编译原理 实验一：词法分析

09019106 牟倪

目录

[一、实验目标 2](#_Toc90478413)

[二、实验内容 2](#_Toc90478414)

[三、基本思路 2](#_Toc90478415)

[四、假设与名词定义 2](#_Toc90478416)

[五、相关FA描述 3](#_Toc90478417)

[5.1 identifier的FA描述 3](#_Toc90478418)

[5.2 integer的FA描述 3](#_Toc90478419)

[5.3 float的FA描述 3](#_Toc90478420)

[5.4 operator的FA描述 4](#_Toc90478421)

[六、核心算法 4](#_Toc90478422)

[6.1 程序框架 4](#_Toc90478423)

[6.2 analysis()函数 6](#_Toc90478424)

[6.3 anaIdentifier()函数 6](#_Toc90478425)

[6.4 anaInteger()函数 7](#_Toc90478426)

[6.5 anaFloat()函数 8](#_Toc90478427)

[6.6 anaOperator()函数 9](#_Toc90478428)

[七、实验 9](#_Toc90478429)

[八、出现问题与解决方案 11](#_Toc90478430)

[九、实验体会 11](#_Toc90478431)

[附录：源代码 12](#_Toc90478432)

# 一、实验目标

* 理解词法分析过程。
* 熟练掌握FA、RE和Grammar之间的转换。
* 自主实现词法分析程序。

# 二、实验内容

* 自主定义token和对应的RE。
* 完成词法分析程序：
  + 输入字符串，输出解析结果token串。
  + 对不合法的输入字符串，报错。

# 三、基本思路

首先利用Thompson's construction rules将RE转换为NFA，利用subset method将NFA转换为DFA，利用partition method将DFA转换为minimum DFA。

然后，对DFA的状态进行标号，构造相应的状态转换逻辑。逐个读入字符，模拟状态转换过程，从而判定字符串是否合法以及token类型。

# 四、假设与名词定义

基本RE定义：

* letter → A | B | … | Z | a | b | … | z
* nonzero\_digit → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
* digit → 0 | nonzero\_digit

token的RE定义：

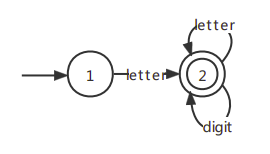
* identifier → letter ( letter | digit )\*
* integer → nonzero\_digit ( digit )\*
* float → ( integer | 0) . ( digit )+
* operator → ( + | - | \* | / )

