1202
20123852

January 27, 2016

1

2

3

1.

2. `Token`

3.

4

`C++` Tokengcc 4.9.2 Ubuntu 12.04 LTS
`LexicalAnalyzer` 1

```
class LexicalAnalyzer
{
public:
  LexicalAnalyzer() = default;
  LexicalAnalyzer(const string& keywordFilePath, const
    string& symbolFilePath);
  ~LexicalAnalyzer() = default;

  void showKeywordAndSymbol();

  void LexicalAnalyze(const string& srcCodeFilePath);
```

```
11
12    void readID(FILE* fp, char& ch);
13    void readCh(FILE* fp, char& ch);
14    void readStr(FILE* fp, char& ch);
15    void readConst(FILE* fp, char& ch);
16    void readSymbol(FILE* fp, char& ch);
17    void readSpaceTabNewline(FILE* fp, char& ch);
18
19    void showTokenSeq();
20    void showIDTab();
21    void showChTab();
22    void showStrTab();
23    void shiowConstTab();
24    void showKeywordTab();
25    void showSymbolTab();
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

```
class Token
{
public:
  Token() = default;
  Token(int myType, int myVal);
  ~Token() = default;

  int type;
  int val;

private:

};
```

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

```
22
bool
break
case
char
continue
default
do
double
else
false
float
for
if
int
long
return
short
switch
true
unsigned
void
while
```

Listing 3: `keyword.txt`

`symbol.txt` 4
DFAkDFADFA `readSymbol`

```
44
(
)
[
]
{
}
+
-
*
/
++
--
=
==
+=
```

```
-=
*=
/=
>
<
>=
<=
>>
<<
>>=
<<=
&
&=
&&
|
|=
||
^
^=
!
!=
~
//
/*
*/
?
:
;
,
```

Listing 4: `symbol.txt`

`LexicalAnalyze` DFA `read` DFADFA

4.2 `readID`

5

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

```
    ch = fgetc(fp);
  }
  // cout << IDOrKeyword << endl;
  auto keywordTabPtr = find(keywordTab.begin(),
   keywordTab.end(), IDOrKeyword);
  auto idTabPtr = find(idTab.begin(), idTab.end(),
   IDOrKeyword);
  if(keywordTabPtr != keywordTab.end()) // is keyword
  {
    tokenSeq.push_back(Token(keywordTabPtr -
   keywordTab.begin() + 4, 0));
  }
  else // is ID
  {
    if(idTabPtr != idTab.end()) // exists in idTab
    {
      tokenSeq.push_back(Token(0, idTabPtr - idTab.
   begin()));
    }
    else // needs to be added to idTab
    {
      idTab.push_back(IDOrKeyword);
      tokenSeq.push_back(Token(0, idTab.size() - 1));
    }
  }
}
```

Listing 5: `readID`

`if-else` DFADFA 1

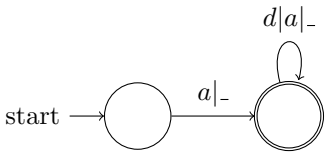

Figure 1: DFA

4.3 `readCh`

6

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

```
  Token token;
  token.type = 1;
  ch = fgetc(fp);
  if(ch == '\\')
  {
    char __ch__;
    __ch__ = fgetc(fp);
    switch(__ch__)
    {
      case 'n': ch = '\n'; break;
      case 't': ch = '\t'; break;
      case '\\': ch = '\\'; break;
      case '\'': ch = '\''; break;
      case '\"': ch = '\"'; break;
    }
  }

  // auto chTabPtr = find(chTab.begin(), chTab.end(),
   ch);
  // if(chTabPtr != chTab.end())
  // {
  //  token.val = chTabPtr - chTab.begin();
  //  tokenSeq.push_back(token);
  // }
  // else
  // {
  //  chTab.push_back(ch);
  //  token.val = chTab.size() - 1;
  //  tokenSeq.push_back(token);
  // }

  chTab.push_back(ch);
  token.val = chTab.size() - 1;
  tokenSeq.push_back(token);

