

编译原理与技术实验二：语法分析程序的设计与实现

实验报告

毛子恒

2019211397

北京邮电大学 计算机学院

日期：2021 年 10 月 26 日

1 概览

1.1 任务描述

编写语法分析程序，实现对算术表达式的语法分析。要求所分析算术表达式由如下的文法产生。

$$E \rightarrow E + T | E - T | T$$

$$T \rightarrow T * F | T / F | F$$

$$F \rightarrow (E) | num$$

要求在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

编写 LL(1) 语法分析程序，要求如下：

1. 为给定文法自动构造预测分析表。
2. 构造 LL(1) 预测分析程序。

1.2 开发环境

- macOS Big Sur 11.6
- Apple clang version 12.0.5
- cmake version 3.19.6
- Clion 2021.2.1
- Visual Studio Code 1.61.2

2 模块介绍

2.1 模块划分

各模块及其关系如图 1。

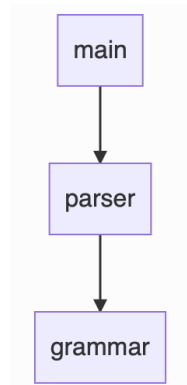


图 1: 模块关系图

其中, `grammar` 模块定义了文法类, 其中实现了文法的输入, 以及消除左递归、消除左公因子、计算 `FIRST` 和 `FOLLOW` 集的算法; `parser` 模块定义了预测分析类, 实现了从 `LL(1)` 文法构造预测分析表的构造函数, 以及利用预测分析表分析字符串的方法。

主函数的流程如下:

- 从文件中读入文法。
- 对文法进行转换并且判断是否是 `LL(1)` 文法。
- 利用文法构建预测分析表
- 从文件中读入需要分析的字符串。
- 对字符串进行预测分析, 输出分析过程。

2.2 文法类

`grammar` 模块中定义了文法类:

```
1 using Symbol = std::string;
2 using SymbolSet = std::unordered_set<Symbol>;
3 using ProductionRight = std::deque<std::string>;
4 using Productions = std::unordered_map<Symbol, std::vector<ProductionRight>>;
5 class Grammar
6 {
7 public:
8     Grammar();
9     void LoadFromFile(std::ifstream &fs);
10    bool ConvertToLL1();
11    const SymbolSet &GetNonterminal() const;
12    const SymbolSet &GetTerminal() const;
13    const Productions &GetProduction() const;
14    const Symbol &GetStart() const;
15    const std::unordered_map<std::string, std::vector<SymbolSet>>
16        &GetCandidateFirst() const;
17    const std::unordered_map<std::string, SymbolSet> &GetFollow() const;
18 private:
19     SymbolSet nonterminal;
20     SymbolSet terminal;
```

```

20     Productions production;
21     Symbol start;
22     std::unordered_map<std::string, std::vector<SymbolSet>> candidate_first;
23     std::unordered_map<std::string, SymbolSet> first;
24     std::unordered_map<std::string, SymbolSet> follow;
25     void EliminateLeftRecursion();
26     void EliminateLeftFactoring();
27     void ConstructFirst(const Symbol &left);
28     void ConstructFirstSet();
29     void ConstructFollow(const Symbol &left, std::unordered_map<Symbol,
    ↪ std::unordered_map<Symbol, bool>> &include_follow);
30     void ConstructFollowSet();
31     bool IsLL1Grammar() const;
32 };
33 std::ostream &operator<<(std::ostream &os, const SymbolSet &rhs);
34 std::ostream &operator<<(std::ostream &os, const ProductionRight &rhs);
35 std::ostream &operator<<(std::ostream &os, const Productions &rhs);

```

该类中包含有文法的非终结符号集合、终结符号集合、产生式集合、起始符号、候选式的 FIRST 集、非终结符的 FIRST 集合非终结符的 FOLLOW 集成员。

2.2.1 文法的输入

LoadFromFile 方法实现了从一个文件输入流中读取文法，一个描述1.1节中文法的文件示例如下：

```

$ Nonterminal symbols
E T F
$ Terminal symbols
+ - * / ( ) num
$ Start symbol
E
$ Productions
E -> E + T $ E - T $ T
T -> T * F $ T / F $ F
F -> ( E ) $ num

```

文件依次输入非终结符号集合、终结符号集合、起始符号、文法产生式集合。每个部分的开始都以一行单独的说明字符串标识，各个部分均可包含多行，各个符号之间以空格分隔。产生式中以 \$ 符号代替 | 符号。

输入时，程序会对文法的合法性进行基本的判断，包括非终结符号集和终结符号集不重合、起始符号是非终结符号、产生式左部是非终结符号、右部是非终结符号或者终结符号。

2.2.2 文法转换为 LL(1) 文法

ConvertToLL1 方法实现了将文法转换为 LL(1) 文法，该方法依次调用 EliminateLeftRecursion、

EliminateLeftFactoring、ConstructFirstSet、ConstructFollowSet、IsLL1Grammar 方法，并在这期间输出调试信息。

消除左递归 EliminateLeftRecursion 方法实现了消除左递归算法，即对于有如下产生式的非终结符 A ：

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$$

其中， $\beta_i (i = 1, 2, \dots, m)$ 不以 A 打头。用如下产生式替代：

$$A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_m A'$$

$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon$$

该算法的伪代码见算法 1。

算法 1: 消除左递归

输入: $G = (N, T, P, S)$

输出: $G_1 = (N_1, T, P_1, S)$

```

1 Function EliminateLeftRecursion()
2    $P_1 \leftarrow \emptyset$ ;
3    $N_1 \leftarrow N$ ;
4   foreach  $A \in N$  do
5     if  $A \rightarrow A\alpha \notin P$  then
6       foreach  $A \rightarrow \beta \in P$  do
7          $P_1 \leftarrow P_1 \cup \{A \rightarrow \beta\}$ ;
8     else
9        $N_1 \leftarrow N_1 \cup \{A'\}$ ;
10       $P_1 \leftarrow P_1 \cup \{A \rightarrow \varepsilon\}$ ;
11      foreach  $A \rightarrow A\alpha \in P$  do
12         $P_1 \leftarrow P_1 \cup \{A' \rightarrow \alpha A'\}$ ;
13      foreach  $A \rightarrow \beta \in P$  do
14         $P_1 \leftarrow P_1 \cup \{A \rightarrow \beta A'\}$ ;

