中间代码生成器设计说明

学号：1952650

姓名：陈子翔

目 录

中间代码生成器题目描述

要求:

1. 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）
2. 注意静态语义错误的诊断和处理
3. 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。

类 C 词法规则:

1. 关键字：int | void | if | else | while | return
2. 标识符：字母（字母|数字）*（注：不与关键字相同）
3. 数值：数字（数字）*
4. 赋值号：=
5. 算符：+ | - | * | / | = | == | > | >= | < | <= | !=
6. 界符：；
7. 分隔符：，
8. 注释号：/* */ | //

类 C 语法规则:

1. <if 语句> ::= if ‘(‘<表达式>’)’ <语句块> [else <语句块>]（注：[ ]中的项表示可选）
2. <表达式>::=<加法表达式>{relop <加法表达式>}(注：relop-><|<=|>|>=|==|!=)
3. <加法表达式> ::= <项> {+ <项> | -<项>}
4. <项> ::= <因子> {* <因子> | /<因子>}
5. <因子> ::=ID|num | ‘(’ <表达式>‘)’

1 程序设计说明

1.1 编程语言选择及环境

编程语言：C++

配置环境: 处理器: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz(8 CPUs), ~2.0GHz

内存：8192MB RAM

开发平台：Qt Creator 4.11.1 (Community)

Qt Creator 是跨平台的 Qt IDE。此 IDE 能够跨平台运行，支持的系统包括 Linux(32 位及 64 位)、Mac OS X 以及 Windows。能够使用强大的 C++ 代码编辑器可快速编写代码。
运行环境：Desktop Qt 5.14.2 MinGW 64-bit

1.2 输入方式

本程序采用了 Qt 开发平台设计了可视化界面，用户可以在程序运行后通过点击“源程序”按钮选择相应的要读入的源代码 txt 文件，按钮下方的文本输入框会将 txt 文件中的代码呈现出来。

1.3 能识别的单词

源程序应为类 C 语法程序，其词法规则如下：
① 支持关键字如下

```
keyword = ['int', 'double', 'char', 'float', 'break', 'continue',
                'do', 'while', 'if', 'else', 'for', 'void', 'return']
```

② 标识符应由字母（A-Z,a-z）、数字（0-9）、下划线“_”组成，并且首字符不能是数字，但可以是字母或者下划线_。
③ 字母：a |....| z | A |....| Z |
④ 数字：0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
数值可为无符号整形、科学计数法或复数
⑤ 赋值号可为+= -= *= /= %= ^= &= |= =
算符可为+ - * / % ^ & | = > < <= >= != ==
界符可为 ；
分隔符可为 ，
注释号可为 /* */ 或 //
括号可为 ( ) [ ] { }

识别单词即类型编码如下所示：

单词符号	类型编码	单词符号	类型编码
int	0	<>	21
double	1	==	22
char	2	!=	23
float	3	+	24
break	4	-	25
continue	5	*	26
do	6	/	27
while	7	=	28
if	8	<	29

else	9	>	30
for	10	(	31
goto	11	)	32
for	12	,	33
void	13	;	34
return	14	.	35
标识符	15	[	36
整数	16	]	37
小数	17	:	38
<=	18	{	39
>=	19	}	40
:=	20	“	41

词法语法规则:

```
start:$ identifier
start:$ integer
start:$ scientific
start:$ complex
start:$ limiter
start:$ operator
operator:+
operator:-
operator:*
operator:/
operator:%
operator:^
operator:&
operator:=
operator:>
operator:<
operator:> equal_tail
operator:< equal_tail
equal_tail:=
limiter:,
limiter:;
limiter:[
limiter:]
limiter:(
```

limiter:)
limiter:{
limiter:}
identifier:_
identifier:alphabet
identifier:_ identifier_tail
identifier:alphabet identifier_tail
identifier_tail:_
identifier_tail:digit
identifier_tail:alphabet
identifier_tail:_ identifier_tail
identifier_tail:digit identifier_tail
identifier_tail:alphabet identifier_tail
integer:digit
integer:digit integer_tail
integer_tail:digit
integer_tail:digit integer_tail
decimal:digit
decimal:digit decimal_tail
decimal_tail:digit
decimal_tail:digit decimal_tail
decimal_tail:e signed_index
signed_index:+ index
signed_index:- index
signed_index:digit
signed_index:digit index_tail
complex:digit
complex:digit complex_first_tail
complex_first_tail:digit complex_first_tail
complex_first_tail:+ complex_second_tail
complex_second_tail:i
complex_second_tail:digit complex_second_tail
index:digit
index:digit index_tail
index_tail:digit
index_tail:digit index_tail
scientific:digit
scientific:digit scientific_tail

```
scientific_tail:digit
scientific_tail:digit scientific_tail
scientific_tail:. decimal
scientific_tail:e signed_index
```

1.4 能分析的语法规则

```
[terminal]
break continue
char int float
if else while
* / + - < > = <= >= != && || ==
( ) [ ] { } ' ; $
ID INT FLOAT CHAR

[variable]
program_
program
main_declaration
type_specifier
compound_statement
declaration_list
declaration_type
statement_list
statement
expression_statement
expression
assignment_expression
unary_expression
logical_or_expression
logical_and_expression
equality_expression
relational_expression
additive_expression
multiplicative_expression
iteration_statement
jump_statement
selection_statement
```

[production]
[0] program_ -> program
[1] program -> main_declaration
[2] program -> declaration_list main_declaration
[3] main_declaration -> type_specifier ID ( ) compound_statement
[4] type_specifier -> char
[5] type_specifier -> int
[6] type_specifier -> float

[7] compound_statement -> { }
[8] compound_statement -> { declaration_list }
[9] compound_statement -> { statement_list }
[10] compound_statement -> { declaration_list statement_list }

[11] declaration_list -> declaration_type
[12] declaration_list -> declaration_list declaration_type

[13] declaration_type -> type_specifier ID ;

[14] statement_list -> statement
[15] statement_list -> statement_list statement

[16] statement -> compound_statement
[17] statement -> expression_statement
[18] statement -> iteration_statement
[19] statement -> jump_statement
[20] statement -> selection_statement

[21] expression_statement -> ;
[22] expression_statement -> expression ;

[23] expression -> assignment_expression

[24] assignment_expression -> ID = assignment_expression
[25] assignment_expression -> logical_or_expression

[26] logical_or_expression -> logical_and_expression

[27] logical_or_expression -> logical_or_expression || logical_and_expression

[28] logical_and_expression -> equality_expression
[29] logical_and_expression -> logical_and_expression && equality_expression

[30] equality_expression -> relational_expression
[31] equality_expression -> equality_expression == relational_expression
[32] equality_expression -> equality_expression != relational_expression

[33] relational_expression -> additive_expression
[34] relational_expression -> relational_expression < additive_expression
[35] relational_expression -> relational_expression > additive_expression
[36] relational_expression -> relational_expression <= additive_expression
[37] relational_expression -> relational_expression >= additive_expression

[38] additive_expression -> multiplicative_expression
[39] additive_expression -> additive_expression + multiplicative_expression
[40] additive_expression -> additive_expression - multiplicative_expression

[41] multiplicative_expression -> unary_expression
[42] multiplicative_expression -> multiplicative_expression * unary_expression
[43] multiplicative_expression -> multiplicative_expression / unary_expression

[44] unary_expression -> ID
[45] unary_expression -> INT
[46] unary_expression -> FLOAT
[47] unary_expression -> CHAR
[48] unary_expression -> (logical_or_expression)

[49] iteration_statement -> while (expression) compound_statement

[50] jump_statement -> continue ;
[51] jump_statement -> break ;

[52] selection_statement -> if (expression) statement
[53] selection_statement -> if (expression) statement else statement

1.5 能实现翻译的表达式类型

1.5.1 说明语句

1. 说明语句作用：用关键字定义名字的性质（数据类型）
   ① 简单类型说明
   ② 数组、记录说明
2. 语义动作: 填符号表、信息向量表、除动态数组外不生成四元式。
3. 文法描述：
   ① $D \to integer\ i$
   ② $D \to real\ i$
   ③ $D \to D, i$

1.5.2 简单算术表达式和赋值语句

1. 定义：只含整型变量的简单算术表达式，并符合运算符的结合规则和优先级规定。
2. 文法描述：
   ① $S \to i \coloneqq E$
   ② $E \to E_1 + E_2$
   ③ $E \to E_1 \times E_2$
   ④ $E \to -E_1$
   ⑤ $E \to (E_1)$
   ⑥ $E \to i$

1.5.3 布尔表达式

1. 定义：用布尔运算符把布尔量、关系表达式联结起来的式子。
   布尔运算符：and, or, not；关系运算符：$<, \leq, =, \neq, >, \geq$
2. 作用：
   ① 控制语句的条件
   ② 计算逻辑值
3. 文法：
   ① $E \to E\ and\ E | E\ or\ E | not\ E | (E) | i\ | i_1\ relop\ i_2$
   ② 运算符优先级：布尔运算由高到低：$not\ and\ or$，同级左结合，关系运算符同级，且高于布尔运算符。

1.5.4 控制流语句

1. 定义：考察 if - then，if - then - else，while – do 语句的翻译，三种语句的形式为：if E then S1，if E then S1 else S2，While E do S1，其中 E 为布尔表达式。
2. 文法：

① $S \rightarrow if\ E\ then\ S$
② $S \rightarrow if\ E\ then\ S\ else\ S$
③ $S \rightarrow while\ E\ do\ S$
④ $S \rightarrow begin\ L\ end$
⑤ $S \rightarrow A$
⑥ $L \rightarrow L;S$
⑦ $L \rightarrow S$

1.5.5 数组元素的引用

1. 问题：解决含有数组元素的表达式和赋值语句的翻译。
2. 关键：找到数组元素的地址。数组元素的地址计算方法与数组的存放方式有关。
3. 文法：$Array\ A[l_1:u_1, l_2:u_2, \dots, l_m:u_m]:type;$

2 头/源文件划分及功能描述

2.1 头文件功能描述

1. KeyWord.h

词法分析中的关键字集合。

2. lexicalanalyzer.h

词法分析程序头文件，包含宏定义、结构体定义、词法分析类定义以及类内函数声明。

3. lritem.h

LR 语法分析程序头文件，包含 LR0，LR1 类定义以及类内函数声明。

4. parser.h

不含语法语义分析过程的分析头文件，包含 Parser 类的定义以及类内分析函数的定义。

5. parsertest.h

包含语法语义分析过程的分析头文件，包含 Parser 类的定义以及类内分析函数的定义。

6. production.h

包含语义分析表达式构造的头文件，包含 Production 类的定义以及类内函数的定义。

7. scanner.h

词法分析器扫描类的定义，包括类内处理关键字、运算符、界符、程序预处理等函数的定义。

8. TokenType.h

词法类别的定义，定义了词法类型类 TokenType。

9. widget.h

Qt 界面头文件，包含了 UI 界面的函数声明、信号槽函数的声明。

2.2 源文件功能描述

1. lexicalanalyzer.cpp

词法分析程序源文件，包含词法分析类内各词法处理函数具体定义。

2. lritem.cpp

LR 语法分析程序源文件，包含 LR0，LR1 类内返回变量函数具体定义。

3. parser.cpp

不含语法语义分析过程的分析源文件，包含 Parser 类类内分析函数的具体定义。

4. parsertest.cpp

包含语法语义分析过程的分析源文件，包含 Parser 类类内分析函数的具体定义。

5. production.cpp

包含语义分析表达式构造的头文件，包含 Production 类内函数的具体定义。

6. scanner.cpp

词法分析扫描类源文件，包含 Scanner 扫描类各功能函数的具体定义。

7. widget.cpp

Qt 界面源文件，包含了整体界面的设计，信号槽函数的具体定义，词法/语法/语义分析的总控函数。

3 详细设计说明

3.1 词法分析器主要函数说明

3.1.1 编译预处理：scanner. Preprocessing()

此函数在词法分析过程使用，用于对读入的源程序进行预处理。

目的：编译预处理，剔除无用的字符和注释。

1. 若为单行注释“//”,则去除注释后面的东西，直至遇到回车换行;
2. 若为多行注释“/* 。。。*/”则去除该内容;
3. 若出现无用字符，则过滤；否则加载；最终产生净化之后的源程序。

3.1.2 具体分析程序：Scanner.getNextToken()

此函数在词法分析过程使用，为分析生成 TOKEN 列表的主体函数。具体实现过程为依次遍历迭代器 iter 获取源程序中的符号，根据对读取信息进行分析，不同种类的 token 调用不同的分析函数。直至全部读取完毕。

3.1.3 错误处理函数：error()

具体实现思路在于按照一定的判别顺序判断读入的信息是否是关键字、标识符、数字、符号；若这些均无法识别，则将读入的信息判别为错误。在此大基础上，在各自判断类别内特判一些个别错误（如开头读入数字，判断后续是否还出现字符等），具体能够识别的错误包括不限于:

