

编译原理

# C-Like语言的语法分析器项目文档

1652817 钟钰琛 计算机科学与技术



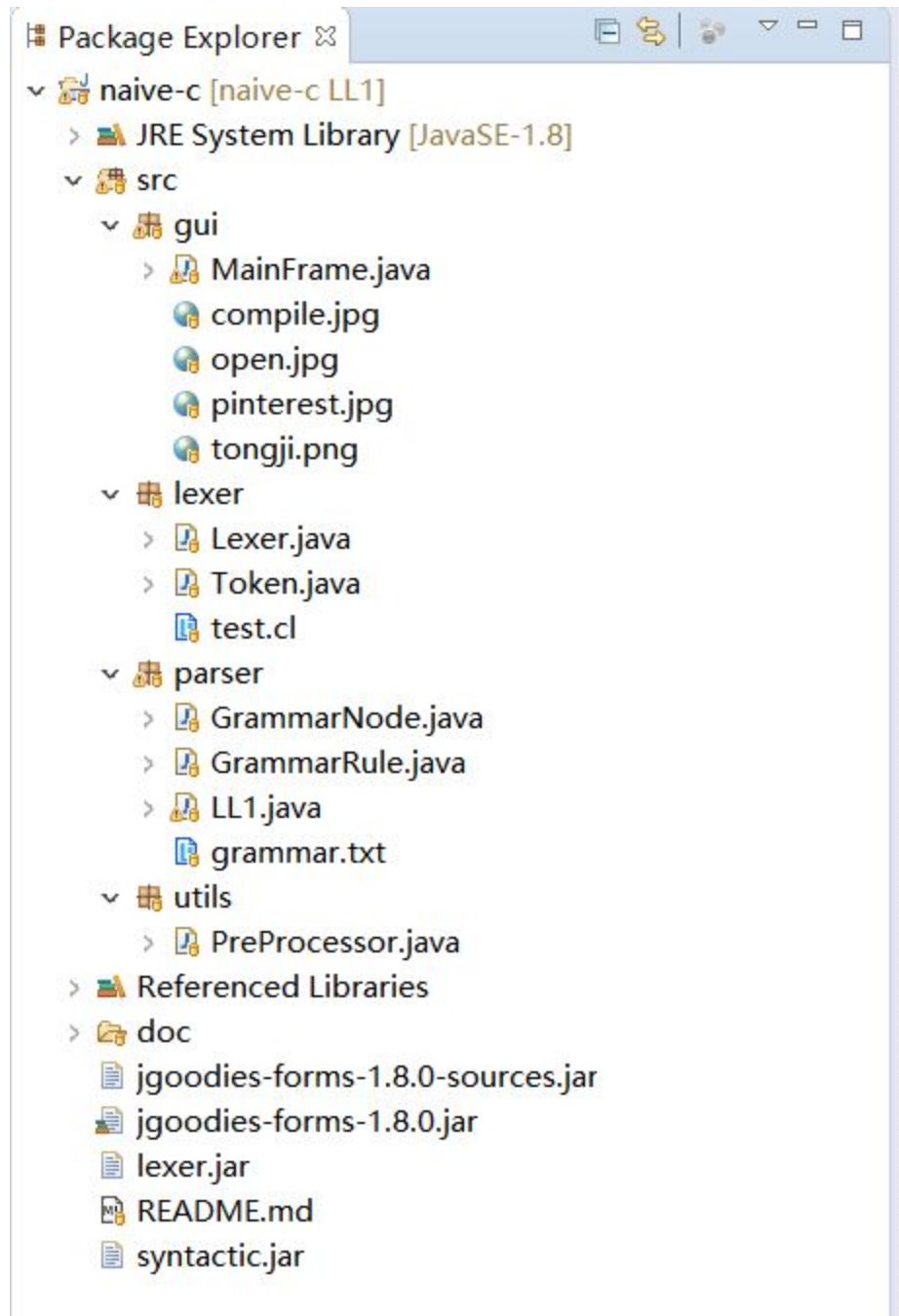
## 简介

语言 : java

JDK: version "1.8.0\_171"

