

编译专题实验报告

语义分析

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: 构建语法制导的语义分析程序能在语法分析的同时生成符号表和中间语言代码, 并输出结果到文件中。
2. 功能:
SLR(1)制导的语义分析框架实现;
中间语言代码形式: 四元式;

实验内容

1. cifa.cpp: 词法分析程序, 生成token序列;

```
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  vector<string> keyword = {"auto", "break", "case", "char", "const",
20                           "continue", "default",
21                           "do", "double", "else", "enum", "extern",
22                           "float", "for", "goto",
23                           "if", "int", "long", "register", "return",
24                           "short", "signed",
```

```

22         "sizeof", "static", "struct", "switch",
    "typedef", "union",
23         "unsigned", "void", "volatile", "while",
    "define", "include"};    //关键字表

```

完整代码见附录1. cifa.cpp。

cifa.cpp由第二次实验修改而来，主要功能是词法分析，生成token序列。从文件 **input.txt** 中读取内容，识别出关键字、标识符、常数、运算符、界符等，并输出到文件 **outputofcifa.txt** 中。作为后面语法分析的输入。

输入：

```

1  s=a+b+c+(a*a )

```

输出：

```

1  (id, s)(OPERATOR, =)(id, a)(OPERATOR, +)(id, b)(OPERATOR, +)(id, c)
    (OPERATOR, +)(DELIMITER, )(id, a)(OPERATOR, *)(id, a)(DELIMITER, ))

```

2. slr.cpp:

```

1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #include<iostream>
5  #define MAX_LEN 1000
6  using namespace std;
7  struct stack {
8      char s[MAX_LEN];
9      int i[MAX_LEN];
10     int point[MAX_LEN];
11     int top;
12 }; // 分析栈数据结构
13
14 struct quadruple {
15     char op[MAX_LEN];
16     char arg1[MAX_LEN];
17     char arg2[MAX_LEN];
18     char result[MAX_LEN];
19 }; // 四元式数据结构
20
21 struct quadruple quad[MAX_LEN]; // 存储四元式
22 int quadTop = 0; // 四元式栈顶
23
24 // 1.S→V=E  2.E→E+T  3.E→E-T  4.E→T  5.T→T*F  6.T→T/F  7.T→F  8.F→(E)
25 // 9.F→i  10.V→i
26 // 表中大于0对应移进，小于0则对应先归约后移进，0为不存在的状态
27 //          GOTO          |          ACTION
28 //i, =, +, -, *, /, (, ), #, S, E, T, F, V
29 int table[20][14] = {{ 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2}, // 0
30                      { 0, 0, 0, 0, 0, 0, 0, 0, 0, -11, 0, 0, 0, 0}, // 1
31                      { 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 2
32                      {-10, -10, -10, -10, -10, -10, -10, -10, -10, 0, 0, 0, 0, 0}, // 3
33                      { 9, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 5, 6, 7, 0}, // 4

```

```

33         {-1,-1,10,11,-1,-1,-1,-1,-1, 0, 0, 0, 0, 0},// 5
34         {-4,-4,-4,-4,12,13,-4,-4,-4, 0, 0, 0, 0, 0},// 6
35         {-7,-7,-7,-7,-7,-7,-7,-7,-7, 0, 0, 0, 0, 0},// 7
36         { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0,14, 6, 7, 0},// 8
37         {-9,-9,-9,-9,-9,-9,-9,-9,-9, 0, 0, 0, 0, 0},// 9
38         { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0,15, 7, 0},//10
39         { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0,16, 7, 0},//11
40         { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0,17, 0},//12
41         { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0,18, 0},//13
42         { 0, 0,10,11, 0, 0, 0,19, 0, 0, 0, 0, 0, 0},//14
43         {-2,-2,-2,-2,12,13,-2,-2,-2, 0, 0, 0, 0, 0},//15
44         {-3,-3,-3,-3,12,13,-3,-3,-3, 0, 0, 0, 0, 0},//16
45         {-5,-5,-5,-5,-5,-5,-5,-5,-5, 0, 0, 0, 0, 0},//17
46         {-6,-6,-6,-6,-6,-6,-6,-6,-6, 0, 0, 0, 0, 0},//18
47         {-8,-8,-8,-8,-8,-8,-8,-8,-8, 0, 0, 0, 0, 0}};//19
48
49 int getindex(char ch) {
50     switch(ch) {
51         case 'i': return 0;
52         case '=': return 1;
53         case '+': return 2;
54         case '-': return 3;
55         case '*': return 4;
56         case '/': return 5;
57         case '(': return 6;
58         case ')': return 7;
59         case '#': return 8;
60         case 'S': return 9;
61         case 'E': return 10;
62         case 'T': return 11;
63         case 'F': return 12;
64         case 'V': return 13;
65         default: return -1;
66     }
67 }
68
69 void printSLR(char *str, struct stack *stk, int now) { // 打印分析状态
70     for(int i = 0; i ≤ stk→top; i++) {
71         printf("%c:%2d  ", stk→s[i], stk→i[i]); // 栈状态
72     }
73     for(int i = 0; i ≤ 60 - stk→top*7; i++) {
74         printf(" ");
75     }
76     for(int i = now; i < strlen(str); i++) {
77         printf("%c", str[i]); // 串状态
78     }
79     printf("\n");
80 }
81
82 void printQuad() { // 打印四元式
83     printf("Quadruples:\n");
84     for(int i = 1; i ≤ quadTop; i++) {
85         printf("(%s, %s, %s, %s)\n", quad[i].op, quad[i].arg1,
quad[i].arg2, quad[i].result);