1. 变量的长度过长（超过 20 位），出现错误;
2. 常量的长度过长（超过 20 位），出现错误;
3. 表示小数时小数点个数大于 1 个（如 1..1），出现错误;
4. 程序中出现无法识别的非法字符;
5. 以数字开头，后续出现非数字的其他字符;
6. 等。

3.2 语法分析器主要函数说明

3.2.1 构造 FIRST 集：Parser.getFirst()

得到终结符的 FIRST 集合。

3.2.2 构造闭包：Parser.GetClosure()

输出表示从产生式推导出的闭包。
计算方式为:
1. 首先将第一个参数加入列表;
（若圆点之后跟着一个非终结符才可进行下述操作）
2. 计算以该非终结符为左侧的产生式的向前搜索符集合;
3. 对以圆点后非终结符为左侧的产生式进行遍历求闭包;
4. 将项目中产生式相同以及圆点位置相同的项目的搜索符做并集。

3.2.3 得到语法分析 Action 表：Parser.getAction()

3.2.4 得到语法分析 Goto 表：Parser.getGo()

3.2.5 具体语法分析过程：Parser.build()

依据已经依据语法规则构建的 ACTION&GOTO 表对经过词法分析得到的 TOKEN 串展开语法分析，得到语法分析结果（符号栈以及语法树）。

3.3 语义分析主要函数说明

3.3.1 语义栈弹出函数：varStackPop ()

语义分析过程中弹出返回语义栈顶部元素。

3.3.2 字符栈弹出函数：Parser.strStackPop()

语义分析过程中弹出返回字符栈顶部元素。

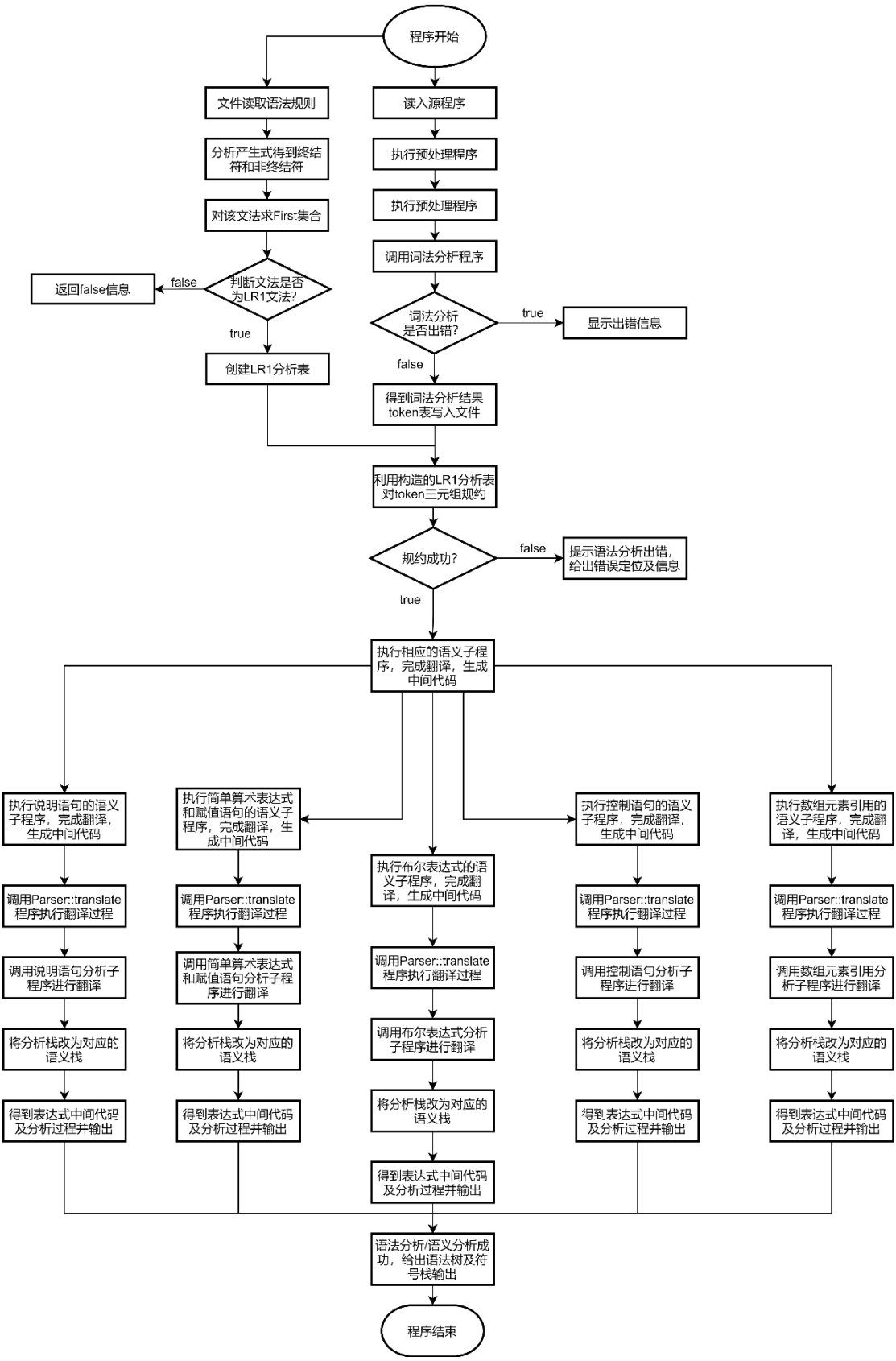
3.3.3 具体语义分析翻译过程：Parser.translate()

在执行语法分析的过程中，对表达式执行相应的语义子程序。完成翻译，生成中间代码。

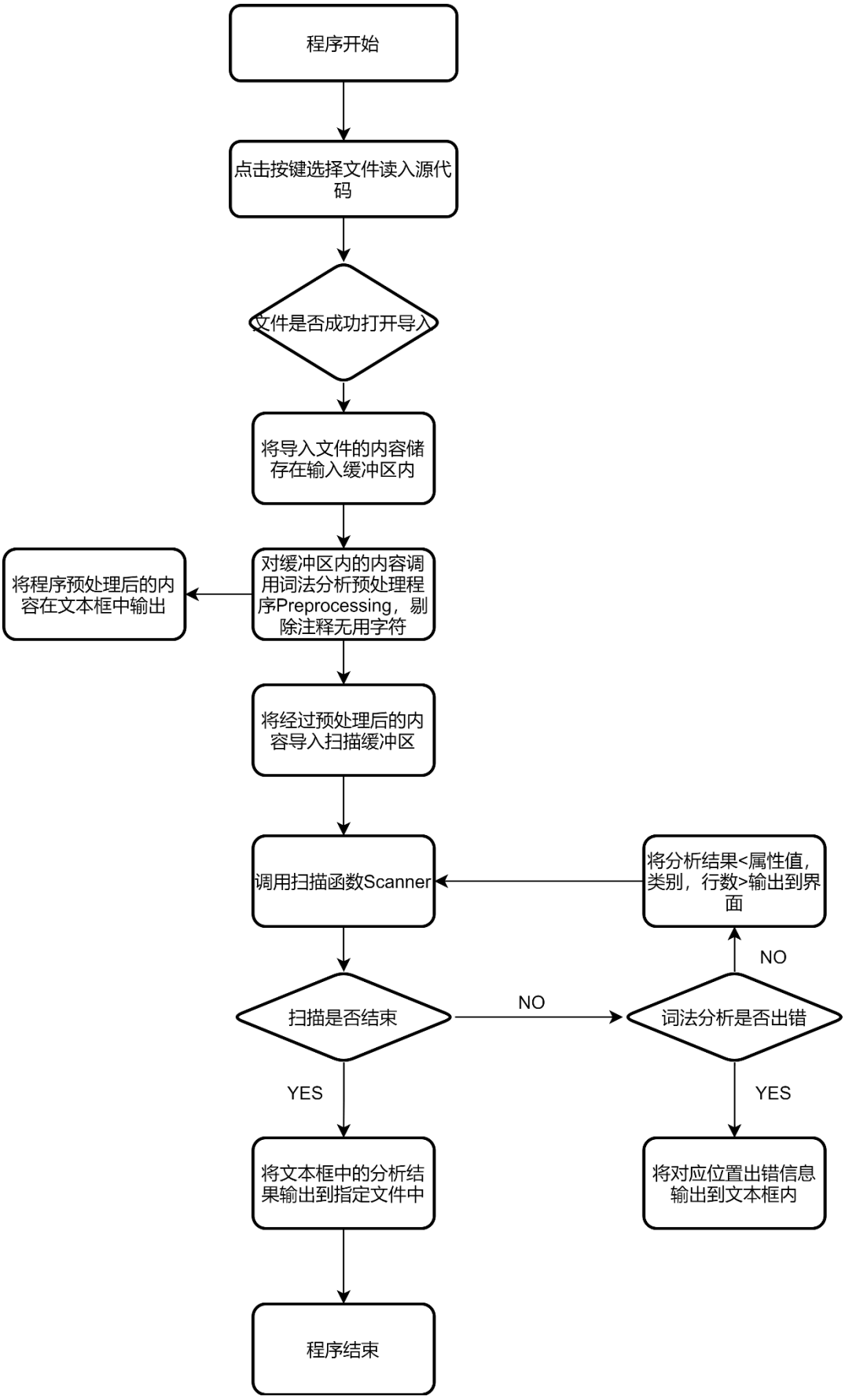
执行不同表达式对应的子程序进行翻译，将语法分析中的分析栈改为对应的语义栈，对栈中元素进行处理，得到表达式及中间代码生成过程并输出。