  // switch(ch)
  // {
  //  case '\n': cout << "\\n" << endl; break;
  //  case '\t': cout << "\\t" << endl; break;
  //  case '\\': cout << "\\\\" << endl; break;
  //  case '\'': cout << "\\\'" << endl; break;
  //  case '\"': cout << "\\\"" << endl; break;
  //  default: cout << ch << endl;
  // }

  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

```
void LexicalAnalyzer::readConst(FILE* fp, char& ch)
// automaton whose shift function is embedded in code
{
  int state = 0;
  double d = 0.0;
  int n = 0;
  int p = 0;
  int m = 0;
  int e = 1;

  while(1)
  {
    switch(state)
    {
      // else default condition err

      case 0:
      n = 0;
      p = 0;
      m = 0;
      e = 1;
      if(isdigit(ch))
      {
        state = 1;
        n = 10 * n + (ch - '0');
      }
      else state = 9;
      break;

      case 1:
      if(isdigit(ch)) n = 10 * n + (ch - '0');
      else if(ch == 'e') state = 4;
      else if(ch == '.') state = 2;
      else state = 8;
      break;

      case 2:
      if(isdigit(ch))
      {
        state = 3;
        n = 10 * n + (ch - '0');
        ++m;
      }
      else state = 9;
      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`

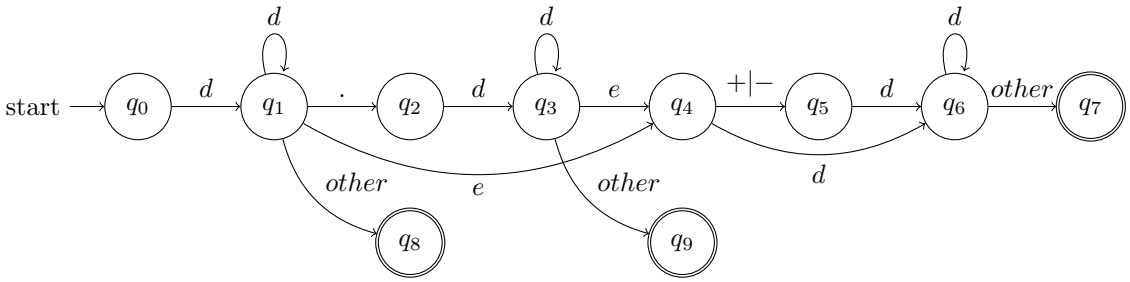

Figure 2: DFA

DFA

`d+(.d+(e(+|-)?d+)?)?`

$q_0q_1q_2q_3q_6q_7$ 2

$Npmnum = N \times 10^{e \times p - m}$

`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
       cause uncertainty
3  {
4    // 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	err
q_9	err

Table 2:

```
  symbolTrie.resetCurPtr();
  string symbol;
  while(symbolTrie.match(ch))
  {
    symbol.push_back(ch);
    ch = fgetc(fp);
  }
  tokenSeq.push_back(
    Token(find(symbolTab.begin(), symbolTab.end(),
   symbol) - symbolTab.begin() + keywordTab.size() +
   4, 0)
    );
  // cout << "symbol identified: " << symbol << endl
   << endl;
}
```

Listing 9: `readSymbol`

Trie C++ DFADFATriematchterminalTrie 7

4.7 readSpaceTabNewline

10

```
void LexicalAnalyzer::readSpaceTabNewline(FILE* fp,
    char& ch)
{
  while(ch == ' ' || ch == '\t' || ch == '\n')
  {
    ch = fgetc(fp);
  }
}
```

Listing 10: `readSpaceTabNewline`

5

6

`test.cpp` 11

```
int f(int i)
{
  /*
  an interesting function
  */
  ++i;
  return i;
}

void g(int& i)
{
  --i;
}

int main()
{
  int x_1_x,__y22,z__123_;
  x_1_x=2*(3+(5/2)-(4<<2));
  __y22 = ~3 + (3 | 5) + (2 ^ 4) + (0 & 1);
  z__123_+=x_1_x;
  z__123_-=__y22;
  z__123_*=z__123_;
  z__123_/=z__123_;

  char _a1_        ='a';
  char b__22       = '\n';
  char __3_333___  = '\t';
  char _a_         = '\\';
  char aa          = '\'';
  char bbb         = '\"';
  char s[] = "abc\n\t\\\'\"-=~!@#$%^&*()_+";
  double d=-3.14e-10;

```

