

2.1 关键字

关键字包括：（小写）

or, and, int, bool, char, while, do, true, false

if, then, else, end, repeat, until, read, write

2.2 操作符

操作符包括：

>, <=, >=, <, =

,, ', {, }, :, :=

(,), +, -, *, /

2.3 分隔符

空格符包括：, \t, \n, 在词法分析后不被保留。

2.4 其他符号

其他符号有四种，他们的正则定义如下：

- 标识符 ID: `letter(letter|digit)*`
- 数字 NUM: `digit digit*`
- 字符串 STRING: `'any char except ''` (不能跨行定义)
- 注释: `{...}` (用花括号标注, 可以跨行定义)

三、程序说明

3.1 编译运行

- Windows下运行

```
1 gcc la.c -o la.exe
2 a.exe
```

- 测试文件位于 \test 文件夹

3.2 数据定义

- 程序中定义的 TOKEN 类型

```
1 // Keyword tokens // |-18 Keywords-----|
2 KEY_OR, KEY_AND, KEY_NOT, // | or | and | not |
3 KEY_INT, KEY_BOOL, KEY_CHAR, // | int | bool | char |
4 KEY_WHILE, KEY_DO, // | while | do | |
5 KEY_IF, KEY_THEN, KEY_ELSE, // | if | then | else |
6 KEY_END, KEY_REPEAT, KEY_UNTIL, // | end | repeat| until |
7 KEY_READ, KEY_WRITE, // | read | write | |
8 KEY_TRUE, KEY_FALSE, // | true | false | |
9
10 // Operator tokens // |-17 operators-|
11 OP_G, OP_L, OP_ASSIGN, // | > | < | := |
```

```

12     OP_GE, OP_LE, OP_EQ,                // | >= | <= | = |
13     OP_COMMA, OP_QUOTA, OP_SEMI,        // | , | ' | ; |
14     OP_LPAREN, OP_RPAREN,              // | ( | ) |   |
15     OP_LBRACE, OP_RBRACE,              // | { | } |   |
16     OP_ADD, OP_SUB,                     // | + | - |   |
17     OP_MUL, OP_DIV,                     // | * | / |   |
18
19     // Other tokens      // |-Each NF-----|-Example-|
20     ID,                  // | letter(letter|digit)*   | c1      |
21     NUM,                 // | digit digit*            | 123     |
22     STRING,              // | ' any character except' ' | 'hi!'   |
23     NONE,                // |                          | 1c      |
24     ANNO                 // | { any character }       | {hi}    |
25 };

```

并提供相关函数获取TOKEN_TYPE，其字符串形式

```

1 char* getTokenName(enum TOKEN_TYPE i);
2 enum TOKEN_TYPE getTokenType(char *token);

```

- 以下是词法分析有关数据结构

```

1 typedef struct TOKEN{                // TOKEN 二元组
2     enum TOKEN_TYPE type;            // 词素
3     char info[SIZE_2];              // 含义
4 }TOKEN;
5
6 FILE* file;                          // 待分析文件的指针
7 TOKEN file_tokens[SIZE_1];          // 保存词法分析结果
8 int file_tokens_num;                // 分析的TOKEN总数
9
10 int current_row;                    // 当前扫描位置（行数）
11 int current_col;                    // 当前扫描位置（列数）
12 char current_word[SIZE_2];          // 当前扫描得到的单词
13 int current_word_ptr;               // 当前扫描单词的长度
14 enum TOKEN_TYPE current_type;       // 当前的预测TOKEN类型
15 int operator_flag = 0;              // 操作符标记
16
17 int error;                          // 词法分析错误处

```

3.3 核心函数

```

1 /**
2  * 初始化：为所有用到的变量（@line 136- 147）赋初始值，若文件无啊打开，提示报错。
3  * @param file_path
4  * @return 1 if the file is opened successfully, otherwise 0.
5  */
6 int la_initial(char file_path[]);
7
8 /**
9  * 根据当前的 current_word 和 current_type 生成 Tokens 并保存在 file_tokens 中。
10 */
11 void la_make_token();
12
13 /**

```

```

14  * 输出词法编译结果：显示出错原因和出错位置（几行几列）
15  * @param message 出错信息
16  * @param row 出错位置（行数）
17  * @param col 出错位置（列数）
18  */
19  void la_show_error(const char* message, int row, int col);
20
21  /**
22   * 在扫描新的一行时进行预处理
23   */
24  void la_update_line();
25
26  /**
27   * 根据下一个字符 c 来更新 current_word 和 current_type。
28   * @param c lookahead symbol
29   */
30  void la_update_word(char c);
31
32  /**
33   * 词法分析入口：执行文件字符扫描和词法分析，在出现错误时或读完程序时终止
34   */
35  void la_start();

```

3.4 主函数

- 用户可以输入文件地址，每次输入一个文件路径，程序读取文件内容并作语法分析，将词法分析结果直接输出。
- 核心部分代码：

```

1  if(la_initial(file_path)){ // 1、初始化，若成功则进入词法分析
2      la_start();           // 2、执行词法分析
3      if(error == 0)        // 3、若没有错误出现
4          la_show_result(); // 4、则输出TOKEN序列
5  }

