

编译专题实验报告

词法分析器

计算机2101 陈实

完成模式：独立完成

实验平台

1. 操作系统: WSL2 Ubuntu 20.04
2. 编程语言: C++
3. g++版本: 13.1.0

实验目的

1. 目的: 构建词法分析程序能将源语言程序作为输入, 并输出词法记号串到文件中。
2. 功能:
 1. 单词设计包含主文法中所有词法单位;
 2. 一遍扫描或含预处理遍, 能删除注解, 并允许空白字符串作为分隔;
 3. 分析框架scanner()允许连续调用直到输入串被扫描完毕, 每调用一次返回一个记号;
 4. 有联合DFA设计结果以及超前搜索功能;
 5. (可选)词法错误处理。

实验内容

1. include和全局变量声明

```
1  #include <algorithm> // 包含 std::transform
2  #include <cctype> // 包含 std::tolower
3  #include <fstream>
4  #include <iostream>
5  #include <map>
6  #include <stdio.h>
7  #include <string>
8  #include <unistd.h>
9  #include <vector>
10
11  using namespace std;
12  int state; // 当前状态指示
13  char C; // 当前读入字符
14  string nowstr; // 当前读入的字符串
15  char *buffer; // 文件缓冲区
16  int forwar = -1; // 向前指针
17  int rows = 1; // 文件行数
18  int sum_char = 0; // 文件总字符数
19
20  vector<string> keyword = {"auto", "break", "case", "char", "const",
    "continue", "default",
```

```

21         "do", "double", "else", "enum", "extern",
    "float", "for", "goto",
22         "if", "int", "long", "register", "return",
    "short", "signed",
23         "sizeof", "static", "struct", "switch",
    "typedef", "union",
24         "unsigned", "void", "volatile", "while",
    "define", "include"}; //关键字表
25
26 std::multimap<string,string> item; //符号表(type,value)

```

1. **state**: 当前状态指示, 用于指示当前状态
 2. **C**: 当前读入字符
 3. **nowstr**: 当前读入的字符串
 4. **buffer**: 文件缓冲区
 5. **forwar**: 向前指针
 6. **rows**: 文件行数
 7. **sum_char**: 文件总字符数
 8. **keyword**: 关键字表
 9. **item**: 符号表(type,value), 用于存储识别出的符号
2. 功能函数:

```

1 void get_char() {
2     forwar=forwar+1;
3     C = buffer[forwar];
4 }
5 //从buf中读一个字符到C中, 向前指针移动。
6
7 void cat() {
8     nowstr.push_back(C);
9 }
10 //将字符C连接到nowstr字符串后面
11
12 std::string toLower(const std::string& str) {
13     // 创建一个副本以避免修改原始字符串
14     std::string lowerStr = str;
15
16     // 使用 std::transform 和 std::tolower 将字符串转换为全小写
17     std::transform(lowerStr.begin(), lowerStr.end(), lowerStr.begin(),
18 ::tolower);
19
20     return lowerStr; // 返回转换后的字符串
21 }
22
23 std::string toUpper(const std::string& str) {
24     // 创建一个副本以避免修改原始字符串
25     std::string upperStr = str;
26
27     // 使用 std::transform 和 std::toupper 将字符串转换为全大写
28     std::transform(upperStr.begin(), upperStr.end(), upperStr.begin(),
29 ::toupper);

```

```

28
29     return upperStr; // 返回转换后的字符串
30 }
31
32 bool is_letter(char ch) {
33     if (isalpha(ch) || ch=='_')
34         return true;
35     else
36         return false;
37 }
38 //判断ch是否为字母或下划线
39
40 int iskeyword() {
41     for (int i = 0; i < keyword.size(); i++){
42         if (nowstr == keyword[i]) {
43             return 1;
44         }
45         if (toLower(nowstr) == keyword[i]) {
46             return 2;
47         }
48     }
49     return 0;
50 }
51 //判断nowstr中是否为关键字，1表示匹配正确，2表示匹配正确但大小写不同，0表示不匹配
52
53 bool iseven() {
54     int num = 0;
55     int i = nowstr.size() - 2;
56     while (nowstr[i] == '\\') {
57         num++;
58         i--;
59     }
60     if (num % 2 == 0)
61         return true;
62     return false;
63 }
64 //判断nowstr中是否为偶数个\结尾

