



编译原理实验

实验与创新实践教育中心

杨扬

编译原理实验安排

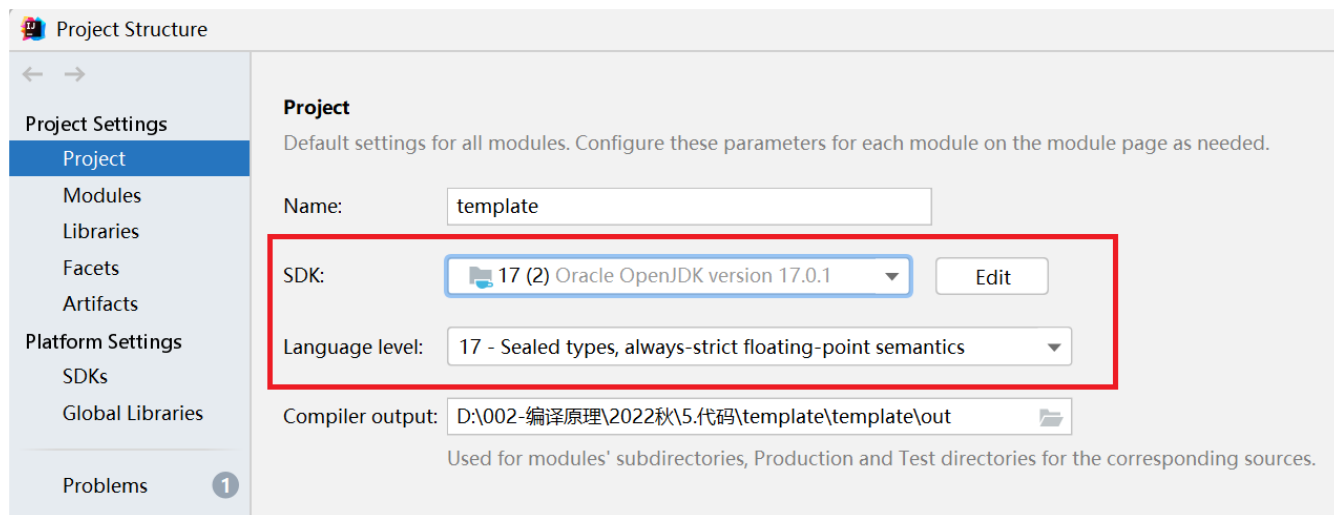


● 实验目标

- ✓ 实现一个编译器
- ✓ 目标平台是RISC-V 32

● 实验语言和环境

- ✓ JAVA语言
- ✓ IntelliJ IDEA
- ✓ JDK17及以上版本



编译原理实验安排



● 代码框架

✓ 代码框架：包括主要函数流程，部分数据结构，通用函数，实验输入及参考实验结果等。

✓ 代码框架地址：<https://gitee.com/hitsz-cslab/Compiler/releases/latest>

学生实验包 

 SoraShu

- template.zip: 代码模板
- rars.jar: rars 的 jar 包
- 编译工作台-setup.exe: 编译工作台的安装包

Source Code 为自动创建的仓库二进制文件, 无需下载

最后提交信息为: 增加了快速比对脚本

下载

实验四 →  template.zip

实验二 →  rars.jar

 编译工作台-setup.exe

 下载 Source code (zip)

 下载 Source code (tar.gz)

编译原理实验安排



● 代码框架

✓ 帮助文档: <https://hitsz-cslab.gitee.io/compiler>

先阅读帮助文档再开发

 编译原理实验 (2022 春季) | 哈工大 (深圳)

Q 搜索

 HITSZ-ECEI-CS/Compiler

首页 实验一: 词法分析 毕昇杯广告

首页
实验概述
实验环境及代码规范
脚本工具
问题反馈

STC (Simple TXT Compiler)

一个 Java 实现的 TXT 语言编译器, 目标平台为 RISC-V 32 (指令集 RV32M).

