

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 1](#_Toc180162382)

[上机练习1 1](#_Toc180162383)

[上机练习2 2](#_Toc180162384)

[系统设计 2](#_Toc180162385)

[系统结构 2](#_Toc180162386)

# 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**

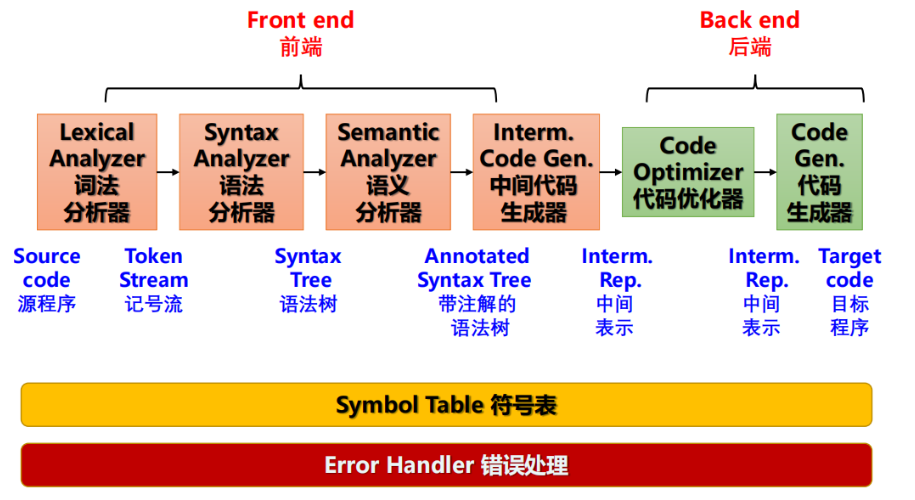
根据上机练习一给出的PL/0语言扩充的巴克斯范式语法描述，利用递归下降的语法分析方法，编写PL/0语言的语法分析程序。

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



编译器的系统结构通常分为前端和后端2个主要部分。前端负责将源代码解析成中间表示，包括词法分析、语法分析和语义分析。此部分确保源代码的正确性并生成抽象语法树。接下来，后端会对中间表示进行各种优化，以提高代码效率。并将优化后的中间表示转换为目标代码，负责代码生成和机器指令的安排。整体上，这一结构帮助编译器高效地将高级语言转换为低级语言，实现程序的可执行性。

## 开始前准备

## 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。

词法分析器在编译器中起到关键作用，为语法和语义分析提供必要的信息。它不仅高效地生成 tokens，还能管理标识符信息，将其记录在符号表中，为后续分析提供支持。

总之，词法分析器通过状态机模型逐字符扫描源代码，识别各种语言成分，生成词法单元供后续编译步骤使用，同时处理无关字符和错误，确保编译器的高效性和准确性。

图 2：词法分析器的工作原理

scss

复制代码

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。

图 1: 词法分析器状态机示意图

lua

复制代码

+---------------+ +-------------+ +------------+

| 起始状态 (S0) | ---> | 关键字 (S1) | ---> | 结束状态 |

+---------------+ +-------------+ +------------+

| |

v v

+-----------+ +----------+

| 标识符 (S2) | | 数字 (S3) |

+-----------+ +----------+

S0（起始状态）：开始扫描字符，根据字符的类型决定下一步的状态。

S1（关键字）：识别关键字或保留字。

S2（标识符）：识别标识符，通常为字母开头。

S3（数字）：识别数字，包括整数和浮点数。

### 词法分析器模块设计

#### 数据结构

词法分析器使用一系列的数据结构来存储字符和词法单元，包括：

符号表（symbol table）：存储标识符及其相关信息。

关键字表：存储保留字，如 if、while 等。

操作符表：存储常见的运算符，如 +、-、\*、/ 等。

cpp

复制代码

unordered\_map<unsigned long, wstring> sym\_map; // 保留字编号与字符串的映射

wstring resv\_table[RSV\_WORD\_MAX] = {

L"odd", L"begin", L"end", L"if", L"then", L"while", L"do", L"call",

L"const", L"var", L"procedure", L"write", L"read", L"program", L"else"

