

编译原理课程报告

162210107蔡蕾



2024-11-18

南京航空航天大学

计算机科学与技术学院

目录

[总任务 2](#_Toc180391304)

[上机练习1 2](#_Toc180391305)

[上机练习2 2](#_Toc180391306)

[系统设计 2](#_Toc180391307)

[系统结构 2](#_Toc180391308)

[开始前准备 2](#_Toc180391309)

[词法分析器 2](#_Toc180391310)

[引言 2](#_Toc180391311)

[词法分析器的基本原理 2](#_Toc180391312)

[词法分析器模块设计 3](#_Toc180391313)

# 总任务

## 上机练习1

**词法分析**

**一个PASCAL语言子集（PL/0）词法分析器的设计与实现**

**PL/0语言的BNF描述（扩充的巴克斯范式表示法）**

**<prog> → program <id>；<block>**

**<block> → [<condecl>][<vardecl>][<proc>]<body>**

**<condecl> → const <const>{,<const>};**

**<const> → <id>:=<integer>**

**<vardecl> → var <id>{,<id>};**

**<proc> → procedure <id>（[<id>{,<id>}]）;<block>{;<proc>}**

**<body> → begin <statement>{;<statement>}end**

**<statement> → <id> := <exp>**

**|if <lexp> then <statement>[else <statement>]**

**|while <lexp> do <statement>**

**|call <id>（[<exp>{,<exp>}]）**

**|<body>**

**|read (<id>{，<id>})**

**|write (<exp>{,<exp>})**

**<lexp> → <exp> <lop> <exp>|odd <exp>**

**<exp> → [+|-]<term>{<aop><term>}**

**<term> → <factor>{<mop><factor>}**

**<factor>→<id>|<integer>|(<exp>)**

**<lop> → =|<>|<|<=|>|>=**

**<aop> → +|-**

**<mop> → \*|/**

**<id> → *l*{*l*|d} （注：*l*表示字母）**

**<integer> → d{d}**

**注释：**

**<prog>：程序 ；<block>：块、程序体 ；<condecl>：常量说明 ；<const>：常量；**

**<vardecl>：变量说明 ；<proc>：分程序 ； <body>：复合语句 ；<statement>：语句；**

**<exp>：表达式 ；<lexp>：条件 ；<term>：项 ； <factor>：因子 ；<aop>：加法运算符；**

**<mop>：乘法运算符； <lop>：关系运算符**

**odd：判断表达式的奇偶性。**

**要求：**

1. **按照P45的算法思想，使用循环分支方法实现PL/0语言的词法分析器，该词法分析器能够读入使用PL/0语言书写的源程序，输出单词符号串及其属性到一中间文件中，具有一定的错误处理能力，给出词法错误提示（需要输出错误所在的行列）。**

## 上机练习2

**语法分析**

根据上机练习一给出的PL/0语言扩充的巴克斯范式语法描述，利用递归下降的语法分析方法，编写PL/0语言的语法分析程序。

要求：

1. 对给出的PL/0语言进行分析，证明其可以进行自上而下的语法分析；
2. 对block、proc、statement、condition、expression、term、factor进行分析，画出语法分析图，在此基础上描述这些子程序的设计思想；
3. 具有一定的语法错误处理能力；

# 系统设计

## 系统结构

在编译器的整体架构中，它可以分为前端（Front End）和后端（Back End）两个主要部分。每一部分又包含不同的处理阶段，依次完成从源代码到目标代码的翻译和优化。下图展示了编译器的流程及其各个组件的相互作用。

**前端：**编译器的前端主要负责分析源代码，确保其结构和语义正确，并将其转换为一种中间表示（Intermediate Representation）。前端的几个主要阶段包括：

**词法分析器（Lexical Analyzer）：** 词法分析器将源代码逐字符扫描，将其分解为最基本的构成单元，即词法单元（token）。这些 token 包含诸如关键字、标识符、常量、操作符等。经过词法分析后，源代码被转换为一个记号流（Token Stream），为后续的语法分析器提供输入。

**语法分析器（Syntax Analyzer）：** 语法分析器负责根据语言的语法规则，分析记号流，构建语法树（Syntax Tree）。这一步骤的目标是确保代码符合语言的语法规范，例如括号是否匹配、语句结构是否正确等。语法树是源代码的结构化表示，反映了程序的嵌套关系和控制结构。

**语义分析器（Semantic Analyzer）：** 语义分析器在语法树的基础上进行语义检查，确保程序的逻辑符合语言的语义规则。它检查变量的定义和使用是否一致、类型是否正确等问题。如果在这一阶段检测到语义错误，编译器将报告相应的错误。语义分析器输出的结果是一个带注解的语法树（Annotated Syntax Tree），为后续代码生成提供更多的上下文信息。

**中间代码生成器（Intermediate Code Generator）：** 这一阶段将带注解的语法树转换为中间代码表示（Intermediate Representation, IR）。中间代码是一种抽象的代码表示形式，独立于具体的机器结构。它提供了一种通用的方式来表示程序的逻辑，方便后续的优化和目标代码生成。

