

1202
20123852

January 27, 2016

1

2

3

- 1.
2. Token
- 3.

4

C++ Tokengcc 4.9.2 Ubuntu 12.04 LTS
LexicalAnalyzer 1

```
1 class LexicalAnalyzer
2 {
3 public:
4     LexicalAnalyzer() = default;
5     LexicalAnalyzer(const string& keywordFilePath, const
        string& symbolFilePath);
6     ~LexicalAnalyzer() = default;
7
8     void showKeywordAndSymbol();
9
10    void LexicalAnalyze(const string& srcCodeFilePath);
```

```

11 void readID(FILE* fp, char& ch);
12 void readCh(FILE* fp, char& ch);
13 void readStr(FILE* fp, char& ch);
14 void readConst(FILE* fp, char& ch);
15 void readSymbol(FILE* fp, char& ch);
16 void readSpaceTabNewline(FILE* fp, char& ch);
17
18 void showTokenSeq();
19 void showIDTab();
20 void showChTab();
21 void showStrTab();
22 void showConstTab();
23 void showKeywordTab();
24 void showSymbolTab();
25
26
27 private:
28     vector<Token> tokenSeq;
29
30     vector<string> idTab;
31     // (0, script in idTab)
32
33     vector<char> chTab; // dynamic
34     // pay attention to \n \t \\ \' \"
35     // (1, script in chTab)
36     vector<string> strTab; // dynamic
37     // (2, script in strTab)
38     vector<double> constTab; // dynamic
39     // (3, script in constTab)
40
41     vector<string> keywordTab; // static
42     // (script in keywordTab + 4, 0)
43     // read it in readID function
44     vector<string> symbolTab; // static
45     // (script in symbolTab + keywordTab.size() + 4, 0)
46     // read it in default condition
47
48     Trie keywordTrie;
49     Trie symbolTrie;
50 };

```

Listing 1: LexicalAnalyzer

show LexicalAnalyze read readID readCh readStr readConst readSymbol
readSpaceTabNewline

—DFA, Definite Finite Automaton Lex Flex3

1. if-else
 $nkDFA_n * k$ if-else DFA if-else
 2. switch-case
if-else switch-case case DFA
 3. DFA
if-else switch-case DFADFADFADFA
- C FILE* C++ ifstream

4.1 Token

Token 2

```

1 class Token
2 {
3 public:
4     Token() = default;
5     Token(int myType, int myVal);
6     ~Token() = default;
7
8     int type;
9     int val;
10
11 private:
12
13 };

```

Listing 2: Token

```

type val idTab chTab strTab constTab keywordTab symbolTab
Token Token tokenSeq idTab chTab strTab constTab tokenSeq keywordTab
symbolTab 1

```

	Token
idTab	(0, index in idTab)
chTab	(1, index in chTab)
strTab	(2, index in strTab)
constTab	(3, index in constTab)
keywordTab	(index in keywordTab + 4, 0)
symbolTab	(index in symbolTab + keywordTab.size() + 4, 0)

Table 1:

```

C++ keyword.txt symbol.txt
keyword.txt 3

```

```

1 22
2 bool
3 break
4 case
5 char
6 continue
7 default
8 do
9 double
10 else
11 false
12 float
13 for
14 if
15 int
16 long
17 return
18 short
19 switch
20 true
21 unsigned
22 void
23 while

```

Listing 3: keyword.txt

symbol.txt 4
DFAkDFADFA readSymbol

```

1 44
2 (
3 )
4 [
5 ]
6 {
7 }
8 +
9 -
10 *
11 /
12 ++
13 --
14 =
15 ==
16 +=

```

```

17 -=
18 *=
19 /=
20 >
21 <
22 >=
23 <=
24 >>
25 <<
26 >>=
27 <<=
28 &
29 &=
30 &&
31 |
32 |=
33 ||
34 ^
35 ^=
36 !
37 !=
38 ~
39 //
40 /*
41 */
42 ?
43 :
44 ;
45 ,