4 算法分析框图

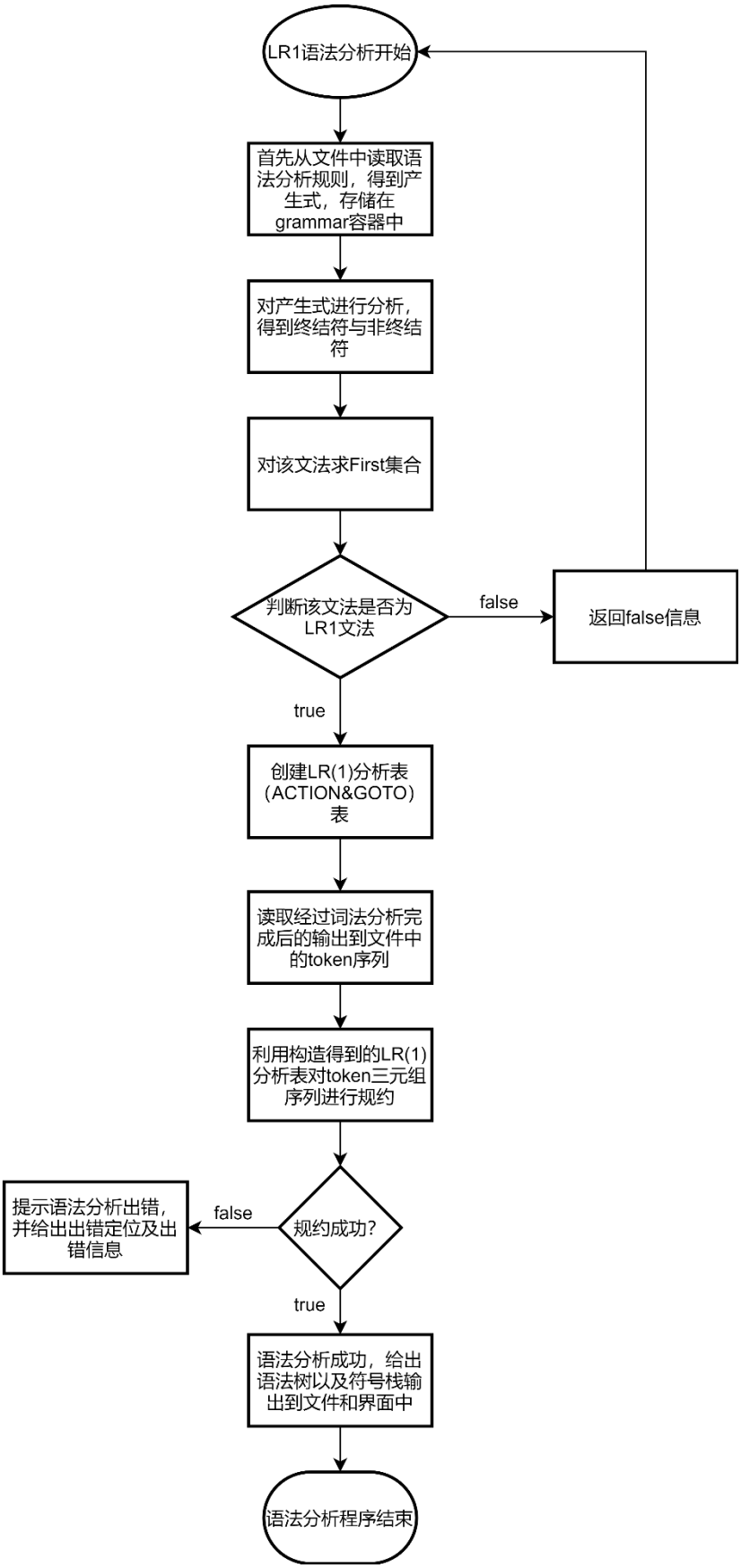
4.1 总程序算法框图



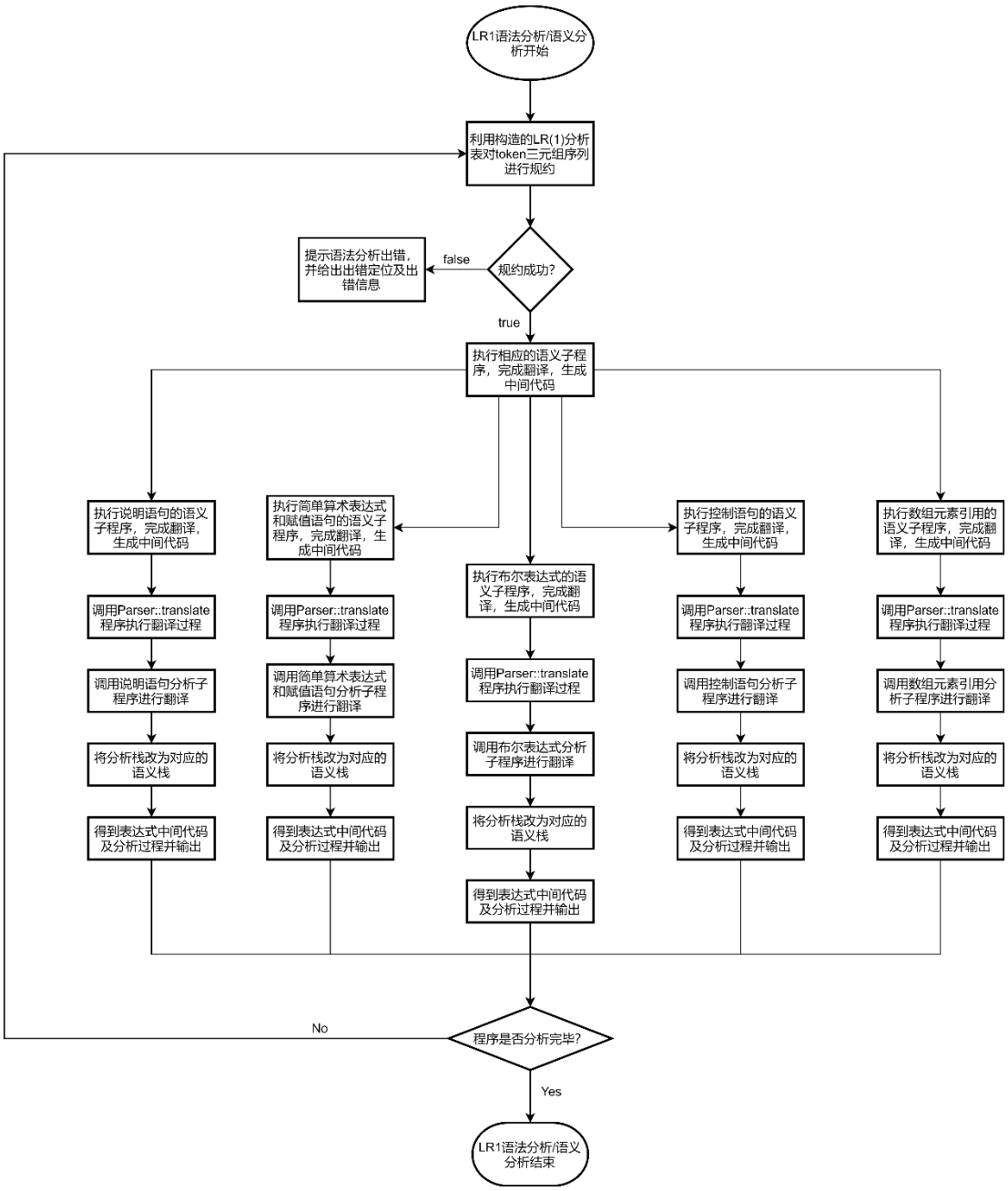
4.2 词法分析算法框图


程序开始
点击按键选择文件读入源代码
文件是否成功打开导入
将导入文件的内容储存在输入缓冲区内
对缓冲区内的内容调用词法分析预处理程序Preprocessing，剔除注释无用字符
将程序预处理后的内容在文本框中输出
将经过预处理后的内容导入扫描缓冲区
调用扫描函数Scanner
将分析结果<属性值，类别，行数>输出到界面
扫描是否结束
NO
词法分析是否出错
NO
YES
YES
将文本框中的分析结果输出到指定文件中
将对应位置出错信息输出到文本框内
程序结束

4.3 语法分析算法框图


LR1语法分析开始
首先从文件中读取语法分析规则，得到产生式，存储在grammar容器中
对产生式进行分析，得到终结符与非终结符
对该文法求First集合
判断该文法是否为LR1文法
false
返回false信息
true
创建LR(1)分析表(ACTION&GOTO)表
读取经过词法分析完成后的输出到文件中的token序列
利用构造得到的LR(1)分析表对token三元组序列进行规约
规约成功?
false
提示语法分析出错，并给出出错定位及出错信息
true
语法分析成功，给出语法树以及符号栈输出到文件和界面中
语法分析程序结束

4.4 语义分析及中间代码生成算法框图



5 静态语义错误的诊断和处理

5.1 使用未经定义的变量

5.1.1 错误诊断

使用未经定义的变量指的是在变量定义之前使用此变量执行运算、赋值等指令。错误诊断思路为：在对表达式执行语义分析时，识别出表达式中的各个变量，查找这些变量是否经过了声明语句的定义，若未声明，则出错。

5.1.2 错误处理

本程序中若出现使用未经定义变量的情况，则直接报错，程序运行结束，同时在中端和 UI 文本框中输出错误及定位信息。

5.1.3 运行实例

5.2 变量名重定义

5.2.1 错误诊断

变量名重定义指的是在一个变量定义之后，又定义了一个相同命名的变量。错误诊断思路为：在执行定义变量指令时，对本次定义的变量名与先前执行的说明语句定义的变量名进行对比，若重复，则出错。

5.2.2 错误处理

本程序中若出现变量名重定义的情况，则直接报错，程序运行结束，同时在中端和 UI 文本框中输出错误及定位信息。

5.2.3 运行实例

6 更为通行的高级语言的语义检查和中间代码生成

6.1 MSVL 程序设计语言

MSVL 语言，是一种时序逻辑程序设计语言，可用于软硬件系统的建模、仿真，验证。MSVL 语言与 C 语言在描述上基本相似，在结构上相较于 C 语言，多了投影语句、并行语句、随机选择语句等复杂结构语句。MSVL 以投影时序逻辑（PTL）为基础，将其与常规程序设计语言结合在一起，使其不仅可以解决传统时序逻辑编程中出现的并发程序设计问题、同步通信问题和框架化问题，于其他的时序逻辑程序设计语言相比，MSVL 具有更强的表达能力。

6.2 LLVM

6.2.1 LLVM 简述

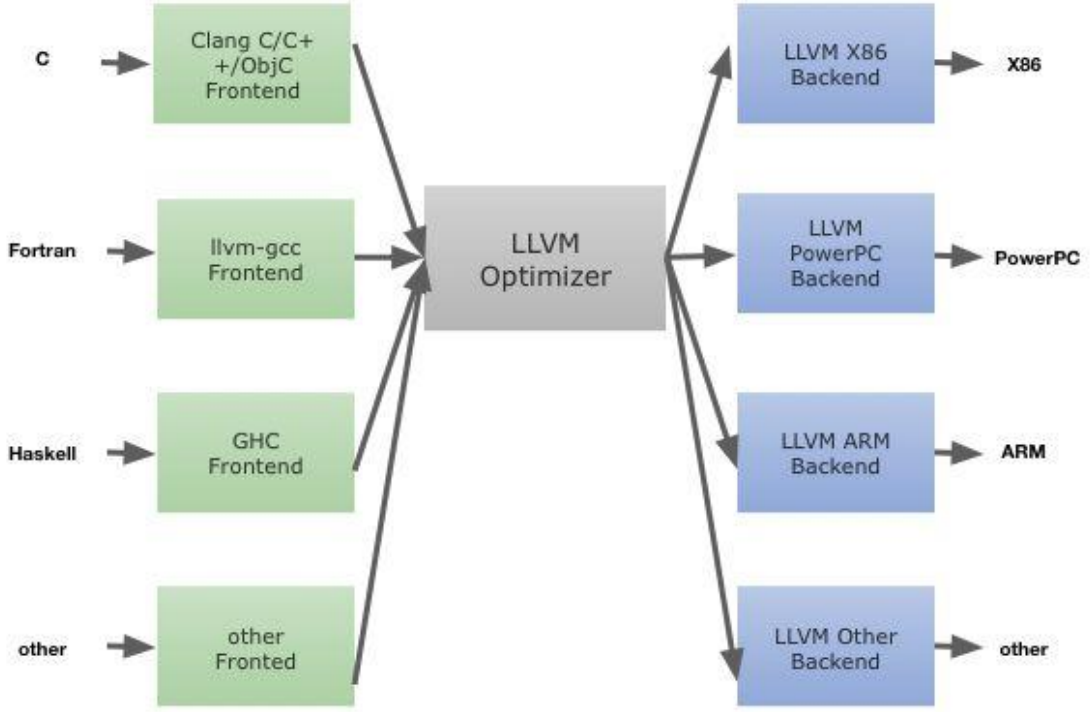
LLVM 是 Low Level Virtual Machine（底层虚拟机）的缩写。LLVM 通过在编译时、链接时、运行时和运行之间的空闲时间为编译器转换提供高级信息，从而支持对任意程序进行透明的、终身的程序分析和转换。LLVM 以静态单赋值（SSA）的形式定义了一种常见的低级代码表示形式，具有如下特性：一个简单的、与语言无关的类型系统，它公开了通常用于实现高级语言特性的原语；一种

用于输入地址运算的指令；一个简单的机制，可以用来实现高级语言的异常处理特性。

6.2.2 LLVM 体系结构

在 LLVM 项目之前，普通的编译器结构通常分为前端、优化和后端。前端部分首先对源码进行分析，确定其没有错误，最后将分析过的源码转换成抽象语法树；优化部分主要是通过对语法树进行大量的等价转换，从而提高效率；后端部分则根据不同的机器生成与之相对应的机器代码。由于语言的不同，需要为各种语言开发不同的前端、优化与后端，重用性较低，有时甚至无法重用。

LLVM 为了简化编译器开发流程，将这三个阶段分离开来，用统一的中间语言 IR 将这三个阶段串联起来。这样就可以无视输入语言的不同带来的问题，只需要设计一个通用的优化器，对于后端模块来说，把 IR 作为它的输入，就能为特定的硬件平台开发位移的后端代码生成器。因此要使用 LLVM 框架开发编译器，需要开发的仅仅是一个输入为 IR 的前端，其他的模块可以不用考虑。LLVM 编译器的结构如下图：



6.2.3 中间代码 IR

LLVM 中间表示代码 IR 是一种静态单赋值语句，IR 代码是一种类似于汇编代码的语言，但它又像其他程序设计语言一样具有函数和语句块。IR 代码不仅能够详细地展示出计算机底层，而且有着良好的可读性。IR 不仅被用于优化器的中间层分析和转换，还被设计用于过程间优化、运行时优化、程序全局分析等。

IR 代码是 LLVM 代码的一种表现形式，LLVM 代码主要有三种表现形式，通常根据它们的存储形式以及存储位置来区分这三种形式。第一种是以二进制的形式存储在内存上的 IR 代码，被称为中间表示 IR，由于其存储在内存上，编译器可以直接、迅速地对它进行分析和处理。第二种通常是以文件的形式来表示，文件存储在磁盘上，代码同样以二进制的形式写入，被称为字节码，在编译期运行时，可以将文件直接读入内存，非常方便。第三种同样为文件的形式，但文件内容是文本，被称为可读的 IR 代码。

6.3 64 位 MSVL 编译器整体架构

6.3.1 语义分析

语义分析要对语法分析产生的语法树进行处理，通过语法分析得到的语法树能够完成对上下文无关语言部分的处理。但是对于那些语法规则涉及不到的语言部分无法处理，而语义分析就是要得到这些语句所代表的含义，根据它们的含义来对它们进行转换。语语法分析相比，语义分析更多的是关注语句自身所代表的含义而不是程序的合法性。语义分析主要实现两个功能，静态语义检查和语法树的转换。静态语义检查包含以下情况：

