南华大学

计算机学院

实 验 报 告

|  |  |
| --- | --- |
| 课程名称 | 编译原理 |
| 实验名称 | 词法分析 |

|  |  |  |  |
| --- | --- | --- | --- |
| 地点 | D205 | 教师 | 许纲理 |

1.实验目的及要求

明确语言的词法规则，根据具体情况，

1. 由同学们自己选取某种语言的一个适当大小的子集，写出基本保留字、标识符、常数、运算符、分隔符，完成单词符号的类别编码，并形成单词符号类别编码表。
2. 画整个词法分析的状态转换图。
3. 编制好程序。

4、上机调试，发现错误，再修改完善，上机调试通过，过程截图。

2.实验步骤和流程

确定 C++ 语言子集及单词符号：查阅 C++ 语言文档，选取常用的保留字、标识符规则、常数类型、运算符和分隔符作为子集。​

进行单词符号类别编码：为每种单词符号分配唯一的类别编码，整理形成单词符号类别编码表。​

绘制状态转换图：分析单词符号的识别规则，绘制状态转换图。​

编写词法分析程序：使用 C++ 语言，依据状态转换图和编码表实现词法分析功能。​

上机调试与修改：运行程序，输入测试用例，检查程序输出结果是否正确。根据调试过程中发现的错误，逐步修改和完善程序。​

记录调试过程：截取程序调试过程中的关键界面，包括输入测试用例、程序运行结果、错误提示信息等，用于实验报告展示。

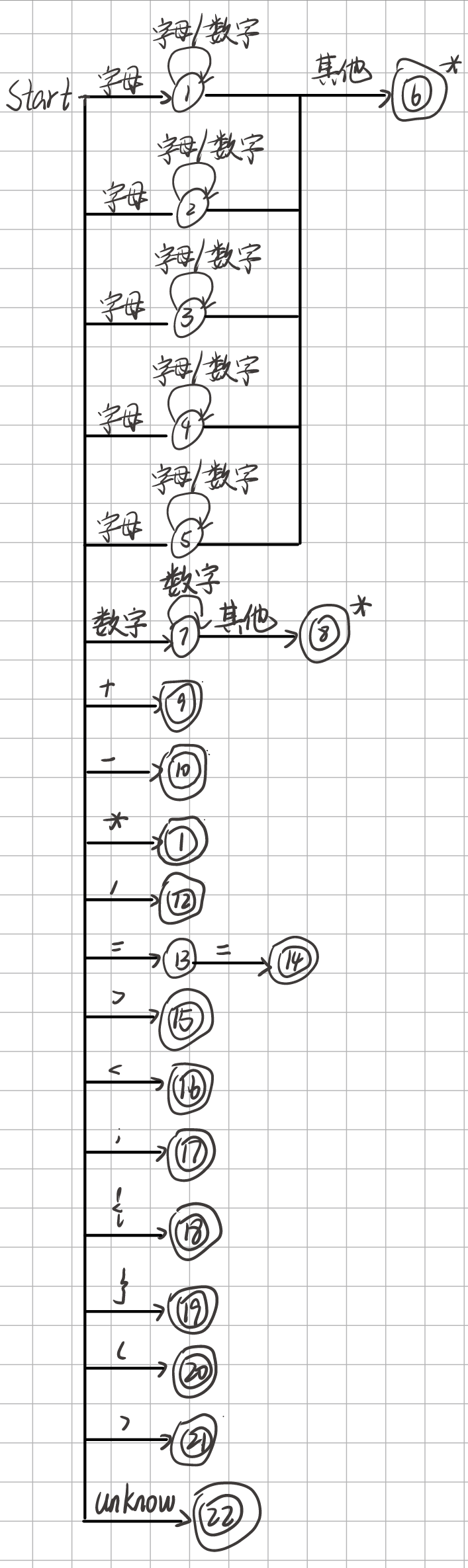
3.实验内容

（一）C++ 语言子集单词符号及类别编码​

单词符号类别编码表如下：​

| **单词符号类别** | **编码** | **示例** |
| --- | --- | --- |
| 数据类型：整型 | 1 | int |
| 数据类型：浮点型 | 2 | float |
| 条件语句：if | 3 | if |
| 条件语句：else | 4 | else |
| 循环语句：while | 5 | while |
| 标识符 | 6 | variable\_name, function\_name |
| 整数常量 | 7 | 1, 100, 255 |
| 浮点常量 | 8 | 1.2, 3.14, 0.001 |
| 运算符：加法 | 9 | + |
| 运算符：减法 | 10 | - |
| 运算符：乘法 | 11 | \* |
| 运算符：除法 | 12 | / |
| 运算符：赋值 | 13 | = |
| 运算符：大于 | 14 | > |
| 运算符：小于 | 15 | < |
| 运算符：等于 | 16 | == |
| 分隔符：分号 | 17 | ; |
| 分隔符：左大括号 | 18 | { |
| 分隔符：右大括号 | 19 | } |
| 分隔符：左小括号 | 20 | ( |
| 分隔符：右小括号 | 21 | ) |

（二）状态转换图绘制​



（三）程序编制​

使用 C++ 语言编写词法分析程序，源代码如下：

#include <iostream>

#include <string>

#include <cctype>

#include <vector>

// 单词符号类别编码

enum TokenType {

    INT = 1, FLOAT = 2, IF = 3, ELSE = 4, WHILE = 5,

    IDENTIFIER = 6, INTEGER\_CONSTANT = 7, FLOAT\_CONSTANT = 8,

    PLUS = 9, MINUS = 10, MULTIPLY = 11, DIVIDE = 12,

    ASSIGN = 13, GREATER = 14, LESS = 15, EQUAL = 16,

    SEMICOLON = 17, LEFT\_BRACE = 18, RIGHT\_BRACE = 19,

    LEFT\_PAREN = 20, RIGHT\_PAREN = 21

};

// 检查是否为保留字

int isReservedWord(const std::string& word) {

    if (word == "int") return INT;

