目录

[一 编译原理基本理论 1](#_Toc41896436)

[1 编译程序的相关概念 1](#_Toc41896437)

[2 编译过程 1](#_Toc41896438)

[3 编译程序的结构 2](#_Toc41896439)

[二 编译程序设计原理概述 3](#_Toc41896440)

[1 程序语言相关概念 3](#_Toc41896441)

[1.1 字母表 3](#_Toc41896442)

[1.2 符号串 3](#_Toc41896443)

[1.3 文法 3](#_Toc41896444)

[2 词法分析 4](#_Toc41896445)

[3 语法分析 5](#_Toc41896446)

[4 语义分析和中间代码生成 6](#_Toc41896447)

[三 编译程序设计于实现 6](#_Toc41896448)

[1 语言规则的设计 6](#_Toc41896449)

[2 文法的设计 6](#_Toc41896450)

[3 词法分析的设计 7](#_Toc41896451)

[4 语法分析的设计 9](#_Toc41896452)

[5 语义分析和中间代码生成的设计 11](#_Toc41896453)

[四 **系统分析与测试** 14](#_Toc41896454)

[1 实验环境 14](#_Toc41896455)

[2 系统分析 14](#_Toc41896456)

[3 词法分析测试 14](#_Toc41896457)

[4 语法分析测试 19](#_Toc41896458)

[5 语义分析和中间代码生成测试 20](#_Toc41896459)

[6 错误管理测试 22](#_Toc41896460)

[**五** **总结与展望** 24](#_Toc41896461)

[六 **参考文献** 25](#_Toc41896462)

[七 **附录** 25](#_Toc41896463)

[八 **致谢** 25](#_Toc41896464)

# 编译原理基本理论

## 编译程序的相关概念

编译程序(Compiler, compiling program)，也称编译器，是指把高级程序设计语言书写的源程序，翻译成等价的机器语言格式目标程序的翻译程序。编译程序属于采用生成性实现途径实现的翻译程序。它以高级程序设计语言书写的源程序作为输入，而已汇编语言或机器语言表示的目标程序作为输出。编译出的目标程序通常还要经历运行阶段，以便在运行程序的支持下运行，加工初始数据，算出所需的计算结果。

作为一个具有实际应用价值的编译系统，除了基本功能外，还应具备语法检查，调试措施，修改手段，覆盖处理，目标程序优化，不同语言合用以及人-机联系等重要功能。

## 编译过程

编译程序把一个源程序翻译成目标程序的工作过程分为五个阶段：词法分析，语法分析，语义分析和中间代码生成，代码优化，目标代码生成。

1. 词法分析

词法分析的任务是对由字符组成的单词进行处理，从左至右逐个字符的对源程序进行扫描，产生一个个单词符号，把作为字符串的源程序改造成为单词符号串的中间程序。执行词法分析的程序成为词法分析程序或扫描器。

源程序中的单词符号经扫描器扫描，一般产生二元式(单词种别，单词的值)。

1. 语法分析

编译程序的语法分析器以单词符号作为输入，分析单词符号串是否形成符合语法规则的语法单位，如表达式，赋值，循环，声明，函数等，最后看是否构成一个合法的程序。语法分析分为自上而下分析法和自下而上分析法。自上而下分析法就是从文法的开始符号出发， 向下推到，推出句子。而自下而上分析法采用的是移进规约法，基本思想是：用一个寄存符号的先进后出栈，把输入符号一个一个的移进栈里，当栈顶形成某个产生式的一个候选式时，即把栈顶的这一部分规约成该产生式的左邻符号。

1. 语义分析和中间代码生成

语义分析和中间代码生成阶段的任务是：对语法分析所识别出的各类语法范畴，分析其含义，并进行初步翻译(产生中间代码)。这一阶段通常包括两个方面的工作。首先对每种语法范畴进行静态语义检查，例如，变量是否定义，类型是否正确等等。如果语义正确，则进行另一方面工作，即进行中间代码生成。中间代码是一种含义明确，便于处理的记号系统，它通常独立于具体的硬件。这种记号系统或者与现代计算机的指令形式有某种程度的接近，或者能够比较容易地把它变换成现代计算机地机器指令。

1. 代码优化

优化的任务在于对前段产生的中间代码进行加工变换，以期在最后阶段能产生出更为高效（省空间和时间）的目标代码。优化的主要方面有：公共子表达式的提取，循环优化，删除无用代码等等。有时，为了便于“并行计算”，还可以对代码进行并行化处理。优化所依循的原则是程序的等价变换规则。

1. 目标代码生成

目标代码生成阶段的任务是把中间代码变换成特定机器上的低级语言代码。这阶段实现了最后的翻译，它的工作有依赖于硬件系统结构和机器指令含义。这阶段工作十分复杂，涉及到硬件系统功能部件的运用，机器指令的选择，各种数据类型变量的存储空间分配，以及寄存器和后援寄存器的调度等。目标代码的形式可以是绝对指令或可重定位的指令代码或汇编指令代码。如目标代码是绝对指令代码，则这种目标代码可立即执行。如果目标代码是汇编指令代码，则需汇编器汇编后才能运行。必须指出，现代多数实用编辑程序所产生的目标代码都是一种可重定位的指令代码。这种目标代码在运行前必须借助一个连接装配程序把各个目标模块（包括系统提供的库模块）连接在一起，确定程序变量（或常数）在主存中的位置，装入内存中指定的起始地址，使之成为一个可以运行的绝对指令代码程序。

