编译原理课程实验报告

实验 2：语法分析

姓名	张恒		院系	计算机		学号	1160300620	
任课教师	辛明影			指导教师	辛明影			
实验地点	格物 208			实验时间	2019.4.20			
实验课表现	出勤、表现得分			实验报告得分		实验总分		
	操作结果得分							

一、需求分析

得分

（1） 实验目的

使用句法分析技术 LR（1）对类高级语言中的基本语句进行句法分析。

（2） 实验内容

在词法分析器的基础上设计实现类高级语言的语法分析器，完成如下基本功能：

一是能识别几类基本的语句，即声明语句（变量声明）、表达式及赋值语句（简单赋值）、分支语句（if_then_else）以及循环语句（do_while）。

二是在随意给出一个文法的情况下，能够自动计算相应的闭包等，并且构造出 LR（1）分析表。

三是具备简单语法错误处理能力，能准确给出错误所在位置，并采用可行的错误恢复策略。输出的错误提示信息格式为：Error at Line [行号]：[说明文字]。

四是能够通过文件导入文法和测试用例，其中测试用例要能够涵盖前面所说的四种语句，并需要设置一些语法错误。

五是系统的输出部分，需要打印出语法分析器的 LR（1）分析表，以及语法分析的结果，即规约时的产生式序列。

（3） 语法分析系统要完成的功能

语法分析(syntax analysis)是编译程序的核心部分，其任务是检查词法分析器输出的单词序列是否是源语言中的句子，亦即是否符合源语言的语法规则。语法分析器的功能位置如图 1-1 所示。

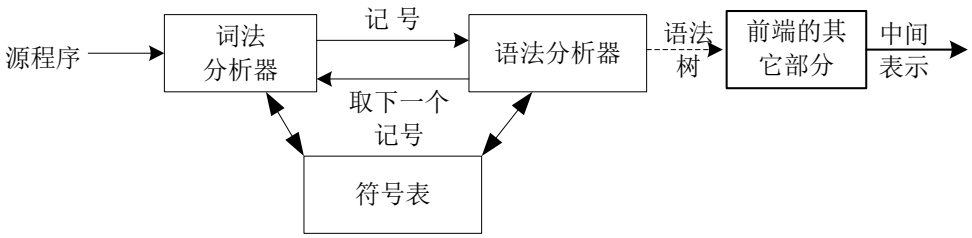

图 1-1 语法分析器的功能位置

二、文法设计

得分	

本次实验中我使用如下的文法作为语法分析器的测试用例，当然，分析器能够处理的文法不限于实验中测试的文法。对该文法，有如下的说明：

（1） 文法中的非终结符均包含的字母符号均是大写字母，终结符包含的字母符号均是小写字母；

（2） 文法中每一行只包含一个推导式；

（3） 如果一个推导式中包含有多个符号（终结符或者非终结符），符号间用空格隔开；

（4） 空串用“epsilon”表示；

（5） 文法必须是拓广文法；

说明	产生式
程序总结构。 描述程序中几种结构的语句可能的顺序	PP->P P->D P->S P->D S S->S S
声明语句	D->D D D->proc id ; D S D->T id ; T->X C T->record D X->int X->float X->bool C->[consti] C C->epsilon
表达式及赋值语句	S->id = E ; S->L = E ; E->E + T E->E - T E->T T->T * F T->F T->id T->consti T->constf T->L L->id [E] L->L [E]
	S->UNMATCHS S->MATCHS MATCHS->if (B) then MATCHS else MATCHS

控制语句，即 条件语句 循环语句	UNMATCHS->if (B) then S UNMATCHS->if (B) then MATCHS else UNMATCHS S->while B do S B->B or B B->B and B B->not B B->(B) B->E RELOP E B->true B->false RELOP-> RELOP-><= RELOP->== RELOP->!= RELOP->>= RELOP->>=
过程调用（暂且不打算支持过程调用，这里仅用于测试语法分析器的 LR（1）分析表 ne 那个否构建成功）	S->call id (ELIST) ELIST->ELIST , E ELIST->E