① 类型检查：类型检查通过将变量、常量以及其他类型存储到一个符号表中来实现。当发现一个操作符跟与它对应的操作数之间的类型无法匹配时，编译器会直接输出错误信息。

② 一致性检查：一致性检查首先要在每个变量定义或声明时通过查询符号表判断其是否已经被定义，还要检查在同一个作用域中是否包含着同名变量，检查选择分支控制语句的分支是否完全相同，若出现相同，编译器发出警告。

③ 控制检查：控制检查要检查苏偶偶的控制流语句，保证它们可以正确地进行程序的跳转。如果无法正确的跳转，发出警告或报错。

④ 名字检查：名字检查要对那些具有相同变量名或代码结构的程序进行一致性检查，若发生冲突则报错。

⑤ 区间长度检查：对与规定长度的 MSVL 语句进行区间长度检查。有一些时序操作符会规定该语句的区间长度，不满足这些操作符的规定无法通过区间长度检查，编译器提示错误。若要进行区间长度检查，首先需要得到该语句的区间长度，对于赋值语句、顺序语句、skip 语句来说，能够直接得到其区间长度，但对于循环语句、分支语句、函数调用语句来说，只有在运行程序后才能够得到其区间长度。

6.3.2 IR 代码生成

编译器要完成目标代码的生成通常会先生成中间代码，也可以直接将源程序翻译为目标程序。中间代码，顾名思义，是位于目标代码与源代码之间的一种全新格式的代码，具有简单的结构、明确的含义，相对于源程序来说更容易被翻译成目标代码。

中间代码的形式有很多，比较常用的有三地址码、后缀表示法等。LLVM 框架所使用的中间代码为三地址码的形式，被称为 IR 代码。IR 代码是一个低级 RISC 类的虚拟指令集，是 LLVM 框架定义的一种有着明确语义的语言，支持轻量级运行时优化以及过程间优化。

在 IR 代码生成模块中，要用 IR 代码构建应用程序接口（API）。因为 LLVM 是通过 C++开发完成的，是一种面向对象的设计，因此 API 都是通过类和类内成员函数的形式提供。IR 代码有着与高级编程语言相似的结构与可读性。IR 代码生成模块将输入的语法树转换为相应的 IR 代码，随后要对语句进行类型检查，提供使用 LLVM 提供的类型系统可以方便快捷地实现类型检查功能。生成 IR 代码后将其存储在内存中，直到将所有的 IR 代码生成完成，将其以一种可读的 IR 格式写入文件并存入硬盘模块类是一种存储 IR 代码的容器，通常需要将 module 的元数据和 IR 代码存入容器中。IR 代码主要以全局变量的定义和声明、自定义类型的定义、函数的定义和生命为主。

6.4 64 位 MSVL 编译器的关键问题

6.4.1 中间代码的生成

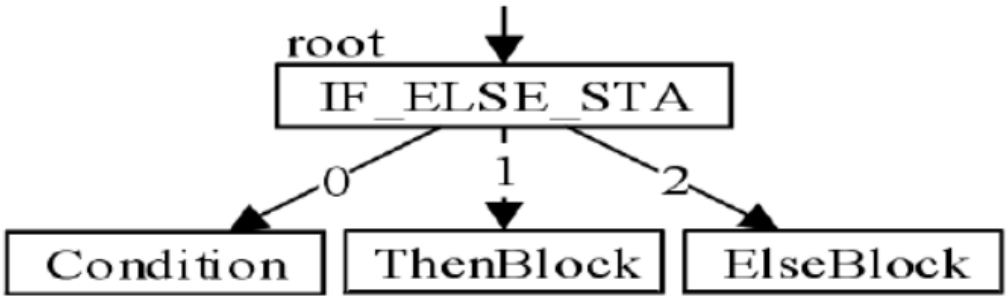
中间代码生成是编译器开发中很重要的一个模块，它是编译器前端的最后一部分，要将之前所有分析结束生成的语法树转换为中间代码。在对 MSVL 编译器的开发中，中间代码生成模块要完成 IR 代码的生成，实际上是要将 MSVL 程序转为 IR 代码。为此，需要将程序中包含的大量的数据类型、语句类型以及运算类型全部转换。例如，结构体、数组、循环语句、分支语句、AND 操作、OR 操作等。

6.4.2 构建 IR 代码

要将 MSVL 程序转换为 IR 代码，主要是对通过词法分析、语法分析、语义分析处理后生成的 MSVL 语法树进行转换。在前期的语法分析语义分析模块将根据给定的规则将 MSVL 程序生成四类语法树：分别是自定义类型树、全局变量树、函数树和主语法树。处理顺序依次为自定义类型树、全局变量树、函数树和主语法树。对于主语法树，对其进行深度优先遍历，从最左的叶子节点开始遍历，直到整个转换完成。对简单的语句，例如算术运算、位运算与逻辑运算等，都可以将其直接翻译成 IR 代码。主要难点在于对函数调用语句、分支语句、循环语句的翻译。

① 分支语句的转换

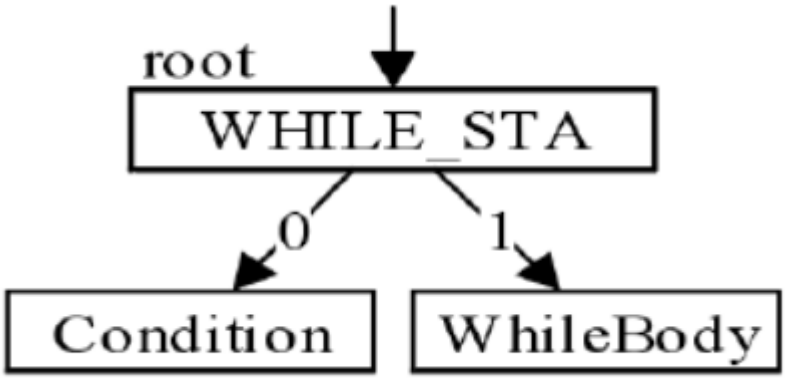
MSVL 的分支语句格式为 if(b) then{p} else{q}。对于分支语句语法树的转换，根据其语法树结构，根节点产生三个后继节点，从左到右分别是条件表达式产生的语法树，条件语句 then 产生的语法树以及条件语句 else 产生的语法树，结构如下图所示：



此部分通过 IR 类中的成员函数 If2IR 来实现，基本过程为：创建一个基本块 ThenBB，并通过 Cond2IR 函数将条件表达式转换为 IR 指令及得到它的返回值 v；随后创建另外两个基本块 ElseBB 和 IfEnd，并通过判断 v 的值的真和假来跳转到相应的基本块。在每个模块完成后会将其翻译完成的 IR 语句插入到相应的模块中。并在每个模块中添加了无条件跳转指令使其可以跳转到结束快 IfEnd，从而完成分支语句的转换。

② 循环语句的转换

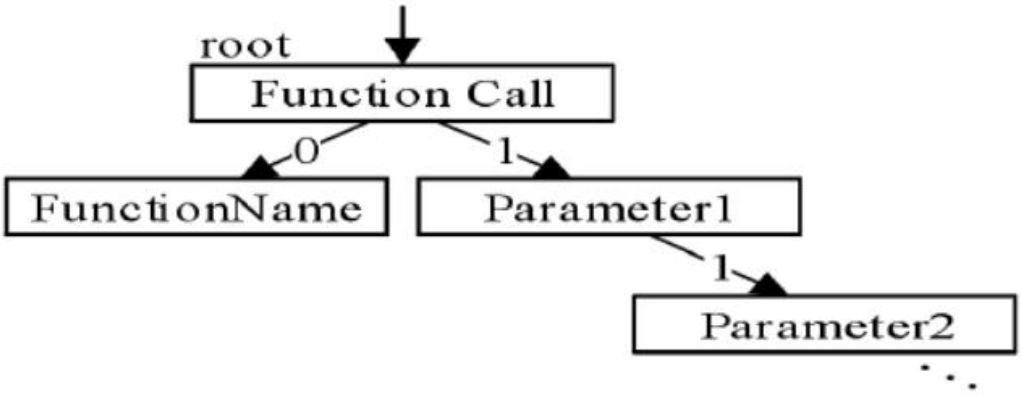
MSVL 的循环语句格式为 while(b) {p}。对于循环语句的语法树转换，根据其语法树的结构，其根节点会产生两个后继节点，分别是循环条件表达式产生的语法树以及循环体产生的语法树，具体结构如下图所示：



该转换会通过 IR 类中的 While2IR 成员函数来实现。与分支语句的转换代码类似，首先得到当前需要转换的函数，随后创建一个基本块 WhileConBB，并且为其创建转换指令，将转换后的 IR 代码添加到 WhileConBB 块中，然后通过之前介绍过的 Cond2IR 函数得到循环条件表达式的值 v，继续创建另外两个基本块。

③ 函数调用语句的转换

MSVL 的内部函数调用语句格式为$fun(x_1, x_2, \ldots x_n)$。对于函数调用语句的语法树转换，根据其语法树结构，根节点有两个后继节点，左边的节点代表的是函数名，右边的节点代表的是函数参数树，当函数含有多个参数时，函数参数语法树将类似于二叉树，即其第二个孩子节点是其兄弟节点。特别的是，函数名有可能是函数指针或返回值为函数指针的表达式，函数调用语句语法树结构如下图所示：



7 程序使用说明

7.1 UI 界面按键功能说明

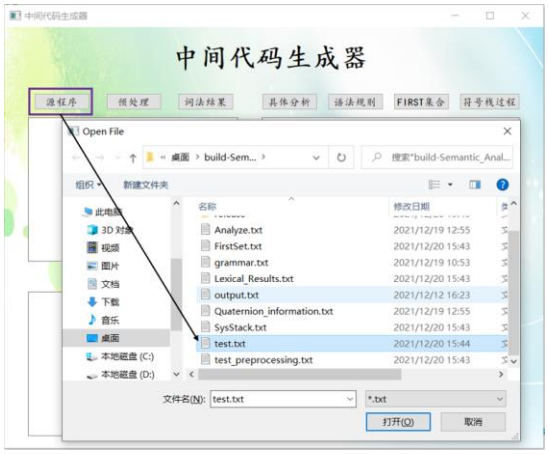
1. 总界面:

该中间代码生成器词法分析部分按键包括: “源程序“、”预处理“、”词法结果“; 语法语义分析部分按键包括: ”具体分析“、”语法规则“、”FIRST 集合“、”符号栈过程“; 中间代码生成部分按键包括: ”四元式“、”语义分析过程“

2. 词法分析按键:

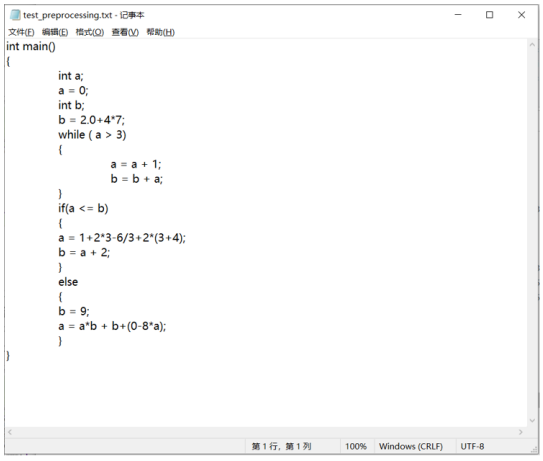
① “源程序”

按下此按键可以进行文件路径选择，选择需要执行词法分析的文件，并将此文件中的内容显示在按键下方的文本框中。



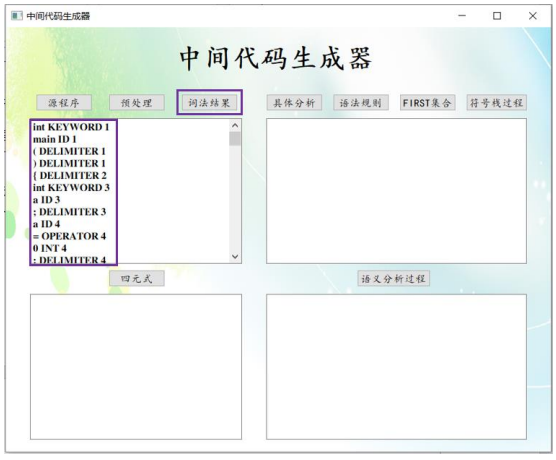
② “预处理”