## 编译程序的结构

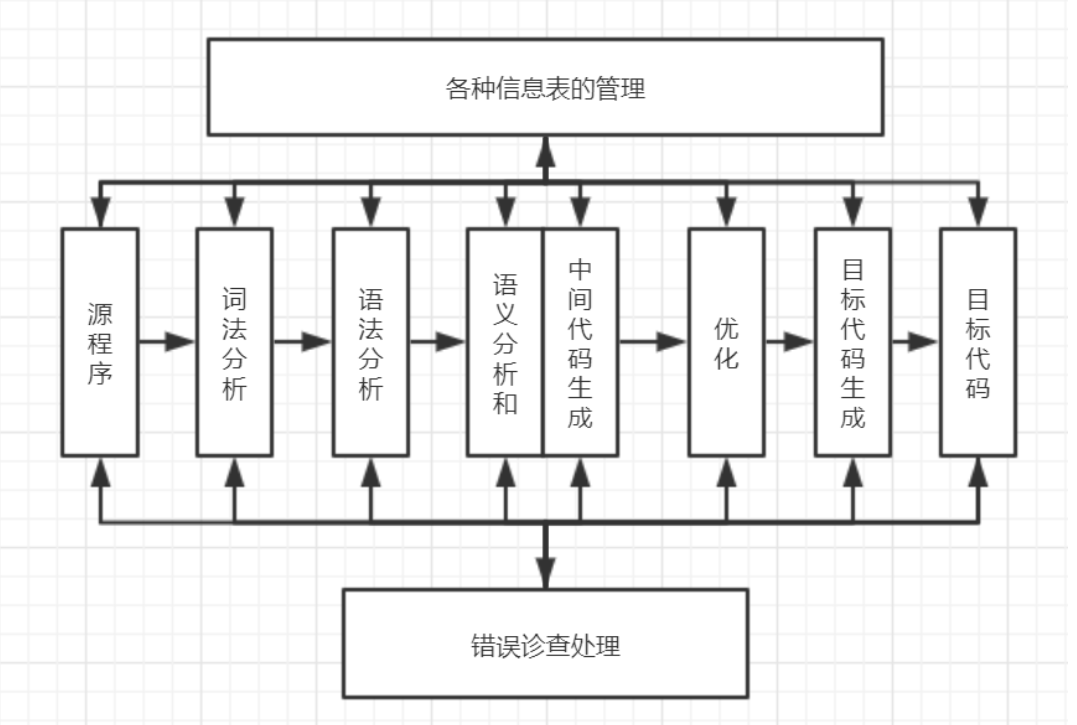


图1 编译程序总框

词法分析器，又称扫描器，输入源程序，进行词法分析，输出单词符号。

语法分析器，简称分析器，对单词符号串进行语法分析（根据语法规则进行推导或归约），识别出各类语法单位，最终判断输入串是否构成语法上正确的“程序”。

语义分析和中间代码生成器，按照语义规则对语法分析器归约出（或推导出）的语法单位进行语义分析并把它们翻译成一定形式的中间代码。

优化器，对中间代码进行优化处理。

目标代码生成器，把中间代码翻译成目标程序。

# 编译程序设计原理概述

## 程序语言相关概念

### 字母表

在介绍词法分析之前，首先需要介绍字母表。

字母表是符号的非空有穷集合，符号是一个抽象实体。任何语言都有自己的字母表，字母表是一个语言字母的全集。

例如C语言字母表：

∑={A~Z,a~z,0~9,+,-,\*,/,<,=,>,<=,>=,==,!=,\_,&,^,%,\*,(,),[,],{,},?,;,”,”,’,’,.,\,|,~,:}

（相邻符号使用,隔开，两端{}是集合括号）

一个C语言程序代码，一定由其字母表组成。

### 符号串

由字母表中符号组成的任何有穷序列，称为该字母表上的符号串，也可称为“字”或“句子”。

ε是∑上的一个符号串。

若a是∑上的符号串，而b是∑的元素，则ab是∑上的符号串。

w是∑上的符号串，当且仅当它由1和2导出。

一段语言代码就是其字母表上的符号串。

### 文法

文法用来描述语法规则，在编译程序中具有十分重要的作用，可以说是一个编译程序的灵魂。文法不仅定义了一个程序设计语言的语法规则，还贯穿编译程序的始终，在词法分析，语法分析，语义分析和中间代码生成过程中都需要使用文法。

一个文法G是一个四元组(Vt, Vn, S, P)，

其中：

Vt是一个非空有穷终结符号集合。

Vn是一个非空有穷非终结符号集合，且Vt∩Vn=Ø。

S∈Vn，是开始符号。

P是有穷非空的规则（产生式）集合，规则形如：A->a，其中A∈Vn，a∈(Vt∪Vn)\*，且开始符号S至少必须在P中的规则左部出现一次。

例如一个文法：

M->int H;

H->i

其含义为C语言中变量声明，如：int a;