```

1. `get_char()`：从buf中读一个字符到C中，向前指针移动。
2. `cat()`：将字符C连接到nowstr字符串后面
3. `toLower()`：将字符串转换为全小写
4. `toUpper()`：将字符串转换为全大写
5. `is_letter()`：判断ch是否为字母或下划线
6. `iskeyword()`：判断nowstr中是否为关键字，1表示匹配正确，2表示匹配正确但大小写不同，0表示不匹配
7. `iseven()`：判断nowstr中是否为偶数个\结尾
8. `toLower()`用于将字符串转换为全小写，用于判断是否是大小写错误的关键字
9. `iseven()`用于判断是否为偶数个\结尾，作用的情况是在遇到"时，如果是偶数个\结尾，则说明是字符串结束，否则说明是转义字符，需要继续读取
10. `iskeyword()`用于判断是否为关键字，如果是关键字，则返回1，如果是大小写错误的关键字，则返回2，否则返回0

3. scanner函数:

```
1 void scanner(){
2     bool isEnd = false;
3     while(!isEnd){
4         get_char();
5         if (C == '\n'){
6             rows++;
7         }
8         if (C == EOF){
9             isEnd = true;
10        }
11        switch (state) {
12            case 0: //初始状态
13                if (is_letter(C)){
14                    state = 1; //id or keyword
15                    cat();
16                }
17                else if (isdigit(C)){
18                    state = 2; //num or wrong id
19                    cat();
20                }
21                else if (C=='0'){
22                    state=28;
23                    cat();
24                }
25                else{
26                    switch (C) {
27                        case '<': state = 8; break;
28                        case '>': state = 9; break;
29                        case '?':
30                            insert_item("OPERATOR", "?");
31                            break;
32                        case ':':
33                            insert_item("OPERATOR", ":");
34                            break;
35                        case '/': state = 11; break;
36                        case '=': state = 12; break;
37                        case '+': state = 13; break;
38                        case '-': state = 14; break;
39                        case '*': state = 15; break;
40                        case '%': state = 16; break;
41                        case '(':
42                            insert_item("DELIMITER", "(");
43                            break;
44                        case ')':
45                            insert_item("DELIMITER", ")");
46                            break;
47                        case ',':
48                            insert_item("DELIMITER", ",");
49                            break;
50                        case ';':
51                            insert_item("DELIMITER", ";");
52                            break;
```

```

53         case '{':
54             insert_item("DELIMITER", "{");
55             break;
56         case '}':
57             insert_item("DELIMITER", "}");
58             break;
59         case '[':
60             insert_item("DELIMITER", "[");
61             break;
62         case ']':
63             insert_item("DELIMITER", "]");
64             break;
65         case '^': state = 17; break;
66         case '|': state = 18; break;
67         case '~': state = 19; break;
68         case '!': state = 20; break;
69         case '&': state = 21; break;
70         case '"': state = 22; cat(); break;
71         case '\\': state = 23; cat(); break;
72         case '.': state = 24; break;
73         case '#': insert_item("OPERATOR", "#"); break;
74         case ' ':
75             case '\\n':
76             case ' ':
77             case EOF : break; //跳过空白符
78             default: cout << "error:第" << rows << "行出现非
法字符" << C << endl;
79                 break;
80         }
81     }
82     break;
83     case 1: //id or keyword
84         if (is_letter(C) || isdigit(C)) {
85             cat();
86             state = 1;
87         }
88         else {
89             forwar--;
90             state = 0;
91             if (C == '\\n') {
92                 rows--;
93             }
94             int flag = iskeyword();
95             if (flag == 1) {
96                 insert_item("keyword", nowstr);
97             } else if (flag == 2) {
98                 insert_item("ERROR", nowstr+" --- Keyword
should be lower case");
99             } else {
100                 insert_item("id", nowstr);
101             }
102             nowstr.clear();
103         }
104         break;

```

```

105         case 2:
106             if (isdigit(C)) {
107                 cat();
108                 state = 2;
109             }
110             else if (C == '.') {
111                 cat();
112                 state = 3;
113             }
114             else if (C == 'E' || C == 'e') {
115                 cat();
116                 state = 5;
117             }
118             //wrong id
119             else if (is_letter(C)) {
120                 cat();
121                 state = 31;
122             }
123             else {
124                 forwar--;
125                 state = 0;
126                 if (C == '\n') {
127                     rows--;
128                 }
129                 insert_item("int", nowstr);
130                 nowstr.clear();
131             }
132             break;
133         case 3: // .
134             if (isdigit(C)) {
135                 cat();
136                 state = 4;
137             }
138             else {
139                 forwar--;
140                 state = 0;
141                 if (C == '\n') {
142                     rows--;
143                 }
144                 nowstr.push_back('0');
145                 insert_item("float", nowstr);
146                 nowstr.clear();
147             }
148             break;
149
150         case 4: // .num
151             if (isdigit(C)) {
152                 cat();
153                 state = 4;
154             }
155             else if (C == 'E' || C == 'e') {
156                 cat();
157                 state = 5;
158             }