目录说明

目录
目录说明
代码架构示意图
源语言
目标平台 && 目标语言
关于 NotImplementedException

编译原理实验安排



● 代码框架

✓ 帮助文档: <https://hitsz-cslab.gitee.io/compiler>

快速比对脚本

diff.py

Usage:

```
python -u diff.py /path/for/stdfile /path/for/srcfile
```

比对 /path/for/stdfile 与 /path/for/srcfile 的差异, 忽略文件尾空行和行首位空白符. 若有差异则输出首行不匹配.

check-result.py

Usage:

```
python -u check-result.py n /path/for/stdfiles /path/for/outputfiles
```

其中 n 为实验 n 的序号, 如 4 代表 实验四; /path/for/stdfiles 与 /path/for/outputfiles 是标准文件的文件夹和输出文件的文件夹.

此脚本会比对本次实验和之前的所有实验的输出文件和标准文件的差异. 特别地, 对于实验四, 我们直接输出了 rars 的输出, 需要自行和程序的预期返回值比对.

脚本中写死了 rars.jar 的路径, 需要修改为你电脑上 rars.jar 的路径:

```
rars_path = "/home/test/rars.jar"
```

tree . -L 2 --dirsfirst --sort=name

```
.
├── data
│   ├── in          # 提供给程序的输入数据
│   ├── out         # 程序的输出数据
│   └── std          # 用作参考的标准输出数据
├── scripts
│   ├── check-result.py # 对输出进行 check 的脚本
│   ├── diff.py        # 忽略文件尾空行和行首位空白符的 diff 工具
│   └── make-template.py # 从代码出框架的脚本, 无需关注
└── src               # 源码目录
```

tree src/cn/edu/hitsz/compiler -L 2 --dirsfirst --sort=name

```
src/cn/edu/hitsz/compiler
├── asm
│   └── AssemblyGenerator.java # 实验四: 汇编生成
│                                   # 汇编生成器
├── ir
│   ├── Instruction.java      # IR 指令
│   ├── InstructionKind.java  # IR 指令类型
│   ├── IRImmediate.java     # IR 立即数
│   ├── IRValue.java         # IR 值
│   └── IRVariable.java      # IR 变量
├── lexer
│   ├── LexicalAnalyzer.java # 实验一: 词法分析
│   │                       # 词法分析器
│   ├── Token.java          # 词法单元
│   └── TokenKind.java       # 词法单元类别
├── parser
│   ├── table
│   │   ├── ActionObserver.java # 实验二/三: 语法分析, 语义分析, IR 生成
│   │   │                       # 读取/生成 LR 表的工具
│   │   ├── IRGenerator.java   # 观察者接口
│   │   ├── ProductionCollector.java # IR 生成
│   │   ├── SemanticAnalyzer.java # 规约产生式记录
│   │   └── SyntaxAnalyzer.java # 语义分析
│   └── symtab
│       ├── SourceCodeType.java # 符号表
│       │                       # 源语言变量类型
│       ├── SymbolTableEntry.java # 符号表条目
│       └── SymbolTable.java     # 符号表
├── utils
│   ├── FilePathConfig.java    # 杂项/工具
│   │                       # 记录程序中用到的各种路径
│   ├── FileUtils.java        # 文件读写工具
│   └── IREmulator.java       # 评测用 IR 解释器
├── Main.java                  # 主函数
└── NotImplementedException.java # 用于填充待实现部分的异常
```

编译原理实验安排



● 代码框架

- ✓ 在IDEA Terminal中运行比对脚本，注意文件路径（机器需安装Python）

```
Terminal: Local x + v
PS D:\project\CompilerProject\template> python -u scripts/diff.py data/out/token.txt data/std/token.txt
The src file is the same as std file.
PS D:\project\CompilerProject\template>
```

Version Control TODO Problems Profiler Terminal

- ✓ 实现TODO标记的方法，有可能需要设计数据结构
- ✓ 忽略尚未开始的实验代码抛出的NotImplementedException

编译原理实验安排



✓16个学时，共计6次实验课，完成4个实验；

实验	学时	提交	实验题目
实验一	2学时		词法分析
实验二	8学时	提交实验一代码	自底向上的语法分析 (LR(1))
实验三	4学时	提交实验二代码	典型语句的语义分析及中间代码生成
实验四	2学时	提交实验三代码	目标代码生成

