

**课程设计报告**

( 2017 – 2018 年度第1学期)

名 称： 编译技术课程设计A

题 目：L语言编译器的设计与实现

院 系： 计算机系

班 级： 软件1502

学 号： 20150902020x

学生姓名：

指导教师： 鲁斌 李继荣

设计周数： 2 周

成 绩：

日期： 2018 年 1 月 26 日

**一、课程设计的目的与要求**

1. 任务：实现一个简单的编译程序，能够对指定程序设计语言进行编译
2. 目的：加深对课堂讲授知识的理解，熟练掌握编译程序设计原理及常用的技术，建立编译程序的整体概念，使得学生初步具有研究、设计、编制和调试编译程序的能力。
3. 要求：熟悉有关定义、概念和实现算法，设计出程序流程框图和数据结构，编写出完整的源程序，进行静态检查，设计出输入数据、显示输出数据；基本功能完善，方便易用，操作无误；通过课程设计学会编译程序设计与实现的常用技术，具备初步分析、设计和开发编译程序的能力，具备分析与检查软件错误、解决和处理实验结果的能力。

**二、课程设计正文**

1. 扫描器设计

该扫描器是一个子程序，其输入是源程序字符串，每调用一次输出一个单词符号。为了避免超前搜索，提高运行效率，简化扫描器的设计，假设程序设计语言中，基本字不能用作一般标识符，如果基本字、标识符和常数之间没有确定的运算符或界符作间隔，则用空白作间隔。

* 1. 具体设计

在该实验中，扫描器子函数放置在一个保存词语的列表WordList类中，而且在该类中放置有一个字符缓冲区类CharBuffer类。而且有一个静态KeyWordTable类，作为Word对象的工厂类。

（1）字符缓冲区

其利用一个保存有char型数组和一个保存数组写入位置的int数的CharBuffer类型实现的。用于接受输入的字符，和输出保存在数组中所形成的字符串。

（2）WordList类

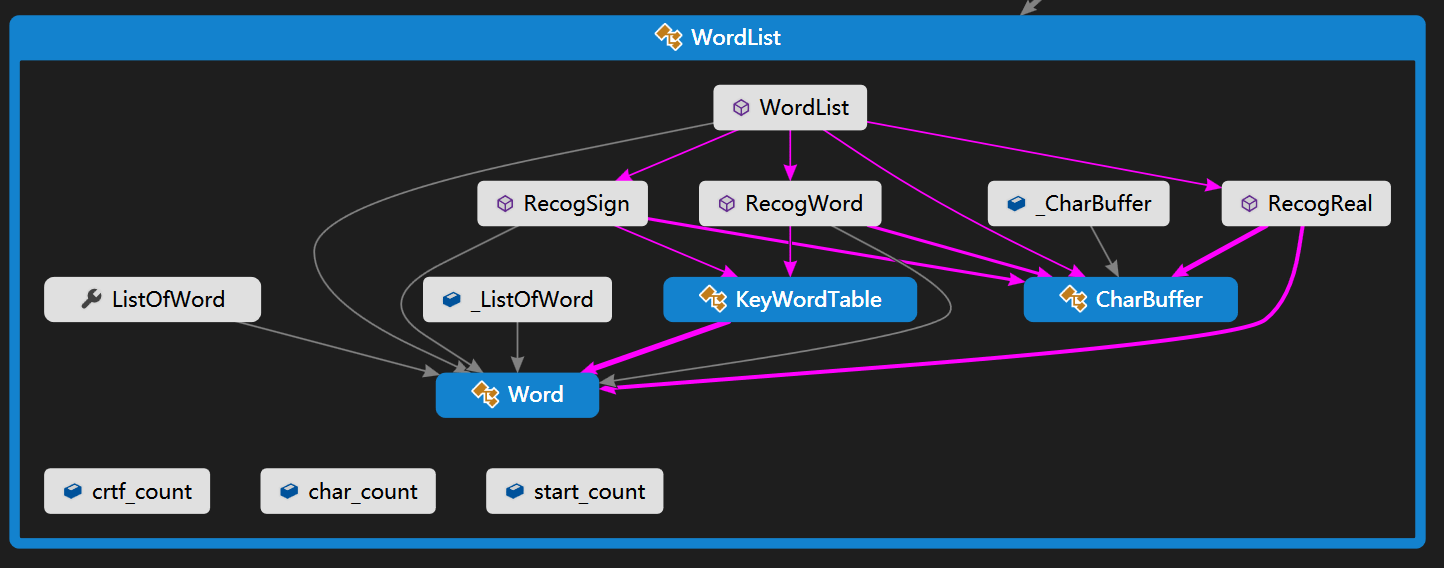
其中保存有三个子函数RecogSign、RecogWord、RecogReal，其扫描到指定的词语后将其字符串传给添加到词语列表中，且返回一个保存有扫描后位置的int数，以供主函数更新扫描指针的位置。而词语的处理利用一个保存有保留字列表的作为Word对象工厂的KeyWordTable类，从缓冲区中读出的字符串传给KeyWordTable类，然后获取返回Word对象放置在词语列表中。