（为了简化情况，减少二义性（如-0.2可以被解析为负0.2或减0.2），假设integer和float均为正数）

# 五、相关FA描述

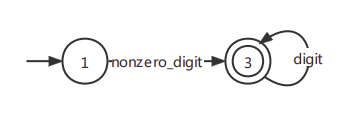
## 5.1 identifier的FA描述

可以直接得到minimum DFA：



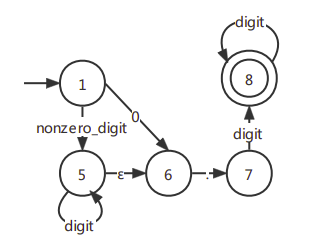
## 5.2 integer的FA描述

可以直接得到minimum DFA：

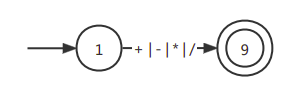


## 5.3 float的FA描述

根据integer的minimum DFA，得到float的NFA：



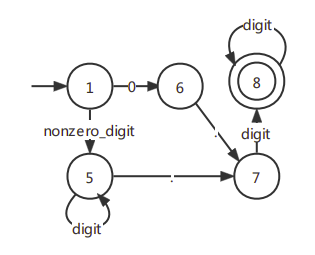
从NFA转化为DFA：



已经是minimum DFA。

## 5.4 operator的FA描述

可以直接得到minimum DFA：



# 六、核心算法

## 6.1 程序框架

构建LexicalAnalysis类。

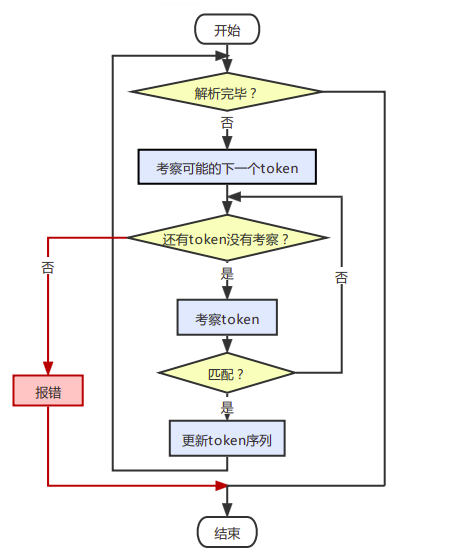
成员变量：

* state，int型变量，表示目前所处的状态。
* input，string型变量，用来记录待解析的字符串。
* pos，int型变量，起指针的作用，指向字符串中已经成功解析的位置。
* cur，int型变量，起指针的作用，指向字符串中目前解析到的位置。
* output，queue<string>型变量，记录解析结果。

成员函数：

* void init()，读入字符串input，为input添加后缀'#'。
* void analysis()，词法分析。
* char curChar()，返回当前字符
* char nextChar()，返回当前字符的下一个字符。
* bool isLetter(char c)、bool isNonzeroDigit(char c)、bool isDigit(char c)、bool isPoint(char c)、 bool isOperator(char c)、bool isZero(char c)，用来判断字符类型。
* bool anaIdentifier()、bool anaInteger()、bool anaFloat()、bool anaOperator()，解析出下一个token，返回值为是否成功。
* void fail()，用来报错。
* void printOutput()，打印解析结果。

利用LexicalAnalysis类，按照如下流程图，可以实现词法分析程序：



在6.2-6.6节，我会详细介绍LexicalAnalysis类的成员函数，将功能与上述流程图对应。

## 6.2 analysis()函数

analysis()函数通过当前字符的类型判断下一个可能的token，并对所有可能的token逐个尝试。如果所有尝试都失败，调用fail()函数。analysis()函数对应前两个带判断条件的while循环。代码如下：

1. void analysis(){
2. while(curChar() != '#'){
3. if(isLetter(curChar())){
4. if(anaIdentifier()) continue;
5. };
6. if(isNonzeroDigit(curChar())){
7. if(anaFloat()) continue;
8. if(anaInteger()) continue;
9. }
10. if(isZero(curChar())){
11. if(anaFloat()) continue;
12. }
13. if(isOperator(curChar())){
14. if(anaOperator()) continue;
15. }
16. fail();
17. }
18. }

## 6.3 anaIdentifier()函数

anaIdentifier()函数用来解析identifier。代码如下：

1. bool anaIdentifier(){
2. cur = pos;
3. char curc = curChar();
4. while(1){
5. switch(state){
6. case 1:
7. ++cur; state = 2;
8. break;
9. case 2:
10. curc = curChar();
11. if(!(isLetter(curc) || isDigit(curc))){
12. output.push("<identifier>");
13. pos = cur;
14. state = 1;
15. return true;
16. }
17. else{
18. ++cur; break;
19. }
20. default:
21. state = 1; cur = pos; return false;
22. }
23. }
24. }

## 6.4 anaInteger()函数

anaInteger ()函数用来解析integer。因为integer和float具有左公共因子，因此额外判断下一个字符是否为'.'，若是则返回false，而非取float的整数部分作为integer。代码如下：

1. bool anaInteger(){
2. cur = pos;
3. char curc = curChar();
4. while(1){
5. switch(state){
6. case 1:
7. ++cur; state = 3;
8. break;
9. case 3:
10. if(!isDigit(curChar())){
11. if(isPoint(curChar())){
12. state = 1;
13. cur = pos;
14. return false;
15. }
16. output.push("<integer>");
17. pos = cur;
18. state = 1;
19. return true;
20. }
21. else{
22. ++cur; break;
23. }
24. default:
25. state = 1; cur = pos; return false;
26. }
27. }
28. }