三、系统设计

得分

（1） 系统概要设计

系统框架图

系统框架如图 3-1 所示。用户界面是用户与分析器的交互媒介，用户通过用户界面调用语法分析器的模块，来进行源代码的语法分析。在这里还需要两个辅助的模块，一个是操作系统提供的文件选择 API，另一个则是前一个实验中的词法分析模块。

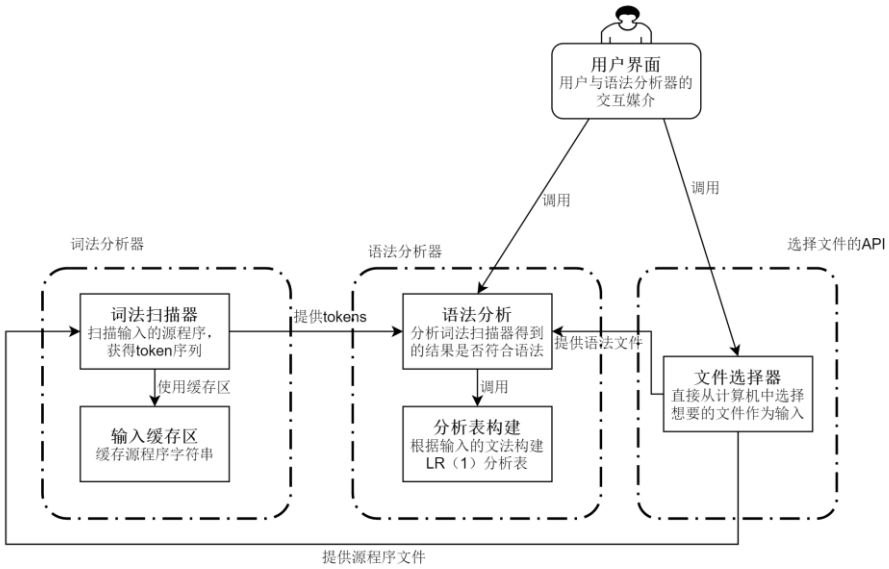

图 3-1 系统框架图

数据流图

程序的数据流起始与用户的输入，先分别用此法处理程序处理源程序输入流，得到 token 流，用语法处理程序中的构建 LR（1）分析表的模块得到 LR（1）分析表。然后语法分析器的分析模块处理这两个输入流就能得到结果。

结果可能会有两种情况，在分析成功时，得到的是分析中的规约式；分析失败时，得到的是分析中得到的错误信息。程序的数据流如图 3-2 所示。

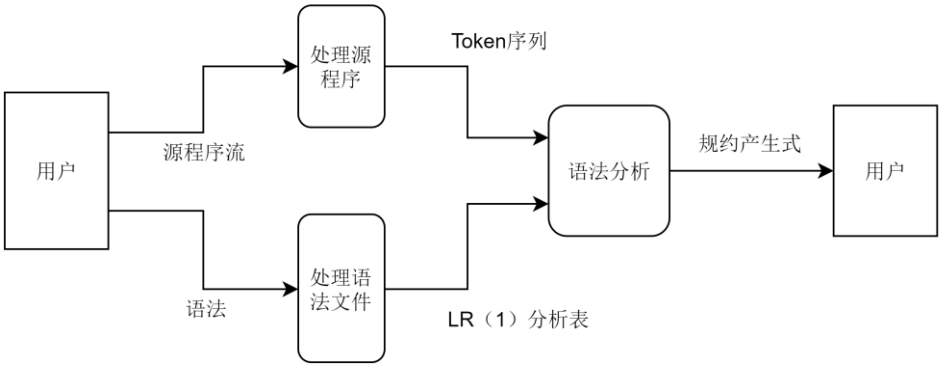

图 3-2 数据流图

功能模块结构图

系统的功能模块结构如图 3-3 所示。程序主要分为两个模块，一是构建 LR（1）分析表，二是分析。但是本次实验中，需要依托词法分析器，因此将词法分析器作为本程序的一个和前两者并列的模块。

