## **1 概述**

通过某种高级语言（如C/C++，Java）实现词法，语法分析器的功能。

## **2 实验目标**

1.理解并掌握词法，语法分析的原理与方法。

2.能够使用某种语言实现词法，语法分析程序。

3.对编译的基本概念，原理和方法有完整和清楚的理解，并能正确而熟练的运用。

## **3 实验描述**

### **3.1 实验要求**

采用词法分析程序作为语法分析程序子程序的方法，实现词法、语法分析。

具体要求如下：

词法分析程序作为语法分析程序的子程序。

输入数据：程序段。

输出结果：语法分析结果，包括错误列表。

### **3.2 本实验概述**

本次实验采用C++编写词法语法分析器，在此基础上，采用递归下降   
分析程序，实现对词法分析程序所提供的单词序列的语法检查和结构分   
析。PL/0语言功能简单，结构清晰，可读性强，而又具备了一般的高级   
程序设计语言的必须部分，所以采用PL/0语言的编译程序能充分体现一   
个高级语言编译程序实现的基本方法和技术，故采用PL/0语言进行词法 语法分析。

## **4 技术分析**

## **4.1 词法语法分析**

本次实验主要用到的是基本的数据结构分析，C++编程技术，编译原理等，   
其中对编译的基本概念抓住词法分析与语法分析两个部分来理解：

* 词法分析

词法分析器根据词法规则识别出源程序中的各个记号，每个记号代表一   
类单词。源程序中常见的记号可以归为几大类：关键字、标识符、字面量和   
特殊符号。词法分析器的输入是源程序，输出是识别的记号流。词法分析器   
的任务是把源文件的字符流转换成记号流。它是编译过程的第一个阶段。其   
主要任务是从左到右依次描描字符串形式的源程序的各个字符，逐个识别出   
其中的单词，并将其转换成为内部编码形式的单词符号串输出，用于进行语 法分析。

概括的说，语法器在其工作过程中，一般应完成下列的任务：

（1）识别出源程序中的各个单词符号，并将其转换成内部编码形式；

（2）删除无用的空白字符、回车字符以及其他非实质性字符；

（3）删除注释；

（4）进行词法检查，报告所发现的错误。

此外，视编译工作流程的组织，一些编译程序在进行词法分析时，还要   
完成将所识别出的标志符登录到符号表的工作。

* 语法分析：

语法分析是编译过程的核心，分析的任务是根据语法规则分析源程序的   
语法结构，并在分析过程中，对源程序进行语法检查，如果语法没有错误，   
则给出正确的语法结构，为语义分析和代码生成做准备。

目前语法分析方法有多种多样，大致分为自顶而下和自底而上两大类。   
自顶而下又分为LL（1）分析方法和递归下降分析方法。自底而上又分为简   
单优先文法、算符优先文法、LR（K）分析方法。下面主要介绍自底而上的   
LR（K）分析方法。

自底向上分析法，也称移进-归约分析法。它的实现思想是对输入符号串   
自左向右进行扫描，并将输入符逐个移入一个后进先出栈中，边移入边分析，   
一旦栈顶符号串形成某个句型的句柄时，(该句柄对应某产生式的右部)，就   
用该产生式的左部非终结符代替相应右部的文法符号串，这称为移步归约。   
重复这一过程直到归约到栈中只剩文法的开始符号时则为分析成功，也就确   
认输入串是文法的句子。否则，分析失败，表示输入符号串不是文法的一个   
句子，其中必定存在语法错误。

根据以上编译的相关知识结合C++编程语言和部分数据结构的知识来写 词法语法分析器。

## **5 设计与实现**

### **5.1 设计思路**

编制一个递归下降分析程序，实现对词法分析程序所提供的单词序列的   
语法检查和结构分析。利用C++编写递归下降分析程序，并对PL/0语言进行   
语法分析。核心思想就是，从开始状态开始，按照文法展开式，逐级进行状 态分析，直   
到分析完毕，如果在此期间出现状态不匹配，即语法错误，停止 分析。当然   
在实际的语法分析器要有错误恢复机制，以发现其他的语法错 误。即一次报   
告多个语法错误。此外要想实现语法分析，必须先有词法分 析，用词法分析   
的结果进行语法分析。

用扩充的BNF表示如下：

编制一个递归下降分析程序，实现对词法分析程序所提供的单词序列的   
语法检查和结构分析。

利用C++编写递归下降分析程序，并对PL/0语言进行语法分析。

\<程序>：：=begin\<语句串>end

\<语句串>：：=\<语句>{；\<语句>}

\<语句>：：=\<赋值语句>

\<赋值语句>：：=ID：=\<表达式>

\<表达式>：：=\<项>{+\<项> | -\<项>}

\<项>：：=\<因子>{\*\<因子> | /\<因子>

\<因子>：：=ID | NUM | （\<表达式>）

输入单词串，以“#”结束，如果是文法正确的句子，则输出成功信息，   
打印“语法分析成功！”，否则输出“语法分析错误（错误原因）”。

例如：

输入 begin a:=4; b:=2\*3; c:=a+b end #

输出 语法分析成功！

输入 x:=a+b\*c end #

输出 缺少begin！

| **单词符号** | **种别编码** |
| --- | --- |
| Begin | 1 |
| If | 2 |
| Then | 3 |
| While | 4 |
| Do | 5 |
| End | 6 |
| 标识符 | 10 |
| 数字 | 20 |
| + | 13 |
| - | 14 |
| \* | 15 |
| / | 16 |
| : | 17 |
| := | 18 |
| \< | 20 |
| \<> | 21 |
| \<= | 22 |
| > | 23 |
| >= | 24 |
| = | 25 |
| ; | 26 |
| ( | 27 |
| ) | 28 |
| # | 0 |

### **5.2 实现方法**

本实验采用C++编码，其中主要编写了以下几个函数及功能：

Void cifa() //词法分析

Void fun\_yufa() //判断语法是否有错误

Void fun\_op() //处理运算符(\\*和/)

Void exp() //处理运算符（+和-）

Void fun\_yuju() //判断是否有语句错误(:=)

Void fun\_end() //判断程序是否结束

Void yufa() //采用递归下降的语法分析