```

```

159         else {
160             forwar--;
161             state = 0;
162             if (C == '\n') {
163                 rows--;
164             }
165             insert_item("float", nowstr);
166             nowstr.clear();
167         }
168         break;
169
170     case 5:        // .numE
171         if (C == '+' || C == '-') {
172             cat();
173             state = 6;
174         }
175         else if (isdigit(C)) {
176             cat();
177             state = 7;
178         }
179         else {
180             forwar--;
181             state = 0;
182             if (C == '\n') {
183                 rows--;
184             }
185             insert_item("ERROR", nowstr + " --- Missing
digit");
            nowstr.clear();
186         }
187         break;
188
189     case 6:        // .numE+ or .numE-
190         if (isdigit(C)) {
191             cat();
192             state = 7;
193         }
194         else {
195             forwar--;
196             state = 0;
197             if (C == '\n') {
198                 rows--;
199             }
200             insert_item("ERROR", nowstr + " --- Missing
digit");
            nowstr.clear();
201         }
202         break;
203
204     case 7:        // .numE+num
205         if (isdigit(C)) {
206             cat();
207             state = 7;
208         }
209     }
210

```

```

211         else {
212             forwar--;
213             state = 0;
214             if (C == '\n') {
215                 rows--;
216             }
217             insert_item("float", nowstr);
218             nowstr.clear();
219         }
220         break;
221
222     case 8:
223         if (C == '=') {
224             insert_item("OPERATOR", "≤");
225         }
226         else if (C == '<') {
227             insert_item("OPERATOR", "<");
228         }
229         else {
230             forwar--;
231             insert_item("OPERATOR", "<");
232             if (C == '\n') {
233                 rows--;
234             }
235         }
236         state = 0;
237         break;
238
239     case 9:
240         if (C == '=') {
241             insert_item("OPERATOR", "≥");
242         }
243         else if (C == '>') {
244             insert_item("OPERATOR", ">");
245         }
246         else {
247             forwar--;
248             insert_item("OPERATOR", ">");
249             if (C == '\n') {
250                 rows--;
251             }
252         }
253         state = 0;
254         break;
255
256     case 11:
257         switch (C) {
258             case '/': // 单行注释
259                 state = 27;
260                 break;
261             case '*': // 多行注释
262                 state = 26;
263                 break;
264             default:

```



```

265         forwar--;
266         insert_item("OPERATOR", "/");
267         state = 0;
268         if (C == '\n') {
269             rows--;
270         }
271         break;
272     }
273     break;
274
275     case 12: // =
276         if (C == '=') {
277             insert_item("OPERATOR", "=");
278         }
279         else {
280             forwar--;
281             insert_item("OPERATOR", "=");
282             if (C == '\n') {
283                 rows--;
284             }
285         }
286         state = 0;
287         break;
288
289     case 13: // +
290         if (C == '+') {
291             insert_item("OPERATOR", "+");
292         }
293         else if (C == '=') {
294             insert_item("OPERATOR", "+=");
295         }
296         else if (isdigit(C)) {
297             cat();
298             state = 2;
299         }
300         else {
301             forwar--;
302             insert_item("OPERATOR", "+");
303             if (C == '\n') {
304                 rows--;
305             }
306         }
307         state = 0;
308         break;
309
310     case 14: // -
311         if (C == '-') {
312             insert_item("OPERATOR", "--");
313         }
314         else if (C == '=') {
315             insert_item("OPERATOR", "-=");
316         }
317         else if (C == '>') {
318             insert_item("OPERATOR", "→");

```

```

319     }
320     else if (isdigit(C)) {
321         cat();
322         state = 2;
323     }
324     else {
325         forwar--;
326         insert_item("OPERATOR", "-");
327         if (C == '\n') {
328             rows--;
329         }
330     }
331     state = 0;
332     break;
333
334 case 15:    // *
335     if (C == '=') {
336         insert_item("OPERATOR", "*=");
337     }
338     else {
339         forwar--;
340         insert_item("OPERATOR", "*");
341         if (C == '\n') {
342             rows--;
343         }
344     }
345     state = 0;
346     break;
347
348 case 16:    // %
349     if (C == '=') {
350         insert_item("OPERATOR", "%=");
351     }
352     else {
353         forwar--;
354         insert_item("OPERATOR", "%");
355         if (C == '\n') {
356             rows--;
357         }
358     }
359     state = 0;
360     break;
361
362 case 17:    // ^
363     if (C == '=') {
364         insert_item("OPERATOR", "^=");
365     }
366     else {
367         forwar--;
368         insert_item("OPERATOR", "^");
369         if (C == '\n') {
370             rows--;
371         }
372     }