    if (word == "float") return FLOAT;

    if (word == "if") return IF;

    if (word == "else") return ELSE;

    if (word == "while") return WHILE;

    return 0;

}

// 词法分析器

std::vector<std::pair<int, std::string>> lexicalAnalyzer(const std::string& input) {

    std::vector<std::pair<int, std::string>> tokens;

    int i = 0;

    while (i < input.length()) {

        if (isspace(input[i])) {

            i++;

            continue;

        }

        if (isalpha(input[i])) {

            std::string word;

            while (i < input.length() && (isalnum(input[i]) || input[i] == '\_')) {

                word += input[i];

                i++;

            }

            int reserved = isReservedWord(word);

            if (reserved) {

                tokens.emplace\_back(reserved, word);

            }

            else {

                tokens.emplace\_back(IDENTIFIER, word);

            }

        }

        else if (isdigit(input[i])) {

            std::string num;

            bool isFloat = false;

            while (i < input.length() && (isdigit(input[i]) || input[i] == '.')) {

                if (input[i] == '.') {

                    if (isFloat) break;

                    isFloat = true;

                }

                num += input[i];

                i++;

            }

            if (isFloat) {

                tokens.emplace\_back(FLOAT\_CONSTANT, num);

            }

            else {

                tokens.emplace\_back(INTEGER\_CONSTANT, num);

            }

        }

        else if (input[i] == '+' || input[i] == '-' || input[i] == '\*' || input[i] == '/' || input[i] == '=' || input[i] == '>' || input[i] == '<') {

            if (input[i] == '=' && i + 1 < input.length() && input[i + 1] == '=') {

                tokens.emplace\_back(EQUAL, "==");

                i += 2;