**后端：**编译器的后端主要负责优化中间代码并生成目标机器代码。后端的各个阶段包括：

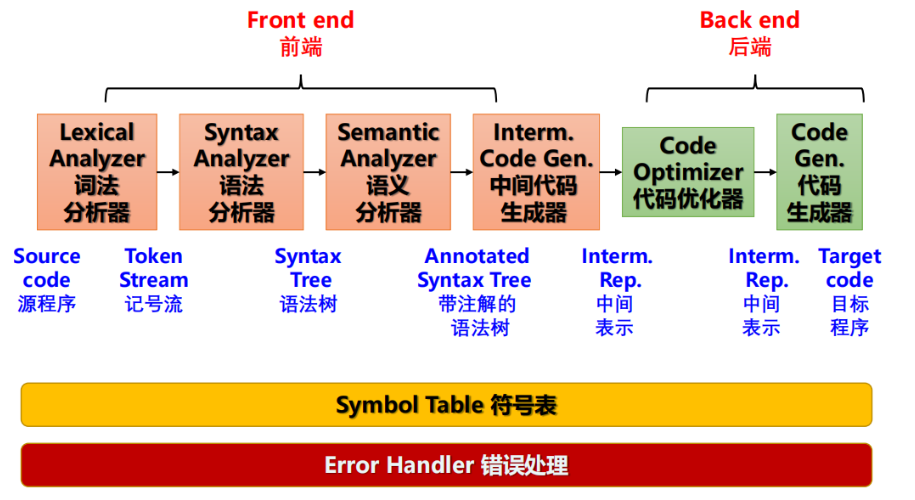
**代码优化器（Code Optimizer）：** 在这个阶段，编译器对中间代码进行优化，旨在提高程序的执行效率或减少资源消耗。例如，它可以通过消除冗余代码、减少内存使用或优化循环等方式改进程序的性能。优化后的中间代码将进一步减少代码的执行时间或空间占用。

**目标代码生成器（Code Generator）：** 最后一步，编译器将优化后的中间代码转换为特定平台的目标代码（Target Code），通常是二进制形式的机器代码或汇编代码。这一阶段的目标是生成可以在特定硬件上执行的程序。

**符号表与错误处理：**贯穿整个编译过程的是两个重要的支持模块：符号表（Symbol Table）和错误处理器（Error Handler）。

**符号表：**符号表在编译的各个阶段中维护着程序中变量、函数、常量等的符号信息。它记录每个符号的类型、作用域、内存地址等详细信息。在词法分析、语法分析、语义分析等阶段，符号表都起到了至关重要的作用，特别是在语义分析阶段，用于检查符号的正确性。

**错误处理器：**错误处理模块则负责在编译过程中发现并报告源代码中的错误。词法错误、语法错误、语义错误都会在相应的阶段被发现并报告给开发者。错误处理器不仅仅是在遇到问题时中止编译，它还会尽量恢复正常的编译过程，以发现更多的潜在问题。



## 初始化

## 词法分析器

### 引言

编译器的设计是计算机科学中的一个重要课题，其中词法分析是编译过程的第一步。词法分析器的主要任务是将源代码转换为一系列的词法单元（tokens），这些词法单元是编译器进行语法分析的基础。词法分析器的设计不仅影响到编译器的整体性能和准确性，还直接关系到编程语言的可解析性和灵活性。

在现代编程语言中，源代码通常由字符组成，这些字符包括字母、数字、操作符、标点符号等。词法分析器需要根据这些字符的组合规则，识别出有意义的成分，如关键字（如 if、while）、标识符（如变量名）、常量（如数字、字符串）和运算符（如 +、-）。此外，词法分析器还需要能够处理空白字符和注释，这些内容在语法分析中是不需要考虑的。

为了实现这一功能，词法分析器通常采用有限状态机（Finite State Machine, FSM）模型。状态机通过不同的状态和状态之间的转换，能够有效地识别出不同类型的词法单元。设计一个高效且准确的词法分析器，要求我们深入理解源语言的语法规则，以及如何利用状态机的机制来实现字符的分类和词法单元的生成。

本报告将详细介绍我们设计的词法分析器模块，重点讨论其实现思路、数据结构、主要函数（如 GetWord）的工作机制，以及如何通过状态机模型来进行词法分析的具体过程。通过对这些内容的分析，我们希望能够展示出词法分析器在编译器中的关键作用，并提供一个有效的解决方案，以支持后续的语法分析和代码生成。

### 词法分析器的基本原理

#### 词法分析器的作用

词法分析器是编译器的重要组成部分，负责将源代码中的字符流转换为一系列的词法单元（tokens），这些 tokens 为后续的语法分析提供基础。它通过逐字符扫描源代码，识别关键字、标识符、常量、运算符和界符，并使用有限状态机（FSM）模型进行有效识别。

词法分析器的主要任务是将字符流划分为有效的词法单元。标识符由字母开头，可以包含字母和数字，而关键字是语言的保留字。它还识别数字常量、运算符和界符，并将这些符号转换为相应的 tokens。在分析过程中，词法分析器跳过空白字符和注释，以专注于提取有效代码成分。

通过状态机模型，词法分析器能够根据当前字符和前一个字符的状态进行转换。当遇到字母时，状态机会进入标识符或关键字状态；遇到数字时，进入数字识别状态；遇到运算符或界符时，直接生成相应的 token。这样设计提高了分析器的灵活性和效率。

此外，词法分析器需处理错误情况，例如非法字符或不完整的词法单元。遇到未识别字符时，分析器应给出错误提示并尽量继续分析后续代码。

图 2：词法分析器的工作原理

源代码 (字符流)

|

v

[词法分析器] --> [跳过空白符号和注释]

|

v

[识别标识符、关键字、运算符等]

|

v

[生成 Token 序列] --> 语法分析器

#### 状态机模型

词法分析器通常采用有限状态自动机（Finite State Machine, FSM）来实现。状态机的每个状态对应一种字符或字符序列的识别状态，状态转换根据输入字符的类型（字母、数字、操作符等）来进行。解释自己画的图

图 3 词法分析器状态机示意图

### 词法分析器模块设计

#### Lexer类

两个表格（一个成员变量，一个成员函数，一张代码截图）

#### GetWord 函数实现

（画21页的流程图+解释）