```

```

373         state = 0;
374         break;
375
376     case 18:        // |
377         if (C == '|') {
378             insert_item("OPERATOR", "|");
379         }
380         else if (C == '=') {
381             insert_item("OPERATOR", "≡");
382         }
383         else {
384             forwar--;
385             insert_item("OPERATOR", "|");
386             if (C == '\n') {
387                 rows--;
388             }
389         }
390         state = 0;
391         break;
392
393     case 19:        // ~
394         if (C == '=') {
395             insert_item("OPERATOR", "≈");
396         }
397         else {
398             forwar--;
399
400             insert_item("OPERATOR", "~");
401             if (C == '\n') {
402                 rows--;
403             }
404         }
405         state = 0;
406         break;
407
408     case 20:        // !
409         if (C == '=') {
410             insert_item("OPERATOR", "≠");
411         }
412         else {
413             forwar--;
414
415             insert_item("OPERATOR", "!");
416             if (C == '\n') {
417                 rows--;
418             }
419         }
420         state = 0;
421         break;
422
423     case 21:        // &
424         if (C == '&') {
425             insert_item("OPERATOR", "&&");
426         }

```

```

427         else if (C == '=') {
428             insert_item("OPERATOR", "&=");
429         }
430         else {
431             forwar--;
432
433             insert_item("OPERATOR", "&");
434             if (C == '\n') {
435                 rows--;
436             }
437         }
438         state = 0;
439         break;
440
441     case 22: // "
442         if (C == '\\') {
443             cat();
444             if (iseven()) {
445                 insert_item("STRING", nowstr);
446                 nowstr.clear();
447                 state = 0;
448             }
449             else {
450                 state = 22;
451             }
452         }
453         else if (C == EOF) {
454             insert_item("ERROR", nowstr + " --- String should
end with \\");
455             nowstr.clear();
456             state = 0;
457         }
458         else {
459             cat();
460             state = 22;
461         }
462         break;
463
464     case 23: // '
465         if (C == '\\') {
466             cat();
467             //判断是否是转义字符
468             if (nowstr.size() == 4 && nowstr[1] == '\\') {
469                 insert_item("CHAR", nowstr);
470             }
471             //单独处理\\'这种情况
472             else if (nowstr[0]=='\\' && nowstr[1]=='\\' &&
nowstr[2]=='\\') {
473                 get_char();
474                 if (C == '\\') {
475                     cat();
476                     insert_item("CHAR", nowstr);
477                 }
478                 else {

```

```

479         forwar--;
480
481     }
482 }
483 else if (nowstr.size() == 3) {
484     insert_item("CHAR", nowstr);
485 }
486 else {
487     insert_item("ERROR", nowstr + " --- Char
should be one character");
488 }
489 nowstr.clear();
490 state = 0;
491 }
492 else if (C == EOF) {
493     insert_item("ERROR", nowstr + " --- Char should
end with '\"");
494     nowstr.clear();
495     state = 0;
496 }
497 else {
498     cat();
499     state = 23;
500 }
501 break;
502
503 case 24:    // .
504     if (isdigit(C)) {
505         cat();
506         state = 4;
507     }
508     else {
509         forwar--;
510
511         insert_item("OPERATOR", ".");
512         if (C == '\n') {
513             rows--;
514         }
515         state = 0;
516     }
517     break;
518
519 case 26:    // /*
520     if (C == '*') {
521         state = 25;
522     }
523     break;
524
525 case 25:    // /*...*
526     if (C == '*') {
527         state = 25;
528     }
529     else if (C == '/') {
530         state = 0;

```

```

531         }
532         else if (C == EOF) {
533             insert_item("ERROR", nowstr + " --- Multi-line
comment should end with */");
534             nowstr.clear();
535             state = 0;
536         }
537         break;
538
539     case 27:    // //
540         if (C == '\n' || C == EOF) {
541             state = 0;
542         }
543         break;
544
545     case 28:    // 0
546         if (C == 'x' || C == 'X') {
547             cat();
548             state = 29;
549         }
550         else if (isdigit(C)) {
551             cat();
552             state = 2;
553         }
554         else if (C == '.') {
555             cat();
556             state = 3;
557         }
558         else {
559             forwar--;
560
561             insert_item("int", nowstr);
562             nowstr.clear();
563             state = 0;
564         }
565         break;
566
567     case 29:    // 0x
568         if (isdigit(C) || (C >= 'a' && C <= 'f') || (C >= 'A'
&& C <= 'F')) {
569             cat();
570             state = 30;
571         }
572         else {
573             forwar--;
574
575             insert_item("ERROR", nowstr + " --- Hexadecimal
number should have at least one digit");
576             nowstr.clear();
577             state = 0;
578         }
579         break;
580
581     case 30:    // 0x...

