**南 京 师 范 大 学**

**《编译原理》**

**课程实验报告**



**题 目： 词法分析**

**学 院： 计算机与电子信息**

**专 业： 计算机科学与技术**

**姓 名： 朱正阳**

**学 号： 19210217**

**任课教师： 蔡维玲**

**计算机与电子信息学院**

2024 年3月10日

# 实验目的

用C或C++语言编写一个简单的词法分析程序，扫描C语言小子集的源程序，根据给定的词法规则，识别单词，填写相应的表。如果产生词法错误，则显示错误信息、位置，并试图从错误中恢复。简单的恢复方法是忽略该字符（或单词）重新开始扫描。

# 实验原理

**相关词法规则**

<标识符>::=<字母>  
<标识符>::=<标识符><字母>  
<标识符>::=<标识符><数字>  
<常量>::＝<无符号整数>  
<无符号整数>::=<数字序列>  
<数字序列>::=<数字序列><数字>  
<数字序列>::=<数字>  
<字母>::=a|b|c|……|x|y|z  
<数字>::=0|1|2|3|4|5|6|7|8|9  
<加法运算符>::＝+|-  
<乘法运算符>::=\*|／  
<关系运算符>::＝<|>|!=|>=|<=|==  
<分界符>::＝,|;|(|)|{|}  
<保留字>::＝main|int|if|else|while|do

**编写词法分析程序的步骤：**

（1）确定所要翻译的语言（或其子集）。

C语言

（2）设计属性字，及各类表格，如标识符表、常量表、符号及其机内表示对照表等。

与词法分析有关的表格：

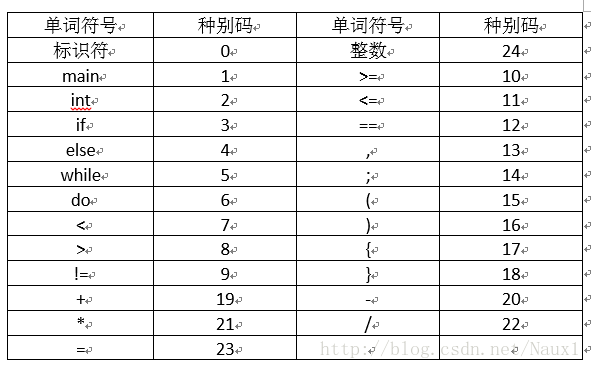
1. **实验完成情况**

与词法分析有关的表格：

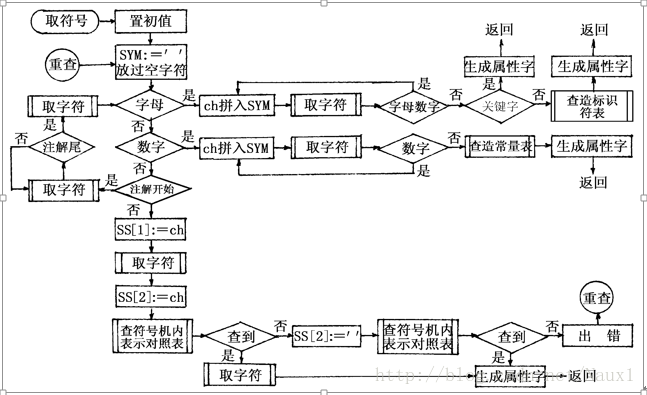
**1. 字符表**

保留字：main，int，if，else，while，do  
字母（全小写）：a|b|c|……|x|y|z  
数字：0,1,2,3,4,5,6,7,8,9  
运算符和界符：<，>，!=，>=，<=，==，,，;，(，)，{，}

**2. 特定单词机内表示表**



**3．画出总控流程图及各个子程序的流程图**



**4. 程序**

输入：一个存放C语言程序的s.txt文件  
输出：存放以（单词，种别码）形式输出的result.txt文件

1. **核心算法和代码**

#include <unordered\_map>

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

class LexicalAnalyzer

{

public:

    enum TokenType

    {

        Identifier = 0, // 标识符

        Constant = 24,  // 常量

        Other           // 其他

    };

public:

    // 标识符、常量及保留字识别函数

    const std::unordered\_map<std::string, int> dic =

        {

            {"main",1,},

            {"int", 2},

            {"if", 3},

            {"else", 4},

            {"while", 5},

            {"do", 6},

            {"<", 7},

            {">", 8},

            {"!=", 9},

            {">=", 10},

            {"<=", 11},

            {"==", 12},

            {",", 13},

            {";", 14},

            {"(", 15},

            {")", 16},

            {"{", 17},

            {"}", 18},

            {"+", 19},

            {"-", 20},

            {"\*", 21},

            {"/", 22},

            {"=", 23}};

public:

    bool isLittleLetter(const char ch) { return 'a' <= ch && ch <= 'z'; }

    bool isdigit(const char ch) {return '0' <= ch && ch <= '9' || ch == '.';}

    // 读入字符串进行分割, 提取出dic中的标识符、常量及保留字

    std::string analyze(const std::string &codeStr)