```

```

86     }
87 }
88
89 int SLR(char *str, struct stack *stk) { // SLR1分析函数
90     quadTop = 0;
91     int i = 0;
92     int next;
93     printf("stack:
          str:                operate:\n");
94     while(i < strlen(str)) {
95         if(stk->top < 0) return 0; // 分析栈不可能为空
96         int y; // 列坐标
97         if (str[i] ≥ 'a' && str[i] ≤ 'z') y = getindex('i'); // 终结
符i
98         else y = getindex(str[i]);
99         if(y == -1 || table[stk->i[stk->top]][y] == 0) { // 表中不存在的
状态, 分析报错
100             return 0;
101         }
102         if(table[stk->i[stk->top]][y] > 0) { // 移进操作
103             next = table[stk->i[stk->top]][y];
104             stk->top++;
105             stk->s[stk->top] = str[i];
106             stk->i[stk->top] = next;
107             stk->point[stk->top] = i;
108             i++;
109             printSLR(str, stk, i);
110         }
111         else if(table[stk->i[stk->top]][y] < 0) { // 归约操作
112             int tmp = -table[stk->i[stk->top]][y]; // 查GOTO表
113             if(tmp == 4 || tmp == 7 || tmp == 9 || tmp == 10) {
114                 stk->top--; // 要归约1位
115             }
116             else if(tmp == 2 || tmp == 3 || tmp == 5 || tmp == 6){
117                 // 生成四元式
118                 quadTop++;
119                 if(tmp == 2) strcpy(quad[quadTop].op, "+");
120                 else if(tmp == 3) strcpy(quad[quadTop].op, "-");
121                 else if(tmp == 5) strcpy(quad[quadTop].op, "*");
122                 else strcpy(quad[quadTop].op, "/");
123                 if(stk->point[stk->top - 2] < 0)
sprintf(quad[quadTop].arg1, "t%d", -stk->point[stk->top - 2]);
124                 else {
125                     char arg1[2] = {str[stk->point[stk->top - 2]],
'\0'};
126                     strcpy(quad[quadTop].arg1, arg1);
127                 }
128                 if(stk->point[stk->top] < 0)
sprintf(quad[quadTop].arg2, "t%d", -stk->point[stk->top]);
129                 else {
130                     char arg2[2] = {str[stk->point[stk->top]], '\0'};
131                     strcpy(quad[quadTop].arg2, arg2);
132                 }
133                 for(int j = 0; j < 90; j++) printf(" ");

```

```

134         printf("t%d = %s %s %s\n", quadTop,
quad[quadTop].arg1, quad[quadTop].op, quad[quadTop].arg2); // 打印语义动
作
135         sprintf(quad[quadTop].result, "t%d", quadTop);
136         stk->top -= 3; // 归约3位
137         stk->point[stk->top + 1] = -quadTop; // 记录归约产生的中
间变量
138     }
139     else if(tmp == 8) {
140         stk->top -= 3; // 归约3位
141         stk->point[stk->top + 1] = stk->point[stk->top + 2];
// 消除括号规约
142     }
143     else if(tmp == 1){
144         quadTop++;
145         strcpy(quad[quadTop].op, "=");
146         if(stk->point[stk->top] < 0)
sprintf(quad[quadTop].arg1, "t%d", abs(stk->point[stk->top]));
147         else {
148             char arg1[2] = {str[stk->point[stk->top]], '\0'};
149             strcpy(quad[quadTop].arg1, arg1);
150         }
151         sprintf(quad[quadTop].arg2, " ");
152         char res[2] = {str[stk->point[stk->top - 2]], '\0'};
153         strcpy(quad[quadTop].result, res);
154         for(int i = 0; i < 90; i++) printf(" ");
155         printf("%s = %s\n", quad[quadTop].result,
quad[quadTop].arg1);
156         stk->top -= 3; // 归约V=E
157     }
158     else stk->top -= 3;
159     if(tmp == 1) {
160         y = getindex('S');
161         next = table[stk->i[stk->top]][y]; // 查ACTION表
162         stk->top++;
163         stk->s[stk->top] = 'S';
164         stk->i[stk->top] = next; // 归约要修改栈顶
165     }
166     else if(tmp == 2 || tmp == 3 || tmp == 4) {
167         y = getindex('E');
168         next = table[stk->i[stk->top]][y];
169         stk->top++;
170         stk->s[stk->top] = 'E';
171         stk->i[stk->top] = next;
172     }
173     else if(tmp == 5 || tmp == 6 || tmp == 7) {
174         y = getindex('T');
175         next = table[stk->i[stk->top]][y];
176         stk->top++;
177         stk->s[stk->top] = 'T';
178         stk->i[stk->top] = next;
179     }
180     else if(tmp == 8 || tmp == 9) {
181         y = getindex('F');

```

```

182         next = table[stk->i[stk->top]][y];
183         stk->top++;
184         stk->s[stk->top] = 'F';
185         stk->i[stk->top] = next;
186     }
187     else if(tmp == 10) {
188         y = getindex('V');
189         next = table[stk->i[stk->top]][y];
190         stk->top++;
191         stk->s[stk->top] = 'V';
192         stk->i[stk->top] = next;
193     }
194     else if(tmp == 11) {
195         return 1;
196     }
197     printSLR(str, stk, i);
198 }
199 }
200 return 0;
201 }
202
203 int main() {
204     char txt[MAX_LEN] = "outputofcifa.txt";
205
206     FILE *fp = fopen(txt, "r");
207     if(fp == NULL) {
208         perror("Error opening file");
209         return 1;
210     }
211
212     char buf[MAX_LEN] = "";
213     char input[MAX_LEN] = "";
214     fgets(buf, MAX_LEN, fp);
215     fclose(fp);
216     cout <<"file content: "<< buf << endl;
217     int j = 0, k = 0;
218     // buf:(id, s)(OPERATOR, =)(id, a)(OPERATOR, +)(id, b)(OPERATOR,
+) (id, c)(OPERATOR, +)(DELIMITER, )(id, a)(OPERATOR, *) (id, a)
(DELIMITER, ))
219     while (k<strlen(buf)){
220
221         if (buf[k] == '(') {
222             while (buf[k]!='(',')') {
223                 k++;
224             }
225             k = k + 2;
226             input[j] = buf[k];
227             j++;
228         } else {
229             k++;
230         }
231     }
232     input[j++] = '#'; // 在串尾添加结束符
233     input[j++] = '\0'; // 确保输入串结束