```

Listing 4: symbol.txt

LexicalAnalyze DFA read DFADFA

4.2 readID

5

```

1 void LexicalAnalyzer::readID(FILE* fp, char& ch)
2 // automaton using if-else sentence
3 {
4     string IDOrKeyword;
5     while(isdigit(ch) || ch == '_' || isalpha(ch))
6     {
7         IDOrKeyword.push_back(ch);

```

```

8     ch = fgetc(fp);
9 }
10 // cout << IDOrKeyword << endl;
11 auto keywordTabPtr = find(keywordTab.begin(),
12     keywordTab.end(), IDOrKeyword);
13 auto idTabPtr = find(idTab.begin(), idTab.end(),
14     IDOrKeyword);
15 if(keywordTabPtr != keywordTab.end()) // is keyword
16 {
17     tokenSeq.push_back(Token(keywordTabPtr -
18     keywordTab.begin() + 4, 0));
19 }
20 else // is ID
21 {
22     if(idTabPtr != idTab.end()) // exists in idTab
23     {
24         tokenSeq.push_back(Token(0, idTabPtr - idTab.
25         begin()));
26     }
27     else // needs to be added to idTab
28     {
29         idTab.push_back(IDOrKeyword);
30         tokenSeq.push_back(Token(0, idTab.size() - 1));
31     }
32 }
33 }
34 }

```

Listing 5: readID

if-else DFADFA 1

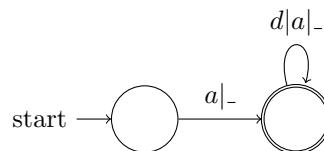


Figure 1: DFA

4.3 readCh

6

```

1 void LexicalAnalyzer::readCh(FILE* fp, char& ch)
2 {

```

```

3   Token token;
4   token.type = 1;
5   ch = fgetc(fp);
6   if(ch == '\\')
7   {
8       char __ch__;
9       __ch__ = fgetc(fp);
10      switch(__ch__)
11      {
12          case 'n': ch = '\n'; break;
13          case 't': ch = '\t'; break;
14          case '\\': ch = '\\'; break;
15          case '\': ch = '\'; break;
16          case '\"': ch = '\"'; break;
17      }
18  }
19
20  // auto chTabPtr = find(chTab.begin(), chTab.end(),
21  // ch);
22  // if(chTabPtr != chTab.end())
23  // {
24  //     token.val = chTabPtr - chTab.begin();
25  //     tokenSeq.push_back(token);
26  // }
27  // else
28  // {
29  //     chTab.push_back(ch);
30  //     token.val = chTab.size() - 1;
31  //     tokenSeq.push_back(token);
32  // }
33
34  chTab.push_back(ch);
35  token.val = chTab.size() - 1;
36  tokenSeq.push_back(token);
37
38  // switch(ch)
39  // {
40  //     case '\n': cout << "\\n" << endl; break;
41  //     case '\t': cout << "\\t" << endl; break;
42  //     case '\\': cout << "\\\" << endl; break;
43  //     case '\': cout << "\\\" << endl; break;
44  //     case '\"': cout << "\\\" << endl; break;
45  //     default: cout << ch << endl;
46  // }
47
48  ch = fgetc(fp);

```

```
48     ch = fgetc(fp);
49 }
```

Listing 6: readCh

4.4 readStr

7

```
1 void LexicalAnalyzer::readStr(FILE* fp, char& ch)
2 {
3     string str;
4     ch = fgetc(fp);
5     while(ch != '\"')
6     {
7         if(ch == '\\')
8         {
9             ch = fgetc(fp);
10            switch(ch)
11            {
12                case 'n': ch = '\n'; break;
13                case 't': ch = '\t'; break;
14                case '\\': ch = '\\'; break;
15                case '\': ch = '\'; break;
16                case '\"': ch = '\"'; break;
17            }
18        }
19        str.push_back(ch);
20        ch = fgetc(fp);
21    }
22    strTab.push_back(str);
23    tokenSeq.push_back(Token(2, strTab.size() - 1));
24    // cout << str << endl;
25    ch = fgetc(fp);
26 }
```

