**XXXX大学**

**《编译原理》实验**

**Mini Pascal设计报告**

|  |  |
| --- | --- |
| 学 院： | 计算机学院 |
| 学　　号： |  |
| 班 级： |  |
| 姓　　名： | XXX |
| 专 业： | 计算机科学与技术 |
| 实验时间： | 2019.05.24 |
| 指导教师： | XX |

**实验第一阶段**

**课堂小结**

**进度**

已完成**2.1-2.5**。在实验课上完成了Codeblocks环境下win flex bison的配置与实验2.1-2.3的内容，课下完成了VS2015环境下win flex bison的配置与实验2.4-2.5的内容。

**实验中遇到的困难**

在控制台使用bison时出错，尝试后发现应将.y文件放到bison安装的文件夹中，并使用win\_bison.exe -d exp.y的命令语句。此时可成功产生exp.tab.h和exp.tab.c。

实验时发现每次对.y文件进行修改后需重新bison生成.tab.h和.tab.c文件。

做实验2.3时网站中的源码有一点问题，所以我选择直接修改2.1中的无符号逆波兰的源码，将逆波兰计算方法改成正常的计算方法。但尝试后发现该程序只能按照从右到左的优先级进行运算，\*、/的优先级并不正常，随后经过查阅资料(<https://rednaxelafx.iteye.com/blog/472113>)使用左结合搞定了\*、/的优先级问题。课后与小组成员讨论时发现还可以使用E+T的文法实现优先级。

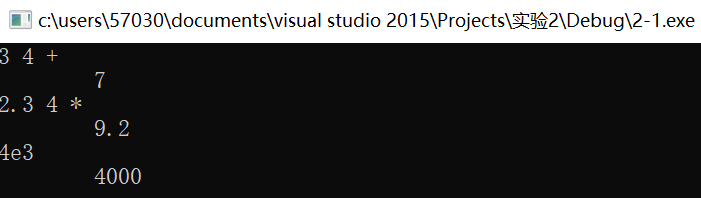
课下做剩余的实验时，由于电脑已安装了VS2015，且了解到VS也能使用bison，所以我根据 <https://www.cnblogs.com/SZxiaochun/p/6803624.html> 等帖子的指导与自己摸索搞定了在VS上跑bison的.y程序。又学习了 <https://blog.csdn.net/xiazdong/article/details/7461272> 该帖子，学会了在控制台运行flex将.l文件生成lex.yy.c，又在电脑上安装了MinGW，可以成功使用gcc命令编译该.c文件并在控制台运行生成的.exe文件。

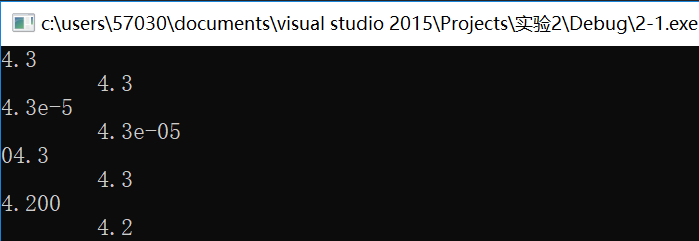
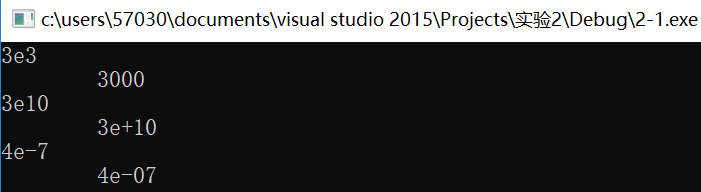
实验中发现VS认为scanf不安全，因此需在生成的程序中加入以下语句：#define \_CRT\_SECURE\_NO\_WARNINGS

另外，由于理论课还没有详细讲语法分析的部分，所以做实验的过程中有些似是而非，只求把代码跑通，没有深究理论方面的东西，这些在日后还需加强。

**实验内容**

**逆波兰表达式实验（无符号数）**

在VS2015上配置环境后编译2-1.y，生成2-1.tab.c和2-1.tab.h，加入#define \_CRT\_SECURE\_NO\_WARNINGS 语句后生成解决方案并运行。

1. 该文法支持浮点数（如上图所示）
2. 支持的浮点数格式如下图所示：
3. 支持科学计数法：
4. 该段程序负责识别浮点数：

if (c == '.' || isdigit (c))

{

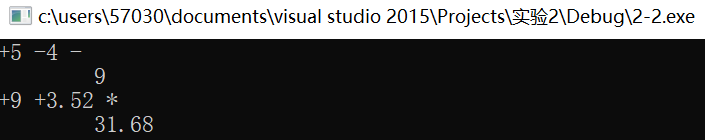
ungetc (c, stdin);

scanf ("%lf", &yylval);

return NUM;