```
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

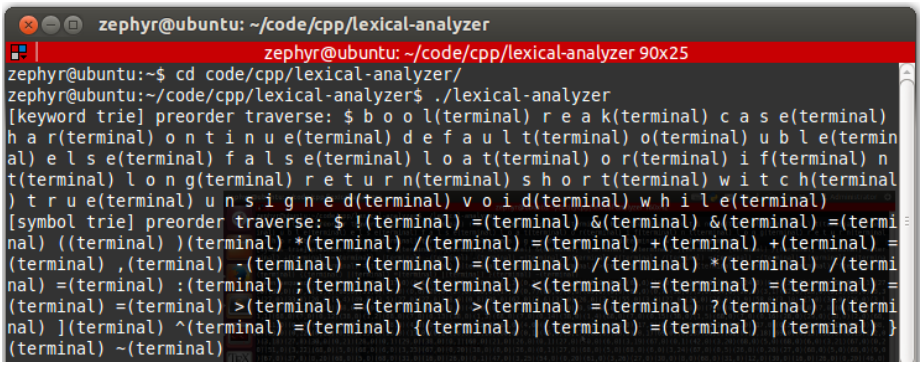

Figure 3: 1keywordTriesymbolTrie

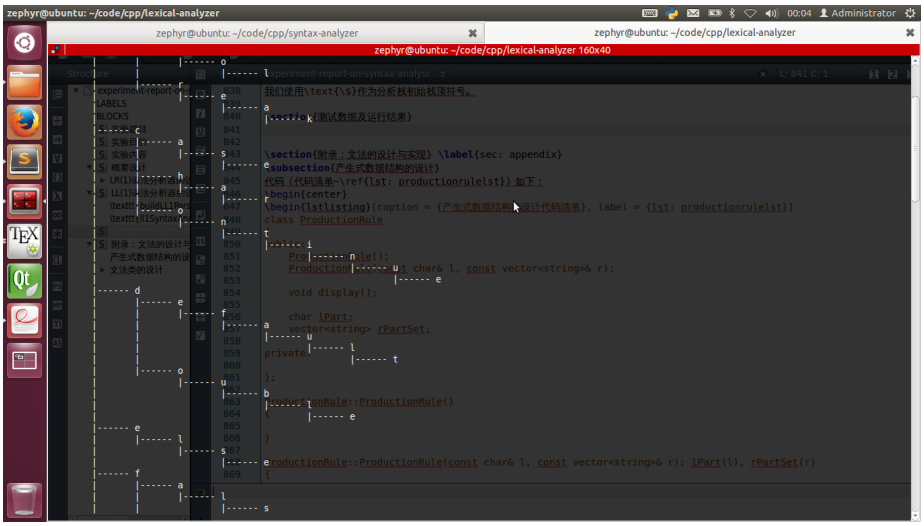

Figure 4: 2keywordTriesymbolTrie

```
class Trie
{
public:
  Trie();
  ~Trie();
  void destruct(Node* p);
  void insert(const string& str);
  void preorder_traverse(Node* p);
  void postorder_traverse(Node* p);
  void preorderTraverse();
  void postorderTraverse();
```

