# 编译原理实践

为了将高级语言程序翻译成目标程序，编译程序首先必须对高级语言源程序进行分析，然后产生目标程序。因此，编译程序分成前后两个阶段：分析阶段和综合阶段。分析阶段根据源语言的定义，分析源程序的结构，检查源程序是否符合语言的规定，并以某种中间代码的形式表示出来。词法分析、语法分析、语义分析和中间代码生成都属于分析阶段。综合阶段跟据分析结果构造所要求的目标代码程序，包括代码优化和目标代码生成。为了记录分析过程中识别出的标识符及有关信息，还需要使用符号表。如果在编译过程中发现源程序有错误，不仅要报告错误，还要进行错误处理，使编译能继续进行。基本的编译程序模型如下图：

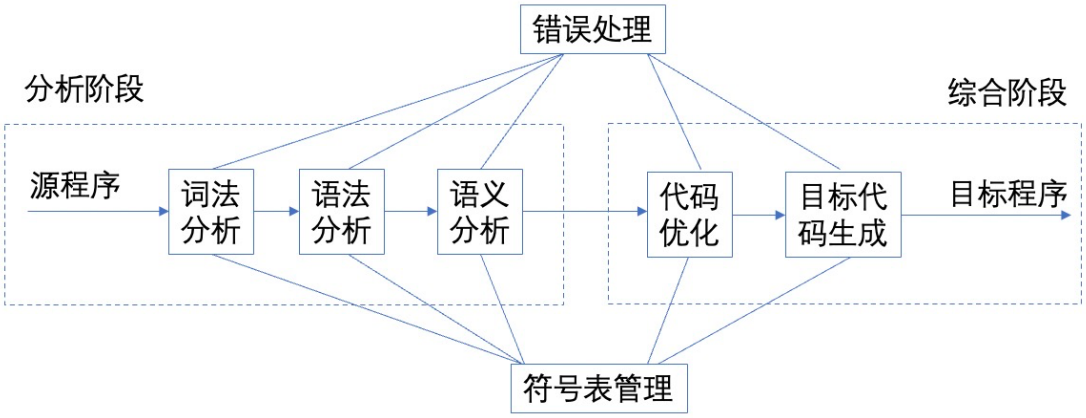


图1. 编译器基本架构

## 1 TEST 语言与编译器

本次实践将设计一种非常小的语言 TEST，一旦能明白 TEST 语言的编译技术，就能够更容易地理解各种语言的编译器了。

## 1.1 TEST 语言

TEST 语言的程序结构很简单，它在语法上相当于 C 的函数体，即由一对大括号括起来的语句序列，没有过程或函数。声明语句、表达式语句以及控制语句的写法都与 C 语言类似，但规定：一条声明语句只能声明一个整型简单变量，没有数组；控制语句只是 if、while 和 for 三个语句，这三个控制语句本身也可包含语句序列；表达式仅限于布尔表达式和整型算术表达式，布尔表达式由对两个算术表达式的比较组成，该比较实用<、<=、>、>=、==和!=比较算符；算术表达式可以包括整型常量、变量以及+、-、*、/这 4 个运算符。另外，还可以有复合语句。为了能实现输入输出，又添加了 read 语句和 write 语句。TEST 语言还可以有注释，注释用/*和*/括起来，但注释不能嵌套。

例如，下列程序就是采用 TEST 语言编制的计算阶乘的程序。

```
{
        int i;
        int n;
        int j;
        j=1;
        read n;
        for(i=1; i<=n; i=i+1)
                j=j*i;
        write j;
}
```

虽然 TEST 缺少真正的程序设计语言所需要的许多特征，但它足以用来体现

编译器的主要特征。

## 1.2 TEST 编译器

TEST 编译器包括以下的 C 文件：

TESTmain.c: 主程序, 先后调用词法分析、语法分析及语义分析和代码生成。

TESTscan.c: 词法分析，接受用 TEST 语言编写的程序，输出的单词符号程序将作为语法分析的输入。

TESTparse.c: 语法、语义分析及 TEST 机的汇编代码生成，如果分析中发现有错误，则报告错误。

## 1.3 TEST 机

我们将用一个抽象机的汇编语言作为 TEST 编译器的目标语言。TEST 机的指令仅能作为 TEST 机的目标。实际上, TEST 机具有精简指令集计算机的一些特性。TEST 机的模拟程序直接从一个文件中读取汇编代码并执行它，因此避免了由汇编语言翻译为机器代码的过程。此外，为了避免与外部的输入输出例程连接的复杂性，TEST 机有内部整型的 I/O 设备；在模拟时，它们都对标准设备读写。