```

消除左公因子 EliminateLeftFactoring 方法实现了消除左公因子算法，即对于每个非终结符 A ，找出它的两个或更多候选式的最长公共前缀 α ，如果 $\alpha \neq \varepsilon$ ，有如下产生式：

$$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma_1 | \gamma_2 | \dots | \gamma_m$$

其中， $\gamma_i (i = 1, 2, \dots, m)$ 表示不以 α 打头的表达式。用如下产生式替代：

$$A \rightarrow \alpha A' | \gamma_1 | \gamma_2 | \dots | \gamma_m$$

$$A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

该算法的伪代码见算法 2。

算法 2: 消除左公因子

输入: $G = (N, T, P, S)$

输出: $G_1 = (N_1, T, P_1, S)$

```

1 Function EliminateLeftFactoring()
2    $P_1 \leftarrow \emptyset;$ 
3    $N_1 \leftarrow N;$ 
4   do
5      $P_0 \leftarrow \emptyset;$ 
6     foreach  $A \in N$  do
7       while  $\exists A \rightarrow \alpha\beta_1 | \alpha\beta_2 \in P$  do
8          $N_1 \leftarrow N_1 \cup \{A'\};$ 
9         foreach  $A \rightarrow \alpha\beta_i \in P$  do
10           $P \leftarrow P - \{A \rightarrow \alpha\beta_i\};$ 
11           $P_0 \leftarrow P_0 \cup \{A' \rightarrow \beta_i\};$ 
12           $P \leftarrow P \cup \{A \rightarrow \alpha A'\};$ 
13        foreach  $A \rightarrow \alpha \in P$  do
14           $P \leftarrow P - \{A \rightarrow \alpha\};$ 
15           $P_1 \leftarrow P_1 \cup \{A \rightarrow \alpha\};$ 
16  while  $P_0 \neq \emptyset;$ 

```

构建非终结符和候选式的 **FIRST** 集 ConstructFirstSet 方法实现了构建非终结符和候选式的 FIRST 集，对于任意产生式 $A \rightarrow \alpha$ ，若 $\alpha \neq \varepsilon$ ，设该产生式为：

$$A \rightarrow Y_1 Y_2 \dots Y_k$$

遍历产生式右部的每一个 Y_i ，如果：

- Y_i 是终结符，则 α 的 FIRST 集中增加 Y_i ，终止遍历；
- Y_i 是非终结符，如果没有求出它的 FIRST 集，则递归求解。之后， α 的 FIRST 集并上 Y_i 的 FIRST 集。此后检查 Y_i 的 FIRST 集中是否包含 ε （即是否能推导出 ε ），若不包含，则终止遍历。

最后， A 的 FIRST 集为各个候选式的 FIRST 集的并。

该算法的伪代码见算法 3。

算法 3: 构建非终结符和候选式的 FIRST 集

输入: $G = (N, T, P, S)$

输出: FIRST

```
1 Function ConstructFirst(A)
2   foreach  $A \rightarrow \alpha \in P$  do
3     if  $\alpha = \varepsilon$  then
4       FIRST(A)  $\leftarrow$  FIRST(A)  $\cup \{\varepsilon\}$ ;
5       FIRST( $\alpha$ )  $\leftarrow \{\varepsilon\}$ ;
6     for  $i \leftarrow 1$  to  $k$  do //  $\alpha = Y_1 Y_2 \dots Y_k$ 
7       if  $Y_i \in T$  then
8         FIRST( $\alpha$ )  $\leftarrow$  FIRST( $\alpha$ )  $\cup \{Y_i\}$ ;
9         break;
10      if FIRST( $Y_i$ ) =  $\emptyset$  then ConstructFirst( $Y_i$ );
11      FIRST( $\alpha$ ) = FIRST( $\alpha$ )  $\cup$  FIRST( $Y_i$ );
12      if  $\varepsilon \notin$  FIRST( $Y_i$ ) then break;
13    if  $i = k$  then FIRST(A)  $\leftarrow$  FIRST(A)  $\cup \{\varepsilon\}$ ;
14    FIRST(A)  $\leftarrow$  FIRST(A)  $\cup$  FIRST( $\alpha$ );

15 Function ConstructFirstSet()
16   foreach  $A \in T$  do
17     if FIRST(A) =  $\emptyset$  then ConstructFirst(A);
```

构建非终结符的 FOLLOW 集 ConstructFollowSet 方法实现了构建非终结符的 FOLLOW 集, 对于非终结符 B , 检查所有右部包含 B 的产生式 $A \rightarrow \alpha B Y_1 Y_2 \dots Y_k$:

遍历每一个 Y_i , 如果:

- Y_i 是终结符, 则 B 的 FOLLOW 集中增加 Y_i , 终止遍历;
- Y_i 是非终结符, B 的 FOLLOW 集并上 Y_i 的 FIRST 集中非空的部分。此后检查 Y_i 的 FIRST 集中是否包含 ε (即是否能推导出 ε), 若不包含, 则终止遍历。

如果遍历完 Y_k 并且 $A \neq B$, 则 A 的 FOLLOW 集包含在 B 的 FOLLOW 集中, 为了处理两个非终结符的 FOLLOW 集互相包含导致无限递归的情况, 采用 include_follow 变量记录非终结符的 FOLLOW 集的包含关系, 以及 finished_construct_follow 变量记录 FOLLOW 集是否构建完成。如果一个非终结符 B 的 FOLLOW 集包含另一个终结符 A 的 FOLLOW 集, 但是 A 的 FOLLOW 集不包含 B 的 FOLLOW 集并且 A 的 FOLLOW 集还没有处理, 则可以递归处理 A 的 FOLLOW 集, 完成后将 B 的 FOLLOW 集并上 A 的 FOLLOW 集。