```

```

234     cout << "input: "<<input << endl;
235     struct stack *stk = (struct stack *)malloc(sizeof(struct stack));
236     if(stk == NULL) {
237         perror("Error allocating memory for stack");
238         return 1;
239     }
240     stk->s[0] = '#';
241     stk->i[0] = 0;
242     stk->point[0] = -1;
243     stk->top = 0; // 初始化分析栈
244
245     if(!SLR(input, stk)) {
246         printf("Grammar illegal\n");
247     } else {
248         printQuad(); // 打印四元式
249     }
250
251     free(stk); // 释放栈内存
252
253     return 0;
254 }
255

```

1. slr.cpp是一个SLR(1)分析程序，用于语法分析，生成四元式。输入为词法分析生成的token序列，输出为四元式序列。Action和Goto表源自第四次实验。
2. 思路：
 1. 读取文件 `outputofcifa.txt` 中的内容，将其转换为输入串；
 2. 初始化分析栈，开始SLR分析；
 3. 通过Action和Goto表，进行移进和归约操作；
 4. 生成四元式；
 5. 输出四元式序列。
 6. 四元式的格式为(op, arg1, arg2, result)。
 7. 当遇到某些规约的时候，例如 $E \rightarrow E+T$ ，会生成四元式 $t = E + T$ 。

实验结果

1. 输入：

```
1 | s=a+b+c+(a*a )
```

```
~/biaryl/lab5
> make run
./cifa
./slr
File content: (id, s)(OPERATOR, ~)(id, a)(OPERATOR, +)(id, b)(OPERATOR, +)(id, c)(OPERATOR, +)(DELIMITER, )(id, a)(OPERATOR, *) (id, a)(DELIMITER, ))
Input: s=a+b+c*(a*a)#
stack:
# 0 S: 3
# 0 V: 2
# 0 V: 2 := 4
# 0 V: 2 := 4 a: 9
# 0 V: 2 := 4 F: 7
# 0 V: 2 := 4 T: 6
# 0 V: 2 := 4 E: 5
# 0 V: 2 := 4 E: 5 +10
# 0 V: 2 := 4 E: 5 +10 b: 9
# 0 V: 2 := 4 E: 5 +10 F: 7
# 0 V: 2 := 4 E: 5 +10 T:15
# 0 V: 2 := 4 E: 5
# 0 V: 2 := 4 E: 5 +10
# 0 V: 2 := 4 E: 5 +10 c: 9
# 0 V: 2 := 4 E: 5 +10 T:15
# 0 V: 2 := 4 E: 5
# 0 V: 2 := 4 E: 5 +10 (: 8 a: 9
# 0 V: 2 := 4 E: 5 +10 (: 8 F: 7
# 0 V: 2 := 4 E: 5 +10 (: 8 T: 6
# 0 V: 2 := 4 E: 5 +10 (: 8 T: 6 *:12 a: 9
# 0 V: 2 := 4 E: 5 +10 (: 8 T: 6 *:12 F:17
# 0 V: 2 := 4 E: 5 +10 (: 8 T: 6
# 0 V: 2 := 4 E: 5 +10 (: 8 E:14 ):19
# 0 V: 2 := 4 E: 5 +10 F: 7
# 0 V: 2 := 4 E: 5 +10 T:15
# 0 V: 2 := 4 E: 5
# 0 S: 1
Quadruples:
(+, a, b, t1)
(+, t1, c, t2)
(*, a, a, t3)
(+, t2, t3, t4)
(=, t4, -, s)
str:
a+b+c*(a*a)#
a+b+c*(a*a)#
a+b+c*(a*a)#
+b+c*(a*a)#
+b+c*(a*a)#
+b+c*(a*a)#
b+c*(a*a)#
+c*(a*a)#
+c*(a*a)#
+c*(a*a)#
t1 = a + b
c*(a*a)#
c*(a*a)#
+(a*a)#
+(a*a)#
+(a*a)#
(a*a)#
(a*a)#
*a)#
*a)#
*a)#
t3 = a * a
)#
)#
)#
t4 = t2 + t3
s = t4
operate:
t1 = a + b
t2 = t1 + c
t3 = a * a
t4 = t2 + t3
s = t4
```

计算机2101 陈实
6月1日

2. 选做实验：

1. 输入：

```
1 | if (a > b) {
2 |     c = a + b
3 | }
```

输出：

```
(> , a, b, t1)
(JUMP_IF_FALSE, t1, -, label1)
(+, a, b, t2)
(=, t2, -, c)
(label1, -, -, -)
```

实验总结

1. 本次实验需要构建语法制导的语义分析程序，能在语法分析的同时生成符号表和中间语言代码，并输出结果到文件中。通过实验，我对语义分析有了更深入的了解。
2. 本次实验需要结合实验2实现的词法分析程序来识别token序列，需要实验4实现的SLR(1)分析表来进行语法分析获取action和goto表。让我对整个编译原理的流程有了更深入的了解。