## 词法分析

词法分析主要任务就是将一段源程序中不同种类的单词识别出来并分类。一个程序语言应当有以下单词符号：

1. 关键字，关键字是由程序语言定义的具有固定意义的标识符。有时称这些标识符为保留字或关键字。例如C语言中的if, while, for, int, double等。这些字不能作为一般标识符。
2. 标识符，标识符用来表示程序代码中各种变量，函数，过程等。
3. 常数，常数指程序中的各种数字，包括整型，浮点型，布尔型，字符串型等等，例如：1，100，3.4，“Hello World!”。
4. 运算符，运算符是程序代码中算术运算符，例如：+，-，\*，/等等。
5. 界符，界符是程序代码中表示一个部分结束的符号，例如：逗号，分号等。

词法分析是从左向右逐个字符对源程序进行扫描，产生一个单词符号。过滤空格，跳过注释，换行符等。词法分析一般输出一个二元式：

（单词种别，单词符号的属性值）

词法分析在编译程序中不是独立阶段，词法分析与语法分析在编译程序中是并列进行的：词法分析器读取字符，识别后交给语法分析器，再取下一个字符重复这一过程。

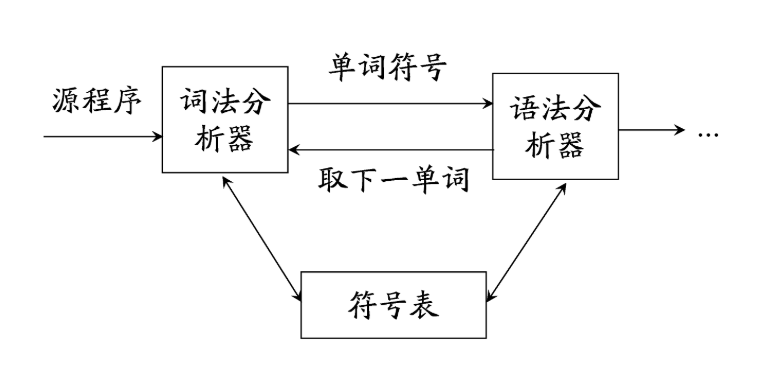


图2 词法分析器功能

## 语法分析

语法分析是编译过程的核心部分。它的任务是在词法分析识别出单词符号串的基础上，分析并判断程序的语法结构是否符合语法规则。语法分析器的工作本质是按文法的产生式，识别输入符号串是否为一个句子。从概念上来说，语法分析需要建立一颗与输入串匹配的语法分析树。根据不同的语法分析过程，对于语法分析树来说，一般分为自上而下分析法和自下而上分析法。

自上而下分析法是对于一个输入串，从文法开始符号（语法分析树的树根）出发，自上而下的推导出目标串。每一部推导都需要进行试探，直到最左子节点是以终结符为标志的子节点，才认为他们匹配，然后进行下一步。若没有找到匹配节点，则需要进行回溯，重新进行推导。由此观之，如果一条文法含有左递归，形如：

A->Aa

那么推导过程就会出现无限循环，于是，对于这种方法无法推导出该文法，于是我们需要消除文法中的左递归。

其次，由于推导的过程要不断回溯，导致语法分析效率极低，我们就需要设计尽量避免回溯的文法。

自下而上分析法是从输入串开始，逐步进行“归约”，直至归约到文法的开始符号，或者说，就是从语法分析树的叶子节点往上搜索，一直归约到根节点为止。

自下而上分析法是一种“移进-归约”法。这种方法的基本思想是，用一个寄存符号的栈，把输入符号一个一个地移进栈里，当栈顶形成某个产生式的一个候选式时，即把栈顶的这一部分归约为该产生式的左部符号。

由于相同的规则左部可以推出不同的规则右部，所以这种分析方法在栈顶归约时，需要找“可归约串”，否则就无法归约出开始符号。对于可归约串的不同，有不同的自下而上分析法。在算符优先分析中，我们用“最左素短语”来刻画“可归约串”，在“规范归约”中，则用“句柄”来刻画可归约串。于是，自下而上分析法面临的中心问题就是如何寻找判断可归约串。

## 语义分析和中间代码生成

在词法分析和语法分析之后，编译程序要做的就是进行静态语义检查和翻译。静态语义检查包括：

1. 类型检查。操作符需要作用于合法的操作数。
2. 控制流检查。控制流语句必须控制转移到合法的地方。例如，C语言中的break，应该跳出最近的一层循环，若不在循环内，应报错。
3. 一致性检查。例如标识符在其生命周期内只能出现一次。
4. 相关名字检查。

在语义分析过程中，需要生成中间代码。中间代码是源程序的中间表示方法之一。常见的有后缀式，三地址代码（三元式，四元式，间接三元式），DAG图等。

# 编译程序设计于实现

## 语言规则的设计

该编译程序编译的是一种类C语言。有两种数据类型，int和double。有一种循环控制，repeat until。有一种分支控制，if then else endif。整体上包含算术表达式，变量声明（不能在声明的时候给变量赋值），循环，分支，以及输入输出函数。

一些例子如下：

变量声明：int a; double a;

赋值：a := 1;

算术表达式：a := a + a \* a / a – a;

循环：repeat a := a + 1; until [ a > 5 ];