```

```

582         if (isdigit(C) || (C ≥ 'a' && C ≤ 'f') || (C ≥ 'A'
&& C ≤ 'F')) {
583             cat();
584             state = 30;
585         }
586         else {
587             forwar--;
588
589             insert_item("int", nowstr);
590             nowstr.clear();
591             state = 0;
592         }
593         break;
594
595     case 31:    //wrong id with num at first
596         if (is_letter(C) || isdigit(C)) {
597             cat();
598             state = 31;
599         }
600         else {
601             forwar--;
602
603             insert_item("ERROR", nowstr + " --- ID should
start with a letter");
604             nowstr.clear();
605             state = 0;
606         }
607         break;
608     default:
609         break;
610 }
611 }
612 }

```

1. `scanner()` 函数用于识别输入的字符，根据状态机的状态进行识别

2. `scanner`识别的类型如下：

1. `id`：标识符
2. `keyword`：关键字
3. `int`：整数
4. `float`：浮点数
5. `STRING`：字符串
6. `CHAR`：字符
7. `OPERATOR`：操作符
8. `DELIMITER`：分隔符
9. `ERROR`：错误

3. `scanner()`函数中的状态机设计如下：

1. `state=0`：初始状态
2. `state=1`：id or keyword
3. `state=2`：num or wrong id
4. `state=3`：.

```

5. state=4: .num
6. state=5: .numE
7. state=6: .numE+ or .numE-
8. state=7: .numE+num
9. state=8: <
10. state=9: >
11. state=11: /
12. state=12: =
13. state=13: +
14. state=14: -
15. state=15: *
16. state=16: %
17. state=17: ^
18. state=18: |
19. state=19: ~
20. state=20: !
21. state=21: &
22. state=22: "
23. state=23: '
24. state=24: .
25. state=26: /*
26. state=27: //
27. state=28: 0
28. state=29: 0x
29. state=30: 0x...
30. state=31: wrong id with num at first

```

4. insert_item和read_file函数:

```

1 //根据传入的type和value, 将其插入符号表
2 void insert_item(string type, string value) {
3     item.insert(pair<string,string>(type,value));
4     sleep(0.5);
5     if (type == "ERROR") {
6         cout << "\033[31m" // 31对应红色
7             << rows << ": (" << type << ", " << value << ")"
8             << "\033[0m" << endl;
9     }
10    else {
11        cout << rows << ": (" << type << ", " << value << ")" << endl;
12    }
13 }
14
15 //根据文件名, 读取文件内容到buffer中
16 void read_file(const char *filename) {
17     FILE *fp = fopen(filename, "r"); // 用二进制模式打开
18     if (fp == NULL) {

```



```

19     cout << "文件打开失败" << endl;
20     exit(0);
21 }
22 fseek(fp, 0, SEEK_END);
23 sum_char = ftell(fp);
24 fseek(fp, 0, SEEK_SET);
25 buffer = new char[sum_char+1];
26 fread(buffer, 1, sum_char, fp);
27 fclose(fp);
28 //Buffer末尾加上EOF
29 buffer[sum_char] = EOF;
30 }

```

1. `insert_item()`: 根据传入的type和value, 将其插入符号表, 并输出, 如果是错误, 则输出为红色
2. `read_file()`: 根据文件名, 读取文件内容到buffer中

5. 主函数:

```

1 int main(int argc, char **argv) {
2     if (argc < 3) { // 检查是否提供了足够的命令行参数
3         std::cerr << "请提供输入和输出文件名作为参数。" << std::endl;
4         return 1;
5     }
6
7     const char *input_filename = argv[1]; // 获取输入文件名
8     const char *output_filename = argv[2]; // 获取输出文件名
9
10    freopen(output_filename, "w", stdout); // 将输出重定向到指定的输出文件
11    read_file(input_filename); // 读取指定输入文件
12    scanner(); // 扫描内容
13    delete[] buffer; // 释放分配的内存
14
15    return 0;
16 }