附录

1. cifa.cpp：词法分析程序，生成token序列；

```
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 std::multimap<string, string> item; // 符号表(type, value)
26 // typelist: KEYWORD, ID, INT, FLOAT, CHAR, STRING, OPERATOR,
    DELIMITER, ERROR
27 void get_char() {
28     forwar = forwar + 1;
29     C = buffer[forwar];
30 }
31 // 从buf中读一个字符到C中，向前指针移动。
32 void cat() {
33     nowstr.push_back(C);
34 }
35 // 将字符C连接到nowstr字符串后面
36 std::string toLower(const std::string& str) {
37     // 创建一个副本以避免修改原始字符串
38     std::string lowerStr = str;
39
40     // 使用 std::transform 和 std::tolower 将字符串转换为全小写
41     std::transform(lowerStr.begin(), lowerStr.end(), lowerStr.begin(),
        ::tolower);
42
43     return lowerStr; // 返回转换后的字符串
44 }
45 std::string toUpper(const std::string& str) {
46     // 创建一个副本以避免修改原始字符串
47     std::string upperStr = str;
48
49     // 使用 std::transform 和 std::tolower 将字符串转换为全小写
50     std::transform(upperStr.begin(), upperStr.end(), upperStr.begin(),
        ::toupper);
51
52     return upperStr; // 返回转换后的字符串
53 }
54
55 bool is_letter(char ch) {
56     if (isalpha(ch) || ch == '_' || ch == '-')

```

```

57         return true;
58     else
59         return false;
60 }
61 //判断ch是否为字母或下划线
62
63 int iskeyword() {
64     for (int i = 0; i < keyword.size(); i++){
65         if (nowstr == keyword[i]) {
66             return 1;
67         }
68         if (toLower(nowstr) == keyword[i]) {
69             return 2;
70         }
71     }
72     return 0;
73 }
74 //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
75
76 bool iseven() {
77     int num = 0;
78     int i = nowstr.size() - 2;
79     while (nowstr[i] == '\\') {
80         num++;
81         i--;
82     }
83     if (num % 2 == 0)
84         return true;
85     return false;
86 }
87
88 //根据传入的type和value，将其插入符号表
89 void insert_item(string type, string value) {
90     item.insert(pair<string,string>(type,value));
91     sleep(0.5);
92     if (type == "ERROR") {
93         cout << "\033[31m" // 31对应红色
94             << rows << ": (" << type << ", " << value << ")"
95             << "\033[0m" << endl;
96     }
97     else {
98         cout << "(" << type << ", " << value << ")";
99     }
100 }
101
102 //根据文件名，读取文件内容到buffer中
103 void read_file(const char *filename) {
104     FILE *fp = fopen(filename, "r"); // 用二进制模式打开
105     if (fp == NULL) {
106         cout << "文件打开失败" << endl;
107         exit(0);
108     }
109     fseek(fp, 0, SEEK_END);
110     sum_char = ftell(fp);

```

```

111     fseek(fp, 0, SEEK_SET);
112     buffer = new char[sum_char+1];
113     fread(buffer, 1, sum_char, fp);
114     fclose(fp);
115     //Buffer末尾加上EOF
116     buffer[sum_char] = EOF;
117 }
118
119 void scanner(){
120     bool isEnd = false;
121     while(!isEnd){
122         get_char();
123         if (C == '\n'){
124             rows++;
125         }
126         if (C == EOF){
127             isEnd = true;
128         }
129         switch (state) {
130             case 0: //初始状态
131                 if (is_letter(C)){
132                     state =1;    //id or keyword
133                     cat();
134                 }
135                 else if (isdigit(C)){
136                     state = 2; //num or wrong id
137                     cat();
138                 }
139                 else if (C=='0'){
140                     state=28;
141                     cat();
142                 }
143                 else{
144                     switch (C) {
145                         case '<': state = 8; break;
146                         case '>': state = 9; break;
147                         case '?':
148                             insert_item("OPERATOR", "?");
149                             break;
150                         case ':':
151                             insert_item("OPERATOR", ":");
152                             break;
153                         case '/': state = 11; break;
154                         case '=': state = 12; break;
155                         case '+': state = 13; break;
156                         case '-': state = 14; break;
157                         case '*': state = 15; break;
158                         case '%': state = 16; break;
159                         case '(':
160                             insert_item("DELIMITER", "(");
161                             break;
162                         case ')':
163                             insert_item("DELIMITER", ")");
164                             break;

```

```

165         case ',':
166             insert_item("DELIMITER", ",");
167             break;
168         case ';':
169             insert_item("DELIMITER", ";");
170             break;
171         case '{':
172             insert_item("DELIMITER", "{");
173             break;
174         case '}':
175             insert_item("DELIMITER", "}");
176             break;
177         case '[':
178             insert_item("DELIMITER", "[");
179             break;
180         case ']':
181             insert_item("DELIMITER", "]");
182             break;
183         case '^': state = 17; break;
184         case '|': state = 18; break;
185         case '~': state = 19; break;
186         case '!': state = 20; break;
187         case '&': state = 21; break;
188         case '"': state = 22; cat(); break;
189         case '\\': state = 23; cat(); break;
190         case '.': state = 24; break;
191         case '#': insert_item("OPERATOR", "#"); break;
192         case ' ':
193         case '\\n':
194         case ' ':
195             case EOF : break; //跳过空白符
196             default: cout << "error:第" << rows << "行出现非法字符" << C << endl;
197                     break;
198             }
199         }
200         break;
201     case 1: //id or keyword
202         if (is_letter(C) || isdigit(C)) {
203             cat();
204             state = 1;
205         }
206         else {
207             forwar--;
208             state = 0;
209             if (C == '\\n') {
210                 rows--;
211             }
212             int flag = iskeyword();
213             if (flag == 1) {
214                 insert_item("keyword", nowstr);
215             } else if (flag == 2) {
216                 insert_item("ERROR", nowstr+" --- Keyword
should be lower case");

```

```

217         } else {
218             insert_item("id", nowstr);
219         }
220         nowstr.clear();
221     }
222     break;
223 case 2:
224     if (isdigit(C)) {
225         cat();
226         state = 2;
227     }
228     else if (C == '.') {
229         cat();
230         state = 3;
231     }
232     else if (C == 'E' || C == 'e') {
233         cat();
234         state = 5;
235     }
236     //wrong id
237     else if (is_letter(C)) {
238         cat();
239         state = 31;
240     }
241     else {
242         forwar--;
243         state = 0;
244         if (C == '\n') {
245             rows--;
246         }
247         insert_item("int", nowstr);
248         nowstr.clear();
249     }
250     break;
251 case 3: // .
252     if (isdigit(C)) {
253         cat();
254         state = 4;
255     }
256     else {
257         forwar--;
258         state = 0;
259         if (C == '\n') {
260             rows--;
261         }
262         nowstr.push_back('0');
263         insert_item("float", nowstr);
264         nowstr.clear();
265     }
266     break;
267
268 case 4: // .num
269     if (isdigit(C)) {
270         cat();