开发IDE : eclipse

文件目录结构 :



parser包：实现语法分析器的主要包

GrammarNode.java 是语法树的节点

GrammarRule.java是记录语法规则

---

LL1.java实现了LL1算法

grammar.txt记录语法规则

lexer包：实现词法分析器的主要包

Lexer.java: 实现词法分析器

Token.java: 包含C-like语言的一些语法规则，以及生成的 <token-name, attribute-value>对

utils包：utilities，一些常用的模块

PreProcessor.java：封装了预处理模块

gui包：实现GUI

MainFrame.java: 带main函数执行入口，GUI的主要定义部分

上次词法分析器已经介绍了除parser以外的部分，这里主要介绍parser

## 设计与实现

**GrammarNode**类：语法树的节点

**数据成员：**

1. 节点的名称

```
private String name;
```

2. 孩子节点的列表

```
public ArrayList<GrammarNode> children;
```

---

## GrammarRule类：记录语法规则的数据结构

### 数据成员：

1.规则左式

```
private String left;
```

2. 规则右式

```
private String[] right;
```

例如， $E \rightarrow T E'$ ， $left = E$   $right[0] = T$   $right[1] = E'$

## LL1类：实现LL1算法

### 数据成员：

1. 规则记录列表

```
private ArrayList<GrammarRule> grammar_list;
```

2. 终结符

```
private TreeSet<String> terminals;
```

3.非终结符

```
private TreeSet<String> non_terminals;
```

4.FIRST集

```
private HashMap<String, ArrayList<String>> first_set;
```

---

## 5.FOLLOW集

```
private HashMap<String, ArrayList<String>> follow_set ;
```

## 6.分析预测表

```
private HashMap<String, GrammarRule> predict_map;
```

## 7.栈

```
private Stack<String> stack;
```

## 8.语法树的根节点

```
private GrammarNode root;
```

## 9.词法分析器

```
public Lexer lexer;
```

## 成员函数：

```
1.private boolean hasNull(String left)
```

判断是否有 $X \rightarrow \$$ 这样的产生式 ( \$代表空，下同 )

```
2.private boolean buildFirst()
```

建立FIRST集

```
3.private boolean buildFollow()
```

建立FOLLOW集

```
4.private void readGrammar(String grammar)
```

读取语法规则

---

5. `private boolean` buildPredictMap()

建立分析预测表

6. `private boolean` syntacticAnalysis(ArrayList<Token> input)

语法分析函数，输入时经过词法分析后的token序列

7. `private void` dfs(GrammarNode cur, `int` depth)()

前序遍历语法树

## 算法设计：

算法参考PPT和课本，不再赘述，这里讲算法的具体实现

### 1. 求FIRST集

首先把所有的终结符加入到各自的FIRST集中，然后注册下非终结符的FIRST集（还是空的）

```
// build FIRST SET
public void buildFirst() {
    ArrayList<String> first;

    // terminals' first set
    for(String terminal : terminals) {
        first = new ArrayList<String>();
        first.add(terminal);
        first_set.put(terminal, first);
    }

    for(String non_terminal : non_terminals) {
        first = new ArrayList<String>();
        first_set.put(non_terminal, first);
    }
}
```