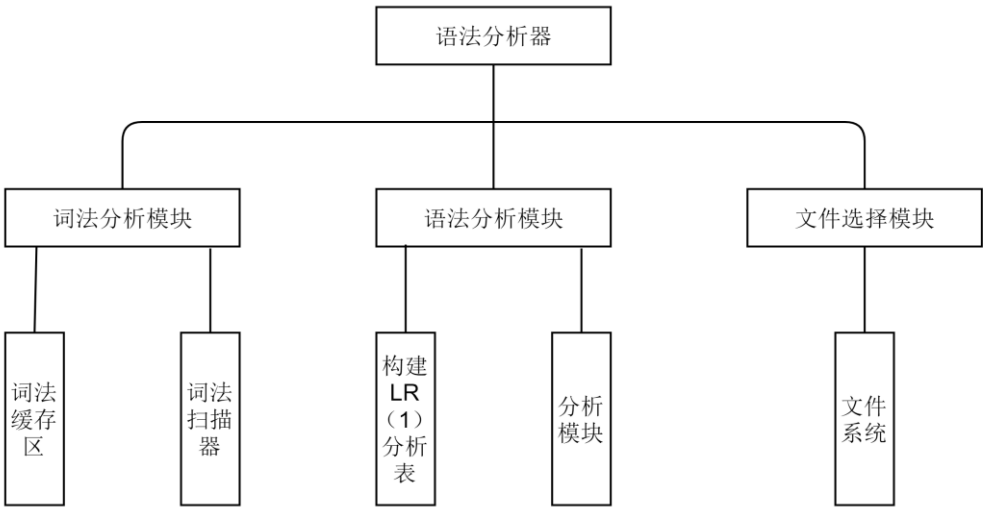

图 3-3 功能模块结构图

（2） 系统详细设计

数据结构的设计

语法分析器中涉及到几个比较重要的数据结构，根据每种数据结构的使用情况，我做如下设计：

一是文法的表示。文法的表示包括推导式、终结符集合和非终结符集合。在构建 LR（1）分析表时，需要用表格的形式来存储，这就需要将每个终结符和非终结符以及推导式对应于一个唯一的序号，因此采用顺序表的方式，用 list 实现。其中每个符号用一个 String 表示。

二是推导式和 LR 项目的表示。LR 的项目可以分为四个部分，推导式左部、推导式右部、圆点位置和后继符号集合。前两者是不需要改变的，但是后两者需要频繁改变，因此我新建了一个数据结构用来表示一个项目，并且将文法中的推导式视为一个特殊的项目。这个数据结构的属性列表如图 3-4 所示。

```
private String left; // 左部非终结符
private List<String> right = new ArrayList<String>(); // 右部符号集合
private int dotPosition; // 圆点的位置
private Set<String> successor = new HashSet<String>();  //  后继符号集合
```

图 3-4 LR（1）项目的数据结构的属性

三是 FIRST 集的表示。在计算时会频繁用来某个非终结符的 FIRST 集，因此我用字典，也就是 java 中的 Map 来实现非终结符和其 FIRST 的对应。并且考虑到 FIRST 需要频繁的做一些集合的操作，且不需要在意集合中符号的顺序，因此每个 FIRST 集我用一个集合 Set 来表示。

四是项目集族的表示。对每个项目集，可以不用关心其中每个项目的顺序，因此我用集合 Set 来表示一个项目集。但是在分析表中，需要得到每个项目集和一个唯一序号的对应，因此项目集族就是项目集的一个有序列表，用 List 实现。

五是分析表。分析表是一个二维的表，其中涉及到集中数据结构，这是难以实现的，

因此我将涉及到的数据全部映射为一个整数。也就是说，我用一个 int 类型的二维数组来表示分析表，其中的行是项目集的编号，列是非终结符或者是终结符的编号。具体来说，我将分析表分为 action 表和 goto 表两个表。