```

```

271         state = 4;
272     }
273     else if (C == 'E' || C == 'e') {
274         cat();
275         state = 5;
276     }
277     else {
278         forwar--;
279         state = 0;
280         if (C == '\n') {
281             rows--;
282         }
283         insert_item("float", nowstr);
284         nowstr.clear();
285     }
286     break;
287
288     case 5:        // .numE
289         if (C == '+' || C == '-') {
290             cat();
291             state = 6;
292         }
293         else if (isdigit(C)) {
294             cat();
295             state = 7;
296         }
297         else {
298             forwar--;
299             state = 0;
300             if (C == '\n') {
301                 rows--;
302             }
303             insert_item("ERROR", nowstr + " --- Missing
digit");
304             nowstr.clear();
305         }
306         break;
307
308     case 6:        // .numE+ or .numE-
309         if (isdigit(C)) {
310             cat();
311             state = 7;
312         }
313         else {
314             forwar--;
315             state = 0;
316             if (C == '\n') {
317                 rows--;
318             }
319             insert_item("ERROR", nowstr + " --- Missing
digit");
320             nowstr.clear();
321         }
322         break;

```

```

323
324     case 7:      // .numE+num
325         if (isdigit(C)) {
326             cat();
327             state = 7;
328         }
329         else {
330             forwar--;
331             state = 0;
332             if (C == '\n') {
333                 rows--;
334             }
335             insert_item("float", nowstr);
336             nowstr.clear();
337         }
338         break;
339
340     case 8:
341         if (C == '=') {
342             insert_item("OPERATOR", "≤");
343         }
344         else if (C == '<') {
345             insert_item("OPERATOR", "<<");
346         }
347         else {
348             forwar--;
349             insert_item("OPERATOR", "<");
350             if (C == '\n') {
351                 rows--;
352             }
353         }
354         state = 0;
355         break;
356
357     case 9:
358         if (C == '=') {
359             insert_item("OPERATOR", "≥");
360         }
361         else if (C == '>') {
362             insert_item("OPERATOR", ">>");
363         }
364         else {
365             forwar--;
366             insert_item("OPERATOR", ">");
367             if (C == '\n') {
368                 rows--;
369             }
370         }
371         state = 0;
372         break;
373
374     case 11:
375         switch (C) {
376             case '/': //单行注释

```

```

377         state = 27;
378         break;
379     case '*':    // 多行注释
380         state = 26;
381         break;
382     default:
383         forwar--;
384         insert_item("OPERATOR", "/");
385         state = 0;
386         if (C == '\n') {
387             rows--;
388         }
389         break;
390     }
391     break;
392
393 case 12:    // =
394     if (C == '=') {
395         insert_item("OPERATOR", "=");
396     }
397     else {
398         forwar--;
399         insert_item("OPERATOR", "=");
400         if (C == '\n') {
401             rows--;
402         }
403     }
404     state = 0;
405     break;
406
407 case 13:    // +
408     if (C == '+') {
409         insert_item("OPERATOR", "++");
410     }
411     else if (C == '=') {
412         insert_item("OPERATOR", "+=");
413     }
414     else if (isdigit(C)) {
415         cat();
416         state = 2;
417     }
418     else {
419         forwar--;
420         insert_item("OPERATOR", "+");
421         if (C == '\n') {
422             rows--;
423         }
424     }
425     state = 0;
426     break;
427
428 case 14:    // -
429     if (C == '-') {
430         insert_item("OPERATOR", "--");

```



```

431     }
432     else if (C == '=') {
433         insert_item("OPERATOR", "-=");
434     }
435     else if (C == '>') {
436         insert_item("OPERATOR", "→");
437     }
438     else if (isdigit(C)) {
439         cat();
440         state = 2;
441     }
442     else {
443         forwar--;
444         insert_item("OPERATOR", "-");
445         if (C == '\n') {
446             rows--;
447         }
448     }
449     state = 0;
450     break;
451
452 case 15:    // *
453     if (C == '=') {
454         insert_item("OPERATOR", "*=");
455     }
456     else {
457         forwar--;
458         insert_item("OPERATOR", "*");
459         if (C == '\n') {
460             rows--;
461         }
462     }
463     state = 0;
464     break;
465
466 case 16:    // %
467     if (C == '=') {
468         insert_item("OPERATOR", "%=");
469     }
470     else {
471         forwar--;
472         insert_item("OPERATOR", "%");
473         if (C == '\n') {
474             rows--;
475         }
476     }
477     state = 0;
478     break;
479
480 case 17:    // ^
481     if (C == '=') {
482         insert_item("OPERATOR", "^=");
483     }
484     else {

```

```

485         forwar--;
486         insert_item("OPERATOR", "^");
487         if (C == '\n') {
488             rows--;
489         }
490     }
491     state = 0;
492     break;
493
494 case 18:    // |
495     if (C == '|') {
496         insert_item("OPERATOR", "||");
497     }
498     else if (C == '=') {
499         insert_item("OPERATOR", "!=");
500     }
501     else {
502         forwar--;
503         insert_item("OPERATOR", "|");
504         if (C == '\n') {
505             rows--;
506         }
507     }
508     state = 0;
509     break;
510
511 case 19:    // ~
512     if (C == '=') {
513         insert_item("OPERATOR", "~=");
514     }
515     else {
516         forwar--;
517
518         insert_item("OPERATOR", "~");
519         if (C == '\n') {
520             rows--;
521         }
522     }
523     state = 0;
524     break;
525
526 case 20:    // !
527     if (C == '=') {
528         insert_item("OPERATOR", "!=");
529     }
530     else {
531         forwar--;
532
533         insert_item("OPERATOR", "!");
534         if (C == '\n') {
535             rows--;
536         }
537     }
538     state = 0;