在处理完所有非终结符的 FOLLOW 集后, 处理两个集合相互包含 (即两个集合相等) 的情况, 此时将两个集合都设为原本的集合的并即可。

该算法的伪代码见**算法 4**。

算法 4: 构建非终结符的 FOLLOW 集

输入: $G = (N, T, P, S)$, **FIRST**

输出: **FOLLOW**

```
1 Function ConstructFollow( $B$ )
2   foreach  $A \in N$  do
3     foreach  $A \rightarrow \alpha B \beta \in P$  do
4       for  $i \leftarrow 1$  to  $k$  do                                     //  $\beta = Y_1 Y_2 \dots Y_k$ 
5         if  $Y_i \in T$  then
6           FOLLOW( $B$ )  $\leftarrow$  FOLLOW( $B$ )  $\cup$   $\{Y_i\}$ ;
7           break;
8         FOLLOW( $B$ )  $\leftarrow$  FOLLOW( $B$ )  $\cup$  (FIRST( $Y_i$ )  $- \{\epsilon\}$ );
9         if  $\epsilon \notin$  FIRST( $Y_i$ ) then break;
10      if  $i = k \wedge A \neq B$  then
11        include_follow[ $B, A$ ] = true;                                // FOLLOW( $A$ )  $\subseteq$  FOLLOW( $B$ )
12        if include_follow[ $A, B$ ] = false  $\wedge$  finished_construct_follow[ $A$ ] = false then
13          ConstructFollow( $A$ );
14        FOLLOW( $B$ )  $\leftarrow$  FOLLOW( $B$ )  $\cup$  FOLLOW( $A$ );
15  finished_construct_follow[ $B$ ] = true;

16 Function ConstructFollowSet()
17    $T \leftarrow T \cup \{\$$ ;
18   FOLLOW( $S$ )  $\leftarrow \{\$$ ;
19   foreach  $A \in N$  do
20     if FOLLOW( $A$ ) =  $\emptyset$  then ConstructFollow( $A$ );
21   foreach  $A, B \in N$  do
22     if include_follow[ $A, B$ ] = true  $\wedge$  include_follow[ $B, A$ ] = true then
23       FOLLOW( $A$ )  $\leftarrow$  FOLLOW( $A$ )  $\cup$  FOLLOW( $B$ );
24       FOLLOW( $B$ )  $\leftarrow$  FOLLOW( $A$ );
```

判断是否是 LL(1) 文法 `IsLL1Grammar` 方法判断文法是否是 LL(1) 文法, 即检查每个产生式 $A \rightarrow \alpha | \beta$, 需要满足:

- **FIRST**(α) \cap **FIRST**(β) = \emptyset
- 如果 $A \rightarrow \epsilon$, **FIRST**(α) \cap **FOLLOW**(A) = \emptyset

2.3 预测分析

`parser` 模块定义了预测分析类:

```

1 using Production = std::pair<Symbol, ProductionRight>;
2 class Parser
3 {
4 public:
5     Parser();
6     explicit Parser(const Grammar &grammar);
7     bool ParseString(const ProductionRight &str);
8 private:
9     using Status = std::pair<std::vector<Symbol>, ProductionRight>;
10    Symbol start;
11    SymbolSet nonterminal;
12    std::vector<Symbol> terminal;
13    std::unordered_map<Symbol, std::unordered_map<Symbol, ProductionRight>>
        ↪ parsing_table;
14    bool NextStep(Status &status, Production &production);
15    friend std::ostream &operator<<(std::ostream &os, const Parser &rhs);
16 };
17 std::ostream &operator<<(std::ostream &os, const std::vector<Symbol> &rhs);
18 std::ostream &operator<<(std::ostream &os, const Production &rhs);
19 std::ostream &operator<<(std::ostream &os, const Parser &rhs);

```

该类中的 `parsing_table` 为预测分析表。

2.3.1 构建预测分析表

`Parser(const Grammar &grammar)` 构造函数通过 LL(1) 文法构造一个预测分析表，构造算法伪代码见算法 5。

算法 5: 构建预测分析表

输入: $G = (N, T, P, S)$, FIRST, FOLLOW

输出: 预测分析表 M

```

1 Function ConstructParsingTable( $G$ )
2   foreach  $a \in \text{FIRST}(\alpha) \cap T$  do  $M[A, a] \leftarrow \{A \rightarrow \alpha\};$ 
3   if  $\varepsilon \in \text{FIRST}(\alpha)$  then
4     foreach  $b \in \text{FOLLOW}(A)$  do  $M[A, a] \leftarrow \{A \rightarrow \alpha\};$ 
5   foreach  $M[A, a] = \emptyset$  do  $M[A, a] = \{\text{error}\};$ 

```

2.3.2 预测分析算法

`ParseString` 方法实现对一个字符串的预测分析，该函数首先初始化一个状态，之后依照状态和当前字符串不返回断调用 `NextStep` 一步步进行预测分析，并且同时输出分析过程。该函数的返回值表示分析是否成功。

非递归预测算法的伪代码见算法 6。

算法 6: 非递归预测分析算法

输入: 输入符号串 ω , $\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{P}, S)$ 及其预测分析表 \mathbf{M}

输出: 若 $\omega \in L(\mathbf{G})$, 则输出 ω 的最左推导 answer , 否则报告错误

```
1 Function ParseString( $\omega$ )
2   stack.push('$');
3   stack.push( $S$ );
4   buffer  $\leftarrow \omega$ ;
5   ip  $\leftarrow 0$ ;
6   do
7      $X \leftarrow \text{stack.top}()$ ;
8      $a \leftarrow \text{buffer[ip]}$ ;
9     if  $X \in \mathbf{T} \cup \{\$\}$  then
10      if  $X = a$  then
11        stack.pop();
12        ip  $\leftarrow \text{ip} + 1$ ;
13      else return error;
14    else
15      if  $\mathbf{M}[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  then
16        stack.pop();
17        for  $i \leftarrow k$  to 1 do stack.push( $Y_i$ );
18        answer.push_back( $X \rightarrow Y_1 Y_2 \dots Y_k$ );
19      else return error;
20  while  $X \neq \$$ ;
21  return answer;
```