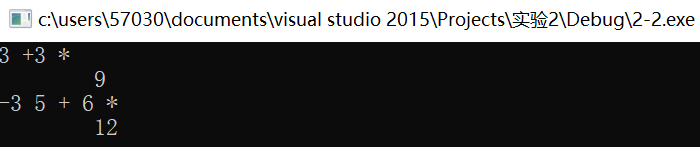
}

**逆波兰表达式（有符号数）**

在VS2015中编译2-2.y，生成2-2.tab.c和2-2.tab.h，加入#define \_CRT\_SECURE\_NO\_WARNINGS 语句后生成解决方案并运行。

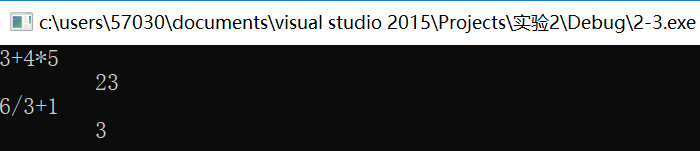
1. 本程序每个数必须带有+或-
2. 如果不带+-号，程序会出错关闭，显示syntax error

对文法进行修改，使其同时支持有符号数与无符号数。在exp声明部分加入语句：| NUM { $$ = $1; }

修改后的程序为2-2b.y，使用bison生成2-2b.tab.c和2-2b.tab.h。运行结果如下图：

**普通表达式实验**

对网站所给源码修改后，加入\*、/功能并使其优先级高于+-。

修改后的程序为2-3.y，使用bison生成2-3.tab.c和2-3.tab.h，运行结果如下图所示。

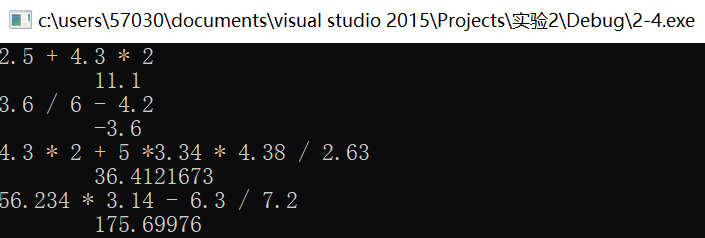
经验证，该程序运算结果正确。

**根据课本无符号浮点数的自动机，修改yylex**

1. 根据课本无符号浮点数的状态图，编写DFA矩阵：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **d** | **.** | **E** | **+|-** |
| **0** | **1** | **3** |  |  |
| **1** | **1** | **2** | **4** |  |
| **2** | **2** |  | **4** |  |
| **3** | **2** |  |  |  |
| **4** | **6** |  |  | **5** |
| **5** | **6** |  |  |  |
| **6** | **6** |  |  |  |

1. 修改yylex函数，支持无符号浮点数的识别
2. 运行修改后的程序，编写多个测试用例进行测试。

修改后的程序为2-4.y，使用bison生成2-4.tab.c和2-4.tab.h，运行结果如下图所示。

经验证，该程序结果正确。

**阅读flex编写的词法分析器源码**

(1) LEX/FLEX词法分析器

(2) 修改正则表达式FLOAT，使其支持有符号浮点数

原程序的float正则表达式为

FLOAT [0-9]\*[.][0-9]+([eE][+-]?[0-9]\*|[0])?f?

若要使其支持有符号浮点数，只需在前面加上 [+-]?即可。即：

**FLOAT [+-]? [0-9]\*[.][0-9]+([eE][+-]?[0-9]\*|[0])?f?**

1. 修改正则表达式FLOAT，使其不允许浮点数具有以下格式：

0.5合法 00.5或01.5非法

01合法（为八进制数） 001非法

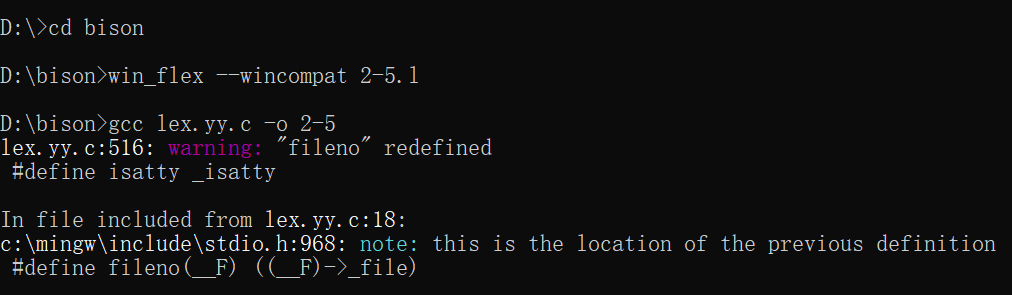
分析原程序中float的正则表达式可知，小数点前为[0-9]\*，因此出现00是合法的，因此对该式进行了修改。修改后的正则表达式为：

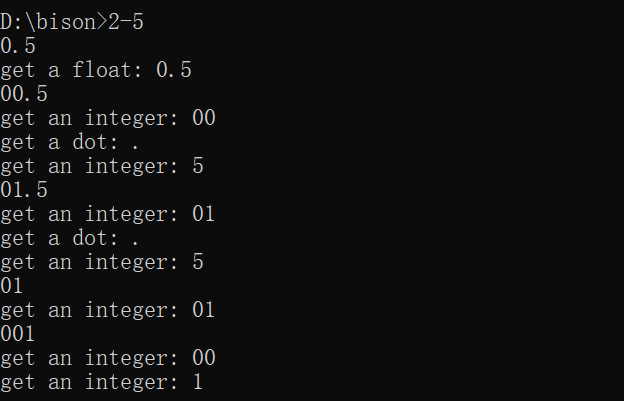
**FLOAT [+-]?([0]|([1-9][0-9]\*))[.][0-9]+([eE][+-]?[0-9]\*|[0])?f?**