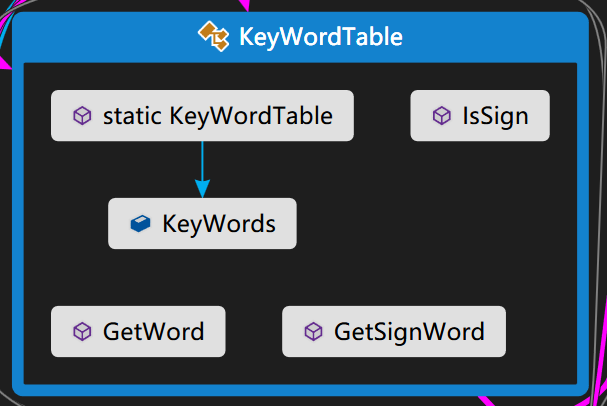
（3）错误处理

在实际实现中，因为还需要返回错误的位置，所以又为其添加了两个变量，crtf\_count、start\_count，分别用以保存当前行数以及当前行的起始位置，而在向WordList对象中添加词语时，将当前字符串位置减去start\_count作为其列号，将词语的列号和行号保存在Word对象中，以供在接受到语法分析器的错误时，以及词法分析时出现错误时，通过其查找错误词语的位置。

（4）KeyWordTable类

其作为Word对象的工厂类，其作用是接受到一个字符串，然后由其返回一个加工好（对其序号进行标记）的Word对象，将其保存在自己的词语列表中。

* 1. 相关对象设计

图 1-1 简化后的WordList类代码图

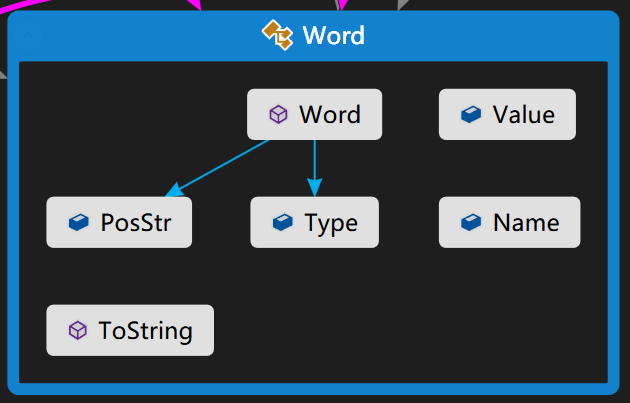
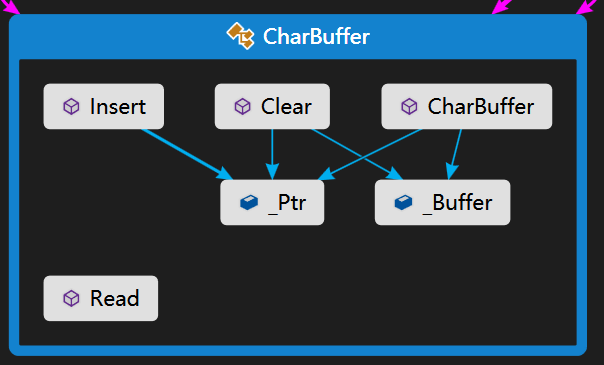
图 1-2 简化后的KeyWordTable的代码图