    {

        std::string curWord;

        std::string res;

        TokenType type = Other;

        // 执行操作函数，封装方便服用

        auto func = [&](int dicVal) -> void

        {

            std::string tmp = "(" + curWord + "," + std::to\_string(dicVal) + ")\n";

            res.append(tmp);

            curWord.clear(); // 清空

            type = Other;

        };

        for (int i = 0; i < codeStr.length(); i++)

        {

            const auto& ch = codeStr[i];

            if (ch == '\n' || ch == ' ' || ch == '\t')

            {

                // 分割出串，进行判断

                if (dic.find(curWord) != dic.end())

                    func(dic.at(curWord));

                else if(!curWord.empty())

                    // 没找到，不是保留字。需要判断是整数还是标识符

                    func(type);

            }

            else if (isLittleLetter(ch))

            {

                if (type == Constant)

                    // 抛出异常，标识符不可以以数字开头

                    throw std::runtime\_error("Identifier can not start with number: " + curWord);

                else if (curWord.empty())

                    type = Identifier;

                curWord.push\_back(ch);

            }

            else if (isdigit(ch))

            {

                if (curWord.empty())

                    type = Constant;

                curWord.push\_back(ch);

            }

            else

            {

                // 遇到操作符，清空上一次的串

                if (!curWord.empty()){

                    if (dic.find(curWord) != dic.end())

                        func(dic.at(curWord));

                    else

                        func(type);

                }

                // 注解开始

                curWord.push\_back(ch);

                if (i + 1 < codeStr.length())

                    curWord.push\_back(codeStr[i + 1]);

                if (dic.find(curWord) != dic.end()){

                    func(dic.at(curWord));

                    i++;

                    continue;

                }

                curWord.pop\_back();

                if (dic.find(curWord) != dic.end()){

                    func(dic.at(curWord));

                    continue;

                }

                throw std::runtime\_error("Unknown operator: " + curWord);

            }

        }

        return res;

    }

};

class IO{

public:

    static std::string readFromFile(const std::string& path){

        std::ifstream fin(path);

        if (!fin)

            throw std::runtime\_error("Unable to open s.txt");

        std::stringstream buffer;

        buffer << fin.rdbuf();

        std::string res = buffer.str();

        fin.close();

        return res;

    }

    static void writeToFile(const std::string& path, const std::string& content){

        std::ofstream fo(path);

        if(!fo)

            throw std::runtime\_error("Unable to open s.txt");

        fo << content;

        fo.close();

    }

};

int main()

{

    LexicalAnalyzer la;

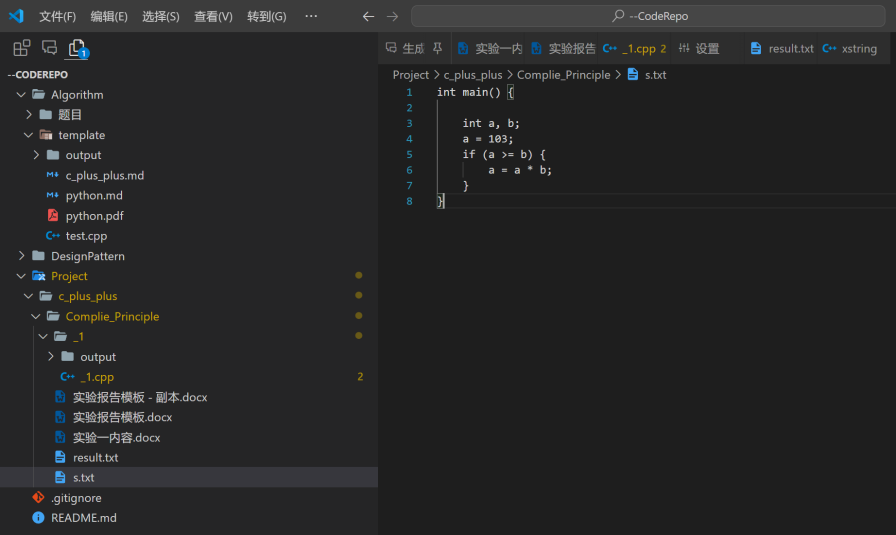
    IO::writeToFile("F:/--CodeRepo/--CodeRepo/Project/c\_plus\_plus/Complie\_Principle/result.txt",

        la.analyze(IO::readFromFile("F:/--CodeRepo/--CodeRepo/Project/c\_plus\_plus/Complie\_Principle/s.txt")));

    return 0;