分支：if [ a>5 ] then a := a + 1; else a := a – 1; endif;

输入函数：read(a);

输出函数：print(a);

## 文法的设计

文法是一个编译程序最重要的部分，在编译程序的各种过程中都需要文法的参与。

在设计文法之前，首先要明确一个基本的程序设计语言由哪几部分组成。

1. Bool表达式
2. 算术表达式
3. 赋值表达式
4. 变量定于语句
5. 循环
6. 分支判断
7. 函数

其中，一个基本的程序设计语言是由赋值表达式，变量定义，循环，分支判断，函数组成的句子。

由上节类C语言的语法规则，设计文法如下（已处理为LL（1）文法）：

1. 整体文法，其中非终结符分别代表定义，赋值，判断，循环，函数语句(未按顺序)

S->Q Q->KQ’ Q->MQ’ Q->NQ’ Q->OQ’ Q->PQ’ Q’->Q Q’->@

2. 算术表达式(其中i表示变量，n表示数字)

A->BA’ A'->+A A'->-A A’->@ B->CB’ B'->\*B B'->/B B'->@

C->(A) C->i C->n

3. 赋值(其中A为算术表达式，意为a := a + b，左边为变量，右边为算术表达式或变量或数字)

K->F:=Z; F->i F->n Z->A

4. Bool表达式(由于设计文法时遇到一些Follow集合的问题，bool表达式需用[]括起来)

L->[GR] R-><G R->>G R-><=G R->>=G R->=G R->!=G

G->A

5. 声明

M->intH; M->doubleH; H->i

6. 循环(I为合法语句，T为bool表达式)

N->repeatIuntilT; T->L I->Q

7. 判断(J为bool表达式，U和Q为合法语句)

O->ifJthenUelseQendif; J->L U->Q

8. 函数(read(a),print(a))

P->read(X); P->print(X); X->YX’ X'->,YX’ X’->@ Y->i

## 词法分析的设计

采用DFA状态转换的方法进行词法分析器的设计。

该语言的语言记号如下表：

表1 语言记号

|  |  |
| --- | --- |
| 保留字 | 特殊符号 |
| if | + |
| then | - |
| else | \* |
| repeat | / |
| until | ( |
| read | ) |
| print | := |
| endif | = |
| int | < |
| double | > |
|  | ; |
|  | { |
|  | } |

设计的DFA如下：

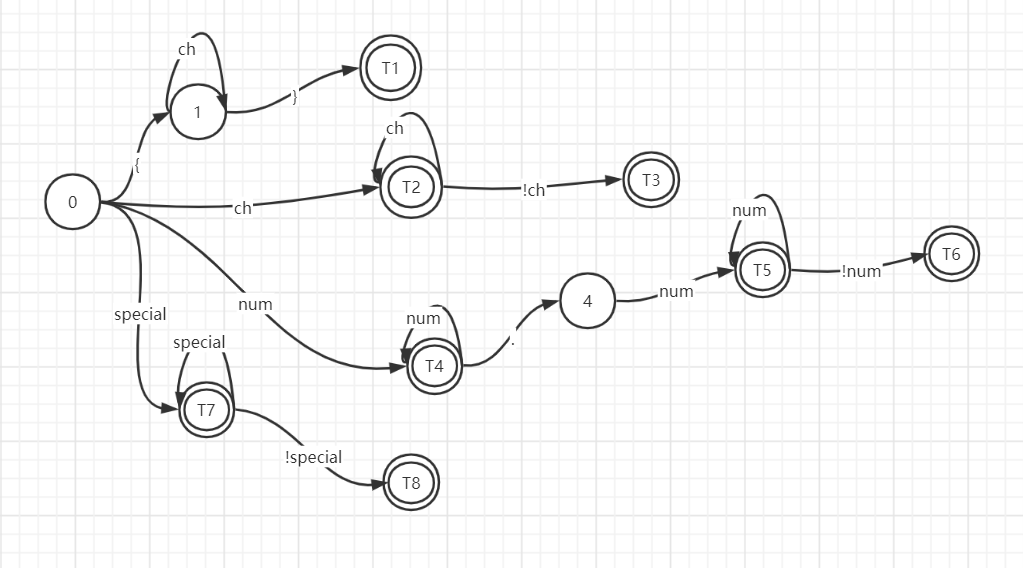


图3 词法分析器DFA

其中：

State 0: 初始状态

State 1，T1: 注释

State T2,T3: 字母符号,在状态中判断是否为保留字，或者是变量名。

State T7，T8: 特殊符号，如运算符，结束符。

State 4，T4，T5，T6: 数字

ch表示字母字符，special表示特殊字符，num表示数字

该词法分析器的工作流程是，每次读取一个字符，放入DFA中进行状态转换，当状态转移到终态的时候就输出一个二元式，表示该单词分析结束。不断读取字符，直到末尾，重复上述过程。

## 语法分析的设计

采用LL(1)分析法进行语法分析器的设计。

LL(1)分析法的需要首先计算出文法的FIRST集合和FOLLOW集合，然后根据FIRST集合和FOLLOW集合构造出LL(1)预测分析表，再使用下推自动机进行语法分析，语法分析的结果是一颗语法树。