3 用户指南

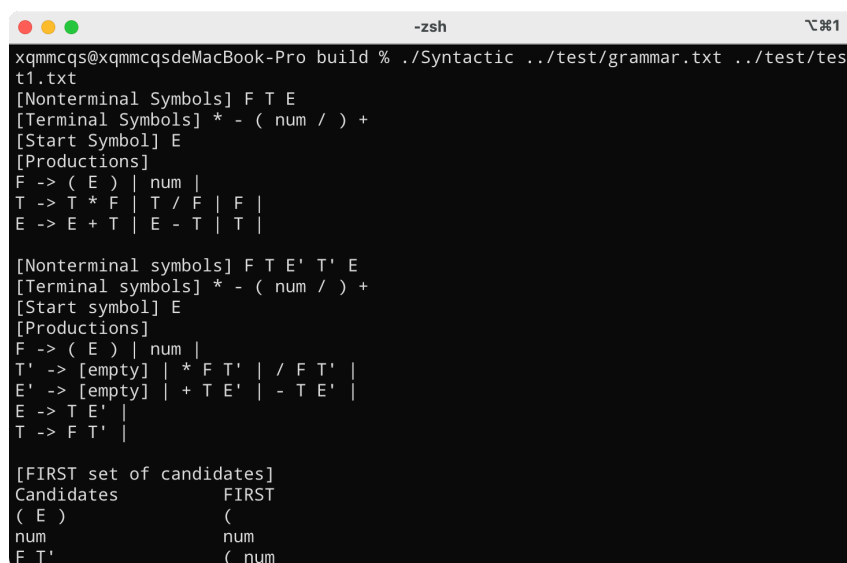
在项目目录中执行以下命令来编译:

```
mkdir build
cd build
cmake ..
make
```

编译完成后, 运行:

```
./Syntactic ../test/grammar.txt ../test/test1.txt
```

运行截图如图 2 所示。



```
xqmmcqs@xqmmcqsdeMacBook-Pro build % ./Syntactic ../test/grammar.txt ../test/tes
t1.txt
[Nonterminal Symbols] F T E
[Terminal Symbols] * - ( num / ) +
[Start Symbol] E
[Productions]
F -> ( E ) | num |
T -> T * F | T / F | F |
E -> E + T | E - T | T |

[Nonterminal symbols] F T E' T' E
[Terminal symbols] * - ( num / ) +
[Start symbol] E
[Productions]
F -> ( E ) | num |
T' -> [empty] | * F T' | / F T' |
E' -> [empty] | + T E' | - T E' |
E -> T E' |
T -> F T' |

[FIRST set of candidates]
Candidates      FIRST
( E )           (
num             num
F T'            ( num
```

图 2: 运行截图

4 测试结果

4.1 测试集 1

此测试集用于简单地测试程序是否正常运行。

4.1.1 输入

文法如 2.2.1 节所示。

num + num

4.1.2 输出

```
[Nonterminal Symbols] F T E
[Terminal Symbols] * - ( num / ) +
[Start Symbol] E
[Productions]
F -> ( E ) | num |
T -> T * F | T / F | F |
E -> E + T | E - T | T |

[Nonterminal symbols] F T E' T' E
[Terminal symbols] * - ( num / ) +
[Start symbol] E
[Productions]
F -> ( E ) | num |
```

$T' \rightarrow [\text{empty}] \mid * F T' \mid / F T' \mid$
 $E' \rightarrow [\text{empty}] \mid + T E' \mid - T E' \mid$
 $E \rightarrow T E' \mid$
 $T \rightarrow F T' \mid$

[FIRST set of candidates]

Candidate	(E)
FIRST	(
Candidate	num
FIRST	num
Candidate	F T'
FIRST	(num
Candidate	[empty]
FIRST	[empty]
Candidate	+ T E'
FIRST	+
Candidate	- T E'
FIRST	-
Candidate	[empty]
FIRST	[empty]
Candidate	* F T'
FIRST	*
Candidate	/ F T'
FIRST	/
Candidate	T E'
FIRST	(num

[FIRST and FOLLOW set of nonterminals]

Nonterminals	F
FIRST	num (
FOLLOW	- / \$ +) *
Nonterminals	T
FIRST	num (
FOLLOW) + \$ -
Nonterminals	E'
FIRST	- + [empty]
FOLLOW	\$)
Nonterminals	T'
FIRST	/ * [empty]
FOLLOW	- \$ +)
Nonterminals	E
FIRST	num (
FOLLOW) \$

[Parsing Table]

Nonterminal	E
Terminal	(
Production	E -> T E'
Terminal	num
Production	E -> T E'

Nonterminal	T'
Terminal	*
Production	T' -> * F T'
Terminal	-
Production	T' -> [empty]
Terminal	\$
Production	T' -> [empty]
Terminal	/
Production	T' -> / F T'
Terminal)
Production	T' -> [empty]
Terminal	+
Production	T' -> [empty]

Nonterminal	E'
Terminal	-
Production	E' -> - T E'
Terminal	\$
Production	E' -> [empty]
Terminal)
Production	E' -> [empty]
Terminal	+
Production	E' -> + T E'

Nonterminal	T
Terminal	(
Production	T -> F T'
Terminal	num
Production	T -> F T'

Nonterminal	F
Terminal	(
Production	F -> (E)
Terminal	num
Production	F -> num

Stack	\$ E
Input	num + num \$
Output	

Stack	\$ E' T
Input	num + num \$
Output	E -> T E'

Stack	\$ E' T' F
Input	num + num \$
Output	T -> F T'

Stack	\$ E' T' num
Input	num + num \$

Output	F -> num
Stack	\$ E' T'
Input	+ num \$
Output	
Stack	\$ E'
Input	+ num \$
Output	T' -> [empty]
Stack	\$ E' T +
Input	+ num \$
Output	E' -> + T E'
Stack	\$ E' T
Input	num \$
Output	
Stack	\$ E' T' F
Input	num \$
Output	T -> F T'
Stack	\$ E' T' num
Input	num \$
Output	F -> num
Stack	\$ E' T'
Input	\$
Output	
Stack	\$ E'
Input	\$
Output	T' -> [empty]
Stack	\$
Input	\$
Output	E' -> [empty]

Parsing successful!

