

词法分析、语法分析程序实验

词法分析、语法分析程序实验

实验目的

实验内容

实验要求

我所实现的扩展内容

实验步骤

词法分析器

(扫描阶段) 双输入缓冲区

(分析阶段)分析器

Token

字符处理

语法分析器

递归下降分析程序

错误分析

实验结果

运行方法

测试样例

测试结果

2019.11.12

此为C++版本

c++程序请用g++ c++11标准编译

(推荐到Github查看, 因为Github对markdown中代码换行处理较好)

[Github地址](#)

2019.11.30 Update

C语言版本

对比C++实现有以下改进:

- 边语法分析边词法分析, 避免使用过大的空间来存储token

其他方面基本一致

实验目的

扩充已有的样例语言TINY, 为扩展TINY语言TINY+构造词法分析和语法分析程序, 从而掌握词法分析和语法分析的构造方法。

实验内容

了解样例语言TINY及TINY编译器的实现, 了解扩展TINY语言TINY+, 用EBNF描述TINY+的语法, 用C语言扩展TINY的词法分析和语法分析程序, 构造TINY+的递归下降语法分析器。

实验要求

将TINY+源程序翻译成对应的TOKEN序列，并能检查一定的词法错误。将TOKEN序列转换成语法分析树，并能检查一定的语法错误

我所实现的扩展内容

由于我所做的扩展不多，EBNF基本与原来的TINY相同，WhileStmt和IfStmt基本一致，BoolExpression多了四组

- STRING 作为新的关键词，同时能创建STRING类型变量
- WHILE 作为新的关键词，表示循环结构
- 扩展Operator，比如 <、>、<=、>=

```
1 Program -> MethodDecl MethodDecl*
2 MethodDecl -> Type [MAIN] Id '(' FormalParams ')' Block
3 FormalParams -> [FormalParam ( ',' FormalParam )* ]
4 FormalParam -> Type Id
5 Type -> INT | REAL | STRING
6
7 Block -> BEGIN Statement* END
8
9 Statement -> Block
10             | LocalVarDecl
11             | AssignStmt
12             | ReturnStmt
13             | IfStmt
14             | WhileStmt
15             | WriteStmt
16             | ReadStmt
17 LocalVarDecl -> INT Id ';' | REAL Id ';'
18 AssignStmt -> Id := Expression ';'
19 ReturnStmt -> RETURN Expression ';'
20 IfStmt -> IF '(' BoolExpression ')' Statement
21         | IF '(' BoolExpression ')' Statement ELSE Statement
22 WriteStmt -> WRITE '(' Expression ',' QString ')' ';'
23 ReadStmt -> READ '(' Id ',' QString ')' ';'
24 QString is any sequence of characters except double quote itself, enclosed
    in double quotes.
25
26 Expression -> MultiplicativeExpr (( '+' | '-' ) MultiplicativeExpr)*
27 MultiplicativeExpr -> PrimaryExpr (( '*' | '/' ) PrimaryExpr)*
28 PrimaryExpr -> Num // Integer or Real numbers
29             | Id
30             | '(' Expression ')'
31             | Id '(' ActualParams ')'
32 BoolExpression -> Expression '==' Expression
33                 | Expression '<' Expression
34                 | Expression '>' Expression
35                 | Expression '<=' Expression
36                 | Expression '>=' Expression
37 ActualParams -> [Expression ( ',' Expression)*]
38
```

实验步骤

词法分析器

(扫描阶段) 双输入缓冲区

在实践中，我们的却需要至少想前看一个字符。因此，我们使用一种双缓冲区方案，能够安全地处理向前看多个符号的问题，并使用“哨兵标记”来节约用于检查缓冲区末端的时间

可用于

- 获取当前所在行数、列数
- 获取当前指针所指字符，（并使指针指向下一个字符）
- 获取当前词素lexeme

```
1  #pragma once
2  #include<fstream>
3  #include<string>
4
5  #define BUF_SIZE 1024
6
7  class Scan
8  {
9  public:
10     explicit Scan(std::string filePath);
11     ~Scan() noexcept;
12
13     std::string getString();
14     void readFromFile();
15     char nextChar();
16     char curChar();
17     int getRow();
18     int getCol();
19     void nextLexeme();
20 private:
21     std::ifstream fileStream;
22
23     char* lexemeBegin;
24     char* forward;
25
26     char readBuffer1[BUF_SIZE];
27     char readBuffer2[BUF_SIZE];
28
29     enum Tag {FirstBuf, SecondBuf};
30     Tag currentBuf = Tag::FirstBuf;
31
32     int row = 1;
33     int col = 1;
34 };
```