Listing 7: readStr

4.5 readConst

8


```

1 void LexicalAnalyzer::readConst(FILE* fp, char& ch)
2 // automaton whose shift function is embedded in code
3 {
4     int state = 0;
5     double d = 0.0;
6     int n = 0;
7     int p = 0;
8     int m = 0;
9     int e = 1;
10
11 while(1)
12 {
13     switch(state)
14     {
15         // else default condition err
16
17         case 0:
18             n = 0;
19             p = 0;
20             m = 0;
21             e = 1;
22             if(isdigit(ch))
23             {
24                 state = 1;
25                 n = 10 * n + (ch - '0');
26             }
27             else state = 9;
28             break;
29
30         case 1:
31             if(isdigit(ch)) n = 10 * n + (ch - '0');
32             else if(ch == 'e') state = 4;
33             else if(ch == '.') state = 2;
34             else state = 8;
35             break;
36
37         case 2:
38             if(isdigit(ch))
39             {
40                 state = 3;
41                 n = 10 * n + (ch - '0');
42                 ++m;
43             }
44             else state = 9;
45             break;

```

```

46
47     case 3:
48         if(isdigit(ch))
49         {
50             n = 10 * n + (ch - '0');
51             ++m;
52         }
53         else if(ch == 'e') state = 4;
54         else state = 7;
55         break;
56
57     case 4:
58         if(ch == '-') e = -1;
59         if(ch == '+' || ch == '-') state = 5;
60         else if(isdigit(ch))
61         {
62             state = 6;
63             p = 10 * p + (ch - '0');
64         }
65         else state = 9;
66         break;
67
68     case 5:
69         if(isdigit(ch))
70         {
71             state = 6;
72             p = 10 * p + (ch - '0');
73         }
74         else state = 9;
75         break;
76
77     case 6:
78         if(isdigit(ch)) p = 10 * p + (ch - '0');
79         else state = 7;
80         break;
81
82     default: break;
83 }
84 if(state != 7 && state != 8 && state != 9)
85 {
86     ch = fgetc(fp);
87 }
88 else
89 {
90     break;
91 }

```

```

92 }
93
94 if(state == 9)
95 {
96     cerr << "Error when parsing constant!" << endl;
97 }
98 else
99 {
100     d = n * pow(10, e * p - m);
101     // cout << d << endl;
102     tokenSeq.push_back(Token(3, constTab.size()));
103     constTab.push_back(d);
104 }
105 }

```

Listing 8: readConst

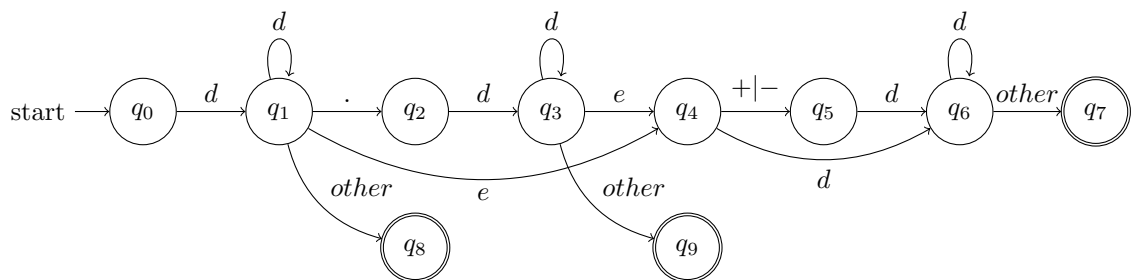


Figure 2: DFA

DFA

$\underbrace{d+}_{q_0q_1q_2q_3q_6q_7} (e(+|-)? \underbrace{d+}_{q_3q_4q_5q_6})?)?)?$

$q_0q_1q_2q_3q_6q_7$ 2
 $N_{pmnum} = N \times 10^{e \times p - m}$
 switch-case DFA case