            }

            else {

                switch (input[i]) {

                case '+': tokens.emplace\_back(PLUS, "+"); break;

                case '-': tokens.emplace\_back(MINUS, "-"); break;

                case '\*': tokens.emplace\_back(MULTIPLY, "\*"); break;

                case '/': tokens.emplace\_back(DIVIDE, "/"); break;

                case '=': tokens.emplace\_back(ASSIGN, "="); break;

                case '>': tokens.emplace\_back(GREATER, ">"); break;

                case '<': tokens.emplace\_back(LESS, "<"); break;

                }

                i++;

            }

        }

        else if (input[i] == ';' || input[i] == '{' || input[i] == '}' || input[i] == '(' || input[i] == ')') {

            switch (input[i]) {

            case ';': tokens.emplace\_back(SEMICOLON, ";"); break;

            case '{': tokens.emplace\_back(LEFT\_BRACE, "{"); break;

            case '}': tokens.emplace\_back(RIGHT\_BRACE, "}"); break;

            case '(': tokens.emplace\_back(LEFT\_PAREN, "("); break;

            case ')': tokens.emplace\_back(RIGHT\_PAREN, ")"); break;

            }

            i++;

        }

        else {

            i++;

        }

    }

    return tokens;

}

int main() {

    std::string input;

    std::cout << "请输入要进行词法分析的内容: ";

    std::getline(std::cin, input);

    std::vector<std::pair<int, std::string>> tokens = lexicalAnalyzer(input);

    for (const auto& token : tokens) {

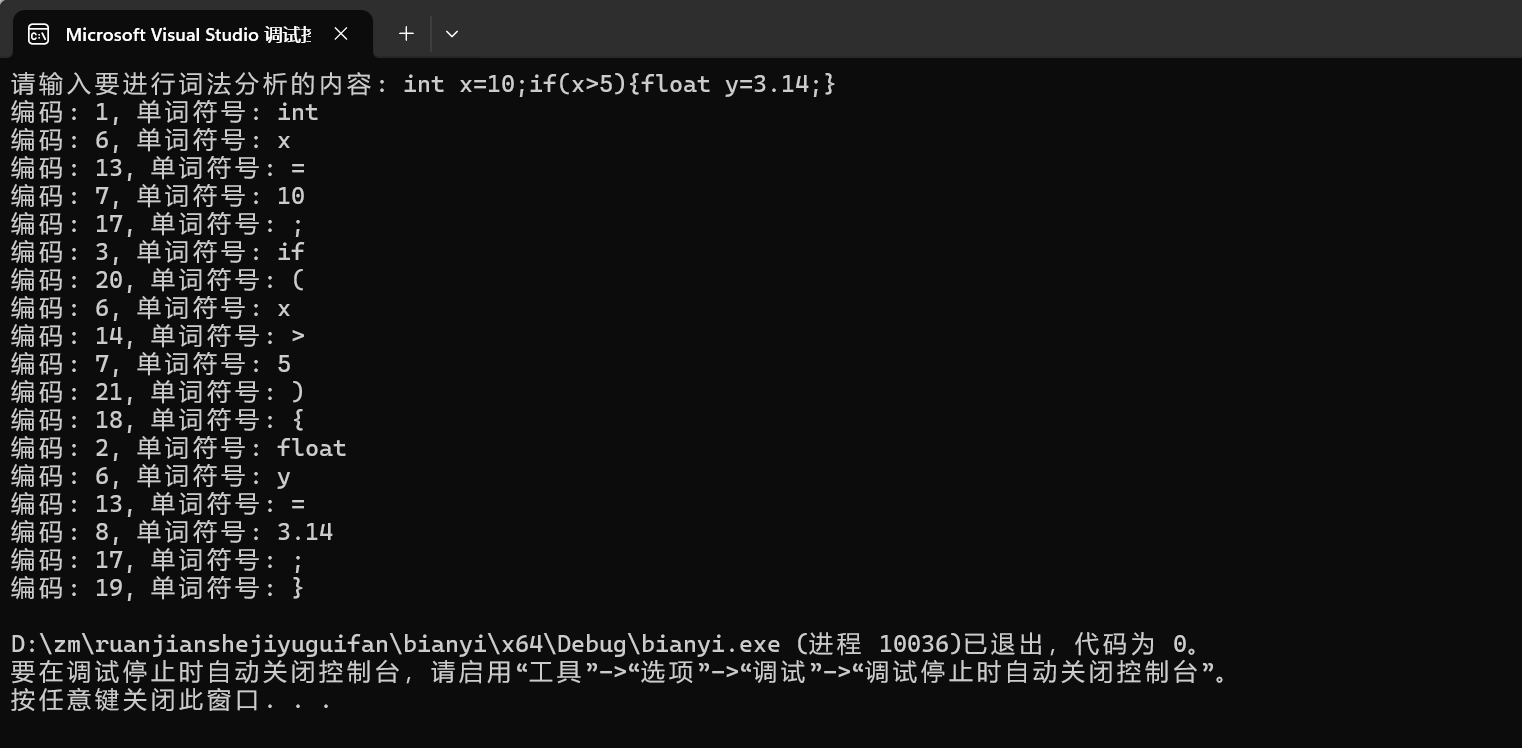
        std::cout << "编码: " << token.first << ", 单词符号: " << token.second << std::endl;