上式可保证不会出现非法的0在前面的现象，又加入了特殊的0.n的情况。

对原程序进行修改，使用flex编写的词法分析器程序为2-5.l

通过命令行win\_flex --wincompat 2-5.l编译2-5.l文件，生成lex.yy.c程序。

使用gcc编译生成的c语言文件：gcc lex.yy.c -o 2-5

在同一文件夹下生成了词法分析程序2-5.exe。使用命令行运行该程序即可验证修改后的程序是否合法。

由上图可知，当输入0.5时，词法分析程序得到一个浮点数0.5。

而当输入00.5时，词法分析程序先得到一个八进制数00，再得到一个符号小数点，再得到一个十进制数5，由此可知在该程序中00.5不是一个合法的浮点数，即00.5是非法的浮点数。01.5同上。

当输入01时，词法分析程序得到一个八进制数01。

当输入001时，词法分析程序得到一个八进制数00和一个十进制数1，由此可知在该程序中001不是一个合法的整数。

**参考文献**

1. bison的运算符优先级一例

<https://rednaxelafx.iteye.com/blog/472113>

1. Windows下flex+bison小例子

<https://www.cnblogs.com/SZxiaochun/p/6803624.html>

1. 使用Bison和Flex（Windows下的基于GNU开源的Yacc和Lex）

<http://www.68idc.cn/help/opersys/windows/20150812497845.html>

1. LEX/FLEX词法分析器

<https://blog.csdn.net/mist14/article/details/48641349>

1. Windows下使用Flex入门

<https://blog.csdn.net/xiazdong/article/details/7461272>

1. 《flex与bison 中文版》
2. 课本

**简单计算器**

1. **实验要求**

利用flex & bison设计简单计算器

（1）参考案例1：编译原理利用Flex+Bison实现简单计算器

https://blog.csdn.net/xiaofeige567/article/details/28301877

（2）编译原理\_计算器\_flex、bison实现\_（从零开始）

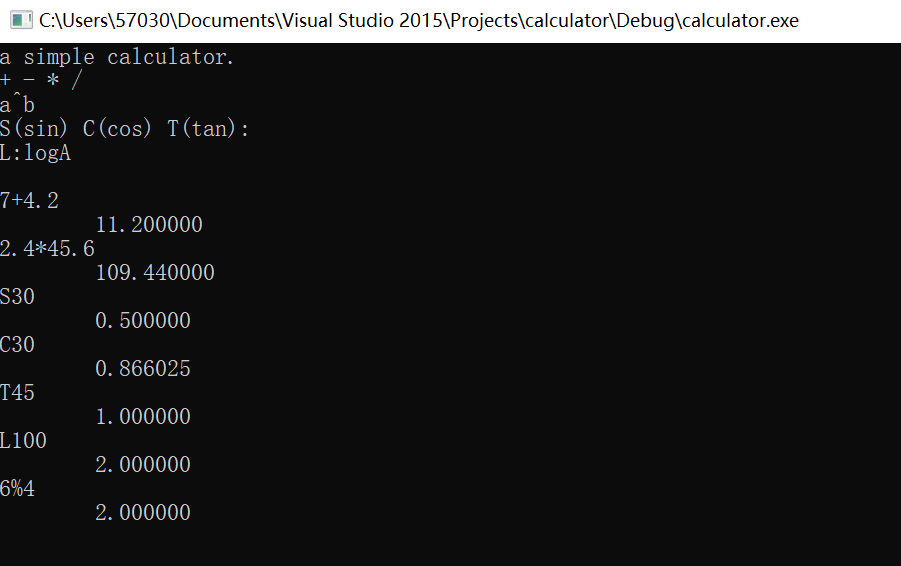
<https://blog.csdn.net/qq_35208390/article/details/78249181>