4.6 readSymbol

9

```

1 void LexicalAnalyzer::readSymbol(FILE* fp, char& ch)
2 // use trie to replace lookahead strategy which might
3   cause uncertainty
4 {
5     // cout << "current character: " << ch << endl;

```

q_0	$N \leftarrow 0, p \leftarrow 0, e \leftarrow 1, m \leftarrow 0$
q_1	$N \leftarrow 10 \times N + [d]$
q_2	
q_3	$m \leftarrow m + 1, N \leftarrow 10 \times N + [d]$
q_4	
q_5	$e \leftarrow 1, if'' - "$
q_6	$p \leftarrow 10 \times p + [d]$
q_7	$num \leftarrow N \times 10^{e \times p - m}$
q_8	<i>err</i>
q_9	<i>err</i>

Table 2:

```

5  symbolTrie.resetCurPtr();
6  string symbol;
7  while(symbolTrie.match(ch))
8  {
9      symbol.push_back(ch);
10     ch = fgetc(fp);
11 }
12 tokenSeq.push_back(
13     Token(find(symbolTab.begin(), symbolTab.end(),
14         symbol) - symbolTab.begin() + keywordTab.size() +
15         4, 0)
16 );
17 // cout << "symbol identified: " << symbol << endl
18 // << endl;
19 }

```

Listing 9: readSymbol

Trie C++ DFADFATriematchterminalTrie 7

4.7 readSpaceTabNewline

10

```

1  void LexicalAnalyzer::readSpaceTabNewline(FILE* fp,
2      char& ch)
3  {
4      while(ch == ' ' || ch == '\t' || ch == '\n')
5      {
6          ch = fgetc(fp);
7      }
8  }

```

5

6

test.cpp 11

```

1  int f(int i)
2  {
3      /*
4      an interesting function
5      */
6      ++i;
7      return i;
8  }
9
10 void g(int& i)
11 {
12     --i;
13 }
14
15 int main()
16 {
17     int x_1_x, __y22, z__123_;
18     x_1_x=2*(3+(5/2)-(4<<2));
19     __y22 = ~3 + (3 | 5) + (2 ^ 4) + (0 & 1);
20     z__123_+=x_1_x;
21     z__123_-=__y22;
22     z__123_*=z__123_;
23     z__123_/=z__123_;
24
25     char _a1_      ='a';
26     char b__22     = '\n';
27     char __3_333___= '\t';
28     char _a_       = '\\';
29     char aa        = '\ ' ;
30     char bbb       = '\"';
31     char s[] = "abc\n\t\\\ ' \"-=~!@#$$%^&*()_+";
32     double d=-3.14e-10;
33

```

```

34  bool b1=true;
35  bool b2=false;
36  int j=2;
37  int a[10];
38
39  // what the fuck
40
41  for(int i=0;i<10;++i,b1=~b1,b2=b2&2)
42  {
43      a[i]=i;
44
45      switch(i)
46      {
47          case 0:i*=2;break;
48          case 1:j <= 3; break;
49          case 2:j=f(i);break;
50          case 3: g(j); break;
51          default:--j;break;
52      }
53
54      if(i<=5&&j!=2)
55      {
56          continue;
57      }
58      else
59      {
60          if(j >= 0)
61          {
62              b1 = false;
63          }
64      }
65  }
66
67  return 0;
68 }