}

1. **运行与测试**

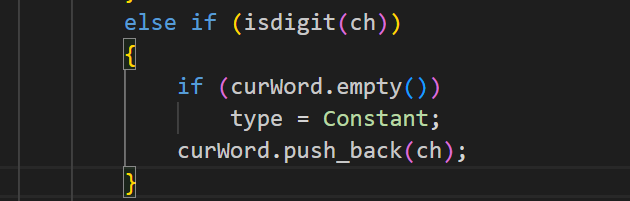
**输入文件：**

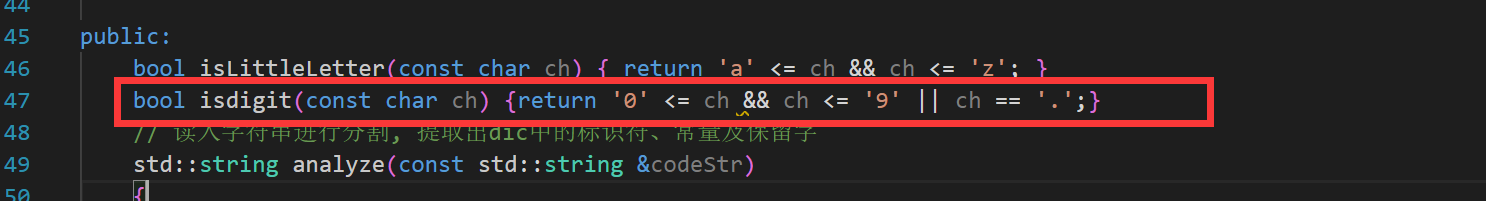
**输出文件：**

# 六、实验总结

**1.关于实现支持小数**

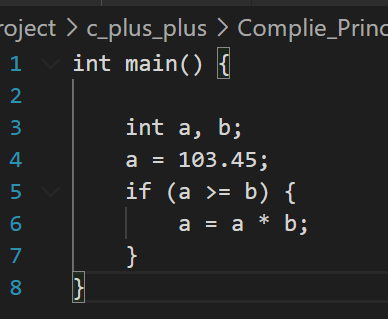
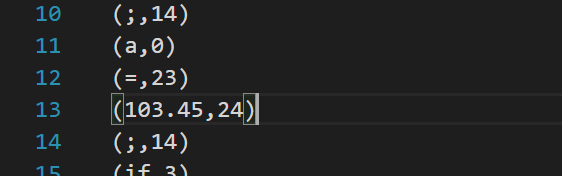
在不考虑”.”符号表示成员符号的情况下，支持将小数视作常量只需要改写isdigit函数的规则。