};

#### GetWord 函数的实现

GetWord 是词法分析器的核心函数，它的作用是从源代码中获取一个词法单元。函数的流程大致如下：

跳过空白和注释：先跳过空格、制表符、换行符和注释，进入实际的代码分析。

读取字符：读取下一个字符，根据该字符的类型判断是关键字、标识符、数字还是操作符。

状态转换：根据字符类型转换到相应的状态，并逐步构建出完整的 token。

保存 token：当识别出一个完整的词法单元后，将其保存，并返回给语法分析器使用。

cpp

复制代码

void Lexer::GetWord() {

strToken = L""; // 初始化词法单元

GetBC(); // 跳过空白字符

if (IsLetter()) { // 如果是字母，可能是关键字或标识符

while (IsLetter() || IsDigit()) {

Concat(); // 拼接字符到 strToken

GetChar(); // 读取下一个字符

}

Retract(); // 回退一个字符，因为已经多读了一个

tokenType = Reserve(); // 判断是否为关键字

if (tokenType == 0) { // 不是关键字，则为标识符

tokenType = IDENT;

}

}

else if (IsDigit()) { // 如果是数字

while (IsDigit()) {

Concat(); // 拼接字符到 strToken

GetChar(); // 读取下一个字符

}

Retract(); // 回退一个字符

tokenType = NUMBER; // 标记为数字类型

}

else if (int opr = IsOperator()) { // 如果是运算符

tokenType = opr; // 标记为运算符类型

Concat(); // 将运算符拼接到 strToken

}

else if (ch == L';') { // 识别分号

tokenType = SEMICOLON;

}

else {

tokenType = NUL; // 其他字符暂时不处理

}

}

### 状态机设计与 GetWord 函数的关系

#### 词法分析状态

GetWord 函数通过状态机模型实现对字符流的分析。根据当前字符的类型（字母、数字、操作符等），GetWord 函数会进入不同的状态，并不断读取下一个字符，直到识别出完整的词法单元。

图 2: GetWord 状态转换流程图

lua

复制代码

+-------------------+ +------------------+

| GetWord() 函数开始 | ------------> | 跳过空格 (GetBC) |

+-------------------+ +------------------+

| |

v v

+---------------------+ +--------------------+

| IsLetter() 判断字母 | | IsDigit() 判断数字 |

+---------------------+ +--------------------+

| |

v v

+-------------------------+ +-------------------+

| 进入关键字或标识符状态 | | 进入数字识别状态 |

+-------------------------+ +-------------------+

| |

v v

+---------------------------+ +-----------------------+

| Reserve() 检查关键字或标识符 | | 识别为数字 token |

+---------------------------+ +-----------------------+

跳过空格（GetBC）：GetWord 函数首先调用 GetBC 跳过源代码中的空格、换行符和注释，确保从有效字符开始分析。

识别标识符或关键字（IsLetter）：如果遇到字母，进入标识符或关键字识别状态，调用 Concat 拼接字符，最终通过 Reserve 判断是否为保留字。

识别数字（IsDigit）：如果遇到数字，进入数字识别状态，同样通过 Concat 拼接数字字符，最终标记为数字类型 token。

识别运算符（IsOperator）：如果遇到操作符，则直接将操作符作为 token 类型返回。

### 错误处理与优化

#### 错误处理

非法字符：词法分析器必须能够识别非法字符，并进行相应处理。在 GetWord 中，如果遇到未定义的字符，设置 tokenType 为 NUL 以表示错误。

#### 性能优化

提前终止：一旦识别出有效的词法单元，词法分析器会立即返回，无需继续读取剩余字符，从而提高性能。

缓存字符：在读取字符时，使用缓冲区存储已读取的字符，减少 I/O 操作，提高分析速度。