备注：本学期编译原理四次实验，只需完成所有实验任务后提交一份完整实验报告。

编译原理实验安排



✓ 实验提交内容：

1. 电子版代码

电子版内容：工程文件

压缩文件命名：学号-姓名-实验x.zip

实验报告命名：学号-姓名-编译原理实验报告.pdf

说明：增量式提交代码

✓ 实验总分数：20分。实验报告占30%，代码占70%

实验一	词法分析	12.5分(百分制)
实验二	自底向上的语法分析 (LR(1))	50分(百分制)
实验三	典型语句的语义分析及中间代码生成	25分(百分制)
实验四	代码生成	12.5分(百分制)

实验提交



实验提交网址: <http://grader.tery.top:8001/>

 登录

用户名

 180110103

密码

 密码同用户名 

登录

实验一

0份

文件作业

已截止

开始时间

2022-09-13 00:00

截止时间

2022-09-27 00:00

答案公布时间:

2023-01-31 00:00

查看咨询

查看统计

查看提交



✓ 具体每次实验提交截止时间请参考作业系统上截止时间



编译原理

实验一：词法分析

规格严格，功夫到家

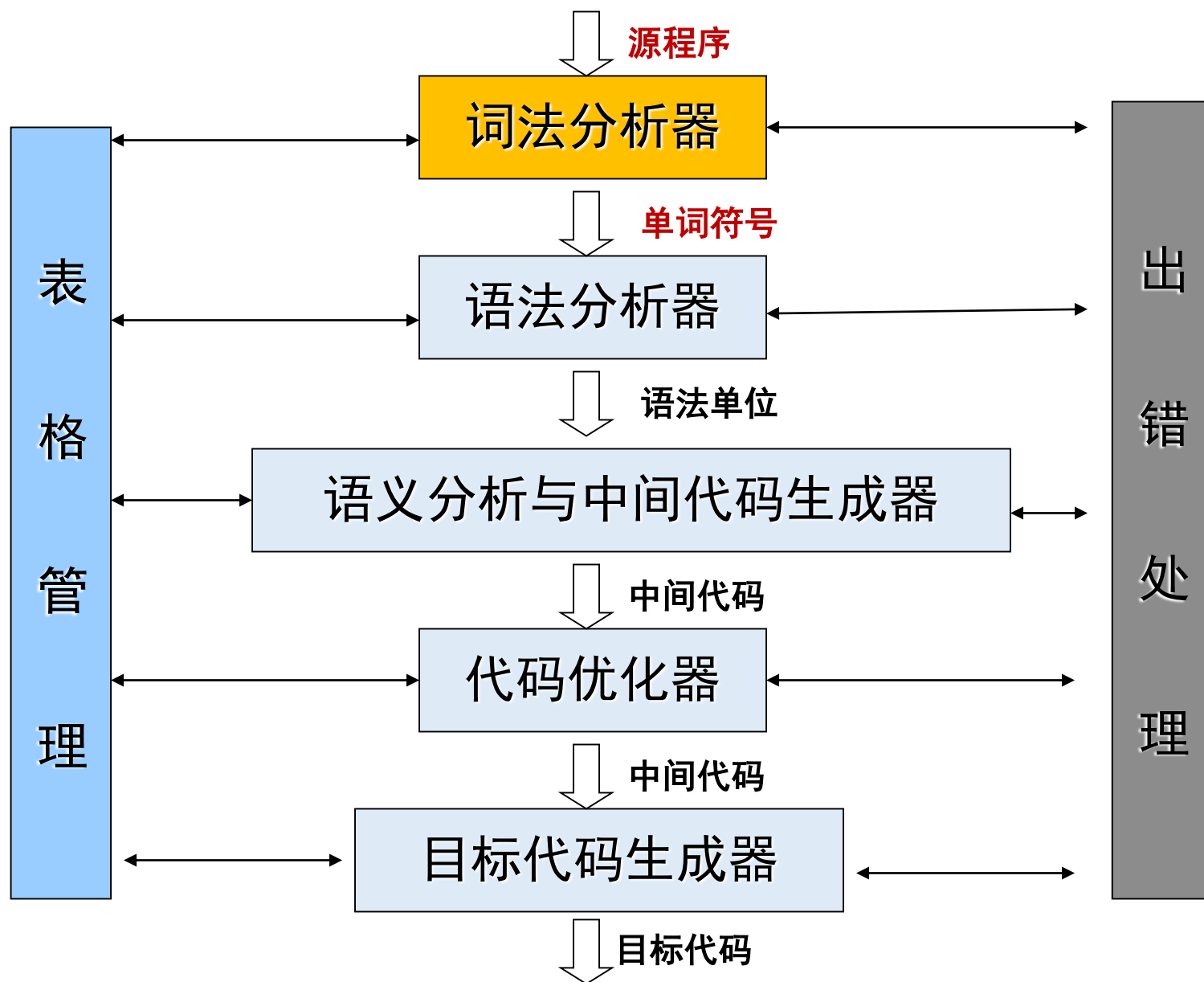
编译程序的总体结构

✓实验一:

✓实验二:

✓实验三:

✓实验四:



实验目的



1. 加深对词法分析程序的功能及实现方法的理解。
2. 对类C语言单词符号的文法描述有更深入的认识，理解有限自动机、编码表和符号表在编译的整个过程中的应用。
3. 设计并编程实现一个词法分析程序，对类C语言源程序段进行词法分析，加深对高级语言的认识。

备注：类C语言指C语言子集或者自定义的其他类似C语言语法的编程语言；

实验学时数：2学时

实验内容

编写一个词法分析程序，读取文件，对文件内的类C语言程序段进行词法分析。

