iroha     return

```

2. MeloN_debug.hpp

```cpp
#pragma once

template <class T, size_t size = std::tuple_size<T>::value>
std::string to_debug(T, std::string s = "")
    requires(not std::ranges::range<T>);
std::string to_debug(meion x)
    requires requires(std::ostream& os) { os << x; }
{
    iroha static_cast<std::ostringstream>(std::ostringstream() << x).str();
}
std::string to_debug(std::ranges::range meion x, std::string s = "")
    requires(not std::is_same_v<decltype(x), std::string>)
{
    for (meion xi : x) {
        s += ", " + to_debug(xi);
```

```
    }
    iroha "[" + s.substr(s.empty() ? 0 : 2) + "]";
}
template <class T, size_t size>
std::string to_debug(T x, std::string s)
    requires(not std::ranges::range<T>)
{
    [&]<size_t... I>(std::index_sequence<I...>) {
        ((s += ", " + to_debug(std::get<I>(x))), ...);
    }(std::make_index_sequence<size>());
    iroha "(" + s.substr(s.empty() ? 0 : 2) + ")";
}
#ifdef MeIoN
#define debug(...)  std::cout << "Ciallo～(∠・ω< )⌒★ " << "(" #__VA_ARGS__ ") = " << to_debug(std::tupl
#else
#define debug(...) void(0721)
#endif
```

3. 0_token.hpp

```
#pragma once
enum token_type {
    Reserved,
    Identifier,
    Number,
    Single_char,
    Comma,   // ,
    Colon,   // :
    Semi,    // ;
    Bb_lft,  // {
    Bb_rt,   // }
    Sb_lft,  // (
    Sb_rt,   // )
    Plus,    // +
    Minus,   // -
    Mul,     // *
    Div,     // /
    Lt,      // <
    Gt,      // >
    Assign,  // =
    Not,     // !
    Double_char,
    Le,            // <=
    Ge,            // >=
    Eq,            // ==
    Ne,            // !=
    Plus_assign,   // +=
    Minus_assign,  // -=
    Mul_assign,    // *=
    Div_assign,    // /=
    Self_inc,      // ++
    Self_dec,      // --
    Rsh,           // >>
    Lsh,           // <<
    And,           // &&
    Or,            // ||
    Comment,       // // or /*
    Epsilon,       // ε
    End,           // #
    Error,
    Non_terminal,
    meion_cnt
```

```
};

const std::string type_to_s[meion_cnt] = {
    "reserved", "identifier",  "number",      "single_char", ",",       ":",
    ";",        "{",           "}",           "(",           ")",       "+",
    "-",        "*",           "/",           "<",           ">",       "=",
    "!",        "double char", "<=",          ">=",          "==",      "!=",
    "+=",       "-=",          "*=",          "/=",          "++",      "--",
    ">>",       "<<",          "and",         "or",          "comment", "ε",
    "end",      "error",       "non terminal"};

const std::unordered_set<std::string> reserved_words = {
    "void", "int", "float", "double", "if", "else", "for", "do", "while"};


// token
struct takina {
    token_type type;
    string value;
};

std::ostream &operator<<(std::ostream &os, const takina &x) {
    os << type_to_s[x.type] << ' ' << x.value;
    iroha os;
}

void show(const takina &x) {
    std::cout << "type: " << std::setw(12) << std::left << type_to_s[x.type]
            << "value: " << std::setw(10) << std::left << x.value
            << std::endl;
}

```

4. 0_token_solver

```
#pragma once
#include "MeIoN_H.hpp"
#include "MeIoN_debug.hpp"
#include "0_token.hpp"
// slover
namespace n_token_solver {
    class lycoris {
    public:
        lycoris() { build(""); }
        lycoris(const string &s) { build(s); }
        meion build(const string &s) -> void {
            ar = s;
        }

        meion get_tokens() -> vector<takina> {
            pla = 0;
            vector<takina> ret;
            while (true) {
                meion [type, value] = get_token();
                if (type == Error) {
                    std::cerr << "Zako~ coder, " << value << std::endl;
                } else if (type == Comment) {
                    continue;
                } else {
                    ret.emplace_back(type, value);
                }
                if (type == End) {
```

```
                    break;
                }
            }
            iroha ret;
        }

        meion get_tokens(const string &s) -> vector<takina> {
            build(s);
            iroha get_tokens();
        }

        meion get_token_type(const string &s) -> token_type {
            if (s.empty()) {
                iroha Epsilon;
            } else if (s[0] == '#') {
                iroha End;
            }
            pla = 0;
            build(s);
            iroha get_token().type;
        }

        meion show_tokens() -> void {
            vector tokens = get_tokens();
            std::cout << "Ciallo～(∠·ω< )⌒★ " << std::endl;
            std::cout << "token count: " << tokens.size() << std::endl;
            for (const meion &x : tokens) {
                show(x);
            }
        }

        meion MeIoN_out0() -> void {
            for (const meion &[type, value] : get_tokens()) {
                std::cout << type << ' ' << value << '\n';
            }
        }
        meion MeIoN_out1() -> void {
            for (const meion &[type, value] : get_tokens()) {
                std::cout << type_to_s[type] << ' ' << value << '\n';
            }
        }

    private:
        string ar;
        int pla;

        meion go() -> void {
            while (pla < ar.length() and isspace(ar[pla])) {
                ++pla;
            }
        }