```

1. 主函数用于读取输入文件, 将输出重定向到指定的输出文件, 然后调用scanner()函数进行扫描
2. 主函数从命令行参数中获取输入文件名和输出文件名

实验结果

1. test1.txt

```

1 #include <stdio.h> // 包含标准输入输出头文件
2 #include <string.h> // 包含字符串操作头文件
3
4 // 定义最大字符串长度
5 #define MAX_LENGTH 100
6
7 // 函数声明
8 void print_greeting(const char *name);
9
10 // 主函数, 程序从这里开始执行
11 int main() {
12     // 定义一个字符数组以存储输入的名字

```

```
13     char name[MAX_LENGTH];
14
15     // 打印提示信息
16     printf("请输入你的名字: ");
17
18     // 从标准输入获取名字, 最多读取 MAX_LENGTH 个字符
19     fgets(name, MAX_LENGTH, stdin);
20
21     // 删除换行符
22     size_t len = strlen(name);
23     if (name[len - 1] == '\n') {
24         name[len - 1] = '\0';
25     }
26
27     // 打印问候语
28     print_greeting(name);
29
30     // 程序结束
31     return 0;
32 }
33
34 // 定义一个函数, 用于打印问候语
35 void print_greeting(const char *name) {
36     // 打印问候语, 使用传入的名字
37     printf("你好, %s! 欢迎使用这个简单的 C 程序.\n", name);
38 }
```

该程序词法正确，输出如下：

```
2: (OPERATOR, >)
5: (OPERATOR, #)
5: (keyword, define)
5: (id, MAX_LENGTH)
5: (int, 100)
8: (keyword, void)
8: (id, print_greeting)
8: (DELIMITER, ())
8: (keyword, const)
8: (keyword, char)
8: (OPERATOR, *)
8: (id, name)
8: (DELIMITER, ))
8: (DELIMITER, ;)
11: (keyword, int)
11: (id, main)
11: (DELIMITER, ())
11: (DELIMITER, ))
11: (DELIMITER, {)
13: (keyword, char)
13: (id, name)
13: (DELIMITER, [])
13: (id, MAX_LENGTH)
13: (DELIMITER, ])
13: (DELIMITER, ;)
16: (id, printf)
16: (DELIMITER, ())
16: (STRING, "请输入你的名字：")
16: (DELIMITER, ))
16: (DELIMITER, ;)
19: (id, fgets)
19: (DELIMITER, ())
19: (id, name)
19: (DELIMITER, ,)
19: (id, MAX_LENGTH)
19: (DELIMITER, ,)
19: (id, stdin)
19: (DELIMITER, ))
19: (DELIMITER, ;)
22: (id, size_t)
22: (id, len)
22: (OPERATOR, =)
22: (id, strlen)
22: (DELIMITER, ())
22: (id, name)
22: (DELIMITER, ))
22: (DELIMITER, ;)
23: (keyword, if)
23: (DELIMITER, ())
23: (id, name)
23: (DELIMITER, [)
```

```

23: (id, name)
23: (DELIMITER, [])
23: (id, len)
23: (OPERATOR, -)
23: (int, 1)
23: (DELIMITER, ])
23: (OPERATOR, ==)
23: (CHAR, '\n')
23: (DELIMITER, ))
23: (DELIMITER, {)
24: (id, name)
24: (DELIMITER, [])
24: (id, len)
24: (OPERATOR, -)
24: (int, 1)
24: (DELIMITER, ])
24: (OPERATOR, =)
24: (CHAR, '\0')
24: (DELIMITER, ;)
25: (DELIMITER, })
28: (id, print_greeting)
28: (DELIMITER, ())
28: (id, name)
28: (DELIMITER, ))
28: (DELIMITER, ;)
31: (keyword, return)
31: (int, 0)
31: (DELIMITER, ;)
32: (DELIMITER, })
35: (keyword, void)
35: (id, print_greeting)
35: (DELIMITER, ())
35: (keyword, const)
35: (keyword, char)
35: (OPERATOR, *)
35: (id, name)
35: (DELIMITER, ))
35: (DELIMITER, {)
37: (id, printf)
37: (DELIMITER, ())
37: (STRING, "你好, %s! 欢迎使用这个简单的 C 程序。 \n")
37: (DELIMITER, ,)
37: (id, name)
37: (DELIMITER, ))
37: (DELIMITER, ;)
38: (DELIMITER, })