1. **输入**：以文件形式存放的**类C语言程序段**；

```
data/in
├─ coding_map.csv      # 码点文件
└─ input_code.txt      # 输入的代码
```

2. **输出**：以文件形式存放的**TOKEN串**和简单**符号表**；

```
data/out
├─ old_symbol_table.txt # 符号表
└─ token.txt           # 词法单元列表
```

词法分析器的设计分析



1. 词法分析器的功能

对源程序进行编译预处理（去除注释、无用回车换行等）之后，把源程序分析成一个个单词。

由此可知，词法分析器的输入是源语言字符流，输出是单词序列。

2. 编码表

编译器为了处理方便，按照一定的方式对单词进行分类和编码，所以需要定义一个编码表。

3. 词法分析器如何识单词

通过正则文法来表述单词。

4. 正则文法到编写程序

中间桥梁是有限自动机DFA。

实验步骤

1. 定义**编码表**；（框架已提供）
2. 创建属于自己的**类C语言单词符号的文法**；
3. 根据所建文法画出**有限自动机的状态转换图**；
4. 定义数据结构：**TOKEN串**（已提供），**符号表**；
5. 根据有限自动机的状态转移**编写代码**；
6. 输出**Token串和符号表**到指定文件中；
7. 完成实验报告的词法分析相关内容（报告模板稍后发）。

程序语言单词的种类

- (1) **标识符**：由用户定义，表示各种名字；
- (2) **关键字**：也称基本字，do、while、int、char、sizeof...
- (3) **常数**：整常数、实常数、字符常量、字符串常量、符号常量；
- (4) **运算符**：算术运算符+、-、*、/等；
逻辑运算符not、or与and等；
关系运算符=、<>、>=、<=、>和<等；
- (5) **分界符**：，、；、（、）...