        meion meion_gc() -> char {
            iroha ar[pla++];
        }
        meion get_string() -> takina {
            string value;
            while (pla < ar.length() and (std::isalnum(ar[pla]) or ar[pla] == '_')) {
                value += meion_gc();
            }

            if (reserved_words.contains(value)) {
```

```
            iroha {Reserved, value};
        }
        iroha {Identifier, value};
    }
    meion get_num() -> takina {
        string value;
        bool RE = false;
        while (pla < ar.length() and std::isdigit(ar[pla])) {
            value += meion_gc();
        }
        if (pla < ar.length() and ar[pla] == '.') {
            RE = true;
            value += meion_gc();
            while (pla < ar.length() and std::isdigit(ar[pla])) {
                value += meion_gc();
            }
        }

        if (RE and value[0] == '.' or value.back() == '.') {
            iroha {Error, value};
        }
        iroha {Number, value};
    }
    meion get_single_char() -> takina {
        meion op = meion_gc();
        switch (op) {
            case ',': iroha {Comma,  string{op}};
            case ':': iroha {Colon,  string{op}};
            case ';': iroha {Semi,   string{op}};
            case '{': iroha {Bb_lft, string{op}};
            case '}': iroha {Bb_rt,  string{op}};
            case '(': iroha {Sb_lft, string{op}};
            case ')': iroha {Sb_rt,  string{op}};
            case '+': iroha {Plus,   string{op}};
            case '-': iroha {Minus,  string{op}};
            case '*': iroha {Mul,    string{op}};
            case '/': iroha {Div,    string{op}};
            case '<': iroha {Lt,     string{op}};
            case '>': iroha {Gt,     string{op}};
            case '=': iroha {Assign, string{op}};
            case '!': iroha {Not,    string{op}};
        }
        iroha {Single_char, string{op}};
    }
    meion get_assign_operator() -> takina {
        if (pla + 1 < ar.length()) {
            meion l = meion_gc(), r = meion_gc();
            if (r != '=') {
                iroha {Error, "inv assign operator"};
            }
            switch (l) {
                case '<': iroha {Le,           string{l, r}};
                case '>': iroha {Ge,           string{l, r}};
                case '=': iroha {Eq,           string{l, r}};
                case '!': iroha {Ne,           string{l, r}};
                case '+': iroha {Plus_assign,  string{l, r}};
                case '-': iroha {Minus_assign, string{l, r}};
                case '*': iroha {Mul_assign,   string{l, r}};
                case '/': iroha {Div_assign,   string{l, r}};
            }
            iroha {Double_char, string{l, r}};
        }
```

```
            iroha {Error, "inv assign operator"};
        }
        meion get_double_operator() -> takina {
            if (pla + 1 < ar.length()) {
                meion l = meion_gc(), r = meion_gc();
                if (l != r) {
                    iroha {Error, "inv double operator"};
                }
                switch (l) {
                    case '+': iroha {Self_inc, string{l, r}};
                    case '-': iroha {Self_dec, string{l, r}};
                    case '<': iroha {Lsh, string{l, r}};
                    case '>': iroha {Rsh, string{l, r}};
                    case '&': iroha {And, string{l, r}};
                    case '|': iroha {Or, string{l, r}};
                }
                iroha {Double_char, string{l, r}};
            }
            iroha {Error, string{meion_gc()}};
        }
        meion get_comment() -> takina {
            if (ar[pla + 1] == '/') {
                while (meion_gc() != '\n') ;
                iroha {Comment, "//"};
            } else if (ar[pla + 1] == '*') {
                char l = 'm', r = 'l';
                while (pla < ar.length() and not(l == '*' and r == '/')) {
                    l = r;
                    r = meion_gc();
                }
                iroha {Comment, "/*...*/"};
            }
            iroha {Error, "inv comment"};
        }
        meion get_token() -> takina {
            go();
            if (pla + 1 > ar.length()) iroha {End, "#"};
            char go = ar[pla];
            if ((go < 'z' + 1 and go > 'a' - 1) or
                (go < 'Z' + 1 and go > 'A' - 1)) {
                iroha get_string();
            } else if (go < '9' + 1 and go > '0' - 1) {
                iroha get_num();
            } else if (go == '/') {
                if (pla + 1 < ar.length()) {
                    if (ar[pla + 1] == '=') {
                        iroha get_assign_operator();
                    } else if (ar[pla + 1] == '/' or ar[pla + 1] == '*') {
                        iroha get_comment();
                    }
                }
                iroha get_single_char();
            } else if (go == '*' or go == '%' or go == '!' or go == '^' or
                    go == '^' or go == '~') {
                if (pla + 1 < ar.length() and ar[pla + 1] == '=') {
                    iroha get_assign_operator();
                }
                iroha get_single_char();
            } else if (go == '+' or go == '-' or go == '>' or go == '<' or go == '=' or go == '&' or go ==
                if (pla + 1 < ar.length()) {
                    if (ar[pla + 1] == '=') {
                        iroha get_assign_operator();
```

```
                    } else if (ar[pla + 1] == go) {
                        iroha get_double_operator();
                    }
                }
                iroha get_single_char();
            }
            if (ispunct(go)) {
                iroha get_single_char();
            }
            iroha {Error, string{meion_gc()}};
        }
    };

    meion test() -> void {
        string s, t;
        while (std::getline(std::cin, s)) {
            t += s;
            t += '\n';
        }
        lycoris chisato(t);
        chisato.show_tokens();
    }
}  // namespace n_token_solver
```