```

计算机2101-陈实-编译实验2-独立完成

完成时间: 2024.04.29

~/flex1 base at 20:47:30
date
Mon Apr 29 20:47:48 CST 2024

```

1 1: (OPERATOR, #)
2 1: (keyword, include)
3 1: (OPERATOR, <)
4 1: (id, stdio)
5 1: (OPERATOR, .)
6 1: (id, h)
7 1: (OPERATOR, >)
8 2: (OPERATOR, #)
9 2: (keyword, include)
10 2: (OPERATOR, <)
11 2: (id, string)
12 2: (OPERATOR, .)
13 2: (id, h)
14 2: (OPERATOR, >)
15 5: (OPERATOR, #)
16 5: (keyword, define)
17 5: (id, MAX_LENGTH)
18 5: (int, 100)
19 8: (keyword, void)
20 8: (id, print_greeting)
21 8: (DELIMITER, ())
22 8: (keyword, const)
23 8: (keyword, char)
24 8: (OPERATOR, *)
25 8: (id, name)
26 8: (DELIMITER, ))

```

```
27 8: (DELIMITER, ;)
28 11: (keyword, int)
29 11: (id, main)
30 11: (DELIMITER, ()
31 11: (DELIMITER, ))
32 11: (DELIMITER, {})
33 13: (keyword, char)
34 13: (id, name)
35 13: (DELIMITER, [])
36 13: (id, MAX_LENGTH)
37 13: (DELIMITER, ])
38 13: (DELIMITER, ;)
39 16: (id, printf)
40 16: (DELIMITER, ()
41 16: (STRING, "请输入你的名字: ")
42 16: (DELIMITER, ))
43 16: (DELIMITER, ;)
44 19: (id, fgets)
45 19: (DELIMITER, ()
46 19: (id, name)
47 19: (DELIMITER, ,)
48 19: (id, MAX_LENGTH)
49 19: (DELIMITER, ,)
50 19: (id, stdin)
51 19: (DELIMITER, ))
52 19: (DELIMITER, ;)
53 22: (id, size_t)
54 22: (id, len)
55 22: (OPERATOR, =)
56 22: (id, strlen)
57 22: (DELIMITER, ()
58 22: (id, name)
59 22: (DELIMITER, ))
60 22: (DELIMITER, ;)
61 23: (keyword, if)
62 23: (DELIMITER, ()
63 23: (id, name)
64 23: (DELIMITER, [])
65 23: (id, len)
66 23: (OPERATOR, -)
67 23: (int, 1)
68 23: (DELIMITER, ])
69 23: (OPERATOR, ==)
70 23: (CHAR, '\n')
71 23: (DELIMITER, ))
72 23: (DELIMITER, {})
73 24: (id, name)
74 24: (DELIMITER, [])
75 24: (id, len)
76 24: (OPERATOR, -)
77 24: (int, 1)
78 24: (DELIMITER, ])
79 24: (OPERATOR, =)
80 24: (CHAR, '\0')
```

```

81 24: (DELIMITER, ;)
82 25: (DELIMITER, })
83 28: (id, print_greeting)
84 28: (DELIMITER, ()
85 28: (id, name)
86 28: (DELIMITER, ))
87 28: (DELIMITER, ;)
88 31: (keyword, return)
89 31: (int, 0)
90 31: (DELIMITER, ;)
91 32: (DELIMITER, })
92 35: (keyword, void)
93 35: (id, print_greeting)
94 35: (DELIMITER, ()
95 35: (keyword, const)
96 35: (keyword, char)
97 35: (OPERATOR, *)
98 35: (id, name)
99 35: (DELIMITER, ))
100 35: (DELIMITER, {)
101 37: (id, printf)
102 37: (DELIMITER, ()
103 37: (STRING, "你好, %s! 欢迎使用这个简单的 C 程序。\\n")
104 37: (DELIMITER, ,)
105 37: (id, name)
106 37: (DELIMITER, ))
107 37: (DELIMITER, ;)
108 38: (DELIMITER, })

```

可以看到，识别出了各种类型的符号，包括关键字、标识符、操作符、分隔符、字符串、字符等，能够正确识别出各种类型的符号

2. test2.txt

下面修改了test1.txt中的一些内容，使得其出现了一些错误，如下：

```

1 Char 1name[MAX_LENGTH];

```

预期能识别画出Keyword大小写错误和ID首字母错误。

实际输出如下：

```

11: (DELIMITER, ))
11: (DELIMITER, {)
13: (ERROR, Char --- Keyword should be lower case)
13: (ERROR, 1name --- ID should start with a letter)
13: (DELIMITER, [)
13: (id, MAX_LENGTH)
13: (DELIMITER, ])
13: (DELIMITER, ;)