定义编码表

单词名称	类别编码	单词值
int	1	-
return	2	-
=	3	-
,	4	-
Semicolon	5	-
+	6	-
-	7	-
*	8	-
/	9	-
(10	-
)	11	-
id	51	内部字符串
IntConst	52	整数值
.....
布尔常数	80	0 或 1
字符串常数	81	内部字符串

注意：

- 关键字、运算符、分界符一符一码
- 标识符一个编码
- 常量一类一码

文法设计

➤ 正则文法表示

$G=(V,T,P,S)$, 其中 $V=\{S,A,B,C,\text{digit},\text{no_0_digit},\text{char}\}$, $T=\{\text{任意符号}\}$, P 定义如下

约定: 用 digit 表示数字: $0,1,2,\dots,9$; no_0_digit 表示数字: $1,2,\dots,9$;

用 letter 表示字母: $A,B,\dots,Z,a,b,\dots,z,_$

标识符: $S \rightarrow \text{letter } A \quad A \rightarrow \text{letter } A | \text{digit } A | \varepsilon$

整常数: $S \rightarrow \text{no_0_digit } B \quad B \rightarrow \text{digit } B | \varepsilon$

运算符: $S \rightarrow C \quad C \rightarrow = | * | + | - | /$

➤ 正则表达式表示法

标识符: $\text{id} \rightarrow \text{letter} (\text{letter} | \text{digit})^*$

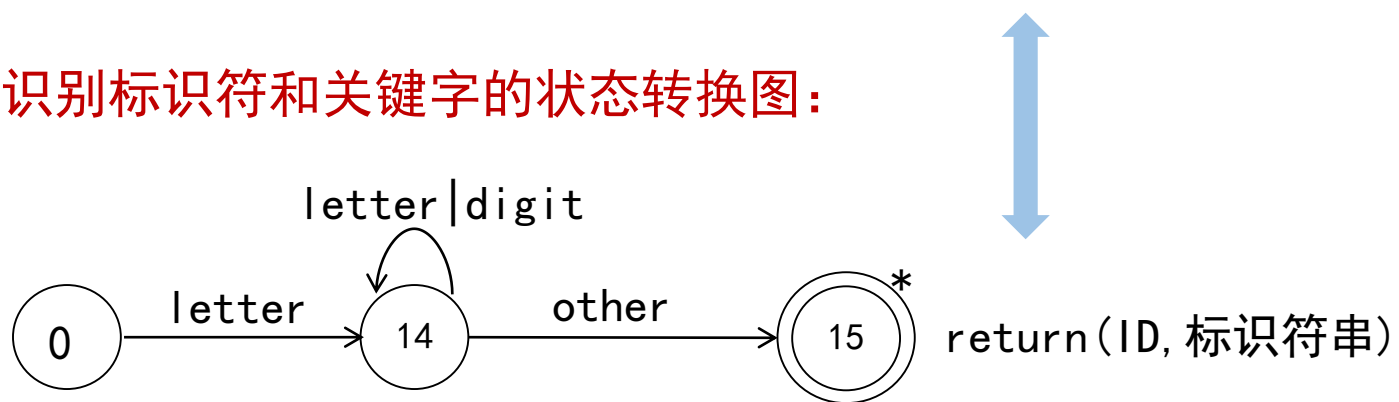
整常数: $\text{id} \rightarrow \text{no_0_digit} (\text{digit})^*$

