**《编译原理》实验二:**

**中间代码生成器设计与实现**

****

**学 号 1953072**

**姓 名 肖鹏飞**

**专 业 计算机科学与技术**

**授课老师 高秀芬**

一．实验要求

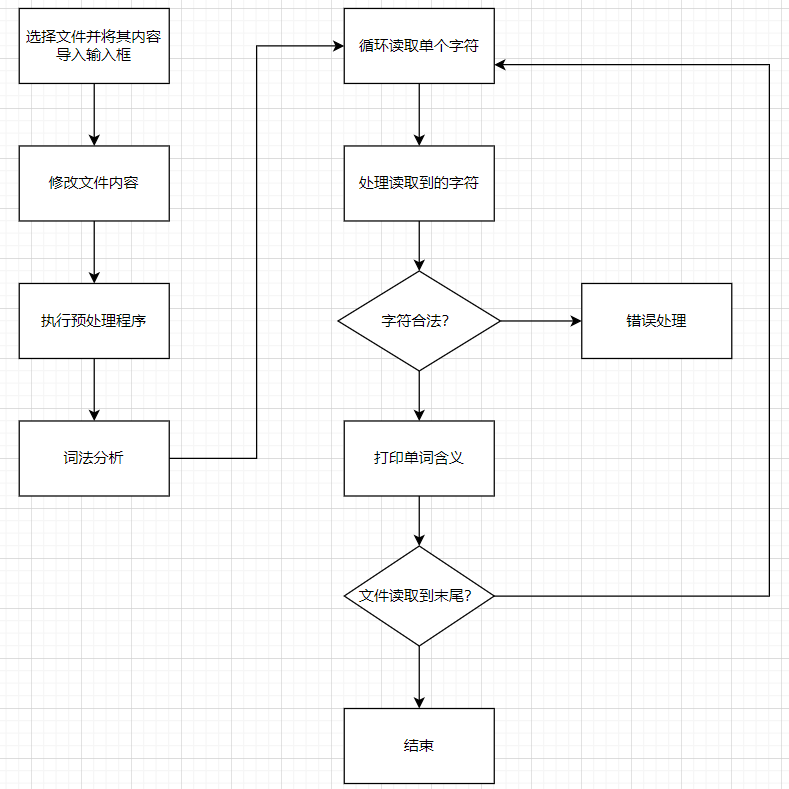
在前面实验的基础上（词法、语法分析），进行语义分析和中间代码生成器的设计，输入源程序，输出等价的中间代码序列（建议以四元式的形式作为中间代码）。