1. **实验进度**
2. 实现简单计算器
3. 修改代码使计算器支持浮点数运算
4. 输出四元式
5. **实验过程**

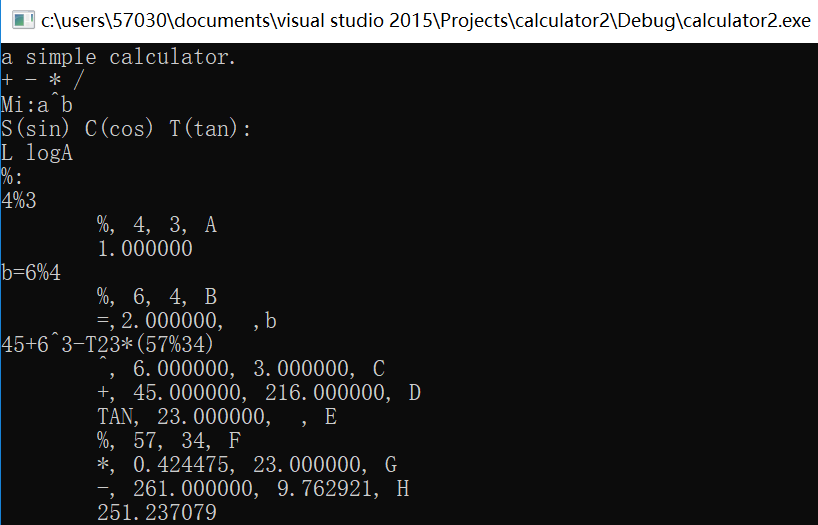
本次实验首先跑通了老师给的简单计算器的网站中的源码，自己摸索出了使用命令行和VS2015两种方法对.y和.l文件进行编译并生成可执行文件。对flex与bison的配合使用有了深入的了解：使用flex进行词法分析，自动生成yylex（）函数，随后使用bison进行语法分析，最终生成语法分析程序。

我们组发现源码并不能实现浮点数的运算，于是对flex文件中的文法进行修改，将只支持整数的文法INTEGER改为整数浮点数都可以的文法NUM，然后参考另一网站加入#define YYSTYPE double语句，对代码稍作修改，对有些数组下标取整后跑通了程序，能够实现浮点数的运算（取余除外，取余本来就只针对整数）。

使用VS2015编译calculator.l，生成calculator.flex.cpp（若是使用命令行则生成lex.yy.c），随后使用VS2015编译calculator.y，生成calculator.tab.h和calculator.tab.cpp（若使用命令行或codeblocks则生成.c文件）。生成解决方案并运行，结果如下图所示：



该计算器可实现+-\*/四则运算和幂运算、三角函数运算、取对数、整数取余等操作。

最后我们对网上找的四元式的例子进行了阅读与理解，决定使用在每个操作后使用C语言输出四元式。四元式是一种更接近目标代码的中间代码形式。由于这种形式的中间代码便于优化处理，因此，在目前许多编译程序中得到了广泛的应用。四元式格式为(op,arg1,arg2,result) 其中，op为一个二元(也可是一元或零元)运算符；arg1,arg2分别为它的两个运算(或操作)对象，它们可以是变量、常数或系统定义的临时变量名；运算的结果将放入result中。结果如下图所示：

参考网站：

https://blog.csdn.net/xiaofeige567/article/details/28301877

https://blog.csdn.net/qq\_35208390/article/details/78249181

https://baike.baidu.com/item/%E5%9B%9B%E5%85%83%E5%BC%8F/8951857?fr=aladdin

**实验第二阶段**

# 简介

在该Mini Pascal编译器的设计过程中，我使用分析、改造的方法学习了课本、网站帖子上现有的例子，之后对其进行改造，自主编写了所有程序并增加了多项功能。

该编译器能实现**变量定义**、基础运算、**布尔表达式（带短路）**、**if-else**语句、**while**语句等Mini Pascal的基础功能，同时还添加了**for循环**语句、**带初始化的变量定义**（且一行支持同时定义和初始化多个变量）、**变量定义和语句定义交叉混合出现**等功能，所有功能均能正常运行，并最终输出**四元式**和**符号表**。达到了老师要求的**B+85**分档，同时有适当的加分空间。

该编译器使用Flex + Bison的方法，使用Flex工具进行词法分析部分的编写，自动生成yylex()函数，使用Bison工具进行语法分析部分的编写和语法制导翻译部分的编写。将.y文件和.l文件分别编译，同时编译生成的两个c语言文件，得到.exe编译器。

# 测试用例与测试结果

## 定义变量并赋值

Program a;

Var Integer a;

Begin

a:=10

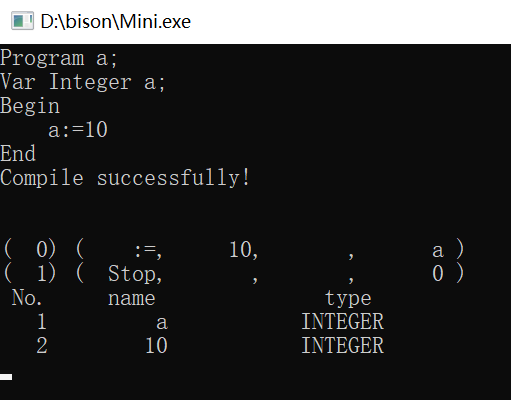
End

**测试用例设计说明：**

定义一个变量并赋值，以测试程序最基本的功能。

**预期测试结果：**

编译成功并生成一个四元式(:=, 10, , a),符号表内存入两个符号a和10.