```

四、测试效果

测试 1

- 测试一段普通的TINY+程序的词法分析结果

```

1  char str;
2  int x, fact;
3  str:= 'sample program in TINY+ language';
4  read x;
5  if x > 0 and x < 100 then { don't do it }
6      fact:=1;
7      while x > 0 do
8          fact:=fact*x;
9          x:=x-1
10         write fact
11     end

```

- 测试结果：如下图所示。

```
命令提示符 - la.exe

Please enter the path of test file: ("q" to quit)
./t1.txt
KEY_CHAR [char]
ID [str]
OP_SEMI [;]
KEY_INT [int]
ID [x]
OP_COMMA [,]
ID [fact]
OP_SEMI [;]
ID [str]
OP_ASSIGN [:=]
STRING ['sample program in TINY+ language']
OP_SEMI [;]
KEY_READ [read]
ID [x]
OP_SEMI [;]
KEY_IF [if]
ID [x]
OP_G [>]
NUM [0]
KEY_AND [and]
ID [x]
OP_L [<]
NUM [100]
KEY_THEN [then]
ID [fact]
OP_ASSIGN [:=]
NUM [1]
OP_SEMI [;]
KEY_WHILE [while]
ID [x]
```

```
命令提示符 - la.exe

KEY_IF [if]
ID [x]
OP_G [>]
NUM [0]
KEY_AND [and]
ID [x]
OP_L [<]
NUM [100]
KEY_THEN [then]
ID [fact]
OP_ASSIGN [:=]
NUM [1]
OP_SEMI [;]
KEY_WHILE [while]
ID [x]
OP_G [>]
NUM [0]
KEY_DO [do]
ID [fact]
OP_ASSIGN [:=]
ID [fact]
OP_MUL [*]
ID [x]
OP_SEMI [;]
ID [x]
OP_ASSIGN [:=]
ID [x]
OP_SUB [-]
NUM [1]
KEY_WRITE [write]
ID [fact]
KEY_END [end]
Please enter the path of test file: ("q" to quit)
```

测试 2

- 测试内容：测试所有关键字和运算符，并给出几个比较特殊的嵌套例子：

{cc'oo'o}、'cco{}coc'

```
1 true false or and not
2 int bool char while do
3 if then else end repeat
4 until read write , ;
5 := + - * /
6 ( ) < = >
7 <= >= a2c 123 'EFG'
8 {cc'oo'o} cco0 'cco{}coc'
```

- 测试结果：所有词都被正确识别并转化为Token

```
命令提示符 - la.exe
C:\Users\17727\Desktop\编译原理\作业\词法分析实验\MyThings>la.exe

+ TINY+ Lexical Analysis Program

Please enter the path of test file: ("q" to quit)
./t2.txt
KEY_TRUE    [true]
KEY_FALSE   [false]
KEY_OR       [or]
KEY_AND      [and]
KEY_NOT      [not]
KEY_INT      [int]
KEY_BOOL     [bool]
KEY_CHAR     [char]
KEY_WHILE    [while]
KEY_DO       [do]
KEY_IF       [if]
KEY_THEN     [then]
KEY_ELSE     [else]
KEY_END      [end]
KEY_REPEAT   [repeat]
KEY_UNTIL    [until]
KEY_READ     [read]
KEY_WRITE    [write]
OP_COMMA     [,]
OP_SEMI      [;]
OP_ASSIGN    [:=]
OP_ADD       [+]
OP_SUB       [-]
OP_MUL       [*]
OP_DIV       [/]
OP_LPAREN    [(]
OP_RPAREN    [)]
OP_L         [<]
OP_EQ        [=]
OP_G         [>]
OP_LE        [<=]
OP_GE        [>=]
ID           [a2c]
NUM          [123]
STRING       ['EFG']
ID           [cco0]
STRING       ['cco{}coc']
Please enter the path of test file: ("q" to quit)

```

测试 3

- 测试内容：测试击中错误代码，分别保存在 t3.txt，t4.txt，t5.txt，t6.txt 中。

```
1 $ hello
2 123hello
3 hello'
4 hello}
```

它们分别代表：非法字符输入、非法ID、字符串不完整、注释不完整。

- 测试结果：

```
+ TINY+ Lexical Analysis Program
Please enter the path of test file: ("q" to quit)
./t3.txt
[ERROR] In 1:1: Illegal Symbol: $
Please enter the path of test file: ("q" to quit)
./t4.txt
[ERROR] In 1:4: Illegal Number/ID
Please enter the path of test file: ("q" to quit)
./t5.txt
[ERROR] In 1:6: Unclosed Statement, lack of '
Please enter the path of test file: ("q" to quit)
./t6.txt
[ERROR] In 1:6: Unclosed Annoation, lack of ']'
Please enter the path of test file: ("q" to quit)

```

实验总结

- 本次词法分析程序实验在TINY语言的基础上进行扩充，为扩展后的TINY+语言构造词法分析程序。实验过程中充分了解了TINY及TINY词法分析器的实现，编写的程序不仅可以将TINY+源程序翻译成对应的TOKEN序列，还能检查一定的词法错误。
- 但编写的代码比较杂乱不易于维护修改，同时，在测试上未必完善，可能还存在未发现的错误。除了直接使用C语言编写词法分析程序之外，另一种选择是使用Lex来构造词法分析程序，这种方法不

仅能确保程序的正确性，还能降低工作量。