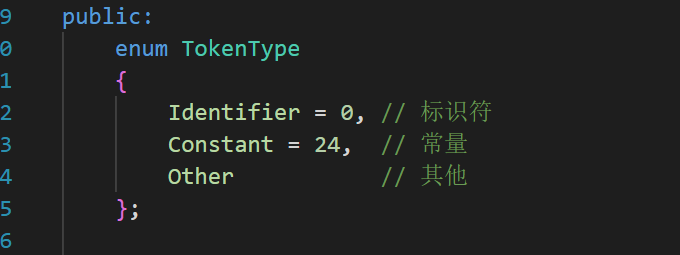
同理，如果需要支持其他大写字母，也可以修改isLetter函数的判断规则。

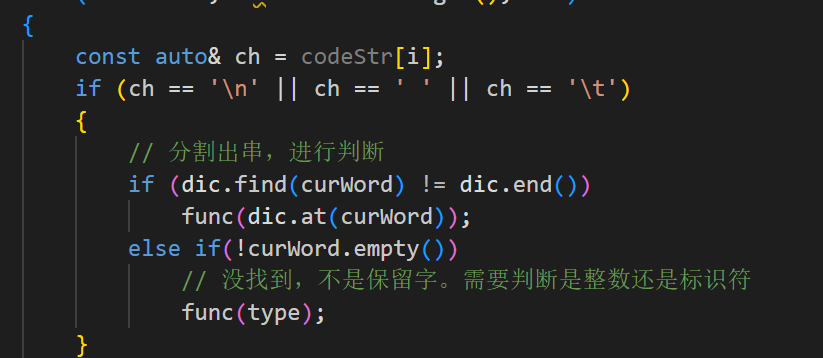
验证结果：



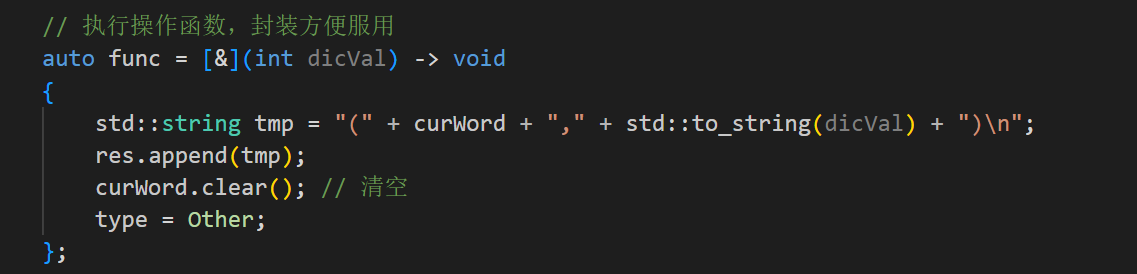
1. **如何区分Identifier和 保留字**

增加枚举类型TokenType，在遇到结束符或者操作符的时候，首先尽可能查找dic中是否定义过。查找成功意味着一定是保留字/定义的操作符，否则根据type类型判断是常量还是Identifier。在实现过程中，增加匿名函数，实现代码复用性提高。