图 1-3 简化后的Word类代码图

图 1-4 简化后的CharBuffer类的代码图

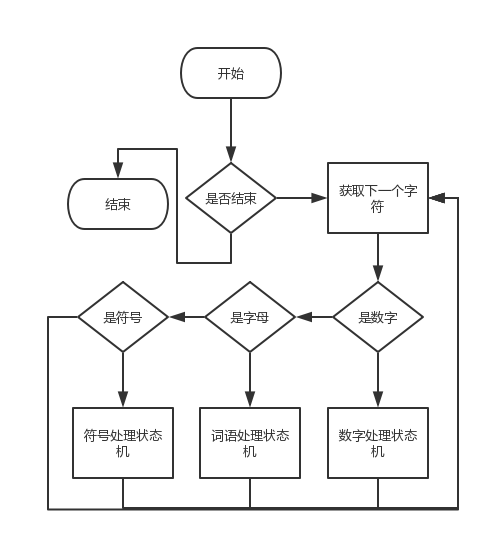
* 1.  程序流程图

图 1-5 词法分析流程图

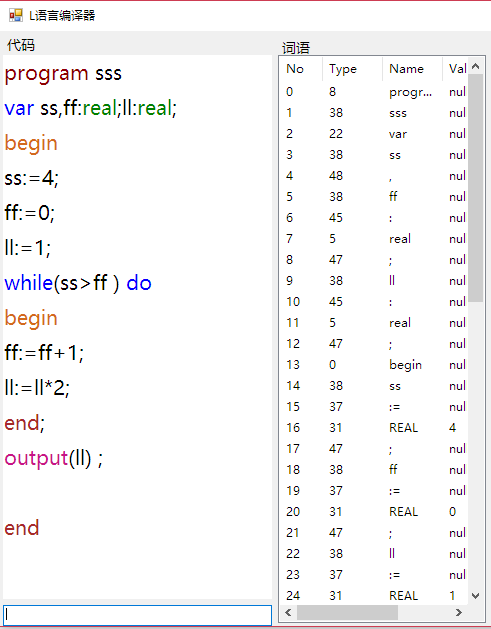
* 1.  程序运行结果

图 1-6 词法分析结果

1. 语法分析器设计

以算法优先分析方法为例，设计一个算符优先语法分析程序。算符优先分析属于自下而上的分析方法，该语法分析程序的输入是终结符号串（即单词符号串，以一个“#”结尾），如果输入串是句子则输出“YES”，否则输出“NO”和错误信息。当然，也可采用预测分析等方法设计语法分析器，具体方法自定

2．1 具体设计

在该实验中，使用的是，自上而下的利用预测分析表的。而在实际设计中使用一个GrammarTree对象，其保存有语法树的头节点，以及对相关的方法进行封装。

1. GrammarItem类

用于建立语法树的节点类，其保存着当前节点的非终结符的内部号码，以及子节点的引用数组以及子节点的内部号，和一个保存插入位置的int。而在其插入词语时，查找自己的子节点，如果其为终结符，则其加入到子节点的数组中，若其为非终结符，则动态从静态类GrammarTable中，利用子节点号码和词语号码获取一个GrammarItem对象，并将其插入到子节点数组中，并调用其插入词语的函数InsertWordInsertWord，将当前词语插入到其中。以实现对树的建立。

1. GrammarTree类

主要用于包装树的头节点以及相关函数，将其头节点的相关数进行封装，以方便调用。

1. GrammarTable类

其为一个静态类，主要保存的是一个字符串化的一维预测分析表，以及相关编号的非终结符的名称。其GetGrammarItem函数，在接受到传来的非终结符节点以及词语的序号，则从预测分析表中查找其节点，返回应当推出的节点，并且将其子节点的内部号写入其列表中以便查找时使用。