**实际测试结果：**

## 定义多个变量并运算

Program a;

Var Integer a,b;

Begin

a:=3;

b:=4;

a:=a+b

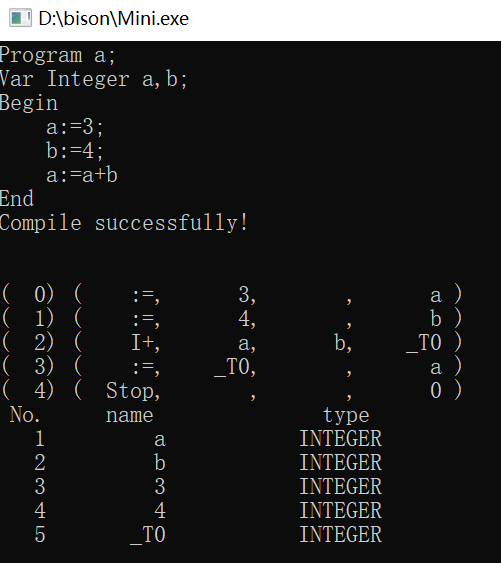
End

**测试用例设计说明：**

定义多个变量并在后面赋值，做简单的运算。

**预期测试结果：**

编译成功并生成两个a、b赋值四元式，一个加法四元式存到临时变量和一个赋值四元式将临时变量赋给a，符号表内存入a、b、3、4、T。

**实际测试结果：**

## If语句

Program a;

Var Integer a,b;

Begin

a:=3;

if(a!=0) a:=4;

b:=5

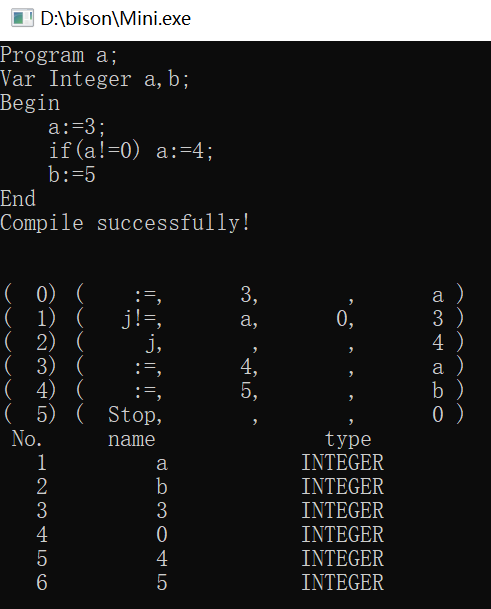
End

**测试用例设计说明：**

编写一个if语句，应在后面添加一个正常赋值语句，体现执行完if语句的跳转情况。

**预期测试结果：**

若满足布尔表达式则跳转到TC所在四元式，否则执行FC所在四元式，跳转到后面的语句。

**实际测试结果：**

## If-else语句

Program a;

Var Integer a,b;

Begin

a:=3;

if(a!=0) a:=4

else a:=5;

b:=3

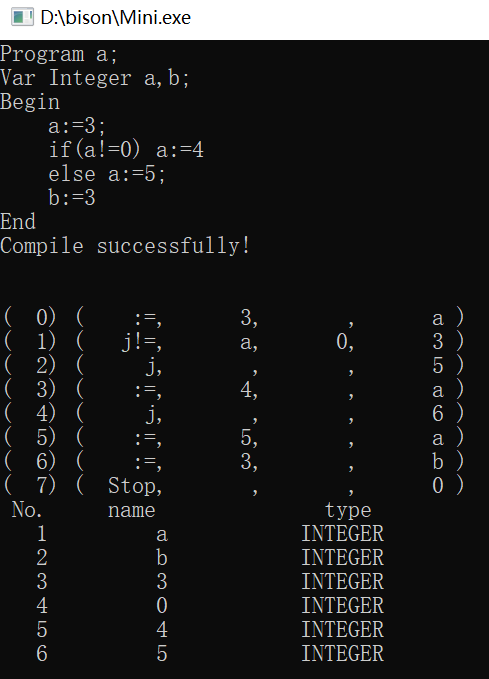
End

**测试用例设计说明：**

编写一个if-else语句，应在该语句后添加正常语句，以体现执行完if-else语句后的跳转情况。

**预期测试结果：**

若满足布尔表达式则跳转到TC所在四元式，否则执行FC所在四元式，跳转到else定义的语句。执行完成后跳转到后面的语句。

**实际测试结果：**

## While语句

Program a;

Var Integer a,b;

Begin

a:=0;

while(a<3)

Begin

a:=a+1;

b:=a\*3

End

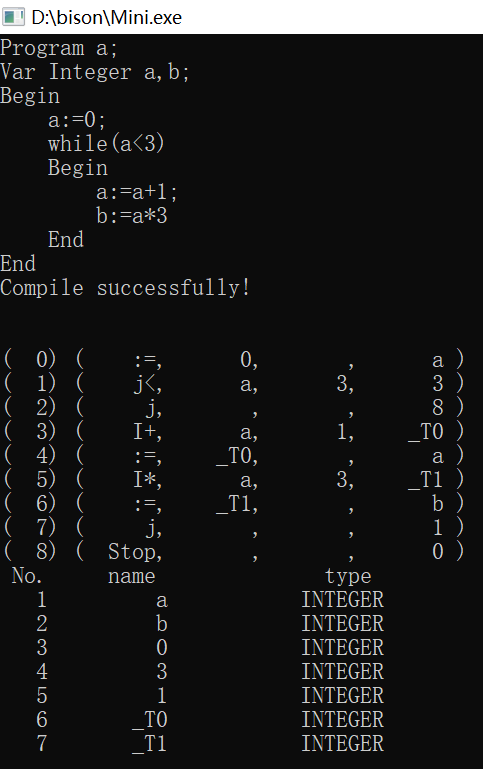
End

**测试用例设计说明：**

编写一个While语句，在while循环后添加正常语句以体现跳转，while语句内使用begin-end嵌套一个复杂语句。

**预期测试结果：**

若满足布尔表达式则应一直循环，每执行完一次跳转到判断语句，直到不满足时跳出到后面语句。

**实际测试结果：**

## While-IF嵌套

Program a;