4.1.3 分析

程序首先输出原文法，之后输出消除左递归和左公因子的文法、候选式的 **FIRST** 集、非终结符的 **FIRST** 集和 **FOLLOW** 集、预测分析表。最后程序给出了分析字符串的过程。

4.2 测试集 2

此测试集测试一个复杂的算术表达式。

4.2.1 输入

文法如2.2.1节所示。

```
( num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num + num ) ) /  
↪ num )
```

4.2.2 输出 (部分)

...

```
Stack          $ E  
Input          ( num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num  
↪ + num ) ) / num ) $  
Output
```

```
Stack          $ E' T  
Input          ( num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num  
↪ + num ) ) / num ) $  
Output         E -> T E'
```

```
Stack          $ E' T' F  
Input          ( num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num  
↪ + num ) ) / num ) $  
Output         T -> F T'
```

```
Stack          $ E' T' ) E (  
Input          ( num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num  
↪ + num ) ) / num ) $  
Output         F -> ( E )
```

```
Stack          $ E' T' ) E  
Input          num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num +  
↪ num ) ) / num ) $  
Output
```

```
Stack          $ E' T' ) E' T  
Input          num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num +  
↪ num ) ) / num ) $  
Output         E -> T E'
```

```
Stack          $ E' T' ) E' T' F  
Input          num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num +  
↪ num ) ) / num ) $  
Output         T -> F T'
```

```
Stack          $ E' T' ) E' T' num  
Input          num * ( ( ( ( num + num ) * num ) + num / num ) - num * ( num +  
↪ num ) ) / num ) $
```

Output	F -> num
Stack	\$ E' T') E' T'
Input	* ((((num + num) * num) + num / num) - num * (num + num
↪)) / num) \$
Output	
Stack	\$ E' T') E' T' F *
Input	* ((((num + num) * num) + num / num) - num * (num + num
↪)) / num) \$
Output	T' -> * F T'
Stack	\$ E' T') E' T' F
Input	((((num + num) * num) + num / num) - num * (num + num)
↪) / num) \$
Output	
Stack	\$ E' T') E' T') E (
Input	((((num + num) * num) + num / num) - num * (num + num)
↪) / num) \$
Output	F -> (E)
Stack	\$ E' T') E' T') E
Input	(((num + num) * num) + num / num) - num * (num + num))
↪	/ num) \$
Output	
Stack	\$ E' T') E' T') E' T
Input	(((num + num) * num) + num / num) - num * (num + num))
↪	/ num) \$
Output	E -> T E'
Stack	\$ E' T') E' T') E' T' F
Input	(((num + num) * num) + num / num) - num * (num + num))
↪	/ num) \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E (
Input	(((num + num) * num) + num / num) - num * (num + num))
↪	/ num) \$
Output	F -> (E)
Stack	\$ E' T') E' T') E' T') E
Input	((num + num) * num) + num / num) - num * (num + num)) /
↪	num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T
Input	((num + num) * num) + num / num) - num * (num + num)) /
↪	num) \$
Output	E -> T E'

Stack	\$ E' T') E' T') E' T') E' T' F
Input	((num + num) * num) + num / num) - num * (num + num)) /
↪ num) \$	
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T') E (
Input	((num + num) * num) + num / num) - num * (num + num)) /
↪ num) \$	
Output	F -> (E)
Stack	\$ E' T') E' T') E' T') E' T') E
Input	(num + num) * num) + num / num) - num * (num + num)) /
↪ num) \$	
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T
Input	(num + num) * num) + num / num) - num * (num + num)) /
↪ num) \$	
Output	E -> T E'
Stack	\$ E' T') E' T') E' T') E' T') E' T' F
Input	(num + num) * num) + num / num) - num * (num + num)) /
↪ num) \$	
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T') E' T') E (
Input	(num + num) * num) + num / num) - num * (num + num)) /
↪ num) \$	
Output	F -> (E)
Stack	\$ E' T') E' T') E' T') E' T') E' T') E
Input	num + num) * num) + num / num) - num * (num + num)) / num
↪) \$	
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T
Input	num + num) * num) + num / num) - num * (num + num)) / num
↪) \$	
Output	E -> T E'
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T' F
Input	num + num) * num) + num / num) - num * (num + num)) / num
↪) \$	
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T' num
Input	num + num) * num) + num / num) - num * (num + num)) / num
↪) \$	
Output	F -> num

Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T'
Input	+ num) * num) + num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T') E'
Input	+ num) * num) + num / num) - num * (num + num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T +
Input	+ num) * num) + num / num) - num * (num + num)) / num) \$
Output	E' -> + T E'
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T
Input	num) * num) + num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T' F
Input	num) * num) + num / num) - num * (num + num)) / num) \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T' num
Input	num) * num) + num / num) - num * (num + num)) / num) \$
Output	F -> num
Stack	\$ E' T') E' T') E' T') E' T') E' T') E' T'
Input) * num) + num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T') E'
Input) * num) + num / num) - num * (num + num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T') E' T')
Input) * num) + num / num) - num * (num + num)) / num) \$
Output	E' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T') E' T'
Input	* num) + num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T' F *
Input	* num) + num / num) - num * (num + num)) / num) \$
Output	T' -> * F T'
Stack	\$ E' T') E' T') E' T') E' T') E' T' F
Input	num) + num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T') E' T' num
Input	num) + num / num) - num * (num + num)) / num) \$
Output	F -> num

Stack	\$ E' T') E' T') E' T') E' T') E' T'
Input) + num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T') E'
Input) + num / num) - num * (num + num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T')
Input) + num / num) - num * (num + num)) / num) \$
Output	E' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T'
Input	+ num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E'
Input	+ num / num) - num * (num + num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T +
Input	+ num / num) - num * (num + num)) / num) \$
Output	E' -> + T E'
Stack	\$ E' T') E' T') E' T') E' T
Input	num / num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T' F
Input	num / num) - num * (num + num)) / num) \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T' num
Input	num / num) - num * (num + num)) / num) \$
Output	F -> num
Stack	\$ E' T') E' T') E' T') E' T'
Input	/ num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T' F /
Input	/ num) - num * (num + num)) / num) \$
Output	T' -> / F T'
Stack	\$ E' T') E' T') E' T') E' T' F
Input	num) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T' num
Input	num) - num * (num + num)) / num) \$
Output	

Output	F -> num
Stack	\$ E' T') E' T') E' T') E' T'
Input) - num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E'
Input) - num * (num + num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T')
Input) - num * (num + num)) / num) \$
Output	E' -> [empty]
Stack	\$ E' T') E' T') E' T'
Input	- num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E'
Input	- num * (num + num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T -
Input	- num * (num + num)) / num) \$
Output	E' -> - T E'
Stack	\$ E' T') E' T') E' T
Input	num * (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T' F
Input	num * (num + num)) / num) \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T' num
Input	num * (num + num)) / num) \$
Output	F -> num
Stack	\$ E' T') E' T') E' T'
Input	* (num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T' F *
Input	* (num + num)) / num) \$
Output	T' -> * F T'
Stack	\$ E' T') E' T') E' T' F
Input	(num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E (