```

```

539         break;
540
541     case 21:    // &
542         if (C == '&') {
543             insert_item("OPERATOR", "&");
544         }
545         else if (C == '=') {
546             insert_item("OPERATOR", "&=");
547         }
548         else {
549             forwar--;
550
551             insert_item("OPERATOR", "&");
552             if (C == '\n') {
553                 rows--;
554             }
555         }
556         state = 0;
557         break;
558
559     case 22:    // "
560         if (C == '\"') {
561             cat();
562             if (iseven()) {
563                 insert_item("STRING", nowstr);
564                 nowstr.clear();
565                 state = 0;
566             }
567             else {
568                 state = 22;
569             }
570         }
571         else if (C == EOF) {
572             insert_item("ERROR", nowstr + " --- String should
end with '\"");
573
574             nowstr.clear();
575             state = 0;
576         }
577         else {
578             cat();
579             state = 22;
580         }
581         break;
582
583     case 23:    // '
584         if (C == '\\'') {
585             cat();
586             //判断是否是转义字符
587             if (nowstr.size() == 4 && nowstr[1] == '\\') {
588                 insert_item("CHAR", nowstr);
589             }
590             //单独处理'\'这种情况
591             else if (nowstr[0]='\\' && nowstr[1]='\\' &&
nowstr[2]='\\') {

```

```

591         get_char();
592         if (C == '\\') {
593             cat();
594             insert_item("CHAR", nowstr);
595         }
596         else {
597             forwar--;
598
599         }
600     }
601     else if (nowstr.size() == 3) {
602         insert_item("CHAR", nowstr);
603     }
604     else {
605         insert_item("ERROR", nowstr + " --- Char
should be one character");
606     }
607     nowstr.clear();
608     state = 0;
609 }
610 else if (C == EOF) {
611     insert_item("ERROR", nowstr + " --- Char should
end with '\\");
612     nowstr.clear();
613     state = 0;
614 }
615 else {
616     cat();
617     state = 23;
618 }
619 break;
620
621 case 24:    // .
622     if (isdigit(C)) {
623         cat();
624         state = 4;
625     }
626     else {
627         forwar--;
628
629         insert_item("OPERATOR", ".");
630         if (C == '\\n') {
631             rows--;
632         }
633         state = 0;
634     }
635     break;
636
637 case 26:    // /*
638     if (C == '*') {
639         state = 25;
640     }
641     break;
642

```

```

643         case 25:      // /*...*
644             if (C == '*') {
645                 state = 25;
646             }
647             else if (C == '/') {
648                 state = 0;
649             }
650             else if (C == EOF) {
651                 insert_item("ERROR", nowstr + " --- Multi-line
comment should end with */");
652                 nowstr.clear();
653                 state = 0;
654             }
655             break;
656
657         case 27:      // //
658             if (C == '\n' || C == EOF) {
659                 state = 0;
660             }
661             break;
662
663         case 28:      // 0
664             if (C == 'x' || C == 'X') {
665                 cat();
666                 state = 29;
667             }
668             else if (isdigit(C)) {
669                 cat();
670                 state = 2;
671             }
672             else if (C == '.') {
673                 cat();
674                 state = 3;
675             }
676             else {
677                 forwar--;
678
679                 insert_item("int", nowstr);
680                 nowstr.clear();
681                 state = 0;
682             }
683             break;
684
685         case 29:      // 0x
686             if (isdigit(C) || (C >= 'a' && C <= 'f') || (C >= 'A'
&& C <= 'F')) {
687                 cat();
688                 state = 30;
689             }
690             else {
691                 forwar--;
692
693                 insert_item("ERROR", nowstr + " --- Hexadecimal
number should have at least one digit");

```

```

694         nowstr.clear();
695         state = 0;
696     }
697     break;
698
699     case 30:    // 0x...
700         if (isdigit(C) || (C ≥ 'a' && C ≤ 'f') || (C ≥ 'A'
&& C ≤ 'F')) {
701             cat();
702             state = 30;
703         }
704         else {
705             forwar--;
706
707             insert_item("int", nowstr);
708             nowstr.clear();
709             state = 0;
710         }
711         break;
712
713     case 31:    //wrong id with num at first
714         if (is_letter(C) || isdigit(C)) {
715             cat();
716             state = 31;
717         }
718         else {
719             forwar--;
720
721             insert_item("ERROR", nowstr + " --- ID should
start with a letter");
722             nowstr.clear();
723             state = 0;
724         }
725         break;
726     default:
727         break;
728     }
729 }
730 }
731
732 int main() {
733     const char *input_filename = "input.txt"; // 默认输入文件名
734     const char *output_filename = "outpufcifa.txt";
735
736     freopen(output_filename, "w", stdout); // 将输出重定向到指定的输出文件
737     read_file(input_filename); // 读取指定输入文件
738     scanner(); // 扫描内容
739     delete[] buffer; // 释放分配的内存
740
741     return 0;
742 }