## 6.5 anaFloat()函数

anaFloat()函数用来解析float。代码如下：

1. bool anaFloat(){
2. cur = pos;
3. char curc = curChar();
4. while(1){
5. switch(state){
6. case 1:
7. if(isZero(curChar())) state = 6;
8. else state = 5;
9. ++cur;
10. break;
11. case 5:
12. if(isDigit(curChar())){
13. ++cur; break;
14. }
15. else if(isPoint(curChar())){
16. state = 7; ++cur; break;
17. }
18. else {
19. state = 1; cur = pos; return false;
20. }
21. case 6:
22. if(isPoint(curChar())){
23. state = 7; ++cur; break;
24. }
25. else {
26. state = 1; cur = pos; return false;
27. }
28. case 7:
29. if(isDigit(curChar())){
30. state = 8; ++cur; break;
31. }
32. else {
33. state = 1; cur = pos; return false;
34. }
35. case 8:
36. if(!isDigit(curChar())){
37. output.push("<float>");
38. pos = cur;
39. state = 1;
40. return true;
41. }
42. else{
43. ++cur; break;
44. }
45. default:
46. state = 1; cur = pos; return false;
47. }
48. }
49. }

## 6.6 anaOperator()函数

anaOperator()函数用来解析operator。代码如下：

1. bool anaOperator(){
2. cur = pos;
3. char curc = curChar();
4. while(1){
5. switch(state){
6. case 1:
7. ++cur; state = 9; break;
8. case 9:
9. output.push("<operator>");
10. pos = cur;
11. state = 1;
12. return true;
13. default:
14. state = 1; cur = pos; return false;
15. }
16. }
17. }

# 七、实验

* 目的：验证identifier是否成功解析。

输入：muni

输出：<identifier>

* 目的：验证integer是否成功解析。

输入：123893827483328234

输出：<integer>

* 目的：验证float是否成功解析。

输入：0.32482734823773732

输出：<float>

* 目的：验证多个token的情形。

输入：123+456+abc

输出：<integer> <operator> <integer> <operator> <identifier>

* 目的：验证多个token的情形。

输入：muni123+haha+yes+123

输出：<identifier> <operator> <identifier> <operator> <identifier> <operator> <integer>

* 目的：检查格式错误的float是否报错。

输入：0123.23232

输出：lexical analysis fail!

* 目的：检查格式错误的float是否报错。

输入：0.123.456.789

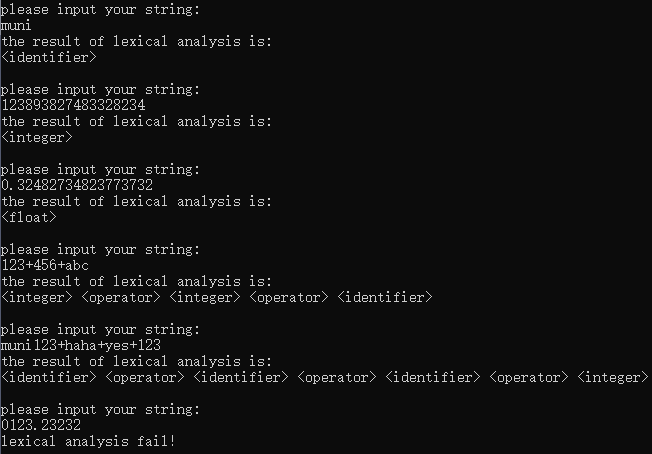
输出：lexical analysis fail!

* 目的：检查格式错误identifier的是否报错。

输入：@a123bdbe

输出：lexical analysis fail!

运行截图（部分）：



可以看出，实验结果正确，程序编写合理。

# 八、出现问题与解决方案

没有出现问题。

# 九、实验体会

经过本次实验，我复习了词法分析的内容，RE、Grammar、FA之间的转换也更熟练了。将抽象的、看起来庞大难以解决的词法分析问题，转化成RE的表达形式，再转化成FA这样的数学模型，最后基于FA编写程序，让我因前人的智慧而震撼赞叹，深刻感受到了数学的魅力与重要性。