Input	(num + num)) / num) \$
Output	F -> (E)
Stack	\$ E' T') E' T') E' T') E
Input	num + num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T
Input	num + num)) / num) \$
Output	E -> T E'
Stack	\$ E' T') E' T') E' T') E' T' F
Input	num + num)) / num) \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T' num
Input	num + num)) / num) \$
Output	F -> num
Stack	\$ E' T') E' T') E' T') E' T'
Input	+ num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E'
Input	+ num)) / num) \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T') E' T +
Input	+ num)) / num) \$
Output	E' -> + T E'
Stack	\$ E' T') E' T') E' T') E' T
Input	num)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E' T' F
Input	num)) / num) \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T') E' T' num
Input	num)) / num) \$
Output	F -> num
Stack	\$ E' T') E' T') E' T') E' T'
Input)) / num) \$
Output	
Stack	\$ E' T') E' T') E' T') E'
Input)) / num) \$
Output	T' -> [empty]

Stack	\$ E' T') E' T') E' T')
Input) / num) \$
Output	E' -> [empty]

Stack	\$ E' T') E' T') E' T'
Input) / num) \$
Output	

Stack	\$ E' T') E' T') E'
Input) / num) \$
Output	T' -> [empty]

Stack	\$ E' T') E' T')
Input) / num) \$
Output	E' -> [empty]

Stack	\$ E' T') E' T'
Input	/ num) \$
Output	

Stack	\$ E' T') E' T' F /
Input	/ num) \$
Output	T' -> / F T'

Stack	\$ E' T') E' T' F
Input	num) \$
Output	

Stack	\$ E' T') E' T' num
Input	num) \$
Output	F -> num

Stack	\$ E' T') E' T'
Input) \$
Output	

Stack	\$ E' T') E'
Input) \$
Output	T' -> [empty]

Stack	\$ E' T')
Input) \$
Output	E' -> [empty]

Stack	\$ E' T'
Input	\$
Output	

Stack	\$ E'
Input	\$
Output	T' -> [empty]

Stack	\$
Input	\$
Output	E' -> [empty]

Parsing successful!

4.2.3 分析

正确分析了此字符串。

4.3 测试集 3

此测试集用于测试一个错误的算术表达式。

4.3.1 输入

文法如2.2.1节所示。

```
( num + num ) * / num
```

4.3.2 输出 (部分)

...

Stack	\$ E
Input	(num + num) * / num \$
Output	

Stack	\$ E' T
Input	(num + num) * / num \$
Output	E -> T E'

Stack	\$ E' T' F
Input	(num + num) * / num \$
Output	T -> F T'

Stack	\$ E' T') E (
Input	(num + num) * / num \$
Output	F -> (E)

Stack	\$ E' T') E
Input	num + num) * / num \$
Output	

Stack	\$ E' T') E' T
Input	num + num) * / num \$

Output	E -> T E'
Stack	\$ E' T') E' T' F
Input	num + num) * / num \$
Output	T -> F T'
Stack	\$ E' T') E' T' num
Input	num + num) * / num \$
Output	F -> num
Stack	\$ E' T') E' T'
Input	+ num) * / num \$
Output	
Stack	\$ E' T') E'
Input	+ num) * / num \$
Output	T' -> [empty]
Stack	\$ E' T') E' T +
Input	+ num) * / num \$
Output	E' -> + T E'
Stack	\$ E' T') E' T
Input	num) * / num \$
Output	
Stack	\$ E' T') E' T' F
Input	num) * / num \$
Output	T -> F T'
Stack	\$ E' T') E' T' num
Input	num) * / num \$
Output	F -> num
Stack	\$ E' T') E' T'
Input) * / num \$
Output	
Stack	\$ E' T') E'
Input) * / num \$
Output	T' -> [empty]
Stack	\$ E' T')
Input) * / num \$
Output	E' -> [empty]
Stack	\$ E' T'
Input	* / num \$
Output	
Stack	\$ E' T' F *

Input	* / num \$
Output	T' -> * F T'

Stack	\$ E' T' F
Input	/ num \$
Output	

Parsing failed!

4.3.3 分析

在分析到错误的/符号时，由于预测分析表中没有对应的项，分析程序报错并退出。

4.4 测试集 4

此测试集用于测试一个错误的算术表达式。

4.4.1 输入

文法如2.2.1节所示。

((num + num) / num

4.4.2 输出 (部分)

...

Stack	\$ E
Input	((num + num) / num \$
Output	

Stack	\$ E' T
Input	((num + num) / num \$
Output	E -> T E'

Stack	\$ E' T' F
Input	((num + num) / num \$
Output	T -> F T'

Stack	\$ E' T') E (
Input	((num + num) / num \$
Output	F -> (E)

Stack	\$ E' T') E
Input	(num + num) / num \$
Output	

Stack	\$ E' T') E' T
Input	(num + num) / num \$
Output	E -> T E'
Stack	\$ E' T') E' T' F
Input	(num + num) / num \$
Output	T -> F T'
Stack	\$ E' T') E' T') E (
Input	(num + num) / num \$
Output	F -> (E)
Stack	\$ E' T') E' T') E
Input	num + num) / num \$
Output	
Stack	\$ E' T') E' T') E' T
Input	num + num) / num \$
Output	E -> T E'
Stack	\$ E' T') E' T') E' T' F
Input	num + num) / num \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T' num
Input	num + num) / num \$
Output	F -> num
Stack	\$ E' T') E' T') E' T'
Input	+ num) / num \$
Output	
Stack	\$ E' T') E' T') E'
Input	+ num) / num \$
Output	T' -> [empty]
Stack	\$ E' T') E' T') E' T +
Input	+ num) / num \$
Output	E' -> + T E'
Stack	\$ E' T') E' T') E' T
Input	num) / num \$
Output	
Stack	\$ E' T') E' T') E' T' F
Input	num) / num \$
Output	T -> F T'
Stack	\$ E' T') E' T') E' T' num
Input	num) / num \$
Output	F -> num

Stack	\$ E' T') E' T') E' T'
Input) / num \$
Output	

Stack	\$ E' T') E' T') E'
Input) / num \$
Output	T' -> [empty]

Stack	\$ E' T') E' T')
Input) / num \$
Output	E' -> [empty]

Stack	\$ E' T') E' T'
Input	/ num \$
Output	

Stack	\$ E' T') E' T' F /
Input	/ num \$
Output	T' -> / F T'

Stack	\$ E' T') E' T' F
Input	num \$
Output	

Stack	\$ E' T') E' T' num
Input	num \$
Output	F -> num

Stack	\$ E' T') E' T'
Input	\$
Output	

Stack	\$ E' T') E'
Input	\$
Output	T' -> [empty]

Stack	\$ E' T')
Input	\$
Output	E' -> [empty]

Parsing failed!