---

然后反复遍历语法规则，直到FIRST集不再变化（flag来控制，flag为真break）

```
while(true) {
    flag = true;
    for(GrammarRule grammarRule : grammar_list) {
        left = grammarRule.getLeft();
        rights = grammarRule.getRight();

        for(String right : rights) {
            // X -> a... or X -> $ => add a or $ into FIRST(X)
            if(terminals.contains(right)) {
                if(!first_set.get(left).contains(right)) {
                    first_set.get(left).add(right);
                    flag = false;
                }
            } // X -> Y... => add FIRST(Y) - $ into FIRST(X)
            else if(non_terminals.contains(right)){
                first = first_set.get(right);
                for(String fi : first) {
                    if(!fi.equals("$")) {
                        if(!first_set.get(left).contains(fi)) {
                            first_set.get(left).add(fi);
                            flag = false;
                        }
                    }
                }
            }
            else {
                System.err.println("YOU GOT SOME BUGS!");
            }

            // if X has 'X -> $', then turn to next right symbol
            if(hasNull(right)) {
                continue;
            } else {
                break;
            }
        }
    }

    if(flag) {
        break;
    }
}
```

对于每个语法规则，遍历其规则右式，如果是形如

( 1 )  $X \rightarrow a...$  这种类型，那么就把a加入到FIRST(X)中，

( 2 )  $X \rightarrow Y...$  这种类型，那么把FIRST(Y) - \$ 加入到FIRST(X)中

---

但是如果右式的当前元素可能推导出 $\epsilon$ ，那么需要继续遍历下一个右式元素

( hasNull那个if - else控制 )

## 2.求FOLLOW集

```
while(true) {
    flag = true;
    for(GrammarRule grammarRule : grammar_list) {
        left = grammarRule.getLeft();
        rights = grammarRule.getRight();

        for(int i = 0; i < rights.length; ++i) {
            right = rights[i];
            // B -> ...A...
            if(non_terminals.contains(right)) {
                flag2 = true;
                for(int j = i + 1; j < rights.length; ++j) {
                    first = first_set.get(rights[j]);
                    for(String fi : first) {
                        if(!follow_set.get(right).contains(fi) && !fi.equals("$")) {
                            follow_set.get(right).add(fi);
                            flag = false;
                        }
                    }

                    if(hasNull(rights[j])) {
                        continue;
                    } else {
                        flag2 = false;
                        break;
                    }
                }
                // end of rights from j = i + 1 to end

                // B -> ...A or B -> ...Ab && b => $
                if(flag2) {
                    left = grammarRule.getLeft();
                    follow = follow_set.get(left);
                    for(String fo : follow){
                        if(!follow_set.get(right).contains(fo)) {
                            follow_set.get(right).add(fo);
                            flag = false;
                        }
                    }
                }
            }
        }
    }
} // end of if non_terminals
} // end of traverse rights
} // end of traverse grammar list
```

---

求FOLLOW ( A ) 会涉及到求FIRST (  $\beta$  ) ，具体方是如果 $\beta = X_1X_2X_3\dots$  先把FIRST( $X_1$ )加入到FIRST( $\beta$ ) 中，然后判断 $X_1$ 是否可能推导出空，如果是那么继续将FIRST( $X_2$ )加入到FIRST( $\beta$ ) 中，否则break，以此类推。



---

FOLLOW只考虑非终结符，所以遍历右式时，只需考虑非终结符就可以了。

如果发现一个终结符，那么

( 1 ) 就算出其后元素的FIRST集，然后加入到FOLLOW集中

( 2 ) 如果正好是最后一个元素或者后续元素能推出空，那么把左式的FOLLOW集加入到FOLLOW集中

### 3.求分析预测表

```
for(GrammarRule grammarRule : grammar_list) {
    left = grammarRule.getLeft();
    rights = grammarRule.getRight();

    rights_first = new ArrayList<String>();

    flag = false;
    for(String right : rights) {
        first = first_set.get(right);
        for(String fi : first) {
            if(!rights_first.contains(fi)) {
                if(fi.equals("$")) {
                    flag = true;
                } else {
                    rights_first.add(fi);
                }
            }
        }

        if(hasNull(right)) {
            continue;
        } else {
            break;
        }
    }

    for(String fi : rights_first) {
        predict_map.put(left + "@" + fi, grammarRule);
    }

    // has A -> $
    if(flag) {
        rights_first.add("$");

        follow = follow_set.get(left);
        for(String fo : follow) {
            predict_map.put(left + "@" + fo, grammarRule);
        }
    }
}
}
```