```

可以看到，识别出了大小写错误的关键字和ID首字母错误，并且输出为红色

```

1 1: (OPERATOR, #)
2 1: (keyword, include)
3 1: (OPERATOR, <)
4 1: (id, stdio)
5 1: (OPERATOR, .)
6 1: (id, h)
7 1: (OPERATOR, >)

```

```
8 2: (OPERATOR, #)
9 2: (keyword, include)
10 2: (OPERATOR, <)
11 2: (id, string)
12 2: (OPERATOR, .)
13 2: (id, h)
14 2: (OPERATOR, >)
15 5: (OPERATOR, #)
16 5: (keyword, define)
17 5: (id, MAX_LENGTH)
18 5: (int, 100)
19 8: (keyword, void)
20 8: (id, print_greeting)
21 8: (DELIMITER, ())
22 8: (keyword, const)
23 8: (keyword, char)
24 8: (OPERATOR, *)
25 8: (id, name)
26 8: (DELIMITER, ))
27 8: (DELIMITER, ;)
28 11: (keyword, int)
29 11: (id, main)
30 11: (DELIMITER, ())
31 11: (DELIMITER, ))
32 11: (DELIMITER, {})
33 13: (ERROR, Char --- Keyword should be lower case)
34 13: (ERROR, lname --- ID should start with a letter)
35 13: (DELIMITER, [])
36 13: (id, MAX_LENGTH)
37 13: (DELIMITER, ])
38 13: (DELIMITER, ;)
39 16: (id, printf)
40 16: (DELIMITER, ())
41 16: (STRING, "请输入你的名字: ")
42 16: (DELIMITER, ))
43 16: (DELIMITER, ;)
44 19: (id, fgets)
45 19: (DELIMITER, ())
46 19: (id, name)
47 19: (DELIMITER, ,)
48 19: (id, MAX_LENGTH)
49 19: (DELIMITER, ,)
50 19: (id, stdin)
51 19: (DELIMITER, ))
52 19: (DELIMITER, ;)
53 22: (id, size_t)
54 22: (id, len)
55 22: (OPERATOR, =)
56 22: (id, strlen)
57 22: (DELIMITER, ())
58 22: (id, name)
59 22: (DELIMITER, ))
60 22: (DELIMITER, ;)
61 23: (keyword, if)
```

```
62 23: (DELIMITER, ()
63 23: (id, name)
64 23: (DELIMITER, [])
65 23: (id, len)
66 23: (OPERATOR, -)
67 23: (int, 1)
68 23: (DELIMITER, ])
69 23: (OPERATOR, ==)
70 23: (CHAR, '\n')
71 23: (DELIMITER, ))
72 23: (DELIMITER, {})
73 24: (id, name)
74 24: (DELIMITER, [])
75 24: (id, len)
76 24: (OPERATOR, -)
77 24: (int, 1)
78 24: (DELIMITER, ])
79 24: (OPERATOR, =)
80 24: (CHAR, '\0')
81 24: (DELIMITER, ;)
82 25: (DELIMITER, })
83 28: (id, print_greeting)
84 28: (DELIMITER, ()
85 28: (id, name)
86 28: (DELIMITER, ))
87 28: (DELIMITER, ;)
88 31: (keyword, return)
89 31: (int, 0)
90 31: (DELIMITER, ;)
91 32: (DELIMITER, })
92 35: (keyword, void)
93 35: (id, print_greeting)
94 35: (DELIMITER, ()
95 35: (keyword, const)
96 35: (keyword, char)
97 35: (OPERATOR, *)
98 35: (id, name)
99 35: (DELIMITER, ))
100 35: (DELIMITER, {})
101 37: (id, printf)
102 37: (DELIMITER, ()
103 37: (STRING, "你好, %s! 欢迎使用这个简单的 C 程序。 \n")
104 37: (DELIMITER, ,)
105 37: (id, name)
106 37: (DELIMITER, ))
107 37: (DELIMITER, ;)
108 38: (DELIMITER, })
```


实验总结

1. 通过本次实验，我学会了如何设计一个简单的词法分析器，能使用C++语言识别出各种类型的符号，包括关键字、标识符、操作符、分隔符、字符串、字符等
2. 通过本次实验，我学会了如何使用状态机设计词法分析器，通过状态机的状态转移来识别各种类型的符号
3. 通过本次实验，我学会了如何使用C++语言读取文件内容，并将输出重定向到指定的输出文件
4. 通过本次实验，我学会了如何实现超前搜索，用于识别++、--等符号
5. 通过本次实验，我学会了如何修改输出颜色，使得错误输出为红色