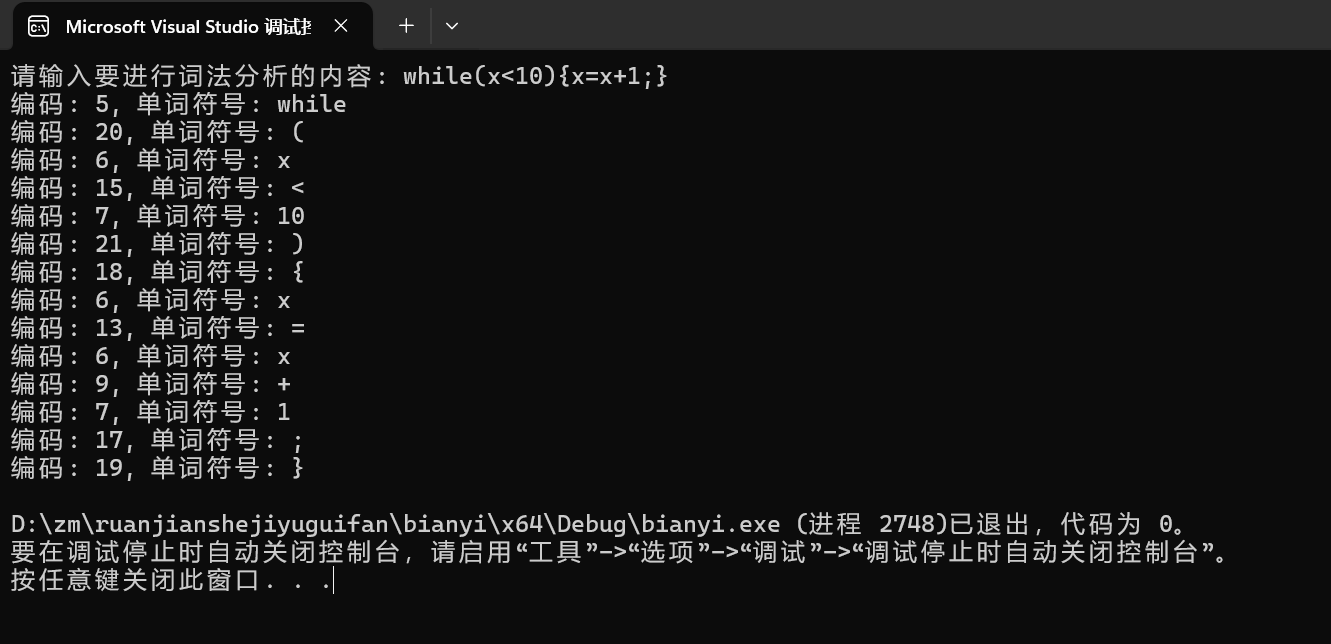
    }

    return 0;