---

---

同样也需要FIRST (  $\beta$  ) , 方法同上。这里存放表的方式是用了HashMap ( 参考 predict\_map数据结构 HashMap<String, GrammarRule> )

Key的String用M[A, a]中的 A@a方式表示

Value则是产生式 ( 实际上只要右式即可 )

#### 4.根据分析预测表进行语法分析

```
while(!stack.empty() && index < len) {
    top = stack.pop();
    in = input.get(index);

    if(top.equals(in.getTokenName())) {
        if(top.equals("#")) {
            break;
        }
        index++;
        continue;
    }

    key = top + "@" + in.getTokenName();

    // System.out.println(cnt + ":" + key);
    grammarRule = predict_map.get(key);
    if(grammarRule != null) {
        rights = grammarRule.getRight();

        if(rights[0].equals("$")) {
            continue;
        }

        result.append(cnt);
        result.append(": ");
        result.append(grammarRule);
        result.append("\n");

        for(int i = rights.length - 1; i >= 0; --i) {
            stack.push(rights[i]);
            tmp = new GrammarNode(rights[i]);
            cur.children.add(tmp);
        }

        cur = tmp; // the last

        cnt++;
    }
    else {
        result.append("Reject!\n");
        return false;
    }
}
```

---

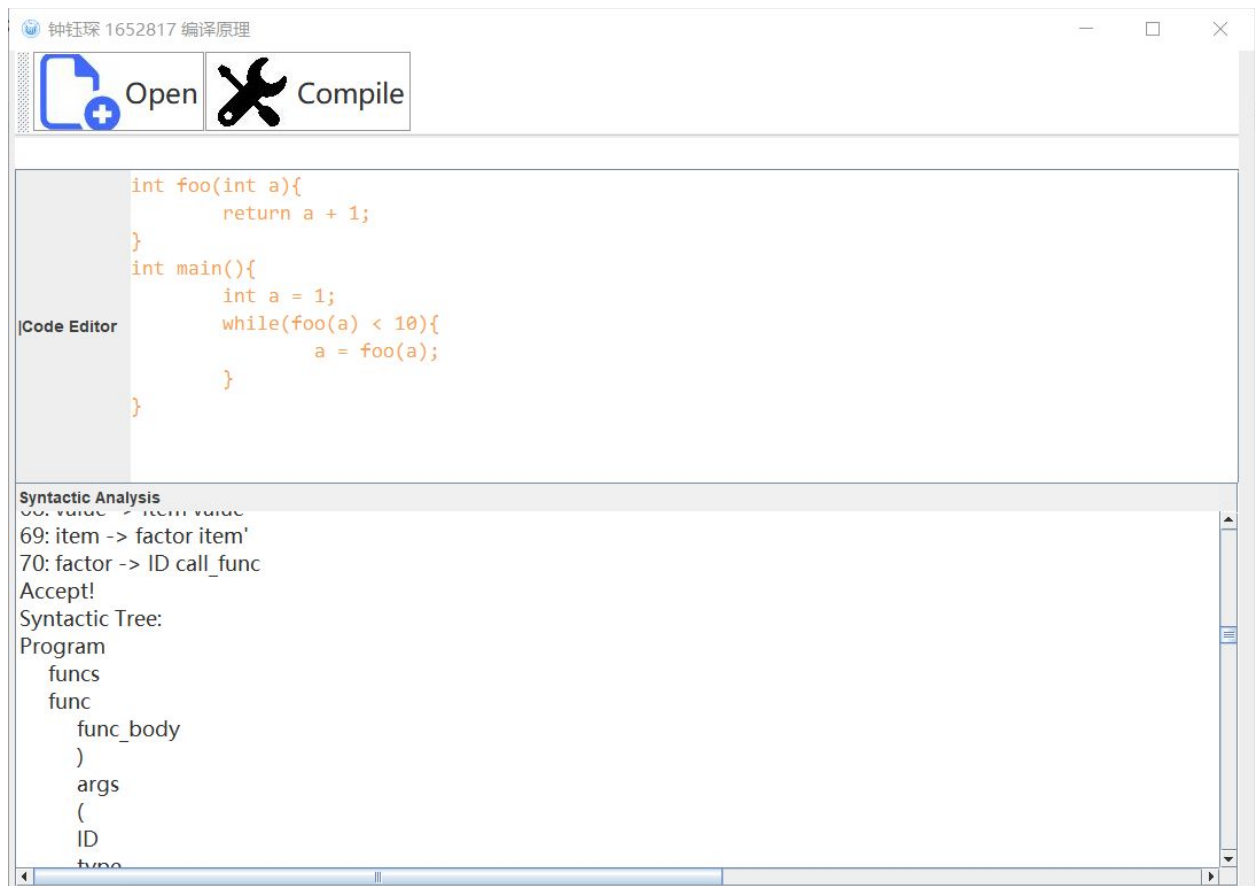
- ( 1 ) 如果栈顶和输入的字符一样，那么都删除
- ( 2 ) 如果查分析预测表成功，那么做最左推导
- ( 3 ) 如果查表不成功，那么报错