# 2. TEST 语言的词法分析

## 2.1 TEST 语言的词法规则及状态图

TEST 语言的所有变量都是整型变量，具有 if、while、for 等控制语句。注释用/*和*/括起来，但注释不能嵌套。TEST 的表达式局限于布尔表达式和算术表达

式。

TEST 语言的单词符号有：

标识符：字母开头,后接字母或数字。

保留字(它是标识符的子集)：if、else、for、 while、do、int。

无符号整数：由 0~9 数字组成。

分界符 :如+、-、*、/、(、)、;、, 等单分界符 ;还有双字符分界符>=、<=、!=、==等。

注释符：用/*和*/括起来。

词法分析程序并不输出注释,在词法分析阶段注释的内容将被删掉。为了从源程序字符流中正确识别出各类单词符号, 相邻的标识符整数或保留字之间至少要用一个空格分开。

TEST 语言的各类单词符号的正则表达式如下:

<identifier>→<letter>|<identifier><letter>I<identifier><digit>

<number>→< digit>|<number><digit>

<letter>→a|b|..|z|A|B|..|Z

<digit>→1|2|…|9|0

<singleword>→+|-|*|/|=|(|)|{|}|:|,|;<|>|!

<doubleword>→>=|<=|!=|==

<comment first>→/*

<commentlast>→*/

由于部分单分界符与双分界符或注释的头符号相同,如>、<、=、和/,所以这些单分界符要在双分界符或注释中处理。考虑到词法分析程序作为一个子程序能用来识别和分析各类单词符号,并且在识别一个单词后将返回调用程序,因此,可将上述状态图合并为一张状态图,具有共同的入口初始状态 S,在识别出一个单词后有一个共同的出口。某些单分界符和双分界符的第一个符号相同,符号 “/” 和注释的第一个符号也相同,因此必须做特殊处理。另外,除上述单词符号规则定义的符号外,如果源程序中出现有其他符号,则认为是非法符号,因此,状态图中还添加了错误状态。当分析处在 S 状态,扫描的字符不属于从 S 射出的弧上标记的字符集时,就进入错误状态。根据以上考虑，词法分析的状态图如图 2 所示。

