实验三 词法分析

一、实验内容

【任务名称】词法分析
【任务性质】必做任务，分值50分。
【任务介绍】根据给定源语言的构词规则，从任意字符串中识别出该语言所有的合法的单词符号，并以等长的二元组形式输出。
【输入】字符串形式的源程序。
【输出】单词符号所构成的串（流），单词以等长的二元组形式呈现。

【题目】设计一个程序，根据给定源语言的构词规则，从任意字符串中识别出该语言所有的合法的单词符号，并以等长的二元组形式输出。注意：
（1）附件一中介绍了一个基于C语法规则设计的源语言LittleC，其单词符号集合是C语言单词符号集合的真子集。
（2）学生可以自行挑选或设计一种源语言，以此为基础来完成本实验和后续实验。该语言的设计应该满足附件二的要求。
要求：
（1）输入文件：input.txt，纯文本。内容是一段经过预处理的“干净的”源程序/片段，不考虑源程序中有错误的情况。
（2）输出文件：output.txt。内容是单词符号流，一行一个单词，其顺序应该与源程序中的出现顺序完全一致。单词的数据结构应该设计为：【单词种别编码，单词属性】，种别码采用整数编码，属性应该是单词的值（直接值）或者指向存储空间的指针（地址）或者指向符号表的指针（表索引）或者为空（一个种别只有一个单词的情况）。
（3）处理过程：要求在实验报告中说明对主要数据结构和算法的设计，包括：单词符号的数据结构设计、符号表设计和核心处理过程（五类单词符号的识别过程等）。
（4）程序代码：要求以附件形式提供能编译运行的程序代码文件（包）。

二、实验设计

1、挑选LittleC为源语言

- 该语言的一个程序由一个块组成，该块中包含可选的声明语句和执行语句
- 该语言只支持一种数据类型，整型，因而也只有一种类型的变量：整型变量和一种类型的常量：整常数。
- 该语言支持+、-、*、/四种算术运算，运算式值为整常数
- 该语言的选择语句和循环语句中允许使用关系运算表达式来作为控制条件，其值为整常数（非零值表示true，零值表示false）
- 运算符的优先级和结合规则按照C语言语法理解
- 该语言不支持数组、结构体、指针等复杂数据类型，不支持函数调用，没有I/O功能。

文法定义：

```
PROG -> BLOCK
BLOCK -> { DECLS STMTS}
DECLS -> DECLS DECL | DECL | empty
DECL -> TYPE NAMES;
TYPE -> int
```

```
NAMES -> NAMES, NAME | NAME
NAME -> id

STMTS -> STMTS STMT | empty
STMT -> id = EXPR;
STMT -> if(BOOL) STMT
STMT -> if(BOOL) STMT else STMT
STMT -> while(BOOL) STMT
STMT -> BLOCK

EXPR -> EXPR ADD TERM | TERM
ADD -> + | -
TERM -> TERM MUL UNARY | FACTOR
MUL -> * | /
FACTOR -> (BOOL) | id | number
REL -> EXPR ROP EXPR
ROP -> > | >= | < | <= | == | !=
```

词法规则：

1. ）名字：由字母打头后跟字母、数字任意组合的字符串；长度不超过15；不区分大小写；把下划线看作第27个字母；
2. ）常数：完全由数字组成的字符串；正数和0前面不加符号，负数在正数前面加-构成；长度不超过15；
3. ）关键字、运算符、分隔符仅包含在文法定义中出现过的单词。
4. ）字母表定义为 1.）～ 3.）出现的字符的集合；不在该集合中的符号都以非法字符对待；

说明

- 结合测试用例，规定**5种**token种类：
  - **number**：上述常数，种别码0
  - **id**：上述“名字”，即C标识符，一个id一个种别码
  - **keyword**：关键字