## 使用说明



1. 打开文件：会判断是否是xxx.cl结尾的文件，如果不是，会在代码编辑框显示红色报错信息。读取成功后，文件的内容会显示在代码编辑框内。
2. 执行语法分析：这里用了compile这个词，是为了后续继续做的考虑。结果将显示在下面的文本框中。
3. 代码编辑框：橙色字体。还没弄代码高亮.....

#### 4. 词法分析结果：会打印 推导过程 和 语法树



The screenshot shows a compiler IDE window titled "钟钰琛 1652817 编译原理". It has two buttons: "Open" (with a file icon) and "Compile" (with a wrench icon). The main area is a "Code Editor" containing the following C code:

```
int foo(int a){  
    return a + 1;  
}  
int main(){  
    int a = 1;  
    while(foo(a) < 10){  
        a = foo(a);  
    }  
}
```

Below the code editor is a "Syntactic Analysis" panel. It displays the following output:

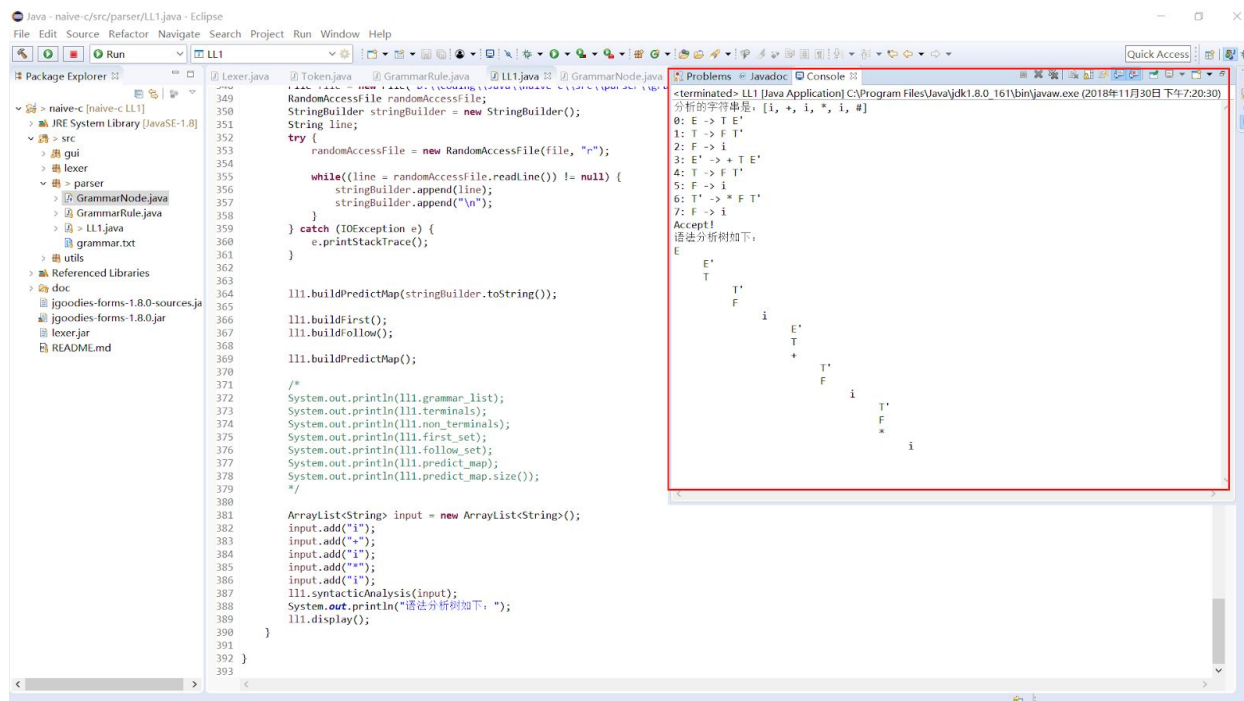
```
68: value -> item value  
69: item -> factor item'  
70: factor -> ID call_func  
Accept!  
Syntactic Tree:  
Program  
  funcs  
  func  
    func_body  
  )  
  args  
  (  
  ID  
  type
```