```

Listing 11: test.txt

7 Trie

Trie 7
terminalterminal
Trie 12

```

zephyr@ubuntu: ~/code/cpp/lexical-analyzer
zephyr@ubuntu:~/code/cpp/lexical-analyzer 90x25
zephyr@ubuntu:~$ cd code/cpp/lexical-analyzer/
zephyr@ubuntu:~/code/cpp/lexical-analyzer$ ./lexical-analyzer
[keyword trie] preorder traverse: $ b o o l (terminal) r e a k (terminal) c a s e (terminal)
h a r (terminal) o n t i n u e (terminal) d e f a u l t (terminal) o (terminal) u b l e (terminal)
e l s e (terminal) f a l s e (terminal) l o a t (terminal) o r (terminal) i f (terminal) n
t (terminal) l o n g (terminal) r e t u r n (terminal) s h o r t (terminal) w i t c h (terminal)
) t r u e (terminal) u n s i g n e d (terminal) v o i d (terminal) w h i l e (terminal)
[symbol trie] preorder traverse: $ ! (terminal) = (terminal) & (terminal) & (terminal) = (terminal)
((terminal) ) (terminal) * (terminal) / (terminal) = (terminal) + (terminal) + (terminal) =
(terminal) , (terminal) - (terminal) - (terminal) = (terminal) / (terminal) * (terminal) / (terminal)
= (terminal) : (terminal) ; (terminal) < (terminal) < (terminal) = (terminal) = (terminal) =
(terminal) = (terminal) > (terminal) = (terminal) > (terminal) = (terminal) ? (terminal) [(terminal)
] (terminal) ^ (terminal) = (terminal) { (terminal) } (terminal) = (terminal) { (terminal) }
(terminal) ~ (terminal)

```

Figure 3: 1keywordTriesymbolTrie

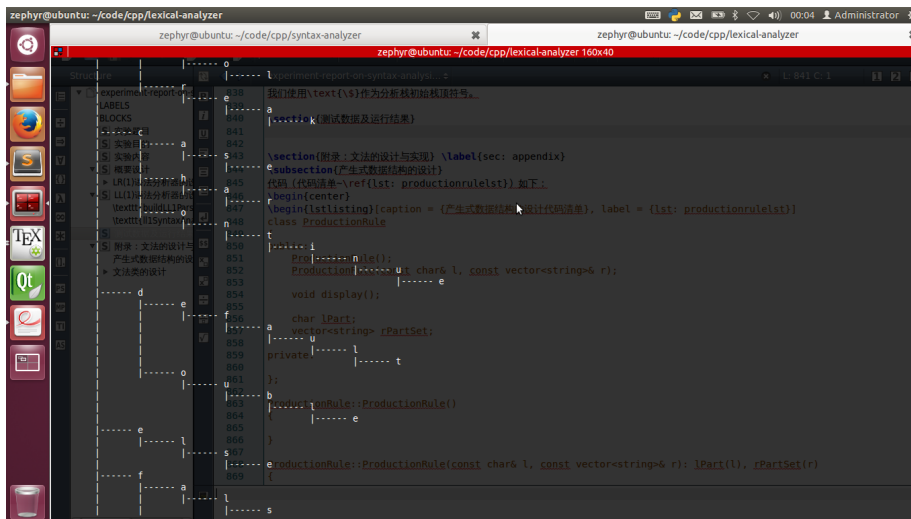


Figure 4: 2keywordTriesymbolTrie

```

1 class Trie
2 {
3 public:
4     Trie();
5     ~Trie();
6     void destruct(Node* p);
7     void insert(const string& str);
8     void preorder_traverse(Node* p);
9     void postorder_traverse(Node* p);
10    void preorderTraverse();
11    void postorderTraverse();