Var Integer a,b;

Begin

a:=0;

while(a<5)

Begin

b:=b+a;

a:=a+1;

if(b<20) b:=b else b:=0

End

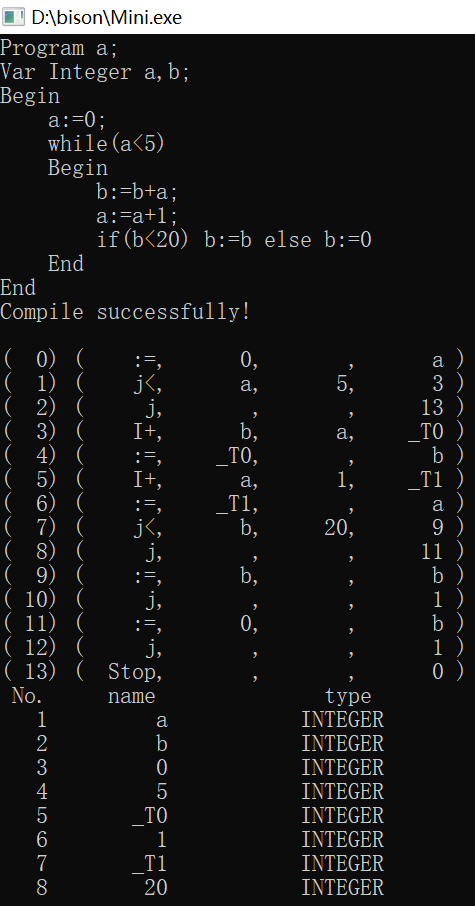
End

**测试用例设计说明：**

将if语句嵌套进while语句内，使用begin-end表示复杂语句。

**预期测试结果：**

满足while中布尔表达式条件时一直循环执行if语句。

**实际测试结果：**

## 多个布尔表达式，布尔表达式短路

Program a;

Var Integer a,b,c;

Begin

a:=3;

b:=5;

c:=a\*(b-2);

if(a<5&b>10) a:=5\*(b+c)

Else if(c>9&(a<2|b+3>9)) a:=6

Else a:=7

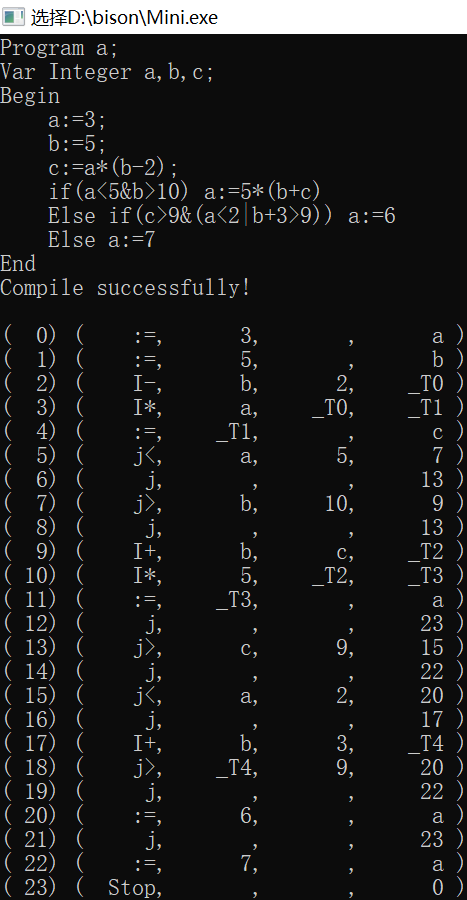
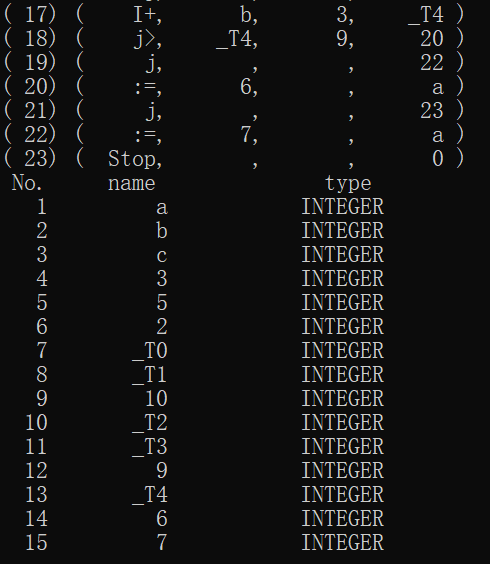
End