## 调试分析

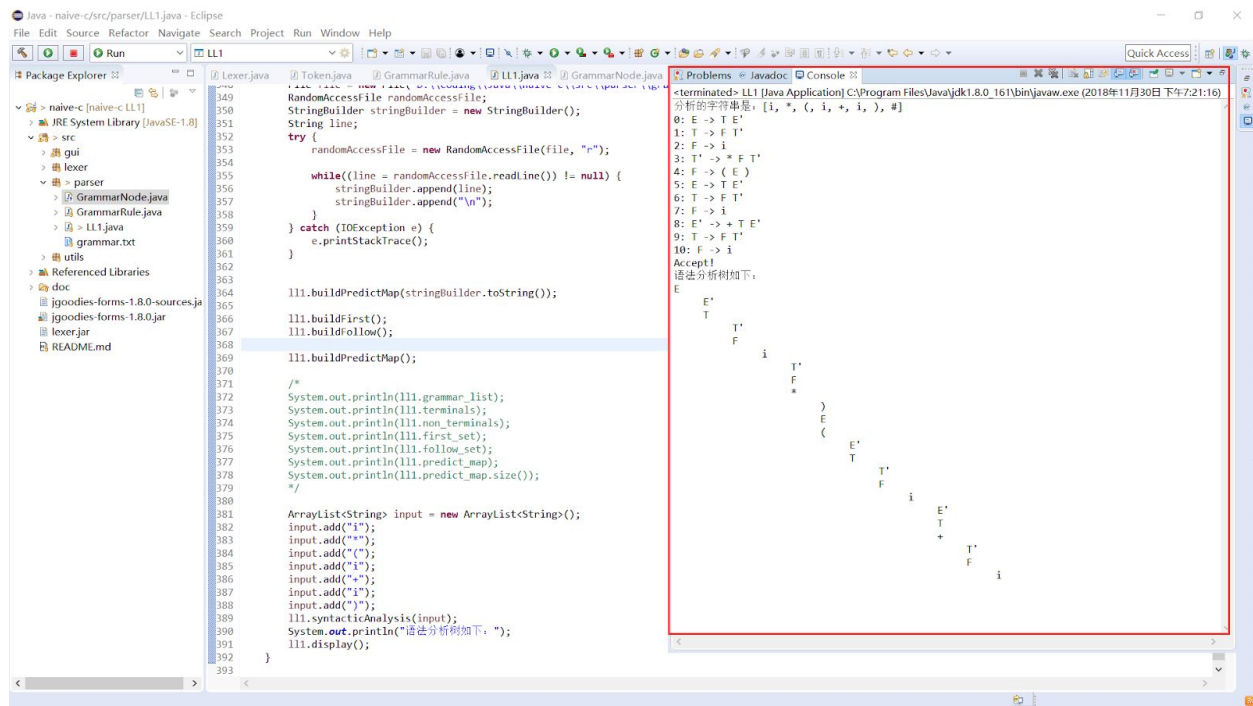
先在控制台测试了PPT上的语法！

**例**

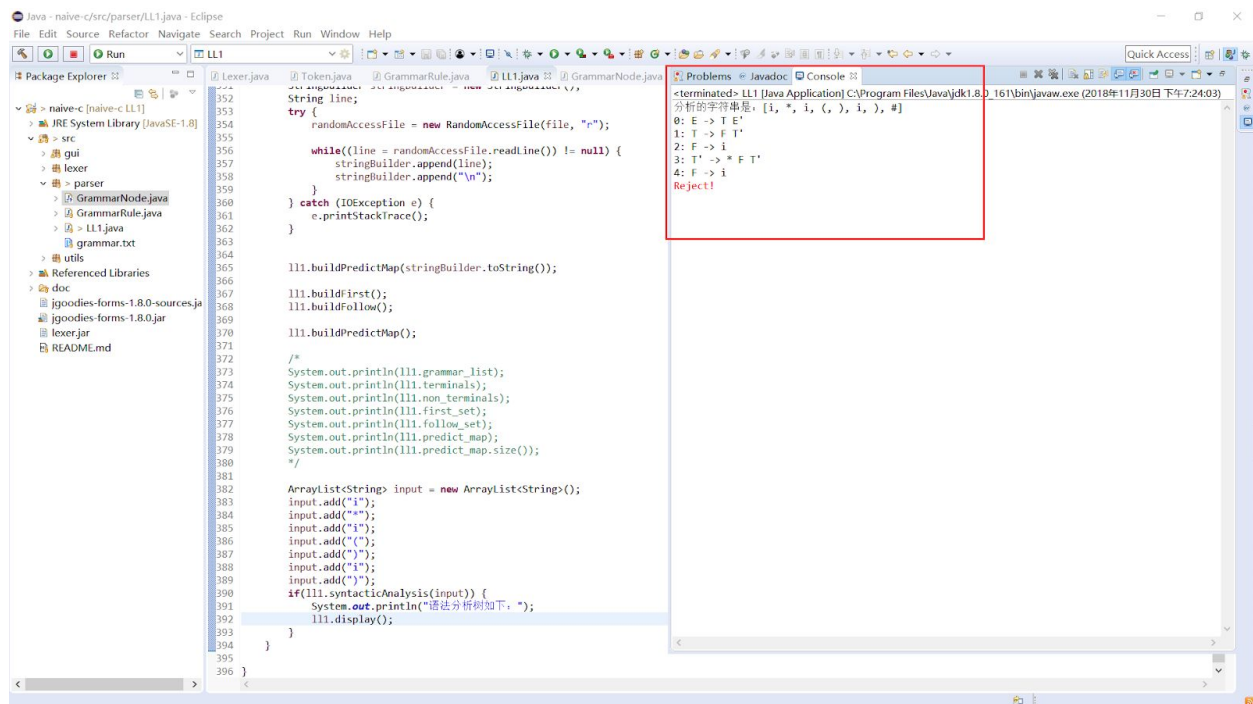
**G:**  $E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \varepsilon$   
 $F \rightarrow (E) \mid i$



测试  $i + i * i$  Accept!