4.4.3 分析

当分析到字符串尾时，由于栈顶的终结符和输入的符号不匹配，故分析程序报错并退出。

4.5 测试集 5

此测试集依照习题 4.5 修改而来。

4.5.1 输入

```
$ Nonterminal symbols
E A B L
$ Terminal symbols
num id ( )
$ Start symbol
E
$ Productions
E -> A $ B
A -> num $ id
B -> ( L )
L -> L E $ E
```

```
( id ( id ( num ) ) ( id ) )
```

4.5.2 输出

```
[Nonterminal Symbols] L B A E
[Terminal Symbols] ) ( id num
[Start Symbol] E
[Productions]
L -> L E | E |
B -> ( L ) |
A -> num | id |
E -> A | B |

[Nonterminal symbols] L L' B A E
[Terminal symbols] ) ( id num
[Start symbol] E
[Productions]
L' -> [empty] | E L' |
B -> ( L ) |
L -> E L' |
A -> num | id |
E -> A | B |

[FIRST set of candidates]
Candidate      E L'
FIRST          num id (
Candidate      [empty]
FIRST          [empty]
Candidate      E L'
FIRST          num id (
Candidate      ( L )
FIRST          (
Candidate      num
```

FIRST	num
Candidate	id
FIRST	id
Candidate	A
FIRST	num id
Candidate	B
FIRST	(

[FIRST and FOLLOW set of nonterminals]

Nonterminals	L
FIRST	(id num
FOLLOW)
Nonterminals	L'
FIRST	(id num [empty]
FOLLOW)
Nonterminals	B
FIRST	(
FOLLOW	(id num \$)
Nonterminals	A
FIRST	id num
FOLLOW	(id num \$)
Nonterminals	E
FIRST	(id num
FOLLOW) \$ num id (

[Parsing Table]

Nonterminal	E
Terminal	(
Production	E -> B
Terminal	id
Production	E -> A
Terminal	num
Production	E -> A

Nonterminal	A
Terminal	id
Production	A -> id
Terminal	num
Production	A -> num

Nonterminal	B
Terminal	(
Production	B -> (L)

Nonterminal	L'
Terminal)
Production	L' -> [empty]
Terminal	(
Production	L' -> E L'
Terminal	id
Production	L' -> E L'

Terminal	num
Production	L' -> E L'

Nonterminal	L
Terminal	(
Production	L -> E L'
Terminal	id
Production	L -> E L'
Terminal	num
Production	L -> E L'

Stack	\$ E
Input	(id (id (num)) (id)) \$
Output	

Stack	\$ B
Input	(id (id (num)) (id)) \$
Output	E -> B

Stack	\$) L (
Input	(id (id (num)) (id)) \$
Output	B -> (L)

Stack	\$) L
Input	id (id (num)) (id)) \$
Output	

Stack	\$) L' E
Input	id (id (num)) (id)) \$
Output	L -> E L'

Stack	\$) L' A
Input	id (id (num)) (id)) \$
Output	E -> A

Stack	\$) L' id
Input	id (id (num)) (id)) \$
Output	A -> id

Stack	\$) L'
Input	(id (num)) (id)) \$
Output	

Stack	\$) L' E
Input	(id (num)) (id)) \$
Output	L' -> E L'

Stack	\$) L' B
Input	(id (num)) (id)) \$
Output	E -> B

Stack	\$) L') L (
Input	(id (num)) (id)) \$
Output	B -> (L)
Stack	\$) L') L
Input	id (num)) (id)) \$
Output	
Stack	\$) L') L' E
Input	id (num)) (id)) \$
Output	L -> E L'
Stack	\$) L') L' A
Input	id (num)) (id)) \$
Output	E -> A
Stack	\$) L') L' id
Input	id (num)) (id)) \$
Output	A -> id
Stack	\$) L') L'
Input	(num)) (id)) \$
Output	
Stack	\$) L') L' E
Input	(num)) (id)) \$
Output	L' -> E L'
Stack	\$) L') L' B
Input	(num)) (id)) \$
Output	E -> B
Stack	\$) L') L') L (
Input	(num)) (id)) \$
Output	B -> (L)
Stack	\$) L') L') L
Input	num)) (id)) \$
Output	
Stack	\$) L') L') L' E
Input	num)) (id)) \$
Output	L -> E L'
Stack	\$) L') L') L' A
Input	num)) (id)) \$
Output	E -> A
Stack	\$) L') L') L' num
Input	num)) (id)) \$

Output	A -> num
Stack	\$) L') L') L'
Input)) (id)) \$
Output	
Stack	\$) L') L')
Input)) (id)) \$
Output	L' -> [empty]
Stack	\$) L') L'
Input) (id)) \$
Output	
Stack	\$) L')
Input) (id)) \$
Output	L' -> [empty]
Stack	\$) L'
Input	(id)) \$
Output	
Stack	\$) L' E
Input	(id)) \$
Output	L' -> E L'
Stack	\$) L' B
Input	(id)) \$
Output	E -> B
Stack	\$) L') L (
Input	(id)) \$
Output	B -> (L)
Stack	\$) L') L
Input	id)) \$
Output	
Stack	\$) L') L' E
Input	id)) \$
Output	L -> E L'
Stack	\$) L') L' A
Input	id)) \$
Output	E -> A
Stack	\$) L') L' id
Input	id)) \$
Output	A -> id
Stack	\$) L') L'

```

Input      ) ) $
Output

Stack      $ ) L' )
Input      ) ) $
Output      L' -> [empty]

Stack      $ ) L'
Input      ) $
Output

Stack      $ )
Input      ) $
Output      L' -> [empty]

Stack      $
Input      $
Output

```

Parsing successful!