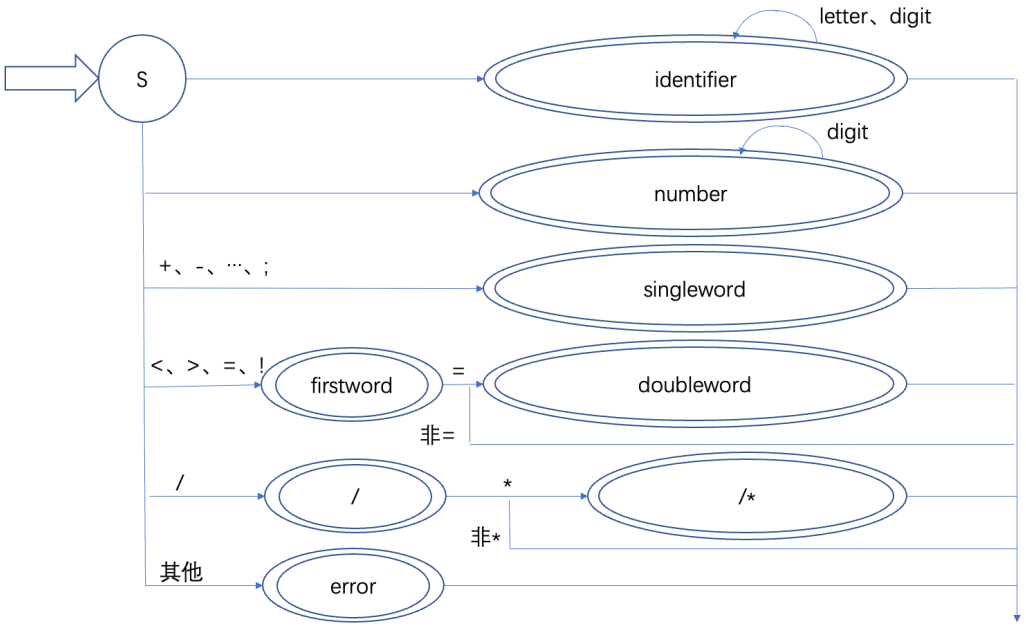


图 2. 单词符号的状态图

## 2.2 TEST 语言词法分析程序的构造

有了状态图后，根据分析的相应动作就可以构造出词法分析的算法流程图,如图 3 所示。

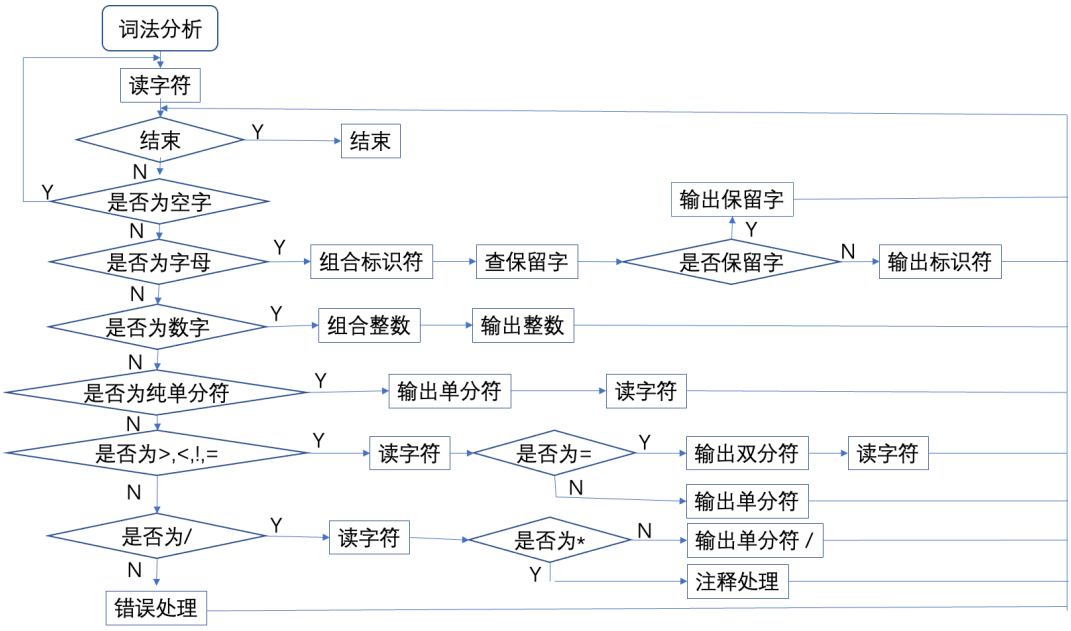


图 3. 词法分析程序流程图

在程序开始时,首先读入一个字符,若为空字符,则继续读,直到读进一个非空字符。读进的字符有如下 6 种情况,要进行不同的处理。

(1)字母。继续读,直到遇见空格、分界符、非字母数字字符或到文件尾。组合标识符,查保留字表。若为保留字,输出相应单词记号;若不是,输出标识符的单词记号及单词值。

(2)数字。继续读,直到遇见空格或非数字字符出现或到文件尾。输出无符号整数的单词记号及数字串。

(3)非=、<、>、等与双分界符首字符不同的单分界字符。输出相应单词记号及单分界符。

(4)=、<、>、!。读下一个字符,判断是否为双字符分界符,若是,组成双字符分界符,输出相应单词记号及双分界符;若不是,输出单分界符记号。

(5)/。读下一个字符。若不是*,输出/的单词记号;若是*,进行注释处理。词法分析不输出 “/*” ,并要跳过整个注释内容直到遇到 “*/” 为止,然后返回开始状

态,继续识别下一个单词符号。

(6)非法字符。如果读进的字符不属于上而任一情况,则说明词法分析程序从源程序读人了一个不合法的字符,即该字符不属于程序语言所定义的所有单词符号的首字符集合。词法分析程序在遇到不合法字符时要进行错误处理,报告错误信息,跳过这个字

## 2.3 TEST 语言的词法分析程序实现

1.输出形式

为了使词法分析的输出含义明确、易于理解本程序对识别出的每个单词符号输出两项内容:一是单词记号,二是单词值。对于保留字、分界符直接输出相同的字符串作为单词记号,此时,单词记号与单词值相同,为了统一输出形式,两者一同输出。对于标识符,其单词记号用 “ID” 表示,单词值是实际标识符。对于无符号整数,单词记号用 “NUM” 表示,具体的无符号整数以数字字符串形式作为单词值输出。