带有哨兵标记的forward指针移动算法

```
1  void Scan::readFromFile() {
2      if (currentBuf == Tag::FirstBuf) {
3          readBuffer2[BUF_SIZE - 1] = EOF;
4          fileStream.read(readBuffer2, BUF_SIZE - 1);
5
6          currentBuf = Tag::SecondBuf;
7          forward = &readBuffer2[0];
```

```

8     }
9     else {
10         readBuffer1[BUF_SIZE - 1] = EOF;
11         fileStream.read(readBuffer1, BUF_SIZE - 1);
12
13         currentBuf = Tag::FirstBuf;
14         forward = &readBuffer1[0];
15     }
16 }
17
18 char Scan::nextChar() {
19     switch (*forward)
20     {
21     case EOF:
22         readFromFile();
23         break;
24     case '\n': //剔除回车
25         col = 1;
26         row++;
27         forward++;
28         return nextChar();
29     default:
30         col++;
31         break;
32     }
33     return *forward++;
34 }

```

(分析阶段)分析器

Token

1.枚举类型，对应Token-name

```

1  typedef enum {
2      //keywords
3      IF,
4      ELSE,
5      WHILE,
6      WRITE,
7      READ,
8      RETURN,
9      BEGIN,
10     END,
11     MAIN,
12     INT,
13     REAL,
14     STRING,
15
16     IDENTIFIER,
17     //Operator such as + - * / == := !=
18     operator,
19     //Delimiter such as ; , ( )
20     Delimiter,
21     Number,
22     String
23 }tokenType;

```

2.保存所有词法单元的结构体，包含<token-name,attribute-value>对以及所在的位置

```
1 typedef struct node{
2     tokenType type;
3     std::string content;
4     int row;
5     int col;
6     int indent;
7
8     node(tokenType _type, std::string str,int _row,int _column) {
9         type = _type;
10        content = str;
11        row = _row;
12        col = _column;
13    }
14 }tokennode;
```

3. Token类，用于输出和匹配字符串和关键词

```
1 #define KEYWORD_SIZE 12
2 #define DELIM_SIZE 4
3 #define OPT_SIZE 11
4
5 class Token {
6 public:
7     std::string keyword[KEYWORD_SIZE] = { "IF","ELSE","WHILE","WRITE",
8     "READ", "RETURN", "BEGIN", "END", "MAIN", "INT", "REAL","STRING"};
9     std::string operation[OPT_SIZE] = { "+","-","*","/","==","!=",
10    "<",">","<=",">=" };
11     std::string delimiter[DELIM_SIZE] = { ";",",","(",")" };
12     tokenType getToken(std::string lexeme);
13     std::string getTokenStr(tokenType token);
14 private:
15 };
```

字符处理

定义了以下的处理类

```
1 class lexicalAnalyzer {
2 public:
3     explicit lexicalAnalyzer(std::string filePath);
4     ~lexicalAnalyzer();
5     void analyse(); //分析，字符处理
6     void logError(std::string errMsg); //错误处理
7     void printToken(tokenType _token); //打印
8     tokennode* getTokenAt(int index); //供语法分析器调用
9     int getTokenSize(); //供语法分析器使用
10 private:
11     Token token; //用于匹配token
12     Scan input; //从输入缓冲区获取字符
13     std::vector<tokennode*> tokensStore; //保存词法单元
14 };
```