注意静态语义错误的诊断和处理。

在此基础上，考虑更为通行的高级语言的语义检查和中间代码生成所需要注意的内容，并给出解决方案。

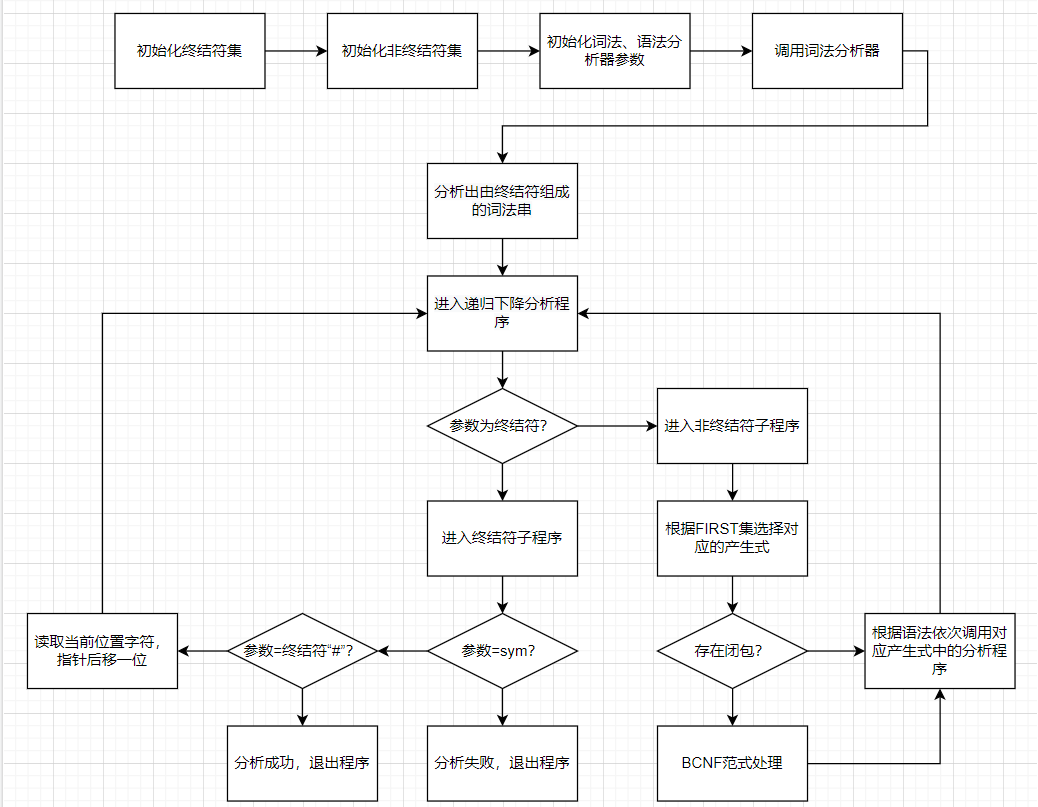
二．分析算法的主程序框图

词法分析器算法基本思想采用有限自动机（DFA）思想进行设计。



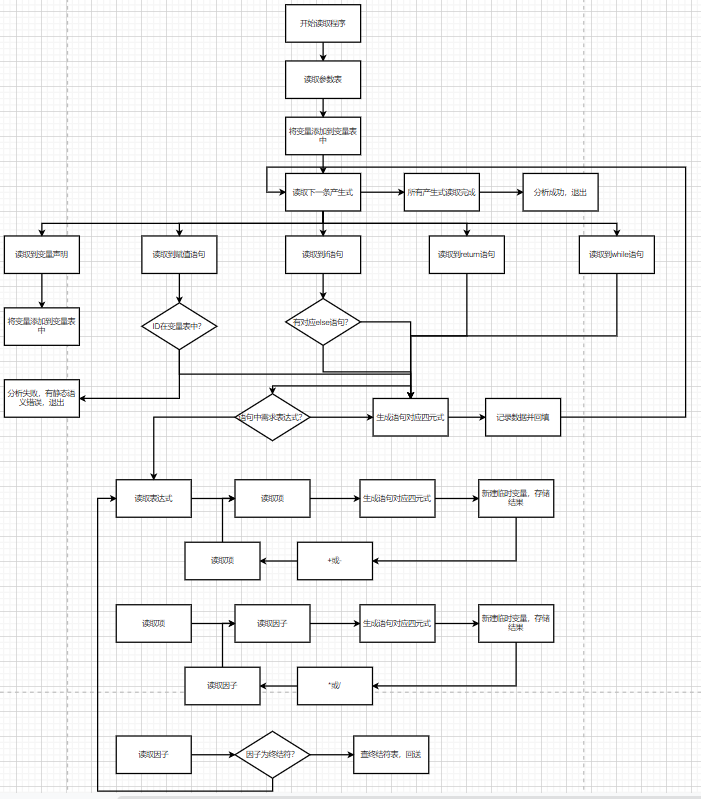
语法分析器采用自上而下语法分析的方法，通过使用LL(1)分析方法，构造递归下降分析程序。在语法分析的过程中调用词法分析，同时在构造程序的过程中应用了扩展的巴科斯范式（EBNF）解析语法。