**测试用例设计说明：**

将if语句判定式中使用多个布尔表达式，并使用与或符号连接。检测布尔表达式短路。

**预期测试结果：**

能够正确判断复杂布尔表达式，当短路条件达成时直接跳转。

**实际测试结果：**

## for循环

Program a;

Var Integer i,b;

Begin

b:=1;

For (i:=0, i<6, i+1)

b:=b\*3;

b:=b+4

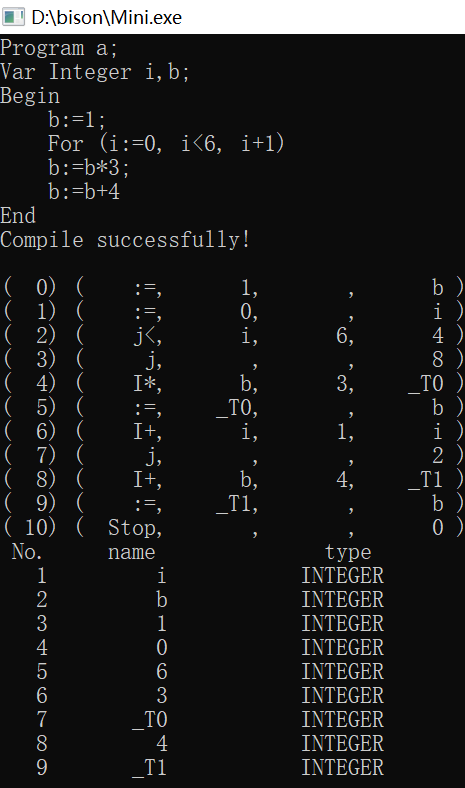
End

**测试用例设计说明：**

编写一个for语句，在后面加上正常语句以体现跳转。

**预期测试结果：**

能够正确运行循环并自加，每执行完一遍跳转至判断语句。不满足条件时跳转出循环。

**实际测试结果：**

## 变量定义并初始化

Program a;

Var Integer a:=5,b:=7;Real c:=3.14,d:=2.48;

Begin

a:=a\*b;

c:=c+d

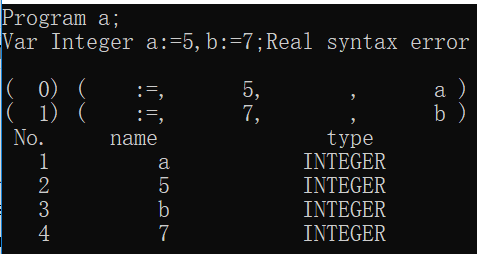
End

**测试用例设计说明：**

在变量定义的同时初始化值。

**预期测试结果：**

变量初始化时即生成四元式。

**实际测试结果：**

## 变量定义与语句定义混合出现

Program a;

Var Integer a;

Begin

a:=10;

Var real b:=1.5;

b:=a+1

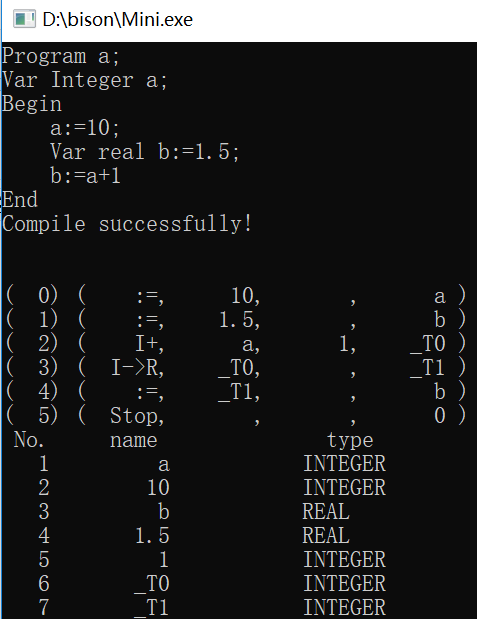
End

**测试用例设计说明：**

变量定义语句在原定义位置以外出现。

**预期测试结果：**

每遇到变量定义语句，就生成定义四元式。

**实际测试结果：**

# 实现过程

## 词法分析部分

首先对字符类型进行定义：标识符、整数、实数

然后定义程序中会出现的关键字:

Program(程序 后接程序名) Var(定义标识符) Real(实数)

Begin(程序开始符号) End(程序结束符号) Integer(整数)

While If Else For(条件语句中出现的关键字)

定义程序中会出现的双字符符号：LE(<=) GE(>=) NE(!=)

定义字符串char strl[20]:用于储存标识符、整数、实数等数字的字符串，当这三种类型被该词法分析器识别时，使用strcpy函数将它们从yytext中拷贝到strl字符串中，用于后续bison程序。

定义文法简写：

letter [A-Za-z]

alnum [A-Za-z0-9]

d [0-9]

white [\t\n\040]