字符处理方法：

1. 检查是否 '_'/'a-z'/'A-Z'，如果是则一直读取直到其不为 '_'/'a-z'/'A-Z'/'0-9'，并将该字符串用作匹配，看是Identifier还是Keyword
2. 忽略空格和换行符
3. 比较查看是否是操作符（有可能需要向后多读一位），如果是 '/' 符号，则需要预读两位看是否是注释
4. 比较查看是否是分隔符，将其token-name设为DELIMITER
5. 当读到双引号时，继续读直到读到另一个双引号，并将其token-name设为STRING
6. 当是数字时，判断是小数还是整数，将其token-name设为NUMBER

错误检测：

当读到的字符不在意料之中（比如冒号后面没有等于号 或者 开头字符不在我们的switch..case..之中的），则报错，并显示其错误位置

5	STRING str;
6	z := 2.0.0;
7	z := x*x - y*y;

```
PS E:\VSprogram\Compiler\Debug> .\Compiler.exe > result.txt
E:\VSprogram\Compiler\Debug\sample.tiny
Row 6 Column13: Unrecognizable characters.
```

部分代码，表示分析器的大致结构

```
1  switch (current)
2  {
3  case ' ':
4  case '\t':
5      break;
6  case '=':
7      if (input.nextChar() == '=') {
8          printToken(Operator);
9          break;
10     }
11     logError("Row" + std::to_string(input.getRow()) + " Column" +
std::to_string(input.getCol()) + ": Unrecognizable characters.");
12 case ':':
13     if (input.nextChar() == '=') {
14         printToken(Operator);
15         break;
16     }
17     logError("Row" + std::to_string(input.getRow()) + " Column" +
std::to_string(input.getCol()) + ": Unrecognizable characters.");
18 case '!':
19     if (input.nextChar() == '=') {
20         printToken(Operator);
21         break;
22     }
23     logError("Row" + std::to_string(input.getRow()) + " Column" +
std::to_string(input.getCol()) + ": Unrecognizable characters.");
24 case ';':
25 case ',':
26 case '(':
27 case ')':
28     printToken(Delimiter);
29     break;
```

```

30 case '+':
31 case '-':
32 case '*':
33     printToken(Operator);
34     break;
35     ...
36 }

```

语法分析器

递归下降分析程序

一个递归下降语法分析程序有一组过程组成，每个非终结符号有一个对应的过程。程序的执行从开始符号对应的过程开始，如果这个过程的过程体扫描了整个输入串，它就停止执行并宣布语法分析成功完成。

通用的递归下降分析技术可能需要回溯，也就是说，他可能重复扫描输入，

这里通过将词法单元保存到vector中的方法来简化获取操作，缺点就是过于占用内存

首先，每一个非终结符号对应一个过程

```

1  class SyntaxAnalyzer {
2  public:
3      explicit SyntaxAnalyzer(std::string filePath);
4      void analyse();
5      void logError(std::string errMsg);
6      bool compareContent(tokennode* tempnode, std::string cmp);
7      bool compareType(tokennode* tempnode, tokenType cmp);
8      void printTree();
9      //High-level program structure
10     void program();
11     void methodDecl();
12     void type();
13     void identifier();
14     void formalParams();
15     void formalParam();
16     //statements
17     void block();
18     void statement();
19     void localVarDecl();
20     void assignStmt();
21     void returnStmt();
22     void ifStmt();
23     void whileStmt();
24     void writeStmt();
25     void readStmt();
26     //expression
27     void expression();
28     void multiplicativeExpr();
29     void primaryExpr();
30     void boolExpression();
31     void actualParams();
32 private:
33     lexicalAnalyzer input;
34     int curIndex;
35     int curDepth;
36 };

```

根据EBNF

```
1 | Program -> MethodDecl MethodDecl*
```

那么我们program过程就为

当我们进入program的过程时，如果能获取到词法单元，就执行methodDecl()过程

只有当我们检测不到词法单元的时候，我们才结束program这个过程

```
1 void SyntaxAnalyzer::program()
2 {
3     while (input.getTokenAt(curIndex) != nullptr) {
4         methodDecl();
5     }
6 }
```