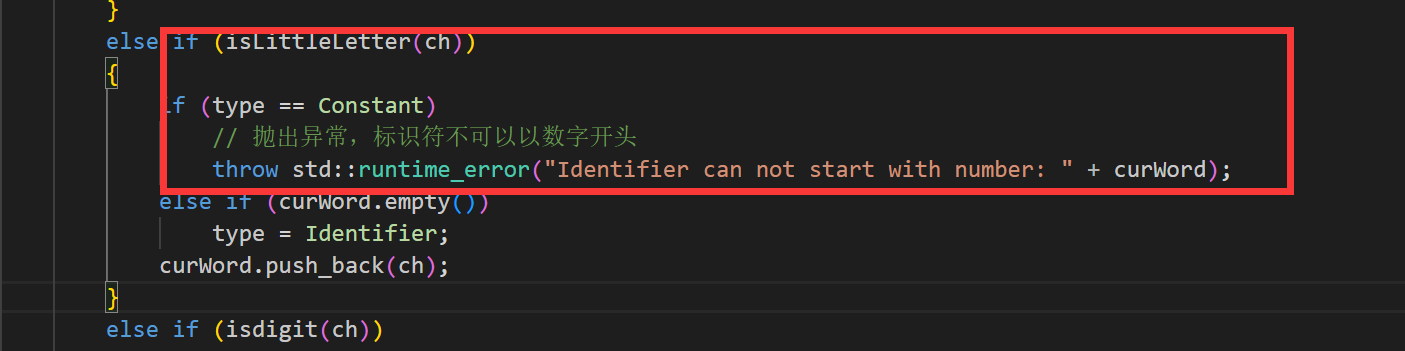


匿名函数, 接受一个种别码。通过type的设置将保留字/标识符/操作符/常量进行了统一。

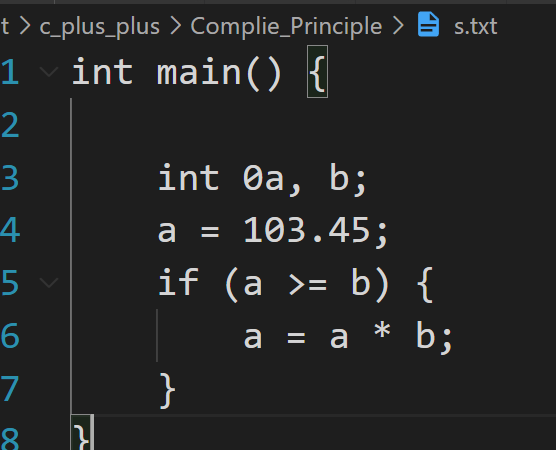


1. **如何提高程序的鲁棒性**

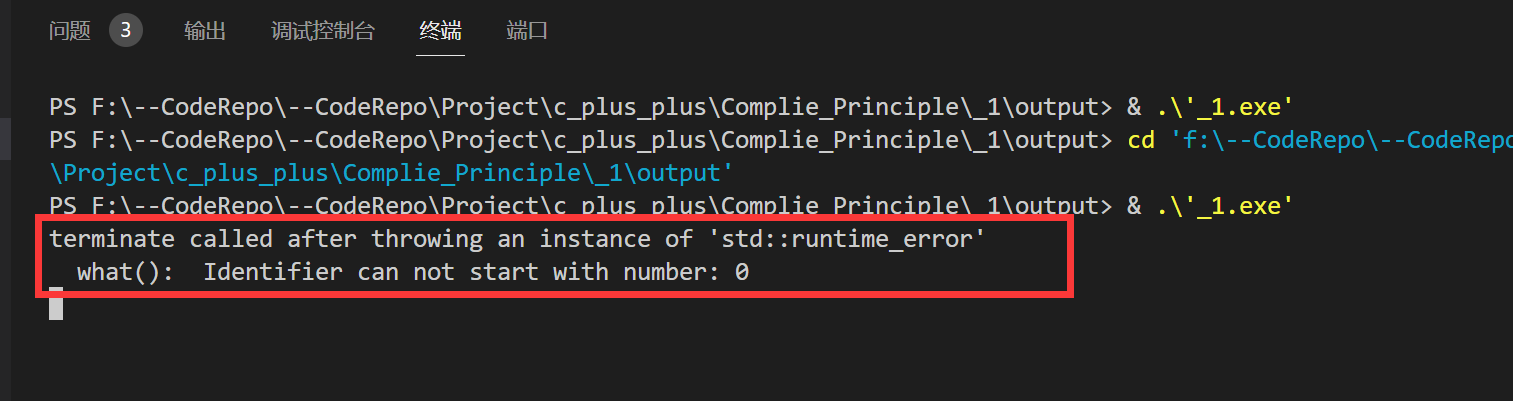
在遇到不合法输入的时候，比如不满足产生规则的标识符（以数字开头），简单的操作可以忽略。而本程序在实现中，通过type的定义，可以有效判断并且给出异常警告。



验证：



报出异常，标识符不可以以0开头。



# 附录：Python实现代码

class LexicalAnalyzer:

    Identifier = 0

    Constant = 24

    Other = 25

    def \_\_init\_\_(self):

        self.dic = {

            "main": 1,

            "int": 2,

            "if": 3,

            "else": 4,

            "while": 5,

            "do": 6,

            "<": 7,

            ">": 8,

            "!=": 9,

            ">=": 10,

            "<=": 11,

            "==": 12,

            ",": 13,

            ";": 14,

            "(": 15,

            ")": 16,

            "{": 17,

            "}": 18,

            "+": 19,

            "-": 20,

            "\*": 21,

            "/": 22,

            "=": 23

        }

    def isLittleLetter(self, ch):

        return 'a' <= ch <= 'z'

    def isdigit(self, ch):

        return '0' <= ch <= '9' or ch == '.'

    def analyze(self, codeStr):

        curWord = ''

        res = ''

        type = self.Other

        def func(dicVal):

            nonlocal res, curWord, type

            tmp = "(" + curWord + "," + str(dicVal) + ")\n"

            res += tmp

            curWord = ''

            type = self.Other

        i = 0

        while i < len(codeStr):

            ch = codeStr[i]

            if ch == '\n' or ch == ' ' or ch == '\t':

                if curWord in self.dic:

                    func(self.dic[curWord])

                elif curWord:

                    func(type)

            elif self.isLittleLetter(ch):

                if type == self.Constant:

                    raise RuntimeError("Identifier can not start with number: " + curWord)

                elif not curWord:

                    type = self.Identifier

                curWord += ch

            elif self.isdigit(ch):

                if not curWord:

                    type = self.Constant

                curWord += ch

            else:

                if curWord:

                    if curWord in self.dic:

                        func(self.dic[curWord])

                    else:

                        func(type)

                curWord += ch

                if i + 1 < len(codeStr):

                    curWord += codeStr[i + 1]

                if curWord in self.dic:

                    func(self.dic[curWord])

                    i += 1

                    continue

                curWord = curWord[:-1]

                if curWord in self.dic:

                    func(self.dic[curWord])

                    continue

                raise RuntimeError("Unknown operator: " + curWord)

            i += 1

        return res

class IO:

    @staticmethod

    def readFromFile(path):

        with open(path, 'r') as fin:

            return fin.read()

    @staticmethod

    def writeToFile(path, content):

        with open(path, 'w') as fo:

            fo.write(content)

def main():

    la = LexicalAnalyzer()

    IO.writeToFile("result.txt", la.analyze(IO.readFromFile("s.txt")))

if \_\_name\_\_ == "\_\_main\_\_":

    main()