完整的语法分析器程序框图如下：

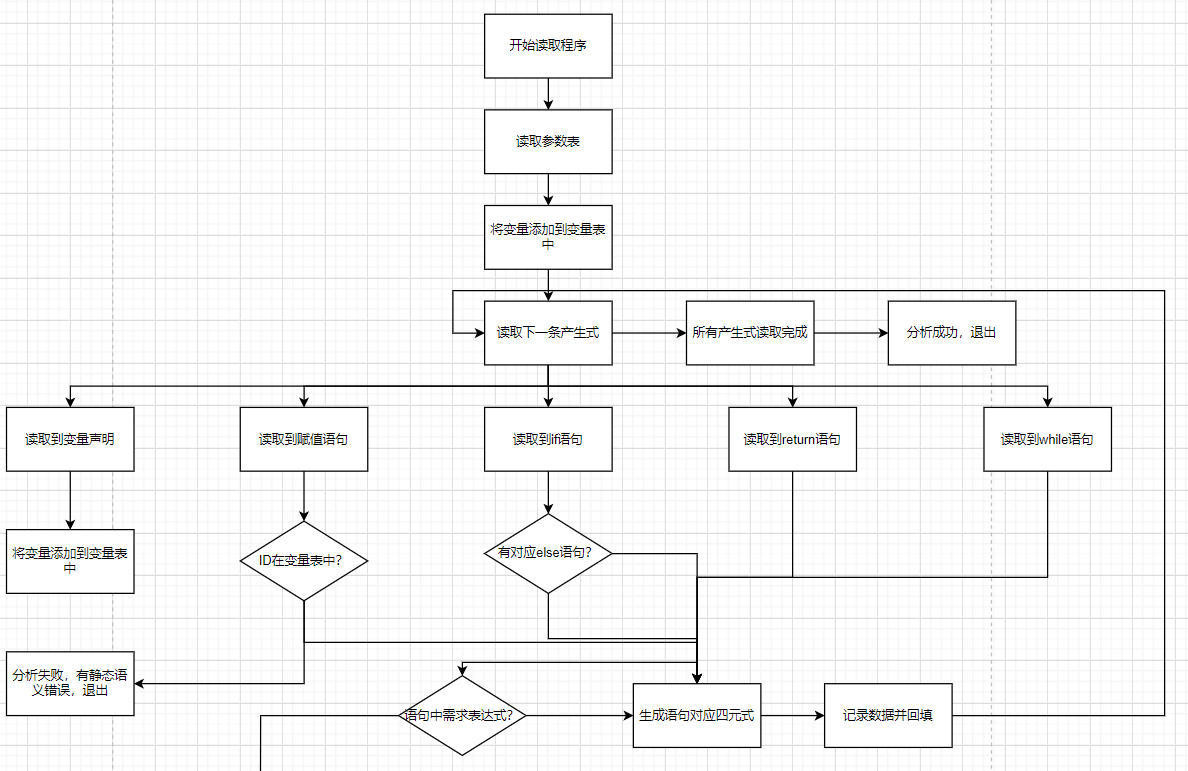


中间代码生成器是在词法和语法分析器的基础上构建而成的。通过在语法分析过程中对产生式进行语义分析，生成了四元式形式的中间代码。

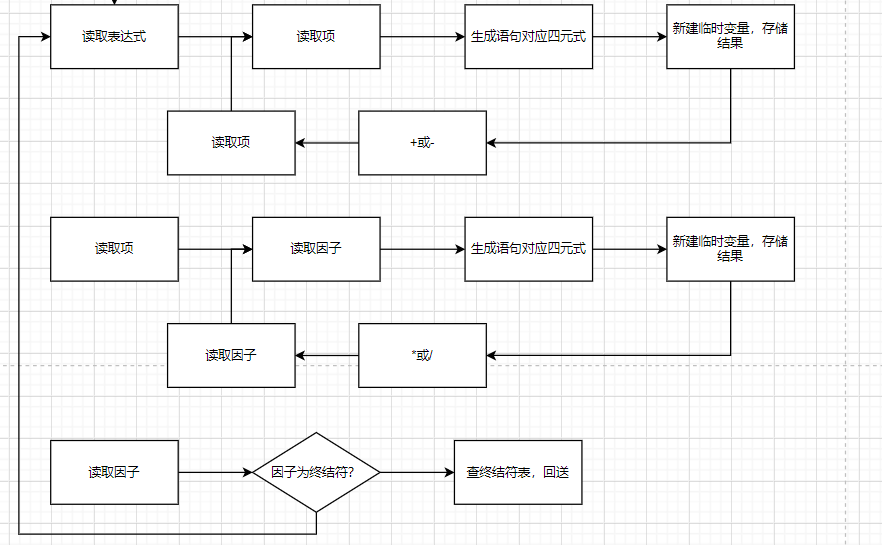
整体架构：



语句级别分析：



表达式级别分析：



三．运行和开发环境

开发平台：

操作系统：Microsoft Windows 10 1903 家庭版

编程语言：C++11

开发框架：Qt 5.13.1

IDE：Qt Creator 4.10.0(Community)

编译器：MinGW 7.3.0 32-bit for C++

软件的运行环境：在以上开发平台环境下能够直接运行，否则需要在可执行文件同一文件夹内添加动态链接库(.dll)用于支持软件运行。

四．能识别的单词、能分析的语法

（1）.能识别的单词（终结符）

关键字：int | void | if | else | while | return

标识符： 字母（字母|数字）\* （注：不与关键字相同）

数值：数字（数字）\*

赋值号： =

算符： + | - | \* | / | = | == | > | >= | < | <= | !=

取非：!