```

```

token sequence: (17,0)(0,0)(26,0)(17,0)(0,1)(27,0)(30,0)(64,0)(0,2)(0,3)(0,4)(65,0)(36,0)(
0,1)(68,0)(19,0)(0,1)(68,0)(31,0)(24,0)(0,5)(26,0)(17,0)(52,0)(0,1)(27,0)(30,0)(37,0)(0,1)
(68,0)(31,0)(17,0)(0,6)(26,0)(27,0)(30,0)(17,0)(0,7)(69,0)(0,8)(69,0)(0,9)(68,0)(0,7)(38,0)
(3,0)(34,0)(26,0)(3,1)(32,0)(26,0)(3,2)(35,0)(3,3)(27,0)(33,0)(26,0)(3,4)(49,0)(3,5)(27,0)
(27,0)(68,0)(0,8)(38,0)(62,0)(3,6)(32,0)(26,0)(3,7)(55,0)(3,8)(27,0)(32,0)(26,0)(3,9)(58,
0)(3,10)(27,0)(32,0)(26,0)(3,11)(52,0)(3,12)(27,0)(68,0)(0,9)(40,0)(0,7)(68,0)(0,9)(41,0)(
0,8)(68,0)(0,9)(42,0)(0,9)(68,0)(0,9)(43,0)(0,9)(68,0)(7,0)(0,10)(38,0)(1,0)(68,0)(7,0)(0,
11)(38,0)(1,1)(68,0)(7,0)(0,12)(38,0)(1,2)(68,0)(7,0)(0,13)(38,0)(1,3)(68,0)(7,0)(0,14)(38
,0)(1,4)(68,0)(7,0)(0,15)(38,0)(1,5)(68,0)(7,0)(0,16)(28,0)(29,0)(38,0)(2,0)(68,0)(11,0)(0
,17)(38,0)(33,0)(3,13)(68,0)(4,0)(0,18)(38,0)(22,0)(68,0)(4,0)(0,19)(38,0)(13,0)(68,0)(17,
0)(0,20)(38,0)(3,14)(68,0)(17,0)(0,21)(28,0)(3,15)(29,0)(68,0)(63,0)(0,22)(0,23)(0,24)(15,
0)(26,0)(17,0)(0,1)(38,0)(3,16)(68,0)(0,1)(45,0)(3,17)(68,0)(36,0)(0,1)(69,0)(0,18)(38,0)(
62,0)(0,18)(69,0)(0,19)(38,0)(0,19)(52,0)(3,18)(27,0)(30,0)(0,21)(28,0)(0,1)(29,0)(38,0)(0
,1)(68,0)(21,0)(26,0)(0,1)(27,0)(30,0)(6,0)(3,19)(67,0)(0,1)(42,0)(3,20)(68,0)(5,0)(68,0)(
6,0)(3,21)(67,0)(0,20)(51,0)(3,22)(68,0)(5,0)(68,0)(6,0)(3,23)(67,0)(0,20)(38,0)(0,0)(26,0)
(0,1)(27,0)(68,0)(5,0)(68,0)(6,0)(3,24)(67,0)(0,5)(26,0)(0,20)(27,0)(68,0)(5,0)(68,0)(9,0
)(67,0)(37,0)(0,20)(68,0)(5,0)(68,0)(31,0)(16,0)(26,0)(0,1)(47,0)(3,25)(54,0)(0,20)(61,0)(
3,26)(27,0)(30,0)(8,0)(68,0)(31,0)(12,0)(30,0)(16,0)(26,0)(0,20)(46,0)(3,27)(27,0)(30,0)(0
,18)(38,0)(13,0)(68,0)(31,0)(31,0)(31,0)(19,0)(3,28)(68,0)(31,0)

```

Figure 5: 3Token

```

id table: f i an interesting function g main x_1 x_ y22 z_ 123_ _a1_ b_ 22_ 3_333_ _a_
aa bbb s d bl b2 j a what the fuck
character table: a
string table: abc
keyword table: bool break case char continue default do double else false float for if int
long return short switch true unsigned void while
symbol table: ( ) [ ] { } + - * / ++ -- = == += -= *= /= > < >= <= >> << >>= <<= & &= && |
|= || ^ ^= ! != ~ // /* */ ? : ; ,

```

Figure 6: 4

```

12 void resetCurPtr();
13 void preorder_traverse_display(vector<int>& stk,
    Node* p, bool b1);
14 void display();
15
16 bool match(char ch);
17
18 Node* curPtr;
19 Node* root;
20
21 private:
22
23 };