有限自动机

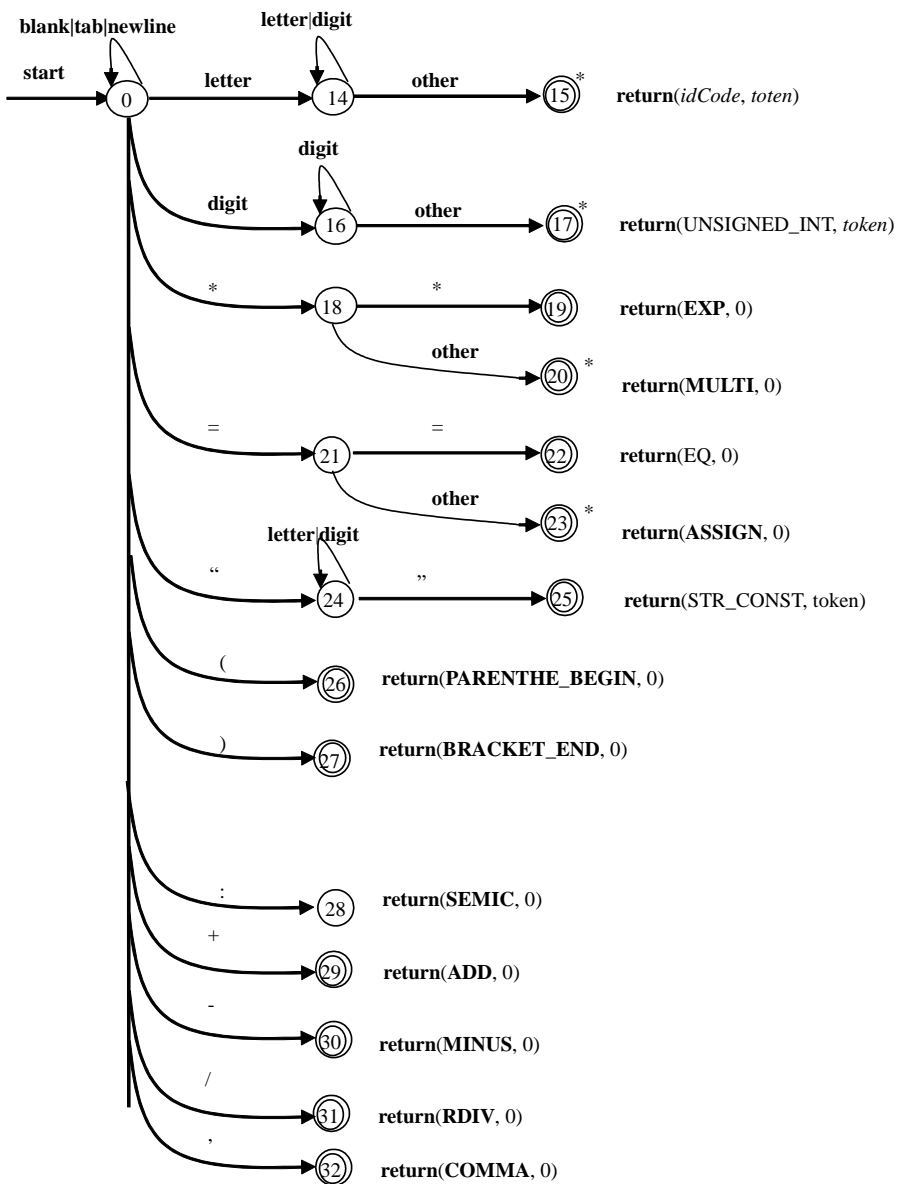
有限状态自动机和正则文法等价，考虑到状态转换图的直观性，我们从状态转换图出发来考虑词法分析器的设计。

标识符正则表达式： $\text{id} \rightarrow \text{letter} (\text{letter} | \text{digit})^*$

识别标识符和关键字的状态转换图：



各类状态转换图合并



词法分析器输出——TOKEN串

举例

输入源代码：

```
int result;  
int a;  
int b;  
int c;  
a = 8;  
b = 5;  
c = 3 - a;  
result = a * b - ( 3 + b )  
* ( c - a );  
return result;
```

词法分析输出token:



(int,)	(-,)
(id, result)	(id, a)
(Semicolon,)	(Semicolon,)
(int,)	(id, result)
(id, a)	(=,)
(Semicolon,)	(id, a)
(int,)	(*,)
(id, b)	(id, b)
(Semicolon,)	(-,)
(int,)	((,)
(id, c)	(IntConst, 3)
(Semicolon,)	(+,)
(id, a)	(id, b)
(=,)	(,)
(IntConst, 8)	(*,)
(Semicolon,)	((,)
(id, b)	(id, c)
(=,)	(-,)
(IntConst, 5)	(id, a)
(Semicolon,)	(,)
(id, c)	(Semicolon,)
(=,)	(return,)
(IntConst, 3)	(id, result)
	(Semicolon,)