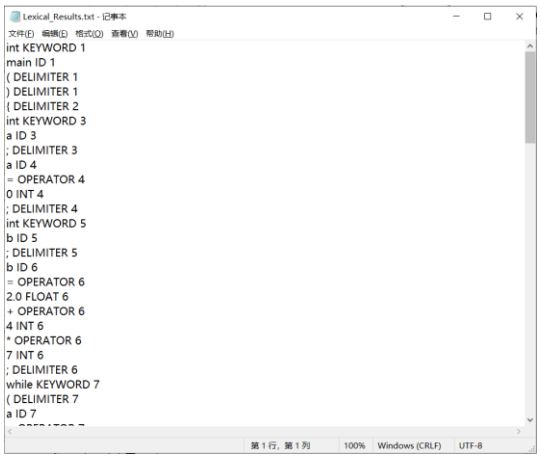
按下此按键对源程序执行预处理操作，去除源代码中的注释部分，将经过预处理后去除注释的源代码输出到文件中，同时在程序终端输出预处理后的源代码。



③ “词法结果”

按下此按键将源程序经过词法分析程序后的结果输出到文件 Lexical_Results.txt 中，同时将词法分析结果（token 表）输出下方文本框中。



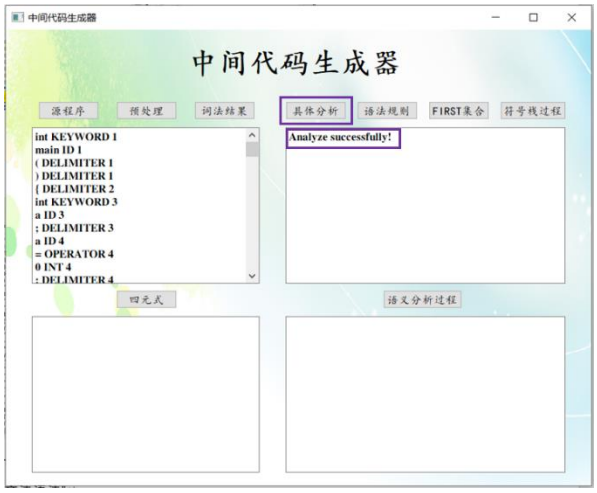


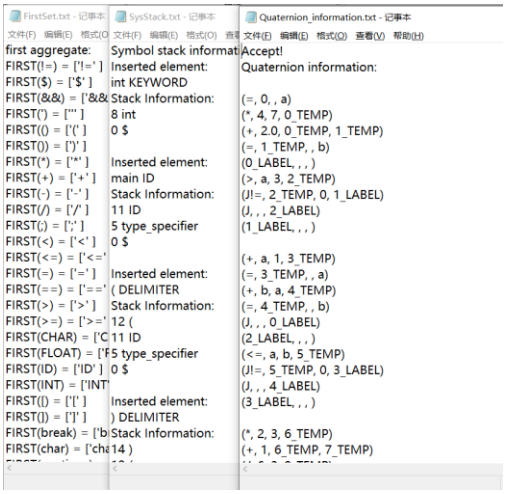
2. 语法语义分析按键：

① “具体分析”

按下此按键执行语法及语义分析程序，对词法分析生成的 token 表按照指定语法规则进行分析，得到 FIRST 集合、符号栈分析过程输出到文件中，同时生成 ACTION&GOTO 分析表。同时在终端输出语法分析结果信息：分析正确、分析错误+出错处信息。

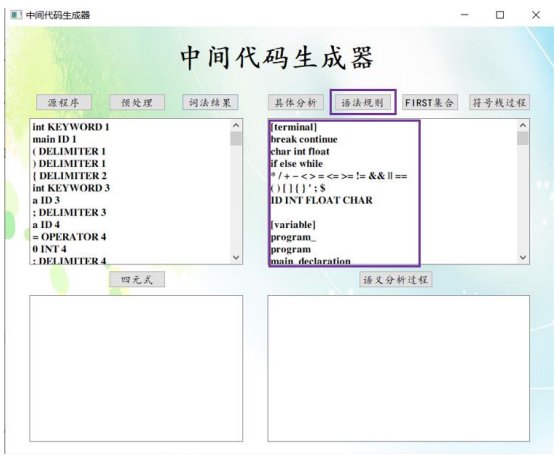
执行语义分析程序，同时将得到的四元式输出到文件 Quaternion_information.txt 中。

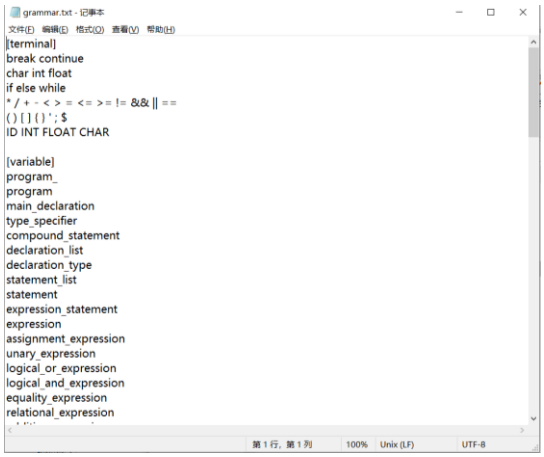




② “语法规则”

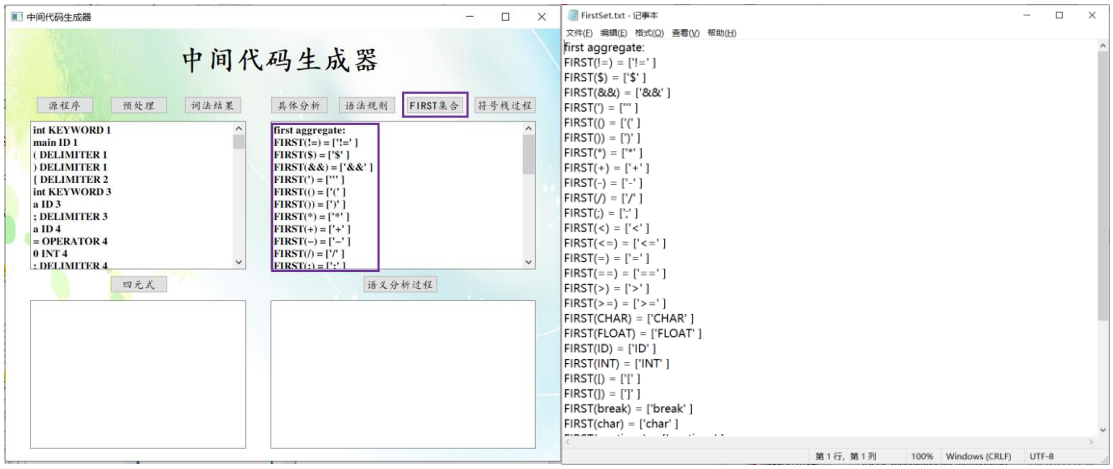
按下此按键显示语法分析文法规则。将文法规则从 grammar.txt 中输出到文本框中。





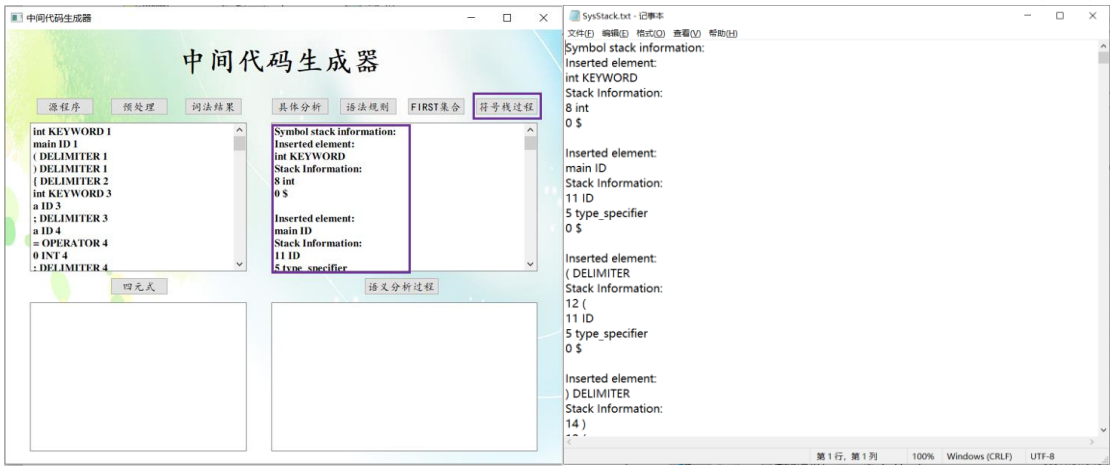
③ “FIRST 集合”

按下此按键显示对 token 表按指定语法分析文法规则得到的 FIRST 集合。将 FIRST 集合从 FirstSet.txt 中输出到文本框中。



④ “符号栈过程”

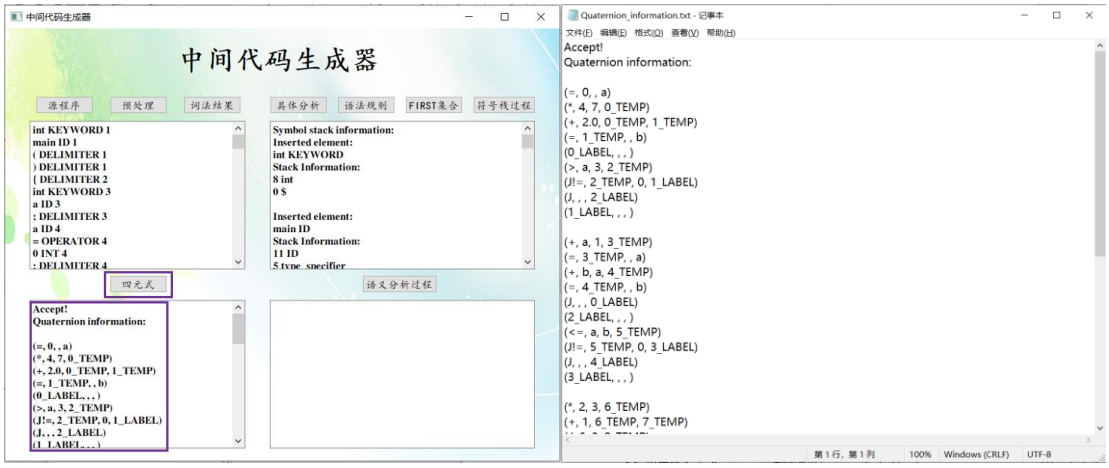
按下此按键显示执行语法分析的符号栈分析过程。将符号栈分析过程从 SysStack.txt 中输出到文本框中。



3. 中间代码生成按键:

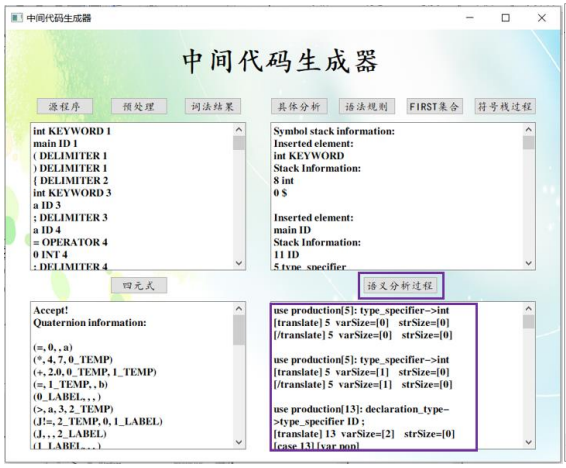
① “四元式”

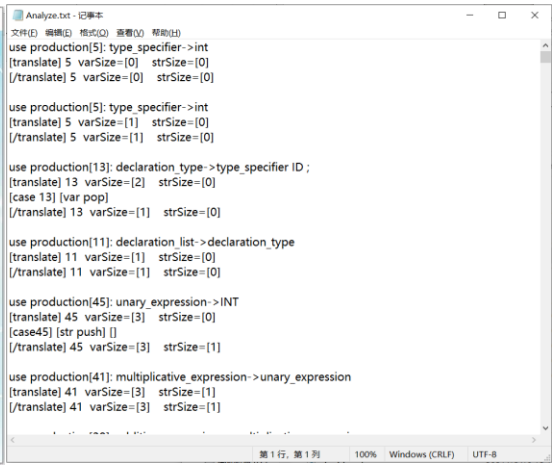
按下此按键显示执行完语义分析生成的中间代码（四元式）。将中间代码（四元式）从 Quaternion_information.txt 中输出到文本框中。



② “语义分析过程”

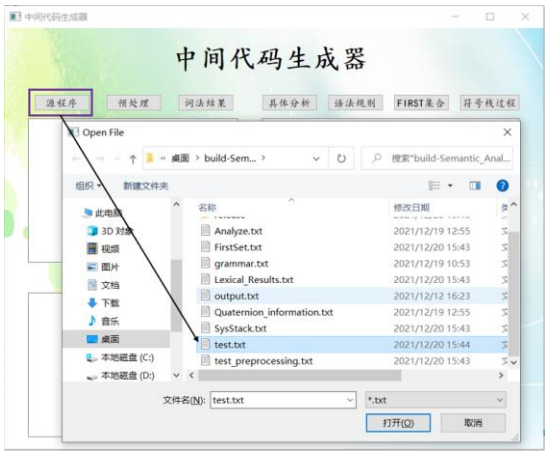
按下此按键显示执行语义分析生成中间代码（四元式）的具体过程。将具体过程从 Analyze.txt 中输出到文本框中。