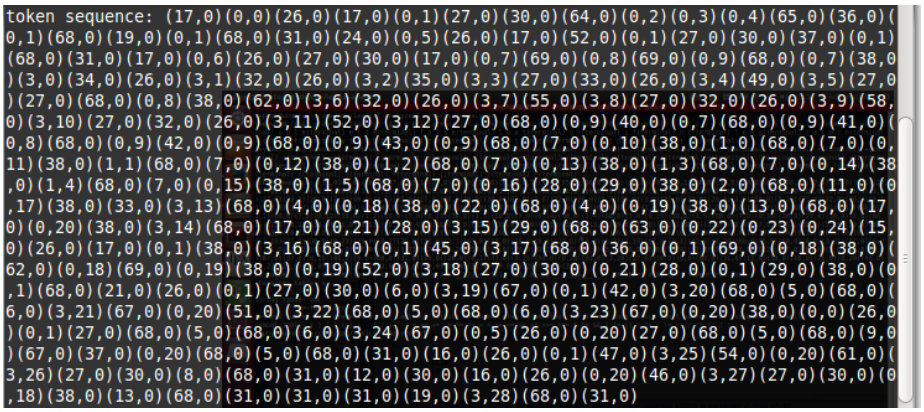

Figure 5: 3Token

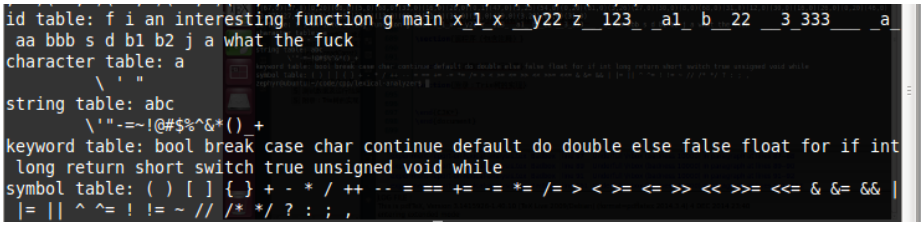

Figure 6: 4

```
  void resetCurPtr();
  void preorder_traverse_display(vector<int>& stk,
   Node* p, bool b1);
  void display();

  bool match(char ch);

  Node* curPtr;
  Node* root;

private:

};
```

Listing 12: `Trie`

Node 13

```
class Node
```

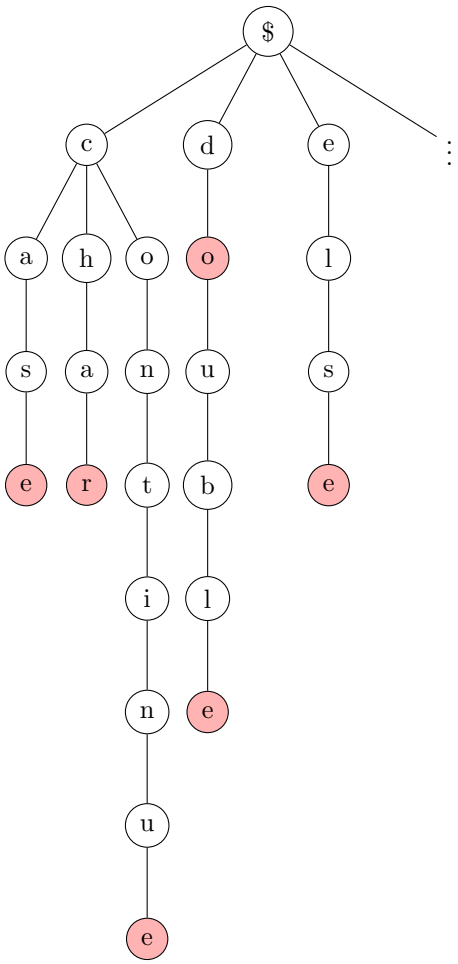

Figure 7: Trie

```
{
public:
  char val;
  vector<Node*> sons;
  bool terminal;

  Node();
  Node(const char& ch);

private:

};
```

Listing 13: `Node`

terminal
Trie`insertmatch insert match`

7.1

14

```
Trie::Trie()
{
  root = new Node('$');
  curPtr = root;
}
```

Listing 14:

`$ Node`

7.2 `insert`

15

```
void Trie::insert(const string& str)
{
  bool existed = false;
  Node* ptr = root;
  auto itr = str.begin();
  bool found = true;
  int mid = 0;

  while(found)
  {
    found = false;
    mid = 0;

    int low = 0;
    int high = ptr->sons.size() - 1;
    while(low <= high)
    {
      mid = (low + high) / 2;
      if(ptr->sons[mid]->val == *itr)
      {
        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 +
    1;
34        else high = mid - 1;
35      }
36
37      result:
38      if(!found)
39      {
40        break;
41      }
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
    (*itr));
57      if(itr == str.end() - 1)
58      {
59        ptr->sons[mid]->terminal = true;
60      }
61      ptr = ptr->sons[mid];
62      ++itr;
63
64      while(itr != str.end())
65      {
```

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