而

```
1 | MethodDecl -> Type [MAIN] Id '(' FormalParams ')' Block
```

于是

根据上述原则，因为只有一个展开式，所以只需要一个一个地匹配

```
1 void SyntaxAnalyzer::methodDecl()
2 {
3     curDepth += 1;
4     type();
5     if (compareType(input.getTokenAt(curIndex) , MAIN)) {
6         input.getTokenAt(curIndex)->indent = curDepth;
7         curIndex++;
8     }
9     identifier();
10
11     if (compareContent( input.getTokenAt(curIndex++), "(")) {
12         input.getTokenAt(curIndex - 1)->indent = curDepth;
13         formalParams();
14     }
15     else {
16         logError("Missing '('");
17     }
18     if (!compareContent(input.getTokenAt(curIndex++), ")")) {
19         logError("Missing ')'");
20     }
21     input.getTokenAt(curIndex - 1)->indent = curDepth;
22     block();
23     curDepth -= 1;
24 }
```

而在statement中，我们多个展开式，


```

1 Statement -> Block
2           | LocalVarDecl
3           | AssignStmt
4           | ReturnStmt
5           | IfStmt
6           | WriteStmt
7           | ReadStmt
8

```

那么我们就需要往前读找到能匹配的展开式，再回溯（我这里用了vector，不需要回溯）

```

1 void SyntaxAnalyzer::statement() {
2     switch (input.getTokenAt(curIndex)->type)
3     {
4         case INT:
5         case REAL:
6         case STRING:
7             localVarDecl();
8             break;
9         case IDENTIFIER:
10            assignStmt();
11            break;
12        case RETURN:
13            returnStmt();
14            break;
15        case IF:
16            ifStmt();
17            break;
18        case WHILE:
19            whileStmt();
20            break;
21        case READ:
22            readStmt();
23            break;
24        case WRITE:
25            writeStmt();
26            break;
27        case BEGIN:
28            block();
29            break;
30        default:
31            curIndex++;
32            logError("Unrecognizable statement, starting with the wrong");
33            break;
34    }
35 }

```

这就是一个递归下降语法分析的例子，关于本次实验的更多代码请参考附件

错误分析

依然是显示出出错的位置和具体的出错信息

出错信息穿插在不同的过程之中，如有匹配不到的情况，则输出对应错误信息

比如

```

1 void SyntaxAnalyzer::ifStmt() {
2     if (!compareType(input.getTokenAt(curIndex++), IF))
3     {
4         logError("Missing \"IF\"");
5     }
6     input.getTokenAt(curIndex - 1)->indent = curDepth;
7     if (!compareContent(input.getTokenAt(curIndex++), "(")) {
8         logError("Missing '('");
9     }
10    input.getTokenAt(curIndex - 1)->indent = curDepth;
11    boolExpression();
12    if (!compareContent(input.getTokenAt(curIndex++), ")")) {
13        logError("Missing ')'");
14    }
15    input.getTokenAt(curIndex - 1)->indent = curDepth;
16    statement();
17    if (compareType(input.getTokenAt(curIndex), ELSE))
18    {
19        input.getTokenAt(curIndex)->indent = curDepth;
20        curIndex++;
21        statement();
22    }
23 }

```

如果匹配不到括号的话，就会输出对应的错误消息。

```

11 BEGIN
12     INT x;
13     READ(x, "A41.input";

```

```

PS E:\VSprogram\Compiler\Debug> .\Compiler.exe > result.txt
E:\VSprogram\Compiler\Debug\sample.tiny
Row 13 Col 24: Missing ')'

```

实验结果

运行方法

打开exe文件后，将tiny文件拖动到命令行窗口/输入绝对路径

测试样例

```

1 /** this is a comment line in the sample program */
2 INT f2(INT x, INT y )
3 BEGIN
4     INT z;
5     STRING str;
6     z := 2.0;
7     z := x*x - y*y;
8     RETURN z;
9 END
10 INT MAIN f1()
11 BEGIN
12     INT x;

```

```

13     READ(x, "A41.input");
14     INT y;
15     READ(y, "A42.input");
16     INT z;
17     z := f2(x,y) + f2(y,x);
18     IF(x >= y)
19     BEGIN
20         z := x + y;
21     END
22     WHILE( x >= y)
23     BEGIN
24         z := x + y;
25     END
26     WRITE (z, "A4.output");
27 END
28