```

2. slr.cpp

```
1 #include<stdio.h>
```

```

2  #include<string.h>
3  #include<stdlib.h>
4  #include<iostream>
5  #define MAX_LEN 1000
6  using namespace std;
7  struct stack {
8      char s[MAX_LEN];
9      int i[MAX_LEN];
10     int point[MAX_LEN];
11     int top;
12 }; // 分析栈数据结构
13
14 struct quadruple {
15     char op[MAX_LEN];
16     char arg1[MAX_LEN];
17     char arg2[MAX_LEN];
18     char result[MAX_LEN];
19 }; // 四元式数据结构
20
21 struct quadruple quad[MAX_LEN]; // 存储四元式
22 int quadTop = 0; // 四元式栈顶
23
24 // 1.S→V=E 2.E→E+T 3.E→E-T 4.E→T 5.T→T*F 6.T→T/F 7.T→F 8.F→(E)
25 // 9.F→i 10.V→i
26 // 表中大于0对应移进, 小于0则对应先归约后移进, 0为不存在的状态
27 //          GOTO          |      ACTION
28 //i, =, +, -, *, /, (, ), #, S, E, T, F, V
29 int table[20][14] = {{ 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2}, // 0
30                      { 0, 0, 0, 0, 0, 0, 0, 0, 0, -11, 0, 0, 0, 0}, // 1
31                      { 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 2
32                      {-10, -10, -10, -10, -10, -10, -10, -10, -10, 0, 0, 0, 0, 0}, // 3
33                      { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 5, 6, 7, 0}, // 4
34                      {-1, -1, 10, 11, -1, -1, -1, -1, 0, 0, 0, 0, 0, 0}, // 5
35                      {-4, -4, -4, -4, 12, 13, -4, -4, -4, 0, 0, 0, 0, 0}, // 6
36                      {-7, -7, -7, -7, -7, -7, -7, -7, 0, 0, 0, 0, 0, 0}, // 7
37                      { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 14, 6, 7, 0}, // 8
38                      {-9, -9, -9, -9, -9, -9, -9, -9, 0, 0, 0, 0, 0, 0}, // 9
39                      { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 15, 7, 0}, // 10
40                      { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 16, 7, 0}, // 11
41                      { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 17, 0}, // 12
42                      { 9, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 18, 0}, // 13
43                      { 0, 0, 10, 11, 0, 0, 0, 19, 0, 0, 0, 0, 0, 0}, // 14
44                      {-2, -2, -2, -2, 12, 13, -2, -2, -2, 0, 0, 0, 0, 0}, // 15
45                      {-3, -3, -3, -3, 12, 13, -3, -3, -3, 0, 0, 0, 0, 0}, // 16
46                      {-5, -5, -5, -5, -5, -5, -5, -5, -5, 0, 0, 0, 0, 0}, // 17
47                      {-6, -6, -6, -6, -6, -6, -6, -6, -6, 0, 0, 0, 0, 0}, // 18
48                      {-8, -8, -8, -8, -8, -8, -8, -8, -8, 0, 0, 0, 0, 0}}; // 19
49
50 int getindex(char ch) {
51     switch(ch) {
52         case 'i': return 0;
53         case '=': return 1;
54         case '+': return 2;

```

```

54         case '-': return 3;
55         case '*': return 4;
56         case '/': return 5;
57         case '(': return 6;
58         case ')': return 7;
59         case '#': return 8;
60         case 'S': return 9;
61         case 'E': return 10;
62         case 'T': return 11;
63         case 'F': return 12;
64         case 'V': return 13;
65         default: return -1;
66     }
67 }
68
69 void printSLR(char *str, struct stack *stk, int now) { // 打印分析状态
70     for(int i = 0; i ≤ stk→top; i++) {
71         printf("%c:%2d  ", stk→s[i], stk→i[i]); // 栈状态
72     }
73     for(int i = 0; i ≤ 60 - stk→top*7; i++) {
74         printf(" ");
75     }
76     for(int i = now; i < strlen(str); i++) {
77         printf("%c", str[i]); // 串状态
78     }
79     printf("\n");
80 }
81
82 void printQuad() { // 打印四元式
83     printf("Quadruples:\n");
84     for(int i = 1; i ≤ quadTop; i++) {
85         printf("(%s, %s, %s, %s)\n", quad[i].op, quad[i].arg1,
86             quad[i].arg2, quad[i].result);
87     }
88 }
89
90 int SLR(char *str, struct stack *stk) { // SLR1分析函数
91     quadTop = 0;
92     int i = 0;
93     int next;
94     printf("stack:
95         str:                operate:\n");
96     while(i < strlen(str)) {
97         if(stk→top < 0) return 0; // 分析栈不可能为空
98         int y; // 列坐标
99         if (str[i] ≥ 'a' && str[i] ≤ 'z') y = getindex('i'); // 终结
100         符i
101         else y = getindex(str[i]);
102         if(y = -1 || table[stk→i[stk→top]][y] = 0) { // 表中不存在的
103             状态, 分析报错
104             return 0;
105         }
106         if(table[stk→i[stk→top]][y] > 0) { // 移进操作
107             next = table[stk→i[stk→top]][y];

```



```

104         stk→top++;
105         stk→s[stk→top] = str[i];
106         stk→i[stk→top] = next;
107         stk→point[stk→top] = i;
108         i++;
109         printSLR(str, stk, i);
110     }
111     else if(table[stk→i[stk→top]][y] < 0) { // 归约操作
112         int tmp = -table[stk→i[stk→top]][y]; // 查GOTO表
113         if(tmp == 4 || tmp == 7 || tmp == 9 || tmp == 10) {
114             stk→top--; // 要归约1位
115         }
116         else if(tmp == 2 || tmp == 3 || tmp == 5 || tmp == 6){
117             // 生成四元式
118             quadTop++;
119             if(tmp == 2) strcpy(quad[quadTop].op, "+");
120             else if(tmp == 3) strcpy(quad[quadTop].op, "-");
121             else if(tmp == 5) strcpy(quad[quadTop].op, "*");
122             else strcpy(quad[quadTop].op, "/");
123             if(stk→point[stk→top - 2] < 0)
124                 sprintf(quad[quadTop].arg1, "%d", -stk→point[stk→top - 2]);
125             else {
126                 char arg1[2] = {str[stk→point[stk→top - 2]],
127                     '\0'};
128                 strcpy(quad[quadTop].arg1, arg1);
129             }
130             if(stk→point[stk→top] < 0)
131                 sprintf(quad[quadTop].arg2, "%d", -stk→point[stk→top]);
132             else {
133                 char arg2[2] = {str[stk→point[stk→top]], '\0'};
134                 strcpy(quad[quadTop].arg2, arg2);
135             }
136             for(int j = 0; j < 90; j++) printf(" ");
137             printf("%d = %s %s %s\n", quadTop,
138                 quad[quadTop].arg1, quad[quadTop].op, quad[quadTop].arg2); // 打印语义动作
139             sprintf(quad[quadTop].result, "%d", quadTop);
140             stk→top -= 3; // 归约3位
141             stk→point[stk→top + 1] = -quadTop; // 记录归约产生的中间变量
142         }
143         else if(tmp == 8) {
144             stk→top -= 3; // 归约3位
145             stk→point[stk→top + 1] = stk→point[stk→top + 2];
146             // 消除括号规约
147         }
148         else if(tmp == 1){
149             quadTop++;
150             strcpy(quad[quadTop].op, "=");
151             if(stk→point[stk→top] < 0)
152                 sprintf(quad[quadTop].arg1, "%d", abs(stk→point[stk→top]));
153             else {
154                 char arg1[2] = {str[stk→point[stk→top]], '\0'};
155                 strcpy(quad[quadTop].arg1, arg1);