界符： ；

域符： ：

分隔符： ，

注释号： /\* \*/ | //

左括号： （

右括号： ）

左大括号： {

右大括号： }

字母：| a |....| z | A |....| Z |

数字：0| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

结束符：#

（2）.能识别的非终结符

"Program"

"类型"

"语句块"

"内部声明"

"内部变量声明"

"语句串"

"语句"

"赋值语句"

"return语句"

"while语句"

"if语句"

"表达式"

"布尔表达式"

"项"

"因子"

"参数表"

（3）.能分析的语法

Program ::= <类型> < ID>’(’<参数表>’)’<语句块>

<类型>::=int | void

<ID>::=字母(字母|数字)\*

<语句块> ::= ‘{’ <内部声明> <语句串>‘}’

<内部声明> ::= 空 | <内部变量声明>;{ <内部变量声明>;}

<内部变量声明>::=int <ID> （注： { }中的项表示可重复若干次）

<语句串> ::= <语句> { <语句> }

<语句> ::= <if语句> |< while语句> | <return语句> | <赋值语句>

<赋值语句> ::= <ID> =<表达式>;

<return语句> ::= return [ <表达式> ] （注：[ ]中的项表示可选）

<while语句> ::= while ‘(’ <布尔表达式> ‘)’ <语句块>

<if语句> ::= if ‘(‘<布尔表达式>’)’ <语句块> [ else <语句块> ]（注：[ ]中的项表示可选）

<表达式> ::= <项> {+ <项> | -<项>}

<项> ::= <因子> {\* <因子> | /<因子>}

<因子> ::=ID|num | ‘(’ <表达式>‘)’

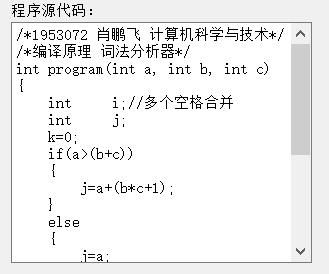
<布尔表达式>::= <表达式> relop  <表达式>

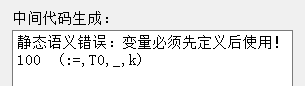
<参数表>::=<内部变量声明> { , <内部变量声明> }

五．扩充的功能

1. 对静态语义错误的诊断和处理

在进行语义分析发过程中，定义了规则：对变量必须先定义后使用。如果出现这个静态语义错误，则停止分析，报对应错误。





在上图中程序符合语法，但是对于变量k在没有定义的情况下就使用，发生了静态语义错误，给出对应的出错信息和截止位置。

2. 在中间代码生成器中，对提供的类C语法规则进行了修改与增加，使得语法分析器能够分析更多类型的文法。

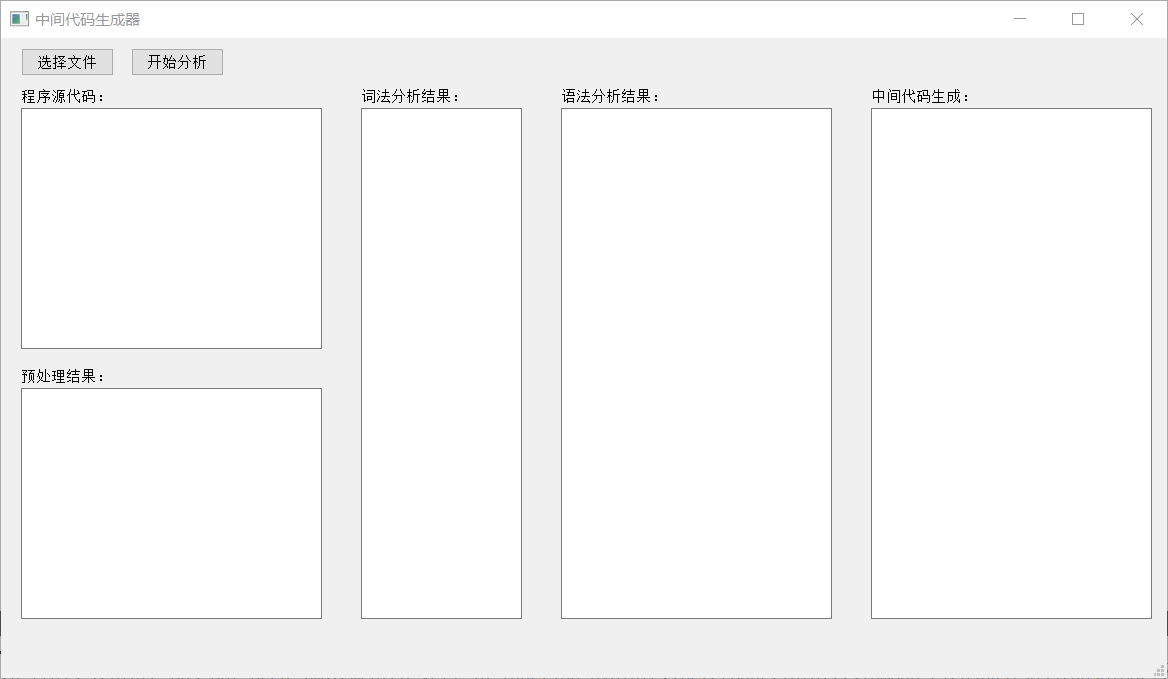
如：增加了<参数表>::=<内部变量声明> { , <内部变量声明> } 用于分析函数定义的形式参数。