```

测试结果

```

1  <Keyword   ,  INT           >
2  <IDENTIFIER,  f2           >
3  <Delimiter ,  (           >
4  <Keyword   ,  INT           >
5  <IDENTIFIER,  x            >
6  <Delimiter ,  ,           >
7  <Keyword   ,  INT           >
8  <IDENTIFIER,  y            >
9  <Delimiter ,  )           >
10 <Keyword   ,  BEGIN         >
11 <Keyword   ,  INT           >
12 <IDENTIFIER,  z            >
13 <Delimiter ,  ;            >
14 <Keyword   ,  STRING        >
15 <IDENTIFIER,  str          >
16 <Delimiter ,  ;            >
17 <IDENTIFIER,  z            >
18 <Operator  ,  :=            >
19 <Number    ,  2.0           >
20 <Delimiter ,  ;            >
21 <IDENTIFIER,  z            >
22 <Operator  ,  :=            >
23 <IDENTIFIER,  x            >
24 <Operator  ,  *            >
25 <IDENTIFIER,  x            >
26 <Operator  ,  -            >
27 <IDENTIFIER,  y            >
28 <Operator  ,  *            >
29 <IDENTIFIER,  y            >
30 <Delimiter ,  ;            >
31 <Keyword   ,  RETURN        >
32 <IDENTIFIER,  z            >
33 <Delimiter ,  ;            >
34 <Keyword   ,  END           >
35 <Keyword   ,  INT           >
36 <Keyword   ,  MAIN          >
37 <IDENTIFIER,  f1           >
38 <Delimiter ,  (           >

```

```

39 <Delimiter , ) >
40 <Keyword , BEGIN >
41 <Keyword , INT >
42 <IDENTIFIER, x >
43 <Delimiter , ; >
44 <Keyword , READ >
45 <Delimiter , ( >
46 <IDENTIFIER, x >
47 <Delimiter , , >
48 <String , "A41.input">
49 <Delimiter , ) >
50 <Delimiter , ; >
51 <Keyword , INT >
52 <IDENTIFIER, y >
53 <Delimiter , ; >
54 <Keyword , READ >
55 <Delimiter , ( >
56 <IDENTIFIER, y >
57 <Delimiter , , >
58 <String , "A42.input">
59 <Delimiter , ) >
60 <Delimiter , ; >
61 <Keyword , INT >
62 <IDENTIFIER, z >
63 <Delimiter , ; >
64 <IDENTIFIER, z >
65 <Operator , := >
66 <IDENTIFIER, f2 >
67 <Delimiter , ( >
68 <IDENTIFIER, x >
69 <Delimiter , , >
70 <IDENTIFIER, y >
71 <Delimiter , ) >
72 <Operator , + >
73 <IDENTIFIER, f2 >
74 <Delimiter , ( >
75 <IDENTIFIER, y >
76 <Delimiter , , >
77 <IDENTIFIER, x >
78 <Delimiter , ) >
79 <Delimiter , ; >
80 <Keyword , IF >
81 <Delimiter , ( >
82 <IDENTIFIER, x >
83 <Operator , >= >
84 <IDENTIFIER, y >
85 <Delimiter , ) >
86 <Keyword , BEGIN >
87 <IDENTIFIER, z >
88 <Operator , := >
89 <IDENTIFIER, x >
90 <Operator , + >
91 <IDENTIFIER, y >
92 <Delimiter , ; >
93 <Keyword , END >
94 <Keyword , WHILE >
95 <Delimiter , ( >
96 <IDENTIFIER, x >