FIRST 集合是由一个产生式所能推导出的所有产生式的第一个终结符所组成的集合，设X为一个文法符号，对于 FIRST 集合的计算存在以下几种情况：

1. 若 XÎVT ，即 X是一个终结符，则FIRST(X) = {X}。
2. 若 XÎVN ，即 X是一个非终结符，且有产生式 X->a…，则aÎFIRST(X)。
3. 若 XÎVN ，且有产生式 X->e，则有eÎFIRST(X)。
4. 若 XÎVN ，Y1 ,Y2 ,...Yn ÎVN ，且有产生式 X->Y1 ,Y2…Yn ，那么当存在eÎFIRST(Yk )(1<= k<=n)时，有FIRST(Yk)-{e} Ì FIRST(X)。

由上述规则，可采用以下方法求解文法的FIRST集合。

首先，把文法右部或关系的推导拆成独立产生式。

如：A->B|C变为A->B A->C

然后将文法左部选为根节点，右部为儿子建成一棵树。所有产生式应当构造出一个森林。

如文法A->BCDE B->FGH G->IJK K->L可构造成如下形式。

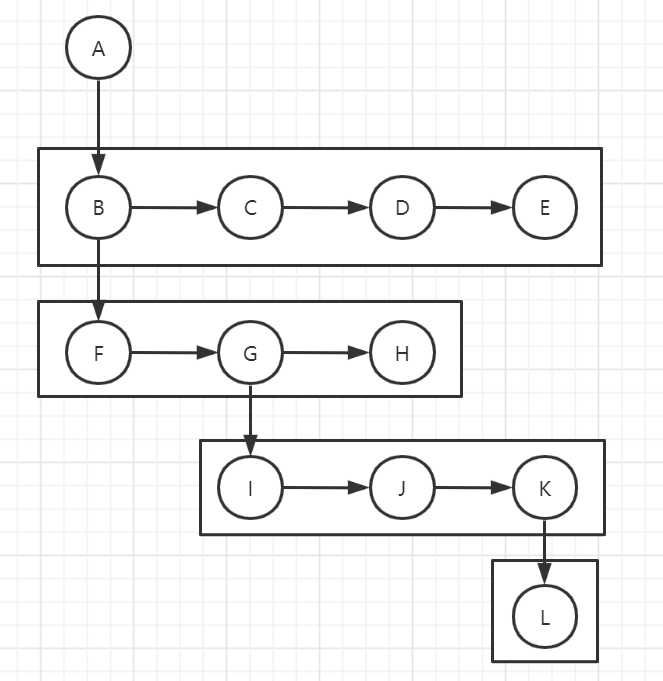


图 4 文法构造出的一棵树

其中带框部分为产生式右部的所有符号，他们从左往右顺次连接。带框部分为一个整体，一个产生式的左部只会和这一整体连接起来。

于是，该树满足递归性质。当被求字符为A时，要遍历其儿子BCDE，B又是一个递归问题，可从FGH中求解，不断通过深度优先搜索的方式就可以求出A的FIRST集合。

若符号A为一个非终结符，FOLLOW后跟符号集合是由句型中紧跟A后的那些终结符组成，记作FOLLOW(A) ：

1. 设 A为文法中的开始符号，把｛$｝加入FOLLOW(A) 中，其中“$”为文法结束符，它是一个终结符，保证了超前搜索算法的一致性；
2. 若 *A->*aBb是一个产生式，则把FOLLOW(b) 的非空元素加到FOLLOW(B) 中。如果*b*-> e，那么原产生式可表示为 *A->*aB，则把FOLLOW(A) 也加入到FOLLOW(B) 中。
3. 反复使用(2)直到每个非终结符的 FOLLOW 集合不再增大为止。

求解FOLLOW集合相较求解FIRST集合简单。

遍历所有规则右边，找当前求解符号，对后面的符号的FIRST进行判断，若后面的FIRST全部含有ε的情况，对规则左边dfs求解即可。

预测分析表的构造需要FIRST集合和FOLLOW集合的参与。

其构造流程如下：

1. 对文法G的每个产生式A->a执行2，3。
2. 对每个终结符xÎFIRST(a),把A->a加入M[A,a]中。
3. 若a最终可推出空，则对任何bÎFOLLOW(A)，把A->a加入M[A,b]中。
4. 把所有无定义的M[A,a]标上错误标记。

构造出预测分析表，就可以使用下推自动机进行语法分析。

下推自动机的工作流程是：

1. 首先向分析栈插入$，表示终止符，再插入文法开始符号。在输入流最后也插入一个$。
2. 读指针读取词法分析输出的输入流。当分析栈推导出该指针指向的终结符后才继续移动。
3. 分析栈栈顶读取预测分析表的第一维，读指针指向终结符为预测分析表的第二维。每一步将栈顶弹出，将预测分析表中的产生式右部逆序插入分析栈。
4. 当分析栈栈顶和读指针都指向$时，语法分析结束。