7.2 程序输入方式

① 通过点击“源程序”按键选择文件路径导入文件中的源代码，程序会将文件中需要执行词法分析的代码显示在文本框中。



② 还可通过直接在文本框中输入要进行词法分析的源代码进行分析。

7.3 结果输出方式

① 程序将经过词法/语法/语义分析所得的结果在界面中的文本框中显示出来。

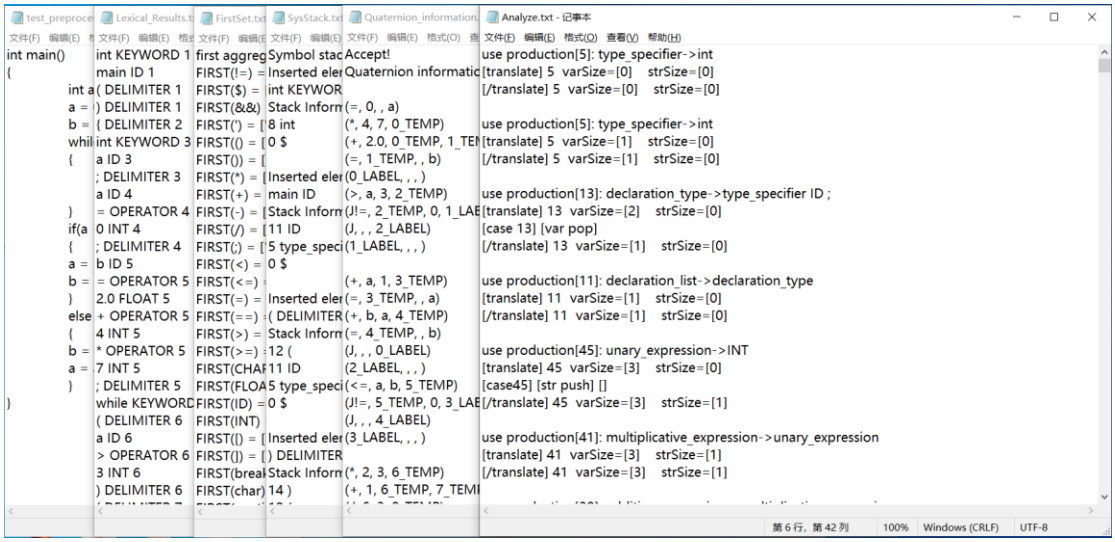
如图在不同的文本框中分别输出源程序预处理结果、词法分析结果 token 表、语法分析 FIRST 集合、语法分析符号栈分析过程、中间代码四元式以及语义分析过程。

② 在程序终端输出词法/语法/语义分析结果信息，若分析成功输出成功信息，若分析失败则给出出错原因提示。

```
Symbol stack information:
Inserted element:
int KEYWORD
Stack Information:
8 int
0 $

Inserted element:
main ID
Stack Information:
11 ID
5 type_specifier
0 $
```

③ 同时程序将经过预处理后的源程序、词法分析的结果（程序 token 表）、语法分析的结果（FIRST 集合、符号栈分析过程）、语义分析过程（中间代码四元式、具体分析过程）分别输出到项目目录下的 test_preprocessing.txt、Lexical_Results.txt、FirstSet.txt、SysStack.txt、Quaternion_information.txt、Analyze.txt 中。



8 执行实例

8.1 用户友好界面

本次词法分析结果主体采用 UI 界面输出。运行程序后，用户可以通过点击“源程序”按钮选择要读入源文件的路径并将程序内容输出到左侧文本框中，直观显示。读入程序内容结束后，点击“预处理”按钮执行源程序预处理操作，并将预处理结果呈现在文本框中，点击“词法结果”按钮，将词法分析结果（TOKEN 表）具体在文本框中呈现；点击“语法规则”可以查看语法规则；点击

“具体分析”按钮具体执行语法、语义分析程序并将程序的结果输出到对应的文本文件中。点击语法相关按键执行语法分析操作，显示语法分析过程的信息（FIRST 集合、符号栈内容）。点击“四元式”、“语义分析过程”按钮在对应的文本框中呈现语义分析相应内容。

总体交互体验不错。

8.2 运行结果截图

8.2.1 语义错误——使用未经定义的变量

如左侧文本框内源代码，变量 b 在未被定义的情况下直接被使用，语义规则不允许，故在右侧执行完具体分析后，文本框内输出“Semantic error at line 5!”错误提示，指出语义分析错误，且错误位置为程序第 5 行。

8.2.2 语义错误——变量名重定义

如左侧文本框内源代码，变量 a 被重复定义，语义规则不允许，故在右侧执行完具体分析后，文本框内输出“Semantic error at line 7!”错误提示，指出语义分析错误，且错误位置为程序第 7 行。

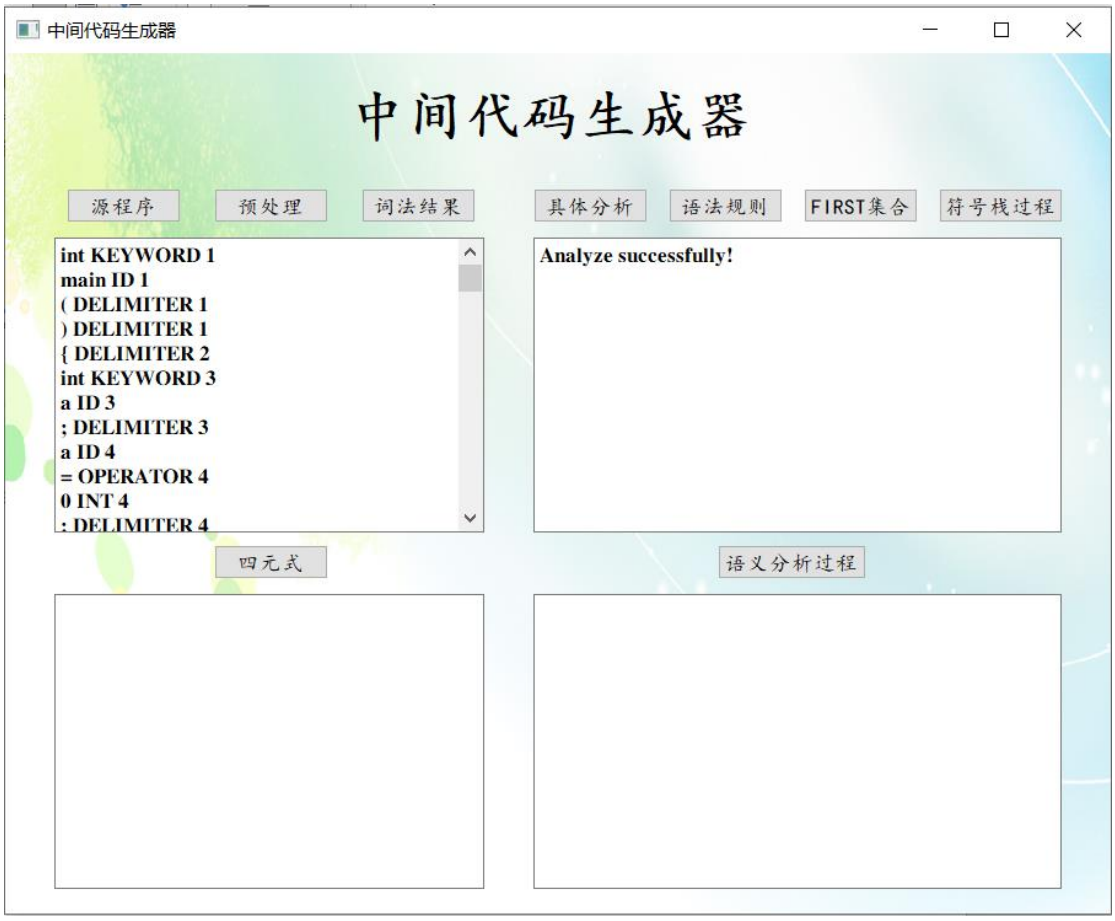
8.2.3 语义分析正确实例

源程序:

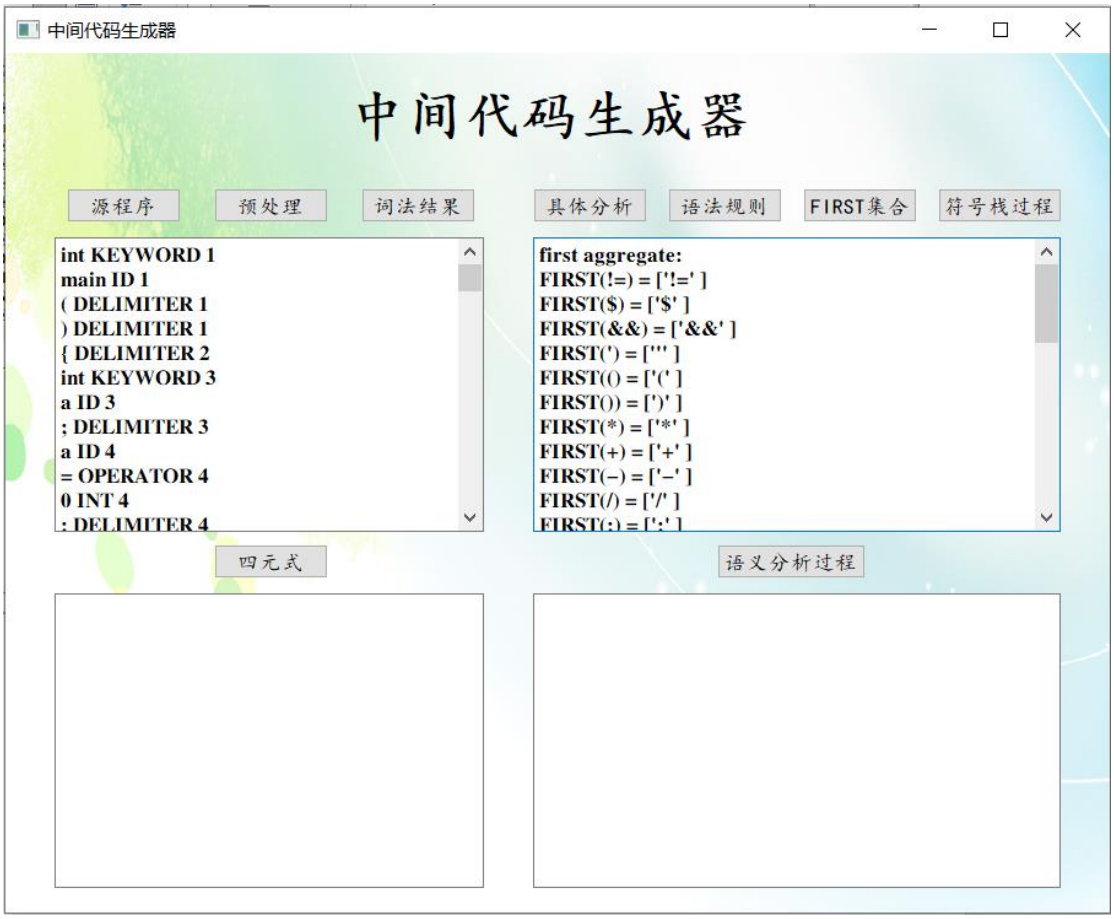
```
int main()
{
    int a;
    a = 0;
    int b;
    b = 2.0+4*7;
    while ( a > 3)
    {
        a = a + 1;
        b = b + a;
    }
    if(a <= b)
    {
        a = 1+2*3-6/3+2*(3+4);
```

```
        b = a + 2;
    }
    else
    {
        b = 9;
        a = a*b + b+(0-8*a);
    }
}
```