```

97	<Operator	,	>=	>
98	<IDENTIFIER,	y		>
99	<Delimiter	,)	>
100	<Keyword	,	BEGIN	>
101	<IDENTIFIER,	z		>
102	<Operator	,	:=	>
103	<IDENTIFIER,	x		>
104	<Operator	,	+	>
105	<IDENTIFIER,	y		>
106	<Delimiter	,	;	>
107	<Keyword	,	END	>
108	<Keyword	,	WRITE	>
109	<Delimiter	,	(>
110	<IDENTIFIER,	z		>
111	<Delimiter	,	,	>
112	<String	,	"A4.output">	
113	<Delimiter	,)	>
114	<Delimiter	,	;	>
115	<Keyword	,	END	>
116	Program			
117	MethodDecl			
118	Type			
119	INT			
120	Identifier			
121	f2			
122	(
123	FormalParams			
124	FormalParam			
125	Type			
126	INT			
127	Identifier			
128	x			
129	FormalParam			
130	Type			
131	INT			
132	Identifier			
133	y			
134)			
135	Block			
136	BEGIN			
137	Statement			
138	LocalVarDecl			
139	Type			
140	INT			
141	Identifier			
142	z			
143	Statement			
144	LocalVarDecl			
145	Type			
146	STRING			
147	Identifier			
148	str			
149	Statement			
150	assignStmt			
151	Identifier			
152	z			
153	:=			
154	Expression			

```

155                                     MultiplicativeExpr
156                                     PrimaryExpr
157                                     2.0
158
159             Statement
160             assignStmt
161             Identifier
162             z
163             :=
164             Expression
165             MultiplicativeExpr
166             PrimaryExpr
167             Identifier
168             x
169             *
170             PrimaryExpr
171             Identifier
172             x
173             -
174             MultiplicativeExpr
175             PrimaryExpr
176             Identifier
177             y
178             *
179             PrimaryExpr
180             Identifier
181             y
182
183             Statement
184             ReturnStmt
185             RETURN
186             Expression
187             MultiplicativeExpr
188             PrimaryExpr
189             Identifier
190             z
191
192     END
193 MethodDecl
194     Type
195     INT
196     MAIN
197     Identifier
198     f1
199     (
200     FormalParams
201     )
202     Block
203     BEGIN
204         Statement
205         LocalVarDecl
206         Type
207         INT
208         Identifier
209         x
210
211         Statement
212         ReadStmt
213         READ
214         (
215         Identifier
216         x

```

213				"A41.input"
214)
215		Statement		
216		LocalVarDecl		
217		Type		
218			INT	
219		Identifier		
220			y	
221		Statement		
222		ReadStmt		
223			READ	
224			(
225		Identifier		
226			y	
227			"A42.input"	
228)	
229		Statement		
230		LocalVarDecl		
231		Type		
232			INT	
233		Identifier		
234			z	
235		Statement		
236		assignStmt		
237		Identifier		
238			z	
239			:=	
240		Expression		
241			MultiplicativeExpr	
242			PrimaryExpr	
243			Identifier	
244			f2	
245			(
246	ActualParams			
247	Expression			
248		MultiplicativeExpr		
249		PrimaryExpr		
250		Identifier		
251			x	
252	Expression			
253		MultiplicativeExpr		
254		PrimaryExpr		
255		Identifier		
256			y	
257)
258			+	
259			MultiplicativeExpr	

260						PrimaryExpr
261						Identifier
262						f2
263					(
264						
265	ActualParams					
266	Expression					
267		MultiplicativeExpr				
268			PrimaryExpr			
269			Identifier			
270				y		
271	Expression					
272		MultiplicativeExpr				
273			PrimaryExpr			
274			Identifier			
275				x		
276)	
277		Statement				
278			IfStmt			
279				IF		
280				(
281				BoolExpression		
282					Expression	
283						MultiplicativeExpr
284	PrimaryExpr					
285		Identifier				
286			x			
287				>=		
288					Expression	
289						MultiplicativeExpr
290	PrimaryExpr					
291		Identifier				
292			y			
293)		
294		Statement				
295			Block			
296			BEGIN			
297					Statement	
298						assignStmt
299	Identifier					

299	
300	z
301	:=
302	Expression
303	MultiplicativeExpr
304	PrimaryExpr
305	Identifier
306	x
307	+
308	MultiplicativeExpr
309	PrimaryExpr
310	Identifier
311	y
312	END
313	Statement
314	whileStmt
315	WHILE
316	(
317	BoolExpression
318	Expression
319	MultiplicativeExpr
320	PrimaryExpr
321	Identifier
322	x
323	>=
324	Expression
325	MultiplicativeExpr
326	PrimaryExpr
327	Identifier
328	y
329)
330	Statement
331	Block
332	BEGIN
333	Statement
334	assignStmt
335	Identifier
336	z
	:=