```

```

150         }
151         sprintf(quad[quadTop].arg2, " ");
152         char res[2] = {str[stk→point[stk→top - 2]], '\0'};
153         strcpy(quad[quadTop].result, res);
154         for(int i = 0; i < 90; i++) printf(" ");
155         printf("%s = %s\n", quad[quadTop].result,
quad[quadTop].arg1);
156         stk→top -= 3; // 归约V=E
157     }
158     else stk→top -= 3;
159     if(tmp == 1) {
160         y = getindex('S');
161         next = table[stk→i[stk→top]][y]; // 查ACTION表
162         stk→top++;
163         stk→s[stk→top] = 'S';
164         stk→i[stk→top] = next; // 归约要修改栈顶
165     }
166     else if(tmp == 2 || tmp == 3 || tmp == 4) {
167         y = getindex('E');
168         next = table[stk→i[stk→top]][y];
169         stk→top++;
170         stk→s[stk→top] = 'E';
171         stk→i[stk→top] = next;
172     }
173     else if(tmp == 5 || tmp == 6 || tmp == 7) {
174         y = getindex('T');
175         next = table[stk→i[stk→top]][y];
176         stk→top++;
177         stk→s[stk→top] = 'T';
178         stk→i[stk→top] = next;
179     }
180     else if(tmp == 8 || tmp == 9) {
181         y = getindex('F');
182         next = table[stk→i[stk→top]][y];
183         stk→top++;
184         stk→s[stk→top] = 'F';
185         stk→i[stk→top] = next;
186     }
187     else if(tmp == 10) {
188         y = getindex('V');
189         next = table[stk→i[stk→top]][y];
190         stk→top++;
191         stk→s[stk→top] = 'V';
192         stk→i[stk→top] = next;
193     }
194     else if(tmp == 11) {
195         return 1;
196     }
197     printSLR(str, stk, i);
198 }
199 }
200 return 0;
201 }
202

```

```

203 int main() {
204     char txt[MAX_LEN] = "outputofcifa.txt";
205
206     FILE *fp = fopen(txt, "r");
207     if(fp == NULL) {
208         perror("Error opening file");
209         return 1;
210     }
211     char buf[MAX_LEN] = "";
212     char input[MAX_LEN] = "";
213     fgets(buf, MAX_LEN, fp);
214     fclose(fp);
215     cout << "file content: " << buf << endl;
216     int j = 0, k = 0;
217     // buf:(id, s)(OPERATOR, =)(id, a)(OPERATOR, +)(id, b)(OPERATOR,
+)(id, c)(OPERATOR, +)(DELIMITER, )(id, a)(OPERATOR, *)(id, a)
(DELIMITER, ))
218     while (k < strlen(buf)) {
219
220         if (buf[k] == '(') {
221             while (buf[k] != ',') {
222                 k++;
223             }
224             k = k + 2;
225             input[j] = buf[k];
226             j++;
227         } else {
228             k++;
229         }
230     }
231     input[j++] = '#'; // 在串尾添加结束符
232     input[j++] = '\0'; // 确保输入串结束
233     cout << "input: " << input << endl;
234     struct stack *stk = (struct stack *)malloc(sizeof(struct stack));
235     if(stk == NULL) {
236         perror("Error allocating memory for stack");
237         return 1;
238     }
239     stk->s[0] = '#';
240     stk->i[0] = 0;
241     stk->point[0] = -1;
242     stk->top = 0; // 初始化分析栈
243
244     if(!SLR(input, stk)) {
245         printf("Grammar illegal\n");
246     } else {
247         printQuad(); // 打印四元式
248     }
249
250     free(stk); // 释放栈内存
251
252     return 0;
253 }

```

3. Makefile

```
1 # 定义编译器
2 CXX = g++
3
4 # 定义目标文件
5 TARGETS = cifa slr
6
7 # 定义源文件
8 SOURCES = cifa.cpp slr.cpp
9
10 # 定义编译选项
11 CXXFLAGS = -Wall -g
12
13 # 默认目标
14 all: $(TARGETS)
15
16 # 规则: 编译cifa
17 cifa: cifa.cpp
18     $(CXX) $(CXXFLAGS) -o cifa cifa.cpp
19
20 # 规则: 编译slr
21 slr: slr.cpp
22     $(CXX) $(CXXFLAGS) -o slr slr.cpp
23
24 # 运行目标: 先运行cifa再运行slr
25 run: all
26     ./cifa
27     ./slr
28
29 # 清理目标: 删除生成的可执行文件
30 clean:
31     rm -f $(TARGETS)
32
33 # 声明伪目标
34 .PHONY: all run clean
```