4.6 测试集 6

此测试集依照习题 4.6 修改而来。

4.6.1 输入

```

$ Nonterminal symbols
E A B L
$ Terminal symbols
num id ( ) ,
$ Start symbol
E
$ Productions
E -> A $ B
A -> num $ id
B -> ( L )
L -> L , E $ E

```

```

( id , ( id , ( num ) ) , ( id ) )

```

4.6.2 输出

```

[Nonterminal Symbols] L B A E
[Terminal Symbols] , ) ( id num

```



```

[Start Symbol] E
[Productions]
L -> L , E | E |
B -> ( L ) |
A -> num | id |
E -> A | B |

```

```

[Nonterminal symbols] L L' B A E
[Terminal symbols] , ) ( id num
[Start symbol] E
[Productions]
L' -> [empty] | , E L' |
B -> ( L ) |
L -> E L' |
A -> num | id |
E -> A | B |

```

```

[FIRST set of candidates]
Candidate      E L'
FIRST          num id (
Candidate      [empty]
FIRST          [empty]
Candidate      , E L'
FIRST          ,
Candidate      ( L )
FIRST          (
Candidate      num
FIRST          num
Candidate      id
FIRST          id
Candidate      A
FIRST          num id
Candidate      B
FIRST          (

```

```

[FIRST and FOLLOW set of nonterminals]
Nonterminals   L
FIRST          ( id num
FOLLOW         )
Nonterminals   L'
FIRST          , [empty]
FOLLOW         )
Nonterminals   B
FIRST          (
FOLLOW         $ , )
Nonterminals   A
FIRST          id num
FOLLOW         $ , )
Nonterminals   E
FIRST          ( id num
FOLLOW         ) , $

```

[Parsing Table]

Nonterminal E

Terminal (

Production E -> B

Terminal id

Production E -> A

Terminal num

Production E -> A

Nonterminal A

Terminal id

Production A -> id

Terminal num

Production A -> num

Nonterminal B

Terminal (

Production B -> (L)

Nonterminal L'

Terminal ,

Production L' -> , E L'

Terminal)

Production L' -> [empty]

Nonterminal L

Terminal (

Production L -> E L'

Terminal id

Production L -> E L'

Terminal num

Production L -> E L'

Stack \$ E

Input (id , (id , (num)) , (id)) \$

Output

Stack \$ B

Input (id , (id , (num)) , (id)) \$

Output E -> B

Stack \$) L (

Input (id , (id , (num)) , (id)) \$

Output B -> (L)

Stack \$) L

Input id , (id , (num)) , (id)) \$

Output

Stack	\$) L' E
Input	id , (id , (num)) , (id)) \$
Output	L -> E L'
Stack	\$) L' A
Input	id , (id , (num)) , (id)) \$
Output	E -> A
Stack	\$) L' id
Input	id , (id , (num)) , (id)) \$
Output	A -> id
Stack	\$) L'
Input	, (id , (num)) , (id)) \$
Output	
Stack	\$) L' E ,
Input	, (id , (num)) , (id)) \$
Output	L' -> , E L'
Stack	\$) L' E
Input	(id , (num)) , (id)) \$
Output	
Stack	\$) L' B
Input	(id , (num)) , (id)) \$
Output	E -> B
Stack	\$) L') L (
Input	(id , (num)) , (id)) \$
Output	B -> (L)
Stack	\$) L') L
Input	id , (num)) , (id)) \$
Output	
Stack	\$) L') L' E
Input	id , (num)) , (id)) \$
Output	L -> E L'
Stack	\$) L') L' A
Input	id , (num)) , (id)) \$
Output	E -> A
Stack	\$) L') L' id
Input	id , (num)) , (id)) \$
Output	A -> id
Stack	\$) L') L'
Input	, (num)) , (id)) \$
Output	

Stack	\$) L') L' E ,
Input	, (num)) , (id)) \$
Output	L' -> , E L'

Stack	\$) L') L' E
Input	(num)) , (id)) \$
Output	

Stack	\$) L') L' B
Input	(num)) , (id)) \$
Output	E -> B

Stack	\$) L') L') L (
Input	(num)) , (id)) \$
Output	B -> (L)

Stack	\$) L') L') L
Input	num)) , (id)) \$
Output	

Stack	\$) L') L') L' E
Input	num)) , (id)) \$
Output	L -> E L'

Stack	\$) L') L') L' A
Input	num)) , (id)) \$
Output	E -> A

Stack	\$) L') L') L' num
Input	num)) , (id)) \$
Output	A -> num

Stack	\$) L') L') L'
Input)) , (id)) \$
Output	

Stack	\$) L') L')
Input)) , (id)) \$
Output	L' -> [empty]

Stack	\$) L') L'
Input) , (id)) \$
Output	

Stack	\$) L')
Input) , (id)) \$
Output	L' -> [empty]

Stack	\$) L'
Input	, (id)) \$
Output	

Output

Stack \$) L' E ,
Input , (id)) \$
Output L' -> , E L'

Stack \$) L' E
Input (id)) \$
Output

Stack \$) L' B
Input (id)) \$
Output E -> B

Stack \$) L') L (
Input (id)) \$
Output B -> (L)

Stack \$) L') L
Input id)) \$
Output

Stack \$) L') L' E
Input id)) \$
Output L -> E L'

Stack \$) L') L' A
Input id)) \$
Output E -> A

Stack \$) L') L' id
Input id)) \$
Output A -> id

Stack \$) L') L'
Input)) \$
Output

Stack \$) L')
Input)) \$
Output L' -> [empty]

Stack \$) L'
Input) \$
Output

Stack \$)
Input) \$
Output L' -> [empty]

Stack \$

```
Input      $
Output

Parsing successful!
```

5 实验总结

本次实验中我编写了一个 LL(1) 语法分析程序，使我对语法分析的流程更加清楚，对相关知识点的掌握更加牢固。

此程序的架构比较简单，主要难点在算法设计方面。对于语法分析中涉及的算法，尤其是消除左公因子和求 FOLLOW 集的算法，书上的介绍比较简单，手算和代码实现的差距比较大。

在成功实现这些算法并且通过测试之后，我尝试再次尝试编写了伪代码，这将会为我未来解题带来很大帮助。在实现期间，我使用了 C++ 的语法特性，使得维护文法产生式和预测分析表等变得更加容易，大大减少编程复杂度。

此外，我的语法分析程序仍然存在许多待改进的地方，比如分析错误时可以输出更多的报错信息、在进行文法转换和求解集合时的鲁棒性仍然有待提高，由于时间所限，这些情况我无法一一考虑周全。

本次实验除了让我对课内知识有了更多的认识，也使我的 C++ 编程能力得到提高，我从中收获颇丰。