其模型如下图：

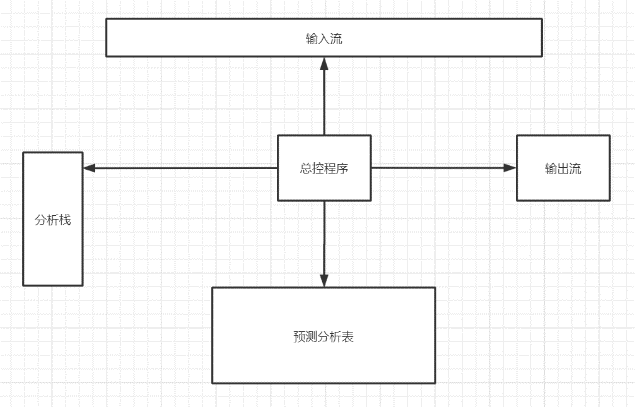


图 5 下推自动机模型

## 语义分析和中间代码生成的设计

语义分析和中间代码生成也需要文法的参与，不仅如此，文法需要扩充为属性文法，才可以完成中间代码生成。

由于在推导的过程中，运算符被当作终结符，一旦推导出来就立刻弹出栈，想要仅仅通过普通的文法进行语义分析和中间代码生成就变得尤为困难。

本编译程序采用一种属性文法（将运算符，保留字当作动作，扩充原有文法）来进行语义分析。

对于算术表达式，将其文法扩充为如下形式：

A->BA' %ADD\_SUB

A'->+A %ADD

A'->-A %SUB

A'->@

B->CB' %MUL\_DIV

B'->\*B %MUL

B'->/B %DIV

B'->@

C->(A) %TRAN\_AC

C->%ASS\_C i

C->%ASS\_C n

其中，ADD SUB MUL DIV意为加减乘除，TRAN\_AC意为将A的值赋给C，ASS\_X意为将产生式右部的终结符放入语义栈的同时，将其赋值给X。

对于赋值表达式，其文法扩充为如下形式：

K->F:=Z%EQ;

F->%ASS\_F i

F->%ASS\_F n

Z->A

其中，EQ意为判断将Z赋值给F

对于bool表达式，其文法扩充为如下形式：

L->[GR] %COMP

R-><G %COMP\_L

R->>G %COMP\_G

R-><=G %COMP\_LE

R->>=G %COMP\_GE

R->=G %COMP\_E

R->!=G %COMP\_NE

G->A

其中COMP\_X意为比较运算符的各种形式。

对于变量声明语句，其文法扩充为如下形式：

M->intH%INT;

M->doubleH%DOUBLE;

H->%ASS\_H i

对于循环语句，其文法扩充为以下形式：

N->repeat%REIuntilT%RE\_BACKPATCH;

T->L

I->Q

其中RE表示循环的开始标记，RE\_BACKPATCH为回填动作。

对于分支判断语句，其文法扩充为以下形式：

O->ifJ%IF\_FJthenU%IF\_FJ\_BACKPATCH%IF\_RJelseQ%IF\_RJ\_BACKPATCHendif;

J->L

U->Q

其中IF\_FJ为FALSE JUMP，即if中bool值为假时的跳转。IF\_FJ\_BACKPATCH为IF\_FJ的回填动作。IF\_RJ为RIGHT JUMP，即跳转到else，IF\_RJ\_BACKPATCH为IF\_RJ的回填动作。

当语法分析栈的栈顶遇到动作符，就需要对语义栈进行一些动作操作，用来正确生成四元式。具体例如如下：

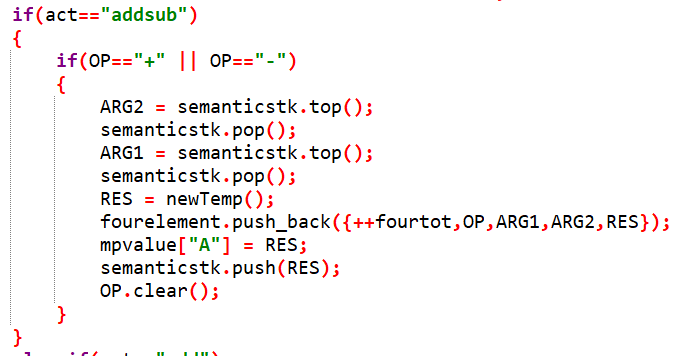


图6 中间代码生成中算术加减的代码

当分析栈栈顶为ADD\_SUB时，首先判断当前的操作符是不是加或减。然后从语义栈中弹出顶部两个结果，其就是算术运算的两个操作数，然后生成一个新的结果变量，并且将运算符，两个操作数和新的结果变量放入四元式中，然后将结果插入语义栈。

其他动作的过程十分类似。

# **系统分析与测试**

## 实验环境

本文的实验环境为Windows 10 Home 1903操作系统，代码开发工具为Dev C++，代码调试环境为Microsoft Visual Studio 2019，采用C++11规范编写，使用STL标准模板库。

## 系统分析

GCC是编译器相关产品中的标杆，其发展至今，已经十分完善，代码体系相当庞大，对程序代码的优化效果十分好。与GCC相比，本文的编译程序功能有限，只能实现基本的词法分析，语法分析和中间代码生成。本文编译程序也没有按照工程的要求来进行编写，代码比较冗余且较难以理解，可维护性一般。存储数据的数据结构也是使用的STL，运行效率较差。总体来说，从代码结构上，代码编写上，可维护性上，数据结构的选择和编写上都有很大的提升空间。