整个语法分析器涉及到的数据结构如图 3-5 所示。

```
public static int acc = 100000; //  接收状态的编码
private List<String> terminals = new ArrayList<String>();   //  终结符的集合
private List<String> varibles = new ArrayList<String>();    //  非终结符的集合
private List<LR1Item> grammers = new ArrayList<LR1Item>();  //  推导式额集合
private Map<String, Set<String>> firsts = new HashMap<String, Set<String>>();   //  非终结符和FIRST集的对应
private List<Set<LR1Item>> LRfamily = new ArrayList<Set<LR1Item>>();    //  项目集族
private int[][] actionTable;    //  action 表
private int[][] gotoTable;  //  goto 表
private List<String> errors = new ArrayList<String>();  //  语法分析的错误信息列表
private List<LR1Item> rules = new ArrayList<LR1Item>(); //  语法分析中的规约式集合
```

图 3-5 语法分析器设计到的数据结构概览

主要功能函数

主要功能函数有五个。

private void computeFirsts()。构建非终结符的 FIRST 集。

private Set<LR1Item> computeClosure(LR1Item item)。计算给定项目的项目集闭包。

private void computeLRfamily()。计算项目集族。

public void buildAnalysisTable(String grammer)。构建所给文法的 LR（1）分析表。

public void parse(List<LexicalToken> tokens)。分析所给 token 序列是否符合文法规则。

程序流程

程序的流程比较简单，如图 3-6 所示。

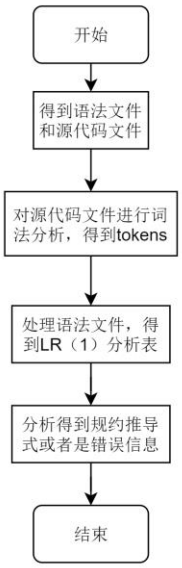

图 3-6 程序的流程图

四、系统实现及结果分析	得分	

要求：对如下内容展开描述。

（1） 系统实现过程中遇到的问题；

（2） 输出该句法分析器的分析表；

（3） 针对一测试程序输出其句法分析结果；

（4） 输出针对此测试程序对应的语法错误报告；

（5） 对实验结果进行分析。

注：其中的测试样例需先用已编写的词法分析程序进行处理。

（1） 遇到的问题

在实现系统的过程中，遇到了比较多的问题。

首先，遇到的问题就是测试用例的问题，在写程序的过程中，求 FIRST 集和闭包以及项目集族时，在课件上是可以找到相应的例子的，但是在计算分析表的时候，课件上涉及到的一些例子均有一些小问题，在排除 bug 的过程中因此花费了较多的时间。为此，我手工对一些例子进行了求解。

然后就是文法的设计。LR（1）文法是一种没有二义性的文法，要构造一个包含基本语句的文法比较有难度。实验指导中给出了一种文法，但是这是一个有二义性的文法，需要对其进行改造。

三是语法分析器中涉及的数据的表示。涉及到比较多的数据，包括项目集族等。为了简单表示这些数据机构，并且兼顾快速的处理，并且能够尽量少使用存储空间。这是一个问题。

四是进行错误处理。错误处理我使用的是恐慌模式，在进行弹栈寻找合适的状态后，恰好不需要丢弃输入符号，此时可能陷入死循环，为此，我规定错误处理时至少丢弃一个输入终结符。

（2） 语法分析器的分析表

语法分析器的分析表比较大，因此程序将语法分析产生的分析表存在了文件 lr1Table.txt 中。

（3） 测试正确程序的结果

如图 4-1 所示，是程序的用户界面。有五个文本显示区域，其中左侧的两个分别是源代码和文法的输入框，支持从文件中直接导入。

最左侧的两个框是语法分析的结果，其中一个是用于展示 LR（1）分析表，另一个是展示分析结果，有两种情况，一个是分析成功时，会展示规约式。另一个是分析失败时，会展示出错的结果。