# 附录：源代码

1. #include<iostream>
2. #include<queue>
3. #include<string>
4. #include<cstring>
5. using namespace std;
6. class LexicalAnalyzer{
7. private:
8. *// 成员变量*
9. string input;
10. queue<string> output;
11. int state;
12. int pos; int cur;
13. public:
14. *// init*
15. void init(string i){
16. pos = cur = 0; state = 1;
17. input = i + '#';
18. }
19. *// utility function*
20. char curChar() {return input[cur];}
21. char nextChar() {return input[cur+1];}
22. void fail(){
23. cout<<"lexical analysis fail!\n";
24. exit(1);
25. }
26. bool isLetter(char c){
27. return (c >= 'a' && c <= 'z')||(c >= 'A' && c <= 'Z');
28. }
29. bool isZero(char c) {return c == '0';}
30. bool isNonzeroDigit(char c){return c >= '1' && c <= '9';}
31. bool isDigit(char c) {return isZero(c) || isNonzeroDigit(c);}
32. bool isPoint(char c) {return c == '.';}
33. bool isOperator(char c){
34. return (c == '+') || (c == '-') || (c == '\*') || (c == '/');
35. }
36. *// get next token*
37. bool anaIdentifier(){
38. cur = pos;
39. char curc = curChar();
40. while(1){
41. switch(state){
42. case 1:
43. ++cur; state = 2;
44. break;
45. case 2:
46. curc = curChar();
47. if(!(isLetter(curc) || isDigit(curc))){
48. output.push("<identifier>");
49. pos = cur;
50. state = 1;
51. return true;
52. }
53. else{
54. ++cur; break;
55. }
56. default:
57. state = 1; cur = pos; return false;
58. }
59. }
60. }
61. bool anaInteger(){
62. cur = pos;
63. char curc = curChar();
64. while(1){
65. switch(state){
66. case 1:
67. ++cur; state = 3;
68. break;
69. case 3:
70. if(!isDigit(curChar())){
71. if(isPoint(curChar())){
72. state = 1;
73. cur = pos;
74. return false;
75. }
76. output.push("<integer>");
77. pos = cur;
78. state = 1;
79. return true;
80. }
81. else{
82. ++cur; break;
83. }
84. default:
85. state = 1; cur = pos; return false;
86. }
87. }
88. }
89. bool anaFloat(){
90. cur = pos;
91. char curc = curChar();
92. while(1){
93. switch(state){
94. case 1:
95. if(isZero(curChar())) state = 6;
96. else state = 5;
97. ++cur;
98. break;
99. case 5:
100. if(isDigit(curChar())){
101. ++cur; break;
102. }
103. else if(isPoint(curChar())){
104. state = 7; ++cur; break;
105. }
106. else {
107. state = 1; cur = pos; return false;
108. }
109. case 6:
110. if(isPoint(curChar())){
111. state = 7; ++cur; break;
112. }
113. else {
114. state = 1; cur = pos; return false;
115. }
116. case 7:
117. if(isDigit(curChar())){
118. state = 8; ++cur; break;
119. }
120. else {
121. state = 1; cur = pos; return false;
122. }
123. case 8:
124. if(!isDigit(curChar())){
125. output.push("<float>");
126. pos = cur;
127. state = 1;
128. return true;
129. }
130. else{
131. ++cur; break;
132. }
133. default:
134. state = 1; cur = pos; return false;
135. }
136. }
137. }
138. bool anaOperator(){
139. cur = pos;
140. char curc = curChar();
141. while(1){
142. switch(state){
143. case 1:
144. ++cur; state = 9; break;
145. case 9:
146. output.push("<operator>");
147. pos = cur;
148. state = 1;
149. return true;
150. default:
151. state = 1; cur = pos; return false;
152. }
153. }
154. }
156. *// analysis*
157. void analysis(){
158. while(curChar() != '#'){
159. if(isLetter(curChar())){
160. if(anaIdentifier()) continue;
161. };
162. if(isNonzeroDigit(curChar())){
163. if(anaFloat()) continue;
164. if(anaInteger()) continue;
165. }
166. if(isZero(curChar())){
167. if(anaFloat()) continue;
168. }
169. if(isOperator(curChar())){
170. if(anaOperator()) continue;
171. }
172. fail();
173. }
174. }
175. *// print result*
176. void printOutput(){
177. cout<<"the result of lexical analysis is:\n";
178. while(!output.empty()){
179. cout<<output.front()<<" "; output.pop();
180. }
181. cout<<"\n\n";
182. }
183. };
184. int main(){
185. LexicalAnalyzer la;
186. string input;
187. while(true){
188. cout<<"please input your string:\n";
189. cin>>input;
190. la.init(input);
191. la.analysis();
192. la.printOutput();
193. }
194. }