## 词法分析测试

词法分析器将源程序的单词分类，过滤多余空格和注释，其输出结果是一个二元式，并且是语法分析器的输入。图7是词法分析器的测试用例，图8是词法分析器的测试输出。

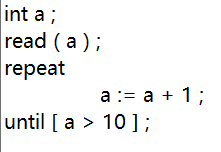


图7 词法分析器测试用例

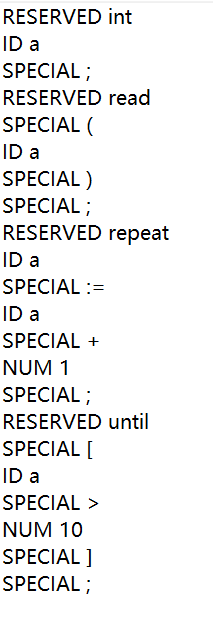


图8 词法分析器输出

## 语法分析测试

语法分析器的输入是词法分析器的输出。该编译程序不仅可以输出语法推导过程，首先可以输出FIRST集合和FOLLOW集合，然后输出预测分析表。之后可以进行语法分析。由于FIRST集合FOLLOW集合和预测分析表输出过于庞大，本文不做展示。图9为词法分析器测试用例的语法推导过程。

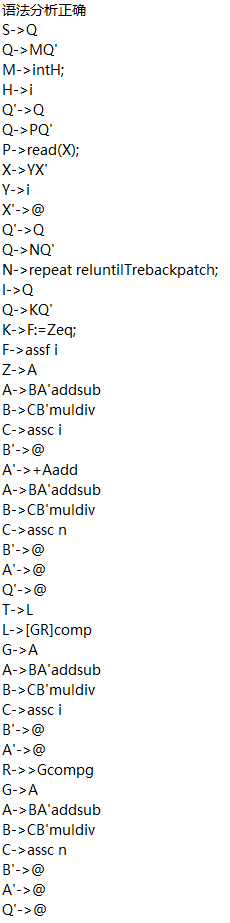


图9 语法分析推导过程

## 语义分析和中间代码生成测试

中间代码生成可以输出源程序对应的四元式，其格式如下：

（操作符，操作数1，操作数2，结果）

首先定义本文程序四元式中的单词标志。

1. RP。RP表示repeat的入口，其动作为直接运行下一句。
2. FJ。FJ表示错误跳转，即条件为假的情况下跳转到结果序号。
3. RJ。RJ表示正确跳转，即条件为真的情况下跳转到结果序号。

图10为中间代码生成的测试用例，图11为中间代码生成的输出结果。

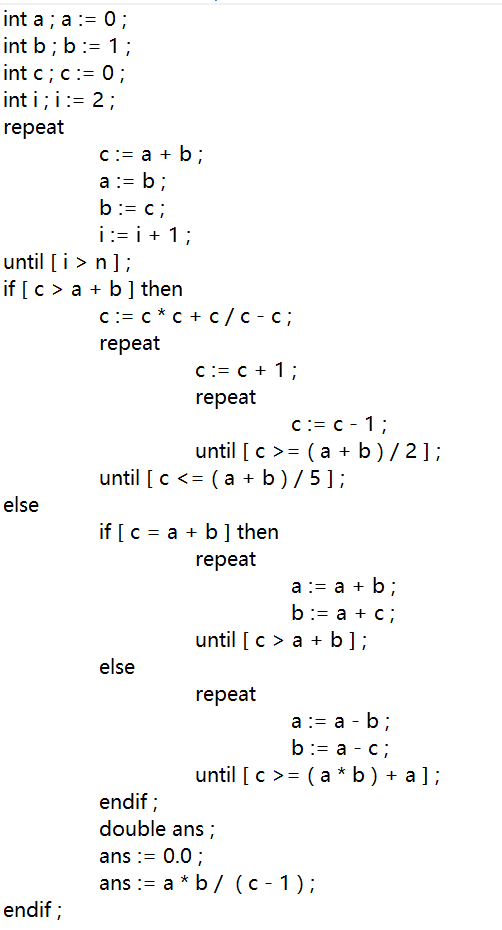


图10 中间代码生成测试用例

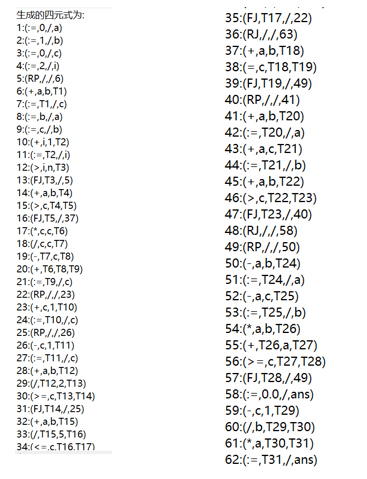


图11 中间代码生成输出结果

通过测试用例和输出可以看出，中间代码可以正常输出，且与源程序匹配。

## 错误管理测试

发现源程序中的错误，定位源程序中错误位置，发现源程序错误原因，是一个编译程序最基本的功能，如果一个编译程序只能报告错误，不能报告错误位置以及错误原因，不仅仅对于编程初学者有巨大的影响，严重打击信心，对于经常编程的人来说也难以快速找到问题。