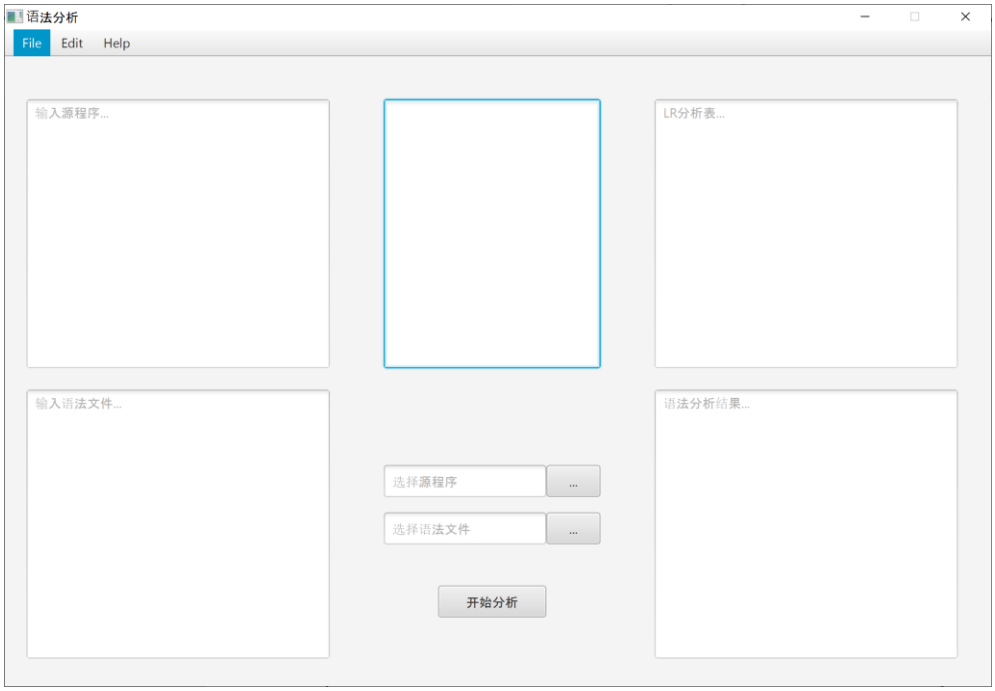

图 4-1 语法分析器的用户界面

如图 4-2 所示是一段正确的源程序

```
int i;
int j;
i=4+i*2;
j=10;
if (i<j) then
while(j<100) do
j=j+1;
```