将内部声明修改为：<内部声明> ::= 空 | <内部变量声明>;{ <内部变量声明>;}，使其能够正确读取作为分隔符的” ; ”。

增加了<布尔表达式>::= <表达式> relop  <表达式> ，对while和if等语句中的判断条件进行处理。

六．运行结果截图

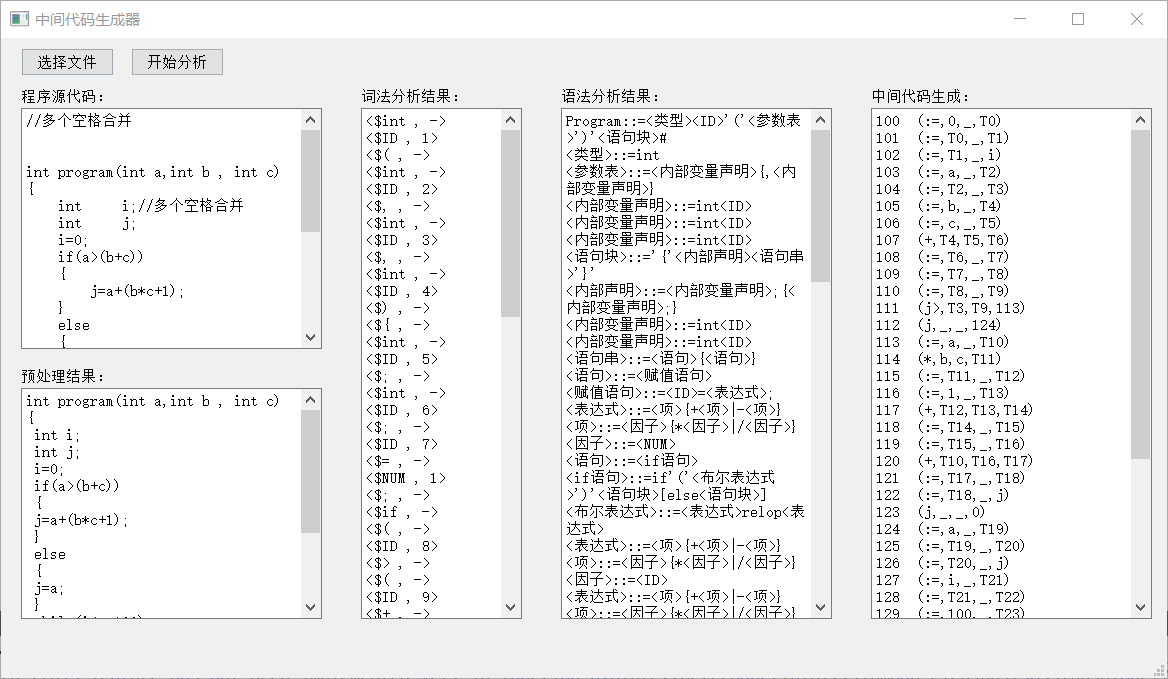
程序界面：



分析成功，示例一：



分析成功，示例二：



分析失败：示例一：



分析失败：示例二：



分析失败：示例三：



七．总结与心得体会

在本次编译原理实验中，我完成了中间代码生成器的设计与代码编写，并撰写了实验报告，将可执行文件和代码一同打包发布并给出了测试用例，完成了实验的全部要求。在此基础上，对程序的功能进行了额外的扩充，增加了静态语义错误处理的功能，同时对于给出的文法进行了进一步的修改和改进。

在实验过程中，我提高了自己对于语义分析和中间代码生成器具体内容和实现方法的认识，通过亲自实践一个完整的中间代码生成器程序，巩固了在课上所学到的知识，增进了对编译原理相关内容的了解。此外，程序界面使用QT框架制作，在过程中学习并使用了C++ 11中的一些新特性，巩固了面向对象程序设计，提高了程序设计能力。