    1. int 2. if 3.else 4.while 5. main 6. return
  - **operator**：运算符

    7. { 8. } 9. ; 10. (11.) 12. \ 13. " 14. ,
  - **separator**：分隔符

    15. = 16. + 17. - 18. * 19. / 20. > 21. < 22. % 23. >= 24. <= 25. == 26. !=
- 文法中没有注释，所以没有处理注释
- 在测试用例中，将%d看作运算符+标识符，\n看作分隔符+标识符

2、算法设计和数据结构

- 核心思路：分词 + 识别
- 技巧：根据文法，operator和separator是非常有利的提取token的工具，从要处理的字符串s头部开始遍历，遇到一个operator或separator就停下来提取它们出前面的部分进行处理，然后再接着重复此操作，直到s末尾，例如：

```
main(){ printf("a"); }
// 遇到(，处理前面的main
// ){
// 遇到(，处理前面的printf
// ("
// 遇到"，处理前面的a
// ...
```

- 利用string按空格分隔读入字符串的特性，又进一步划分了原程序段
- 利用C++STL中map存放5中token和它们的种别码，相当于运用散列法，知道token就知道了它的种别码，提高了效率。因为C++STL中的map有自动按照键值排序的功能，为了提高效率用了unordered_map
- 每一个token用结构体表示，存储它的值，单词中别，单词种别码

  所有的token按出现的顺序存在一个数组中，最后输出到文件里
- 利用<cctype>中的isdigit识别number
- 利用regex正则表达式识别id

```
regex e("[a-zA-Z_][a-z0-9A-Z_]{0,18}", regex_constants::icase);
```

- 如果遇到无法识别的token则直接报错输出Error

3. 驱动模块伪代码

```
while(遍历读入的字符串s) {
    i <- 找到的operator或separator位置
    识别i之前的token
    识别i位置的operator或separator
    如果s中没有operator和separator，那么识别s整体
    以上都无法识别出合法的token，报错
}
```

4. 识别合法number伪代码

```
bool alldigit(string s):
    BEGIN
        IF s[0]='-' THEN 去除负号
        WHILE(遍历s每个字符)
            如果不是数字，返回FALSE
        如果长度大于15，返回FALSE
        返回TRUE
    END
```

5. 识别合法的token（除运算符和分隔符）

```
bool istoken(string s):
    BEGIN
        IF s是keyword THEN 作为keyword插入符号表
        ELSE IF s是number THEN 作为number插入符号表
        ELSE IF s是id THEN
            IF s是新id THEN 作为id插入符号表
            ELSE 找到id已有的种别码，插入符号表
        ELSE RETURN FALSE;
        RETURN TRUE;
    END
```

6. 识别合法的operator和separator

```
bool isso(string s, int i)
    BEGIN
        IF s[i]是separator THEN 作为separator插入符号表
        ELSE IF s[i]s[i+1]是双字operator THEN 作为operator插入符号表
        ELSE IF s[i]是单字operator THEN 作为operator插入符号表
        ELSE RETURN FALSE;
        RETURN TRUE;
    END
```

三、实验环境

- 语言：C++
- 编译环境：Code Blocks 17.12 自带MinGW中的g++，需要在编译器设置中勾选C++14
- 输入：按照输入提示输入文件（input1.txt, input2.txt, input3.txt, input4.txt)
- 输出：文件Word_List.txt
- 关于输出格式控制：已用setw设置宽度，ios::left设置左对齐但是结果并不理想，最终也没想到解决方案QAQ

```
ofs.setf(ios::left);
for(int i=0; i<ans.size(); i++) {
    ofs << setw(15) << i+1
        << setw(20) << ans[i].type
        << setw(20) << ans[i].val
        << setw(15) << ans[i].num << endl;
}
```

Word_List.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1	keyword	int	1
2	keyword	main	5
3	separator	(	10
4	separator	)	11
5	separator	{	7
6	keyword	int	1
7	id	a	27
8	separator	;	9
9	keyword	int	1
10	id	b	27