图 4-2 一段正确的源程序的实例

这段程序中包括了声明、赋值、条件和循环四种基本的类型。对这段代码利用上面定义的文法分析，得到的结果如图 4-3 所示

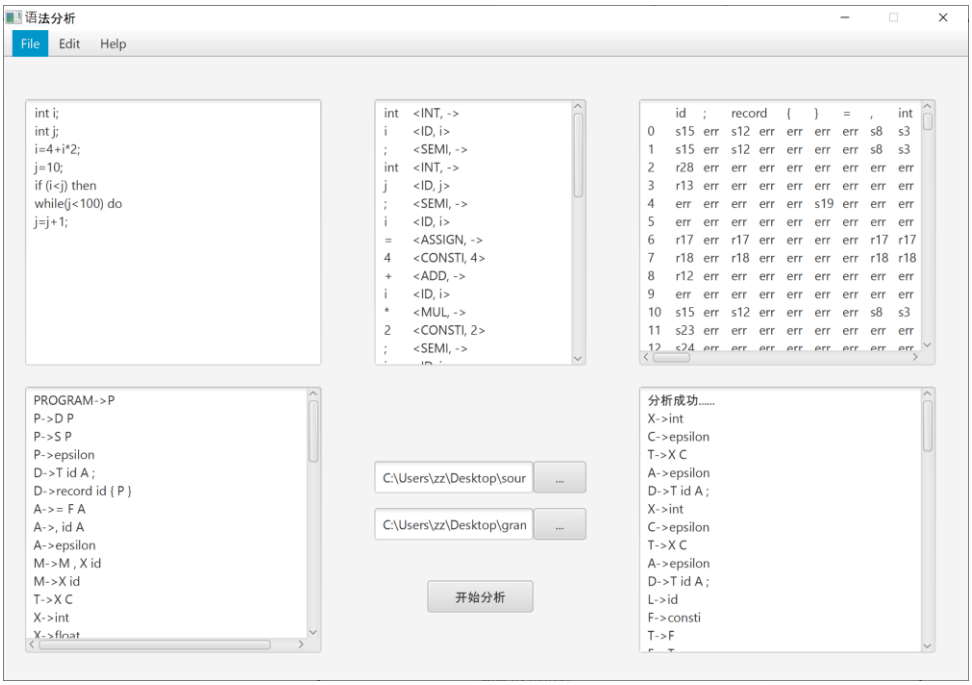

图 4-3 正确源程序的分析结果

可以看到，分析是成功的，并且在右下角的框中列出了规约式序列。

（4） 测试错误程序的结果

对正确的源程序稍作修改，引入两处错误，如图 4-4 所示。

```
int i;
int j;
i=4+i*2;
j=10;@
if (i<j) then ;
while(j<100) do
j=j+1;
```

图 4-4 错误的源程序

对该源程序进行分析的结果如图 4-5 所示，可以看到，分析失败了，并且提示除了两处错误。实际上，语法分析器采用恐慌模式恢复错误，会试图尽可能多的发现程序中的错误。

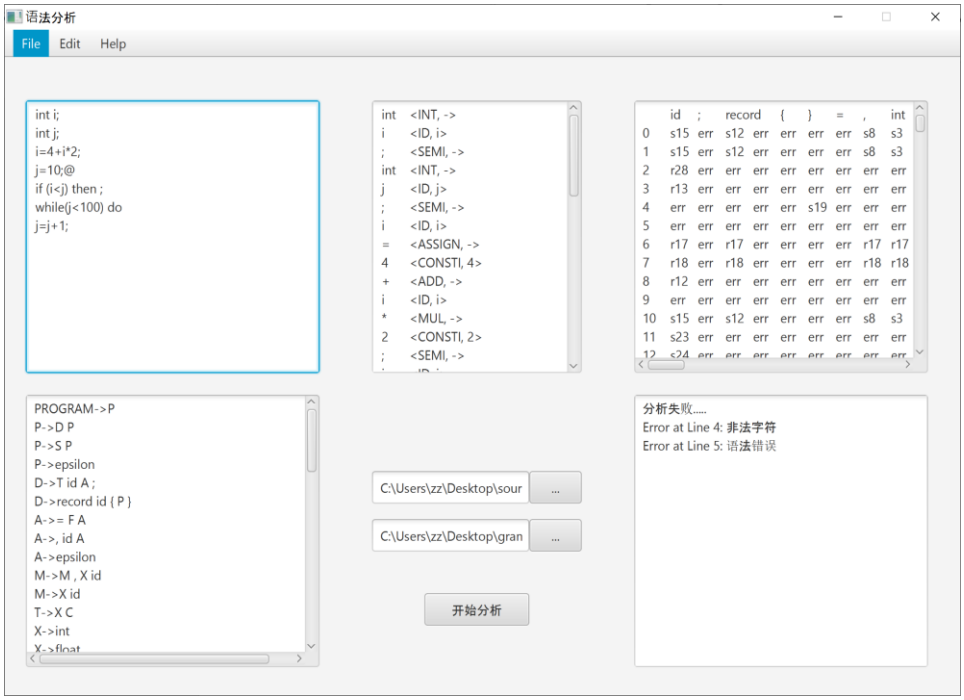

图 4-5 错误源程序的分析结果

（5） 结果分析

可以看到，该语法分析器基本能够正常的工作，能够根据所给的文法自动产生 LR（1）分析表，并根据这个表来分析输入的源程序是否符合语法。在分析成功时，能够将规约式输出。在不符合语法时，能够输出错误信息，包括错误的行。

程序有很多不足的地方，比如程序只能检测文法是否有冲突，但是并不能自动解决冲突。

指导教师评语：

日期：