本文的编译程序可以找到错误，并且发现错误原因。

本文只是选取少量错误作为测试用例，其他所有错误都可以报告。

1. 缺少右括号

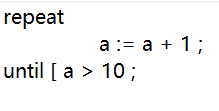


图12 源程序的错误样例

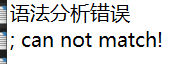


图13 源程序的错误检测结果

1. 缺少分号

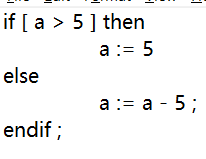


图14 源程序的错误样例2

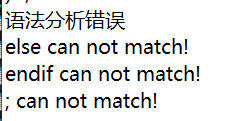


图15 源程序的错误检测2

# **总结与展望**

编译是人类语言与机器语言之间连接的桥梁，没有编译程序，人类基本是无法进行编程的。编译技术作为计算机科学领域中一个十分重要的部分，涉及的编程基础知识和技能很广，从存储各种数据的数据结构，到实现完整功能的算法，再到全部整合起来的工程项目，无一不需要开发人员具有极强的功底。其中，将复杂的程序设计语言抽象成文法，将复杂的待解决问题抽象成数学问题，都是十分困难的，编译程序是人类思想的精华。如果可以更深刻的了解编译，无论是对学生还是开发人员，都会大有裨益。

本文简略的阐述了一个不完整的编译程序的实现过程，主要包括：

1. 文法的设计，左递归的消除，回溯的避免，左因子的提取。
2. 词法分析器的设计与实现。给出了词法分析器的有穷自动状态机并介绍了其工作流程。
3. 语法分析器的设计与实现。给出了自上而下分析法和自下而上分析法的理论过程，给出了自上而下分析法的LL(1)分析法的实现过程。
4. 语义分析和中间代码生成部分的设计与实现。给出了属性文法的工作方式，将文法扩展为属性文法并且简单的介绍了其工作流程。
5. 最后针对本文的系统进行了一些简单的测试，基本各种结果基本符合预期。

本文首先改进了一种类C语言，并且针对该语言设计了其文法。然后针对该语言设计了可以实现词法分析，语法分析和中间代码生成的简单的编译程序。能够支持int, double数据类型，可以支持分支判断，循环两种控制语句。可以进行算术表达式的运算与赋值。

但是，由于时间和个人能力有限，本文的编译程序还有很多不足之处。后续的研究工作主要有以下几个方面：

1. 数据结构的优化。使用更为高效的数据结构对数据进行存储，提高编译程序的执行速度。
2. 可维护性。将各个部分封装成类，并且提供给维护人员接口，以最少的代码修改量进行良好的代码维护。
3. 编译的后续阶段。本文仅仅只实现了编译程序的一小部分，对于编译程序的后续部分完全没有涉及。下一步对编译程序的后续阶段进行研究。
4. 良好的错误管理。本文只是实现了简单的错误管理功能，下一步需要设计更简单方便地错误管理功能。

# **参考文献**

[1] 陈火旺, 刘春林等. 程序设计语言编译原理[M]. 国防工业出版社,2000.

[2] xiangxianzhang. 基于Java的C语言编译器. [https://github.com/xiangxianzhang/Compiler 2020年5月17](https://github.com/xiangxianzhang/Compiler%202020年5月17)日访问.

[3] 高云云. C语言编译系统的研究与实现[D].南京邮电大学,2019.

[4] 王秉睿. 神经网络专用编程语言[D].中国科学技术大学,2019.

[5] 徐金光. C编译器语法分析的设计与实现[D].电子科技大学,2016.

# **附录**

checkError.cpp

FIRST.cpp

FOLLOW.cpp

WORDANALYSE.cpp

wordtest.txt

output.txt

result.txt

tbbs.txt

init.txt

Action.txt

# **致谢**

一学期的课程将要进入尾声。虽然2020年春季学期的一半是在家中远程教学，但是依然没有阻止我们学生的学习，也没有阻止我国和全世界人民共同抗疫的决心。

在短短的几个月的学习过程中，我从一个只会用编译器的学生成长为一个了解了编译程序简单原理的学生，这离不开我的老师郑关胜老师的教学和指导。郑关胜老师不仅是我编译原理课程的任课老师，还是我校ACM集训队的教练。我在郑关胜老师的教导下度过了大学的三年时间，这三年时间我不仅仅学会了一些专业知识，老师的认真治学的态度也深刻感染了我，使我从一个未经世事，见识短浅的孩子成长为一个还算有点理想抱负的成年人。我很庆幸我在大学初期就加入了校ACM集训队，在那里，每个同伴都胸怀壮志。感谢学长学姐们不论在学校还是在寒暑假中无私的分享，感谢老师在休息时间指导我们训练和学习。我在集训队度过了快乐的，最有意义的三年。

感谢唐俊同学在程序设计时提出的宝贵建议。唐俊同学不仅是我的队友，也是我的挚友。感谢唐俊同学在困难时无私帮助我，在生活中对我的关心。十分感谢。

感谢我的同学们给我的关心和鼓励，和你们相聚大学校园是一件十分美好的事情，感谢你们。

还要感谢我的父母，你们的支持是我最大的动力。

最后，我要感谢每一个帮助过我的人，是你们无私的帮助让我茁壮成长。