}

4.实验结果





从输出结果可以看出，程序能够正确识别输入字符串中的保留字、标识符、运算符、分隔符和常数，并按照预先设定的类别编码进行输出，验证了词法分析程序的正确性。

4.实验总结分析

（一）实验过程中面临的问题及解决办法​

保留字与标识符的区分问题：在编写词法分析程序时，程序难以准确判断输入字符串是保留字还是普通标识符。最初的代码仅通过判断字符串首字符是否为字母或下划线来区分，导致保留字被误判为标识符。例如，将int识别成了普通标识符。​

解决办法：创建一个包含所有保留字的字符串数组，在识别到疑似标识符的字符串后，遍历该数组进行比对。若字符串与数组中的某个元素相同，则判定为保留字，赋予其对应的保留字编码；否则认定为普通标识符，赋予标识符编码，从而有效解决了这一问题。​

字符处理边界条件问题：在字符处理过程中，可能遇到多种边界情况，如输入字符串为空、仅包含空白字符或特殊字符等。这些情况可能导致程序出错或产生意外结果。​

解决办法：在开始词法分析前，先对输入字符串进行预处理。检查字符串是否为空，若为空则直接返回错误提示。对于仅包含空白字符的字符串，可将其视为无效输入进行处理。对于特殊字符，在词法规则中明确其处理方式，例如单独定义特殊字符的词法单元。在扫描字符过程中，针对每一种字符类型，如字母、数字、特殊字符等，分别设置边界条件判断。当扫描到字符串末尾时，确保所有正在处理的词法单元（如标识符、字符串常量等）都已正确结束，若未结束则抛出相应异常，以此全面解决字符处理中的边界条件问题。​

（二）心得体会​

通过本次实验，我对编译原理中词法分析的理论知识有了更深刻的理解，真正将课本上的概念转化为了实际的编程实践。在解决各种问题的过程中，不仅锻炼了自己独立思考和调试代码的能力，还学会了如何通过查阅资料、借鉴他人经验来攻克技术难题。同时，也意识到严谨的逻辑思维在程序设计中的重要性，一个小的逻辑漏洞就可能导致程序出现错误的结果。此外，实验过程中团队协作和交流也非常重要，与同学讨论问题时往往能获得新的思路和启发，这让我明白在学习和工作中，合作与沟通是不可或缺的。​

（三）程序优点​

错误处理机制：针对常见的词法错误，如非法字符、错误的浮点数格式、未终止的字符串常量等，程序设置了相应的异常抛出机制，能够及时反馈错误信息，有助于定位和解决问题。​

编码规范：代码遵循 C++ 语言的编码规范，变量和函数命名直观易懂，合理使用注释对关键代码段进行解释，提高了代码的可读性和可理解性。​

（四）值得改进的地方​

语言子集覆盖不足：目前选取的 C++ 语言子集较为基础，对于复杂的语法结构，如模板、lambda 表达式、类型转换等涉及的单词符号尚未处理。后续可以进一步拓展语言子集的范围，使词法分析器能够处理更丰富的 C++ 代码。​

性能优化：在处理较长的输入字符串时，程序的性能可能会受到影响。例如，在判断保留字时，遍历数组的方式效率较低。可以考虑使用哈希表等数据结构来提高查找保留字的速度，从而提升程序整体性能。​

词法单元合并：对于一些复合运算符（如+=、-=），目前程序将其拆分为单个字符进行处理，后续可以改进词法分析逻辑，实现对复合运算符的识别和处理，使词法分析结果更符合语法规则。