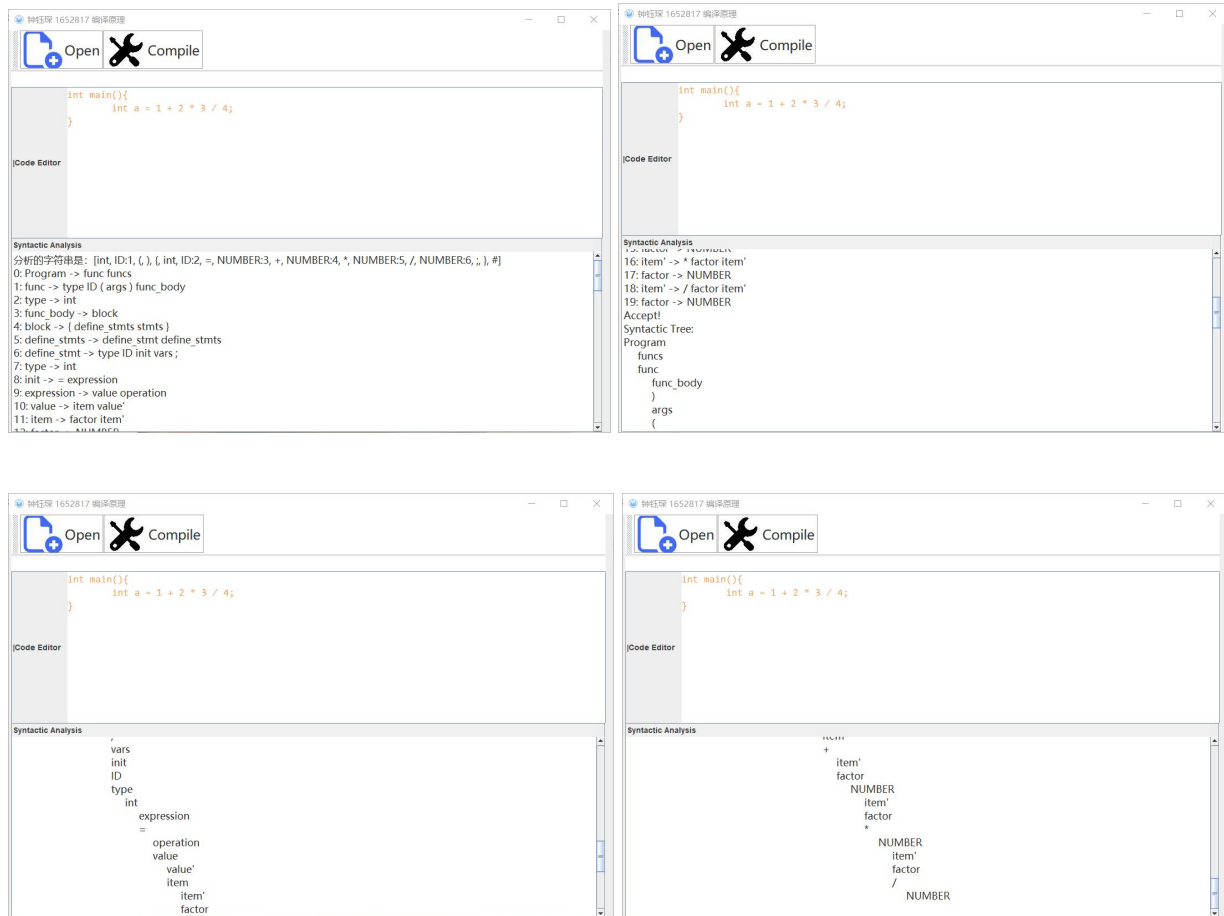


测试  $i * (i + i)$  Accept!



测试  $i * i ( ) i$  Reject!

然后在类C语法上做了测试！



( 输出太长了，故截了四张图 )



这里reject是因为缺少分号！

项目开源地址：<https://github.com/zhongyuchen/naive-c>