对于如下的一段 TEST 语言程序:

```
{
    int a;
    a=100;
}
```

词法分析识别出{后,输出 “{ {” ; 识别出 int,输出 “int int” ;而识别出标识符 a 后,应输出 “ID a” ;识别出无符号整数 100 后,应输出 “NUM 100” 。

2.词法分析程序

该词法分析程序要采用 C 语言编制,程序中的变量及有关过程均在程序注释中说明为了便于与后面的语法分析及语义分析相连接,在此将词法分析程序设置为一个函数,名为 TESTscan(),而保留字表用一个字符指针数组保存,名为 keyword[keywordSum]。

在这个词法分析程序中,为了简化程序设计,使初学者能尽快掌握词法分析的设计,程序中使用的语句、设计思想及实现方法都比较简单,极容易实现。程序中将保留字保存在一个指针数组中,可随时增加和修改。分界符有单分界符和双分界符之分。对于那些和双分界符或其他符号没有冲突的单分界符将它们连接起来,作为一个字符串并保存在字符数组 singleword 中,这样,通过判断读入的字符是否是 singleword 中的字符,就可识别出单分界符。双分界符需要单独处理,本程序处理的双分界符第二个字符都相同,所以可将所有双分界符的第一个符号连接成字符串并保存在字符数组 doubleword 中,当判断字符是 doubleword 中的字符时,要再读入一个字符。如果这个字符是=,则识别出双分界符;如果不是则输出相应的单分界符。

对于注释和单分界符/要特别处理。当读入的字符是/时,要读入下一个字符,如果不是*,则输出单分界符/;如果是*,则进行注释处理。处理方法是:增加一个新变量 ch1,ch 保存前一个字符,ch1 为新读入的字符。如果 ch 是*且 ch1 是/,则遇到注释尾,否则令 ch=ch1,ch1 则读下一个字符,再判断,直到遇到注释尾,这样就可将注释去掉。如果还想加入其他的双分界符甚至三分界符,就需要单独处理。不过,掌握了处理双分界符的方法,处理其他形式的分界符应该不再是难事。

本词法析程序处理的保留字不区分大小写,但标识符区分大小写,如果想不区分大小写,可在组合标识符结束后统一转换成大写或小写字母即可。词法分析函

数返回值为整型,当返回值为 0 时表示词法分析没有发现错误，返回值为 1、2 时表示输入或输出文件有错误，返回值为 3 时表示有非法符号。

该词法分析程序处理的源程序文件名和输出文件名保存在字符数组 Scanin 和 Scanout 中,执行词法分析程序前,首先要建立词法分析的源程序。并在运行词法分析程序时按提示输入源程序文件名和输出文件名。如果输入或输出文件与词法分析程序在同一目录下,直接输入文件名即可,否则应输入包含路径的文件名。词法分析程序运行后,屏幕提示词法分析是否成功,然后可用任意文本编辑器打开输出文件查看词法分析结果。

【例 3-3】假设输入文件 AAA.T 中程序如下，运行词法分析程序，给出输出结果。

```
{
    int a;
    a=100;
}
```

要求运行结束后,屏幕提示词法分析成功，并将分析结果打印在屏幕和文件 BBB.T 中。词法分析结果如下:

```
{                    {
int                  int
ID                   a
;                    ;
ID                   a
=                    =
```

NUM 100

; ;

} }

其中，第一列是单词记号，第二列是单词值。

词法分析程序可以按如下开始(也可以按自己的方式开始)：

```
#include <stdio.h>
#include <ctype.h>
//下面定义保留字表，为简化程序，实用字符指针数组保存所有保留字
//如果想增加保留字，可继续添加，并修改保留字数目 keywordSum
#define keywordSum 10
char * keyword[keywordSum]={``if``,``else``, ``for``, ``while``, ``do``, ``int``,
``read``, ``write``};
//下面定义纯单分界符
char singleword[50] = ``+-*(){};,:``;
//下面定义双分界符的首字符
char doubleword[10]= ``><=!``;
extern char Scanin[300], Scanout[300];
//用于接收输入输出文件名，在 test_main.c 中定义
int TESTscan()
{ … }
```

主程序可以如下：

```
#include <stdio.h>
#include <ctype.h>
extern int TESTscan();
char Scanin[300], Scanout[300];
FILE * fin, *fout;
void main() {
    int es = 0;
    es = TESTscan();
    if(es > 0) printf( “词法分析有误，编译停止！” );
    else printf( “词法分析成功！\n” );
}
```