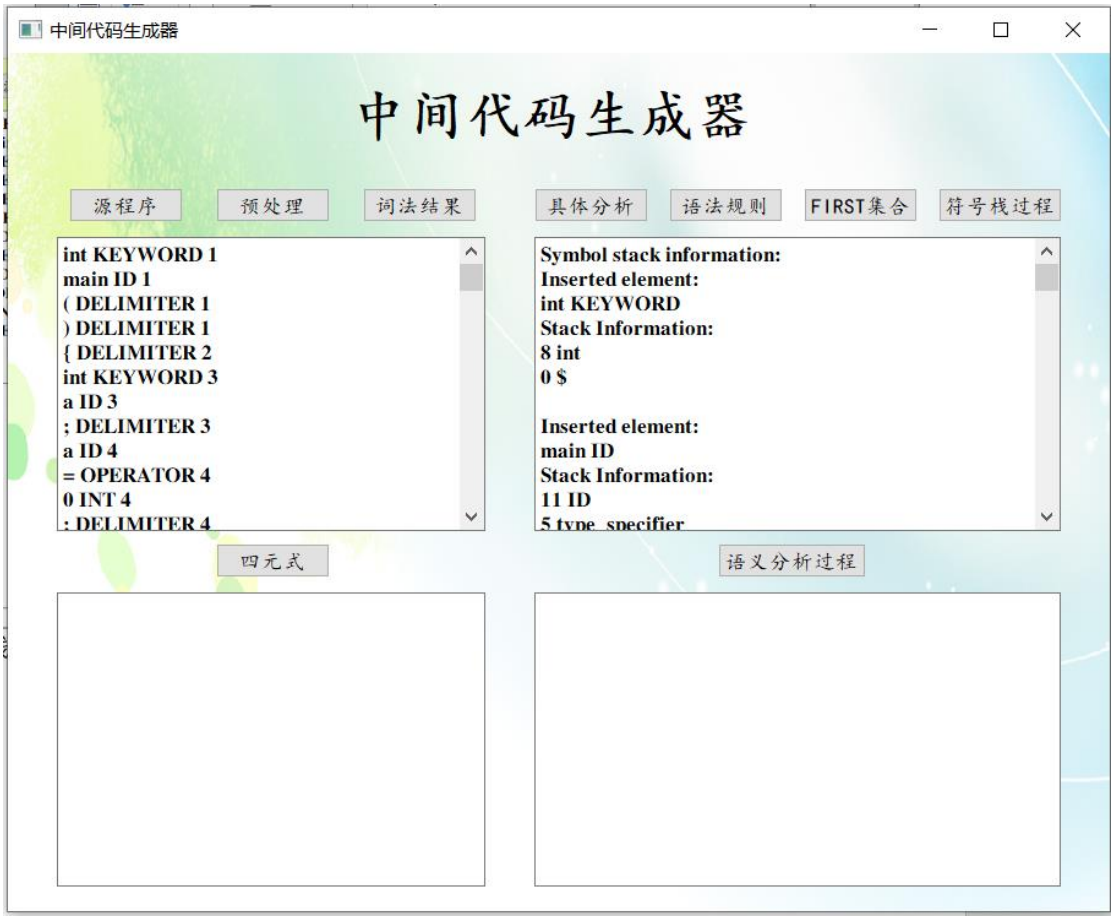
词法分析结果&语法/语义分析成功



FIRST 集合:


中间代码生成器
中间代码生成器
源程序
预处理
词法结果
具体分析
语法规则
FIRST集合
符号栈过程
int KEYWORD 1
main ID 1
(DELIMITER 1
) DELIMITER 1
{ DELIMITER 2
int KEYWORD 3
a ID 3
; DELIMITER 3
a ID 4
= OPERATOR 4
0 INT 4
first aggregate:
FIRST(!=) = ['!=']
FIRST($) = ['$']
FIRST(&&) = ['&&']
FIRST(') = [''']
FIRST(() = ['(']
FIRST()) = [')']
FIRST(*) = ['*']
FIRST(+) = ['+']
FIRST(-) = ['-']
FIRST(/) = ['/']
四元式
语义分析过程

符号栈过程:


中间代码生成器
中间代码生成器
源程序
预处理
词法结果
具体分析
语法规则
FIRST集合
符号栈过程
int KEYWORD 1
main ID 1
(DELIMITER 1
) DELIMITER 1
{ DELIMITER 2
int KEYWORD 3
a ID 3
; DELIMITER 3
a ID 4
= OPERATOR 4
0 INT 4
; DELIMITER 4
Symbol stack information:
Inserted element:
int KEYWORD
Stack Information:
8 int
0 $
Inserted element:
main ID
Stack Information:
11 ID
5 type_specifier
四元式
语义分析过程

四元式&语义分析过程:

同时，程序还将相关信息输出到对应的文本文件中。

9 程序源代码

9.1 widget.cpp

```
#include "widget.h"
#include "ui_widget.h"
#include <iostream>
#include <cstdio>
#include "scanner.h"
#include "parser.h"
#include "parsertest.h"
#include <fstream>
#include <QDebug>
#include <QFile>
#include <QFileDialog>
#include <QPainter>

//#pragma warning(disable:4996)
using namespace std;

Widget::Widget(QWidget *parent)
```

```
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);

    setWindowTitle("中间代码生成器");


QObject::connect(ui->SourceProgramButton,SIGNAL(clicked(bool)),this,S
LOT(readButtonSlot()));

QObject::connect(ui->PreProcessingButton,SIGNAL(clicked(bool)),this,S
LOT(PreprocessingButtonSlot()));

QObject::connect(ui->LexResultsButton,SIGNAL(clicked(bool)),this,SLOT
(LexicalResultsButtonSlot()));

QObject::connect(ui->SynGrammarButton,SIGNAL(clicked(bool)),this,SLOT
(SynGrammarButtonSlot()));

QObject::connect(ui->FirstSetButton,SIGNAL(clicked(bool)),this,SLOT(F
irstSetButtonSlot()));

QObject::connect(ui->SysStackButton,SIGNAL(clicked(bool)),this,SLOT(S
ysStackButtonSlot()));

QObject::connect(ui->AnalyzeButton,SIGNAL(clicked(bool)),this,SLOT(An
alyzeButtonSlot()));

QObject::connect(ui->AnalyzeTestButton,SIGNAL(clicked(bool)),this,SLO
T(AnalyzeTestButtonSlot()));

QObject::connect(ui->QuaternionInformationButton,SIGNAL(clicked(bool)
),this,SLOT(QuaternionInformationButtonSlot()));
}

void Widget::paintEvent(QPaintEvent *)
{
    //创建画家，指定绘图设备
    QPainter painter(this);
    //创建 QPixmap 对象
    QPixmap pix;
```

```
    //加载图片
    pix.load(":/picture/background1.jpg");
    //绘制背景图
    painter.drawPixmap(0, 0, this->width(), this->height(), pix);
}

void Widget::AnalyzeButtonSlot()
{
    Scanner scanner;
    auto tokens = scanner.getTokens("test_preprocessing.txt");
    Parser parser;
    parser.openFile("grammar.txt");
    parser.build();
    parser.analyse(tokens);

    //输出文件重定向
    ofstream fout;
    fout.open("FirstSet.txt", ios::out);
    if (fout.is_open() == 0)
    {
        cout << "output open failed" << endl;
        exit(-1);
    }

    //以下输出终结符的 first 集
    std::cout << "first aggregate:" << std::endl;
    fout << "first aggregate:" << std::endl;
    for (auto i : parser.terminalSet)
    {
        std::set<std::string> iter = parser.getFirst(i);
        std::cout << "FIRST(" << i << ") = [";
        fout << "FIRST(" << i << ") = [";
        for (std::set<std::string >::iterator it = iter.begin();
it != iter.end(); ++it)
        {
            std::cout << '\'' << *it << '\'' << " ";
            fout << '\'' << *it << '\'' << " ";
        }

        fout << "]" << std::endl;
    }
    fout.close();
```

```
    ui->textEdit_2->setText("Analyze successfully!");
}

void Widget::AnalyzeTestButtonSlot()
{
    Scanner scanner;
    auto tokens = scanner.getTokens("test_preprocessing.txt");
    ParserTest parsertest;
    parsertest.openFile("grammar.txt");
    parsertest.build();
    parsertest.analyse(tokens);

    QFile file( "Analyze.txt" );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
    ui->textEdit_3->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void Widget::readButtonSlot()
{
    QString fileName = QFileDialog::getOpenFileName(
                this,
                tr("Open File"),
                "",
                tr("*.txt"));
    QFile file( fileName );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
```

```
    QString line = in.readAll();//读取所有
    ui->textEdit->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void Widget::PreprocessingButtonSlot()
{
    //输出文件重定向
    ofstream fout;
    fout.open("test_preprocessing.txt", ios::out);
    if (fout.is_open() == 0)
    {
        cout << "output open failed" << endl;
        exit(-1);
    }

    //输入文件重定向
    ifstream infile;
    infile.open("test.txt", ios::in);
    if (infile.is_open() == 0)
    {
        cout << "input open failed" << endl;
    }

    std::string line;

    std::string inn;
    while (getline(infile, line))
    {
        inn += line + '\n';
    }

    cout << "source program:" << endl << inn << endl;
    Scanner scanner;
    inn = scanner.Preprocessing(inn, inn.size());

    fout << inn;

    cout << "after program preprocessing:" << endl << inn << endl;
```

```
    QString qstrinn = QString::fromStdString(inn);

    ui->textEdit->setText(qstrinn);

    fout.close();
    infile.close();
}

void Widget::LexicalResultsButtonSlot()
{
    Scanner scanner;

    auto tokens = scanner.getTokens("test_preprocessing.txt");

    //输出文件重定向
    ofstream fout;
    fout.open("Lexical_Results.txt", ios::out);
    if (fout.is_open() == 0)
    {
        cout << "output open failed" << endl;
        exit(-1);
    }
    //cout << "Lexical results:" << endl;
    for (auto i : tokens)
        fout << i.getName() <<" " << i.getTypeOutput() << " " <<
i.getLine() << endl;
    fout.close();

    QFile file( "Lexical_Results.txt" );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
    ui->textEdit->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}
```

```
void Widget::SynGrammarButtonSlot()
{
    QFile file( "grammar.txt" );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
    ui->textEdit_2->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void Widget::FirstSetButtonSlot()
{
    QFile file( "FirstSet.txt" );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
    ui->textEdit_2->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void Widget::SysStackButtonSlot()
{
    QFile file( "SysStack.txt" );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
```

```
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
    ui->textEdit_2->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void Widget::QuaternionInformationButtonSlot()
{
    QFile file( "Quaternion_information.txt" );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug()<<"no read";
        return;
    }
    qDebug()<<"Yes read";
    QTextStream in(&file);
    QString line = in.readAll();//读取所有
    ui->textEdit_4->setText(line);
    //设置文本框只读
    //ui->textEdit->setReadOnly(true);
    file.close();
}

void getToken()
{
    Scanner scanner;
    string FileName =
"C:/Users/C.X/Desktop/Semantic_Analyzer/test_preprocessing.txt";

    auto tokens = scanner.getTokens(FileName);

    for (auto token : tokens)
    {
        cout << "    " << token.getName() << " " <<
TokenDict[token.getType()] << "  " << token.getLine() << endl;
    }
}
```

```
void getGrammar()
{
    Parser parser;
    if (parser.openFile("grammar.txt"))
        cout << "ok" << endl;
    else
    {
        cout << "oh no!" << endl;
        return;
    }
    parser.build();
    auto grammar = parser.getGrammar();
    auto closurelist = parser.getClosurelist();
    auto closuremap = parser.getClosuremap();
    for (auto lr1set : closurelist)
    {
        cout << "I" << closuremap[lr1set] << endl;
        for (auto lr1 : lr1set)
        {
            cout << "  ";
            auto id = lr1.getLeft().getLeft();
            auto point = lr1.getLeft().getRight();
            auto str = lr1.getRight();
            cout << grammar[id].getLeft() << "->";
            auto right = grammar[id].getRight();
            for (unsigned int i = 0; i < right.size(); i++)
            {
                if (i == point)
                    cout << ".";
                cout << right[i];
            }
            if (point == right.size())
                cout << ".";
            cout << "|" << str << endl;
        }
    }
    cout << endl;
    auto transfer = parser.getTransfer();
    for (unsigned int i = 0; i < transfer.size(); i++)
    {
        if (transfer[i].size())
        {
```

```
            cout << "from I" << i << ":" << endl;
            for (auto e : transfer[i])
            {
                cout << "  str=|" << e.first << "|  to  I" <<
e.second << endl;
            }
        }
    }
    cout << endl;

    cout << "\t";
    for (auto e : parser.getTerminalSet())
        cout << e << "\t";
    for (auto e : parser.getVariableSet())
        cout << e << "\t";
    cout << endl;

    auto action = parser.getAction();
    auto go = parser.getGo();
    for (unsigned int i = 0; i < action.size(); i++)
    {
        cout << "I" << i << "\t";

        for (auto e : parser.getTerminalSet())
        {
            for (auto one : action[i])
                if (one.first == e)
                    cout << one.second.first << one.second.second;
            cout << "\t";
        }
        for (auto e : parser.getVariableSet())
        {
            for (auto one : go[i])
                if (one.first == e)
                    cout << one.second << "\t";
            cout << "\t";
        }
        cout << endl;
    }
    cout << endl;
}
```

```
void analyse()
{
    Scanner scanner;

    auto tokens = scanner.getTokens("test_preprocessing.txt");

    for (auto i : tokens)
        cout << i.getName() <<" " << i.getTypeOutput() << " " <<
i.getLine() << endl;
    cout << endl;
    system("pause");
    Parser parser;
    parser.openFile("parser/grammar.txt");
    parser.build();
    parser.analyse(tokens);

    std::cout << "first 集: " << endl;
    for (auto i : tokens)
    {
        std::set<std::string> iter = parser.getFirst(i.getName());
        std::cout << i.getName() << ": ";
        for (std::set<std::string >::iterator it = iter.begin();
it != iter.end(); ++it)
            std::cout << *it << " ";
        std::cout << endl;
    }

}

Widget::~Widget()
{
    delete ui;
    }
```