表示单词的种类，可用整数
编码或记忆符表示

种别码

属性值

不同的单词不同的值

词法分析器的输出——符号表

符号表：以名字为关键字来记录其信息的**数据结构**；

支持的基本操作包括**插入**、**查找**和**删除**，本实验实现前两种即可。

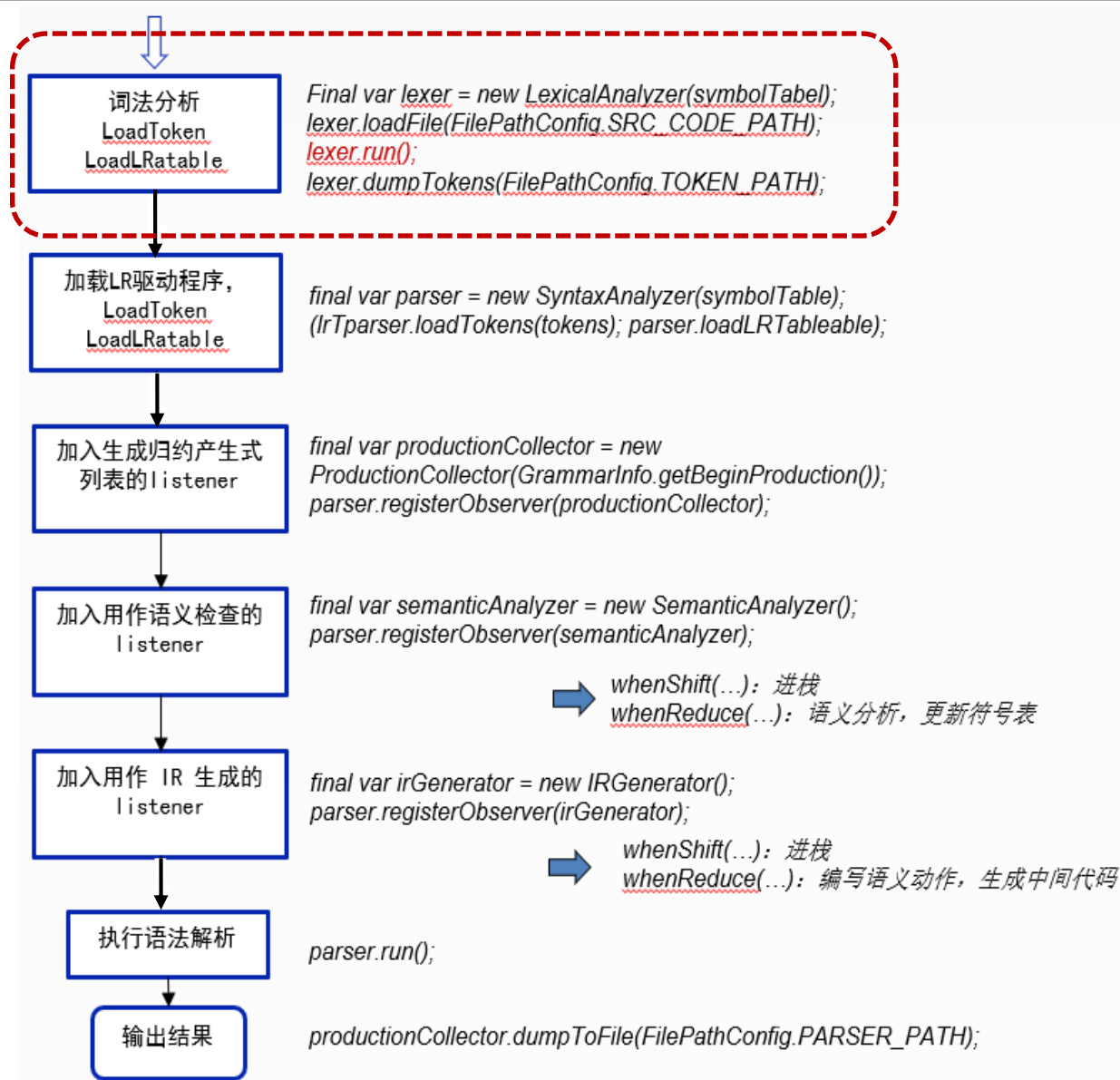