（1）规则部分

首先定义关键字的规则：即各字母不区分大小写，如begin：

[Bb][eE][Gg][iI][Nn] {ECHO;return Begin;} 其中，ECHO是将其存入yyout中，为了在输出中将原程序再显示出来。

接着定义几种数据类型的文法正则表达式：

标识符{letter}{alnum}\*

实数{d}+\.{d}\*|{d}\*\.{d}+

整数{d}+

空格{white}+

最后定义各种标点符号，返回它们自身。

（2）附加部分

该词法分析器暂时没有附加c语言的必要，因此只需使用yywrap函数即可。

int yywrap(){return 1;}

## 语法分析部分

yyin、str1为在lex中定义的变量和数组，因此应在Bison中声明为exter变量。结构体Quaterlist和Varlist分别为四元式和符号表。%start指定文法的开始符号，若不使用则默认第一条规则为开始符号。%union与%type配合，表示非终结符的属性值。

OutputQ()函数用于输出四元式，Entry(char \*name)用于将变量名填入符号表，FillType用于填入数据类型，Merge用于合并退出链，并返回链首，BackPatch用于回填跳转类四元式的result，GEN函数用于生产新的四元式，NewTemp函数用于定义一个临时变量，OutputIList用于输出符号表。

* If '(' BoolExpr ')'{BackPatch($3.TC,NXQ);$$=$3.FC;}

用NXQ回填布尔表达式的TC链，将FC属性保留，作为CH（chain）方便后续合并和回填。

* Wd : W '(' BoolExpr ')' {BackPatch($3.TC,NXQ);$$.QUAD=$1;$$.CH=$3.FC;};

W : While{$$=NXQ;};

将BoolExpr第一四元式序号记录下来，以便在翻译完statement后产生一个跳转到此处的四元式。翻译BoolExpr后使用当前NXQ（Statement的第一四元式序号）回填TC链，将FC属性传递给CH属性，方便后续合并和回填。

* Fd : For '(' FExpr ')'

{BackPatch($3.AC,NXQ);$$.QUAD=$3.AC;$$.CH=$3.CH;$$.D=$3.CC;

$$.N=$3.BC;};

FExpr : AsignState ',' BoolExpr ',' Expr '+' Const

{$$.AC=$3.TC;$$.CH=$3.FC;$$.CC=$1;$$.BC=$7.place; };

用AC传递布尔表达式的TC，CH传递布尔表达式的FC，方便后续回填，CC传递AsignState语句的变量名，BC传递自增的数值。此处生成赋值四元式、布尔表达式的TC和FC链。与While相同，For语句也将布尔表达式的第一四元式记录下来，方便statement语句之后生成跳转四元式跳回来。

* BoolExpr :BoolExpr '&' {BackPatch($1.TC,NXQ);}

BoolExpr{$$.TC=$4.TC;$$.FC=Merge($1.FC,$4.FC);}

| BoolExpr '|' {BackPatch($1.FC,NXQ);}

BoolExpr{$$.FC=$4.FC;$$.TC=Merge($1.TC,$4.TC);}

布尔表达式短路：遇到且符号时，第一个布尔表达式的TC回填第二个布尔表达式的TC，将两个表达式的FC链合并，整个布尔表达式的TC链为第二个布尔表达式的TC，FC为合并的新链链首。

遇到或符号时，第一个布尔表达式FC回填第二个布尔表达式的TC，两个表达式的TC链合并，作为整个表达式的TC，第二个布尔表达式的FC作为整个表达式的FC。

* VarList : VarList ',' Asignment{FillType($3,$1);}

| VarList ',' Varable{FillType($3,$1);}

| Type Asignment{$$=$1 ;FillType($2,$$);}

| Type Varable{$$=$1 ;FillType($2,$$);};

Asignment : Varable':''='Expr {GEN(":=",$4.place,0,$1);};

变量初始化：添加一个赋值语句Asignment代替变量Varable，赋值时生成四元式。

变量任意位置定义（定义语句与其他语句可交替出现）：在Statement语法声明中加入|VarDef; 即可实现在任意位置定义并初始化变量。

# 调试记录

4.26实验课及课下完成LEX词法分析器的编写和调试。

5.5 实验课及课下完成Bison文法分析部分，包括赋值、布尔表达式、if-else、while等功能的框架。

5.10实验课及课下完成Bison一些基本功能的语法制导翻译部分，编写四元式的函数等。

5.17实验课及课下添加for函数功能。

5.17-5.23添加变量定义初始化和变量任意位置定义。

5.23-5.29调试程序、修改测例、撰写实验报告。

# 遇到的问题及解决方法

* 刚开始Flex和Bison使用不熟练

通过前几节实验课的练习，加上自学网络上各种帖子，逐渐掌握了Flex+Bison的使用方法。

* scanf和strcpy等函数会报错

VS2015会认为这两个函数不安全，编译时加上一句就可解除warning。后来我为了省事直接使用命令行和gcc进行编译，反而更方便。

* 文法规则在测试时会出现syntax error，应该是哪里出了问题

仔细检查语法后发现是语法有一点小错误，少了一个分号，编写测例时只需把最后一个分号删去，或者在语法定义时在END前加上一个分号即可解决。

* For循环生成的四元式太麻烦

学习理论课的ppt仿照While生成的四元式，将自加四元式放到statement之后，即可简化四元式。