9.2 parser.cpp

```
#include "parser.h"
#include "production.h"
#include <vector>
#include <string>
#include <fstream>
#include <fstream>
#include <cctype>
```

```cpp
#include <queue>
#include <stack>
#include <sstream>
//#define test

#include <iostream>
using namespace std;

std::stack<std::string> strStack;
std::stack<std::string> varStack;
int tempNum;
int labelNum;

std::string itoTemp(int i)
{
    std::ostringstream temp;
    temp << i << "_TEMP";
    return temp.str();
}

std::string itoLabel(int i)
{
    std::ostringstream temp;
    temp << i << "_LABEL";
    return temp.str();
}

std::string varStackPop()
{
    auto var = varStack.top();
    varStack.pop();
    return var;
}

std::string strStackPop()
{
    auto str = strStack.top();
    strStack.pop();
    return str;
}
```

```
std::string cmd(std::string a, std::string b, std::string c,
std::string d)
{
    return  "(" + a + ", " + b + ", " + c + ", " + d + ")";
}

int Parser::translate(int id, std::string name)
{
#ifdef test
    cout << "[translate] " << id << "  varSize=[" << varStack.size()
<< "]";
    cout << "    strSize=[" << strStack.size() << "]" << endl;
#endif // test
    switch (id)
    {
    case 3:
    {
        varStackPop();
        varStackPop();
#ifdef test
        cout << "[case 3] [var pop]" << endl;
        cout << "[case 3] [var pop]" << endl;
#endif // test
        break;
    }

    case 9: case 10:
    {
        auto str = strStackPop();
        std::ostringstream os;
        os << std::endl << str;
        strStack.push(os.str());
        break;
    }

    case 13:
    {
        varStackPop();
#ifdef test
        cout << "[case 13] [var pop]" << endl;
#endif // test
        break;
```

```
    }

    case 15:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        strStack.push(os.str());
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        varStack.push(lvar);
#ifdef test
        cout << "[case 15] [str pop] " << endl;
        cout << "[case 15] [str pop] " << endl;
        cout << "[case 15] [str push] " << os.str() << endl;
#endif // test
        break;
    }

    case 24:
    {
        auto r = varStackPop();
        auto l = varStackPop();
        std::ostringstream os;
        os << strStackPop();
        os << cmd("=", r, "", l) << std::endl;
        strStack.push(os.str());
        varStack.push(l);
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 27:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
```

```
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("or", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 29:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("and", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 31:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("==", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
```

```
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 32:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("!=", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 34:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("<", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
```

```
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 35:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd(">", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 36:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("<=", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
```

```
    }
    case 37:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd(">=", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 39:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        os << lstr << rstr;
        auto rvar = varStackPop();
        auto lvar = varStackPop();
        os << cmd("+", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 40:
    {
        auto rstr = strStackPop();
```

```
            auto lstr = strStackPop();
            std::ostringstream os;
            os << lstr << rstr;
            auto rvar = varStackPop();
            auto lvar = varStackPop();
            os << cmd("-", lvar, rvar, itoTemp(tempNum)) << std::endl;
            strStack.push(os.str());
            varStack.push(itoTemp(tempNum));
            tempNum++;
#ifdef test
            cout << "[case" << id << "] [str pop] " << endl;
            cout << "[case" << id << "] [str pop] " << endl;
            cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
            break;
        }
        case 42:
        {
            auto rstr = strStackPop();
            auto lstr = strStackPop();
            std::ostringstream os;
            os << lstr << rstr;
            auto rvar = varStackPop();
            auto lvar = varStackPop();
            os << cmd("*", lvar, rvar, itoTemp(tempNum)) << std::endl;
            strStack.push(os.str());
            varStack.push(itoTemp(tempNum));
            tempNum++;
#ifdef test
            cout << "[case" << id << "] [str pop] " << endl;
            cout << "[case" << id << "] [str pop] " << endl;
            cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
            break;
        }
        case 43:
        {
            auto rstr = strStackPop();
            auto lstr = strStackPop();
            std::ostringstream os;
            os << lstr << rstr;
            auto rvar = varStackPop();
```

```
        auto lvar = varStackPop();
        os << cmd("/", lvar, rvar, itoTemp(tempNum)) << std::endl;
        strStack.push(os.str());
        varStack.push(itoTemp(tempNum));
        tempNum++;
#ifdef test
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str pop] " << endl;
        cout << "[case" << id << "] [str push] " << os.str() << endl;
#endif // test
        break;
    }
    case 44: case 45: case 46: case 47:
    {
        strStack.push("");
#ifdef test
        cout << "[case" << id << "] [str push] []" << endl;
#endif // test
        break;
    }

    case 49:
    {
        auto rstr = strStackPop();
        auto lstr = strStackPop();
        std::ostringstream os;
        auto rvar = varStackPop();
        auto lvar = varStackPop();

        os << cmd(itoLabel(labelNum), "", "", "") << std::endl;
        os << lstr;
        os << cmd("J!=", lvar, "0", itoLabel(labelNum + 1)) <<
std::endl;
        os << cmd("J", "", "", itoLabel(labelNum + 2)) << std::endl;
        os << cmd(itoLabel(labelNum + 1), "", "", "") << std::endl;
        os << rstr;
        os << cmd("J", "", "", itoLabel(labelNum)) << std::endl;
        os << cmd(itoLabel(labelNum + 2), "", "", "") << std::endl;
        labelNum += 3;
        strStack.push(os.str());
        varStack.push(lvar);
        break;
```

```
}

case 52:
{
    auto rstr = strStackPop();
    auto lstr = strStackPop();
    std::ostringstream os;
    auto rvar = varStackPop();
    auto lvar = varStackPop();
    os << lstr;
    os << cmd("J!=", lvar, "0", itoLabel(labelNum)) << std::endl;
    os << cmd("J", "", "", itoLabel(labelNum + 1)) << std::endl;
    os << cmd(itoLabel(labelNum), "", "", "") << std::endl;;
    os << rstr;
    os << cmd(itoLabel(labelNum + 1), "", "", "") << std::endl;
    labelNum += 2;
    strStack.push(os.str());
    varStack.push(lvar);
    break;
}

case 53:
{
    auto rstr = strStackPop();
    auto midstr = strStackPop();
    auto lstr = strStackPop();

    std::ostringstream os;
    auto rvar = varStackPop();
    auto midvar = varStackPop();
    auto lvar = varStackPop();

    os << lstr;
    os << cmd("J!=", lvar, "0", itoLabel(labelNum)) << std::endl;
    os << cmd("J", "", "", itoLabel(labelNum + 1)) << std::endl;
    os << cmd(itoLabel(labelNum), "", "", "") << std::endl;;
    os << midstr;
    os << cmd("J", "", "", itoLabel(labelNum + 2)) << std::endl;
    os << cmd(itoLabel(labelNum + 1), "", "", "") << std::endl;
    os << rstr;
    os << cmd(itoLabel(labelNum + 2), "", "", "") << std::endl;
    labelNum += 3;
```

```
            strStack.push(os.str());
            varStack.push(lvar);
            break;
        }

        default:
            break;
        }
#ifdef test
        cout << "[/translate] " << id << "  varSize=[" << varStack.size()
<< "]";
        cout << "    strSize=[" << strStack.size() << "]" << endl;
        cout << endl;
#endif // test
        return 1;
}

// analyze the tokens with grammar
int Parser::analyse(const std::vector<Token>& tokens)
{
        tempNum = 0;
        labelNum = 0;
        build();

        //输出文件重定向
        ofstream fout;
        fout.open("SysStack.txt", ios::out);
        if (fout.is_open() == 0)
        {
            cout << "output open failed" << endl;
            exit(-1);
        }

        std::cout << std::endl;
        std::cout << "Symbol stack information:" << std::endl;

        //在此输出到文件
        fout << "Symbol stack information:" << std::endl;

        //system("pause");

        std::stack<std::pair<unsigned int, std::string>> st;
```

```
    st.push({ 0,"$" });
    auto iter = tokens.cbegin();
    //在这里输出程序运行中插入 token 的信息
    std::cout << "Inserted element:" << std::endl;
    std::cout << iter->getName() << " " << iter->getTypeOutput() <<
std::endl;

    //在此输出到文件
    fout << "Inserted element:" << std::endl;
    fout << iter->getName() << " " << iter->getTypeOutput() <<
std::endl;
    for (; ; )
    {
        auto I = st.top().first;
        std::string type;
        if (iter->getType() == TokenType::ID || iter->getType() ==
TokenType::CHAR
                || iter->getType() == TokenType::INT ||
iter->getType() == TokenType::FLOAT)
            type = TokenDict[iter->getType()];
        else
            type = iter->getName();

        if (action[I].find(type) != action[I].end())
        {
            auto act = action[I][type];
            if (act.first == "S")
            {
                if (iter->getType() == TokenType::ID ||
iter->getType() == TokenType::CHAR
                        || iter->getType() == TokenType::INT ||
iter->getType() == TokenType::FLOAT)
                {
                    varStack.push(iter->getName());
                    //cout<<"[pushVAR] "<<iter->getName() <<"   [at
line "<<iter->getLine()<<"]"<<endl;
                }
                st.push({ act.second, type });

                //在这里输出程序运行中栈的信息
                std::cout << "Stack Information:" << std::endl;
```

```
                for (std::stack<std::pair<unsigned int, std::string>>
dump = st; !dump.empty(); dump.pop())
                    std::cout << dump.top().first << " " <<
dump.top().second << '\n';
                std::cout << std::endl;

                //在此输出到文件
                fout << "Stack Information:" << std::endl;
                for (std::stack<std::pair<unsigned int, std::string>>
dump = st; !dump.empty(); dump.pop())
                    fout << dump.top().first << " " <<
dump.top().second << '\n';
                fout << std::endl;

                //system("pause");

                iter++;
                //在这里输出程序运行中插入 token 的信息
                std::cout << "Inserted element:" << std::endl;
                std::cout << iter->getName() << " " <<
iter->getTypeOutput() << std::endl;

                //在此输出到文件
                fout << "Inserted element:" << std::endl;
                fout << iter->getName() << " " <<
iter->getTypeOutput() << std::endl;
            }
            else if (act.first == "r")
            {
                auto id = act.second;
                auto right = grammar[id].getRight();
                for (unsigned int i = 0; i < right.size(); i++)
                    if (right[i] != "@")
                        st.pop();
                auto newI = st.top().first;

st.push({ go[newI][grammar[id].getLeft()],grammar[id].getLeft() });

                // translate
#ifdef test
                //cout<<"use production["<<id<<"]:
"<<grammar[id].getLeft()<<"->";
```

```
                //for(auto e:right)cout<<e<<" ";
                //cout<<endl;
#endif // test
                if (!translate(id, iter->getName()))
                {
                    std::cout << "ERROR! at line " << iter->getLine()
<< std::endl;
                    return iter->getLine();
                }

                /*
                //在这里输出程序运行中插入 token 的信息
                std::cout << "插入的元素：" << std::endl;
                std::cout << iter->getName() << " " <<
iter->getTypeOutput() << std::endl;
                //在这里输出程序运行中栈的信息
                std::cout << "栈中信息：" << std::endl;
                for (std::stack<std::pair<unsigned int, std::string>>
dump = st; !dump.empty(); dump.pop())
                    std::cout << dump.top().first << " " <<
dump.top().second << '\n';
                system("pause");
                */

            }

            else if (act.first == "acc")
            {
                //关闭 SysStack.txt 文件
                fout.close();

                ofstream fout;
                fout.open("Quaternion_information.txt", ios::out);
                if (fout.is_open() == 0)
                {
                    cout << "output open failed" << endl;
                    exit(-1);
                }

                std::cout << "Accept!" << std::endl;
                //输出到文件
                fout << "Accept!" << std::endl;
```

```
                //在这里输出最后的四元式
                std::cout << "Quaternion information:" << std::endl;
                std::cout << strStack.top() << std::endl;
                //输出到文件
                fout << "Quaternion information:" << std::endl;
                fout << strStack.top() << std::endl;
                fout.close();
                return 0;
            }
        }

        else
        {
            std::cout << "ERROR! at line " << iter->getLine() <<
std::endl;
            return iter->getLine();
        }
    }
    return 0;
}
```