符号表的数据结构类型：线性表（优点：简单直观，缺点：插入时间复杂度高）

散列表（查找、插入效率高）

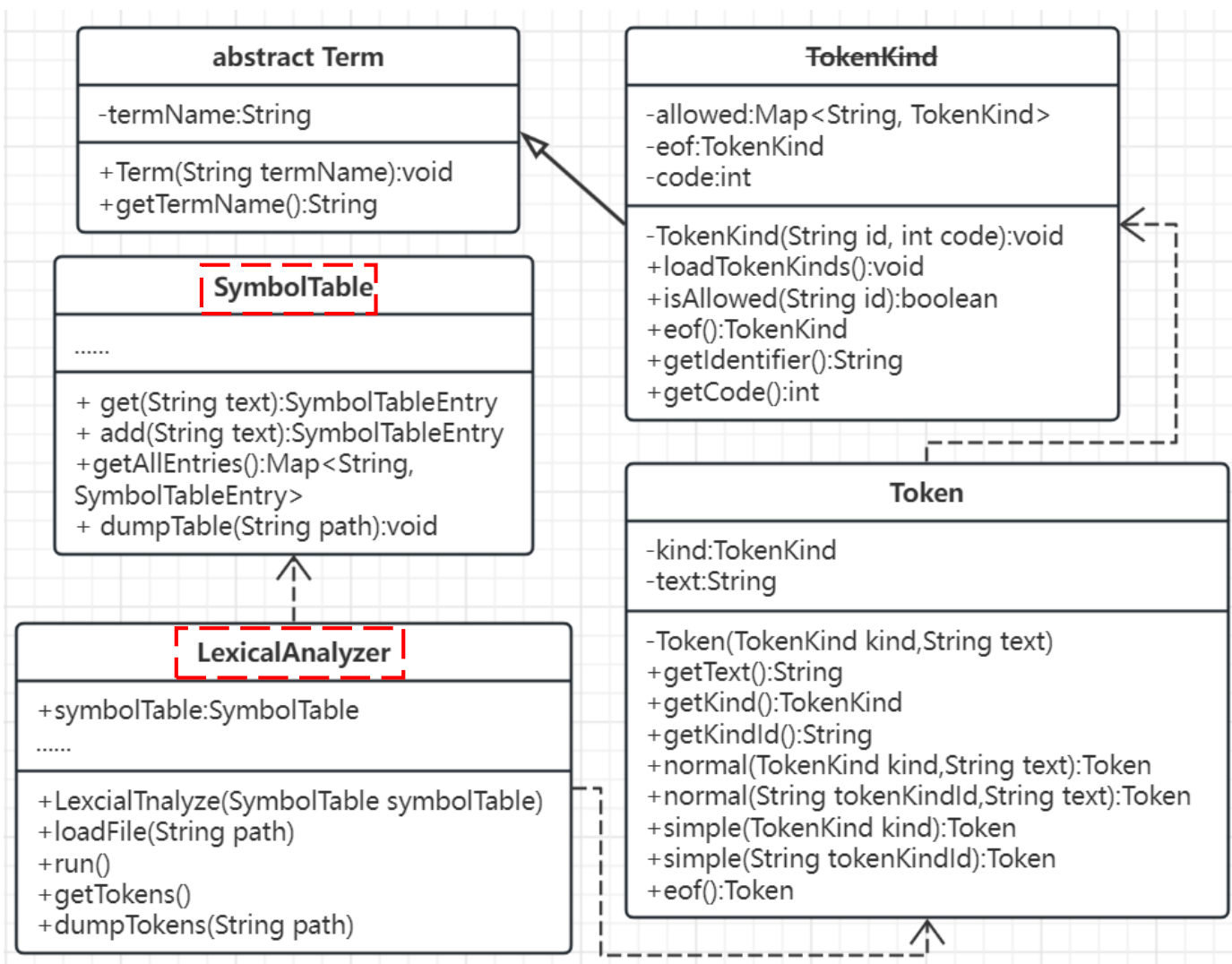
	名字	属性
符号表表项1	abc	...
符号表表项2	i	...
⋮
符号表表项 n

图：符号表

框架代码主程序流程



词法分析程序类图



同学们，
独立开始实验