```

Listing 12: Trie

Node 13

```

1 class Node

```

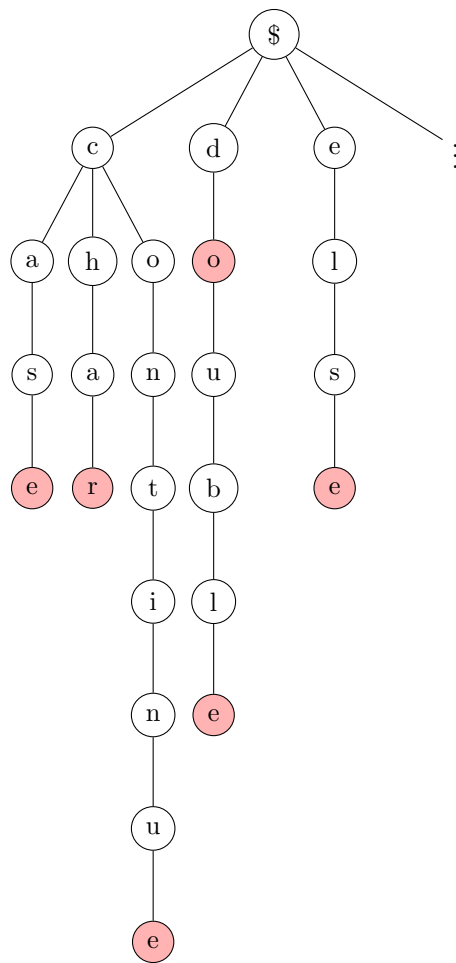



Figure 7: Trie

```

2 {
3 public:
4     char val;
5     vector<Node*> sons;
6     bool terminal;
7
8     Node();
9     Node(const char& ch);
10
11 private:
12
13 };

```

Listing 13: Node

```
terminal
Trieinsertmatch insert match
```

7.1

14

```
1 Trie::Trie()
2 {
3     root = new Node('$');
4     curPtr = root;
5 }
```

Listing 14:

```
$ Node
```

7.2 insert

15

```
1 void Trie::insert(const string& str)
2 {
3     bool existed = false;
4     Node* ptr = root;
5     auto itr = str.begin();
6     bool found = true;
7     int mid = 0;
8
9     while(found)
10    {
11        found = false;
12        mid = 0;
13
14        int low = 0;
15        int high = ptr->sons.size() - 1;
16        while(low <= high)
17        {
18            mid = (low + high) / 2;
19            if(ptr->sons[mid]->val == *itr)
20            {
21                found = true;
```

```

22         ptr = ptr->sons[mid];
23         ++itr;
24
25         if(itr == str.end())
26         {
27             existed = true;
28             goto next;
29         }
30
31         goto result;
32     }
33     else if(ptr->sons[mid]->val < *itr) low = mid +
34 1;
35     else high = mid - 1;
36 }
37
38 result:
39 if(!found)
40 {
41     break;
42 }
43
44 next:
45 if(!existed)
46 {
47     if(ptr->sons.empty())
48     {
49         mid = 0;
50     }
51     else if(ptr->sons[mid]->val < *itr)
52     {
53         ++mid;
54     }
55
56     ptr->sons.insert(ptr->sons.begin() + mid, new Node
57 (*itr));
58     if(itr == str.end() - 1)
59     {
60         ptr->sons[mid]->terminal = true;
61     }
62     ptr = ptr->sons[mid];
63     ++itr;
64
65     while(itr != str.end())
66     {

```

```

66     ptr->sons.push_back(new Node(*itr));
67     if(itr == str.end() - 1)
68     {
69         ptr->sons[ptr->sons.size() - 1]->terminal =
70         true;
71     }
72     ptr = ptr->sons[ptr->sons.size() - 1];
73     ++itr;
74 }
75 else
76 {
77     ptr->terminal = true;
78 }
79 }

```

Listing 15: insert

terminal

7.3 match

16

```

1 bool Trie::match(char ch)
2 {
3     for(size_t i = 0; i < curPtr->sons.size(); ++i)
4     {
5         if(curPtr->sons[i]->val == ch)
6         {
7             curPtr = curPtr->sons[i];
8             return true;
9         }
10    }
11    return false;
12 }

```

Listing 16: match

terminal