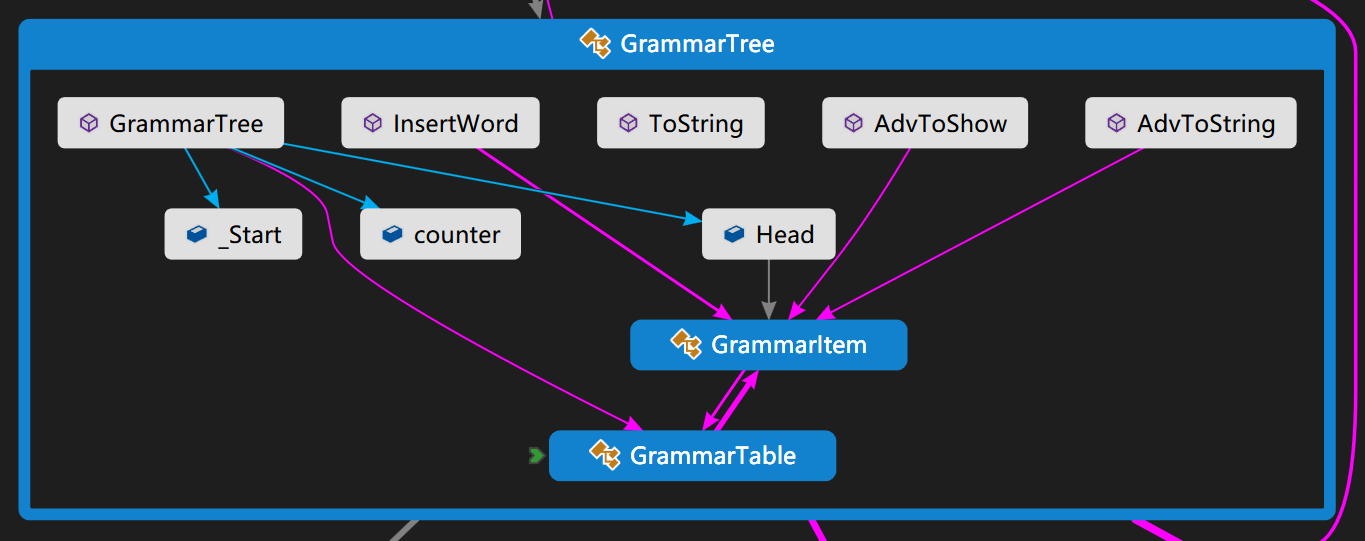
2．2 相关对象设计

图 2-1 简化后的GrammarTree类代码图

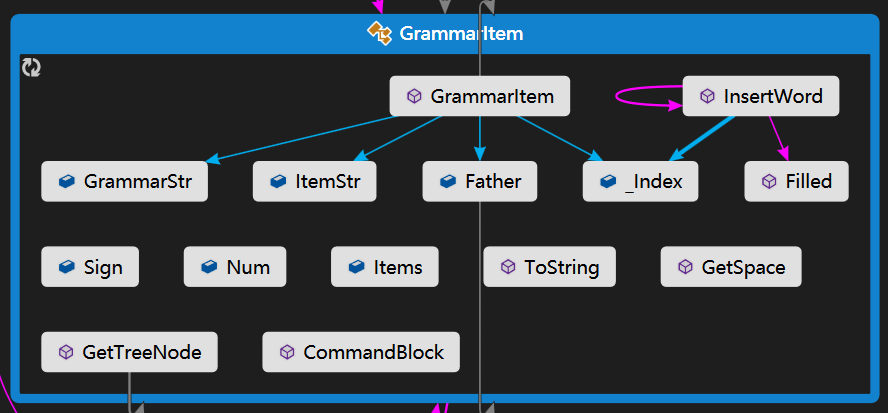


图 2-2 简化后的未加入语义分析的GrammarItem类代码图

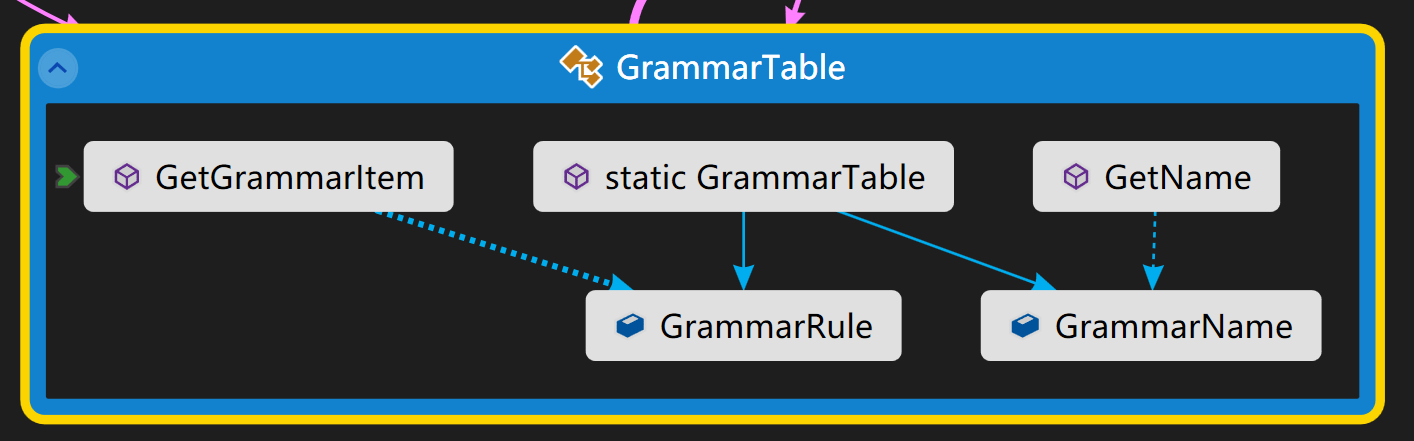


图 2-3 GrammarTable类代码图

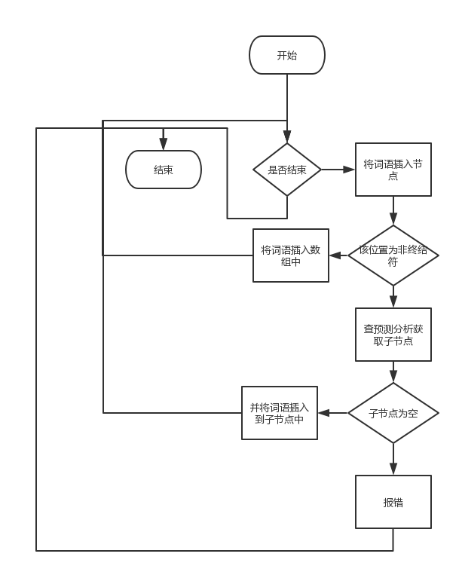
2．3 程序流程图

图 2-4 语法树节点获取单词的流程图

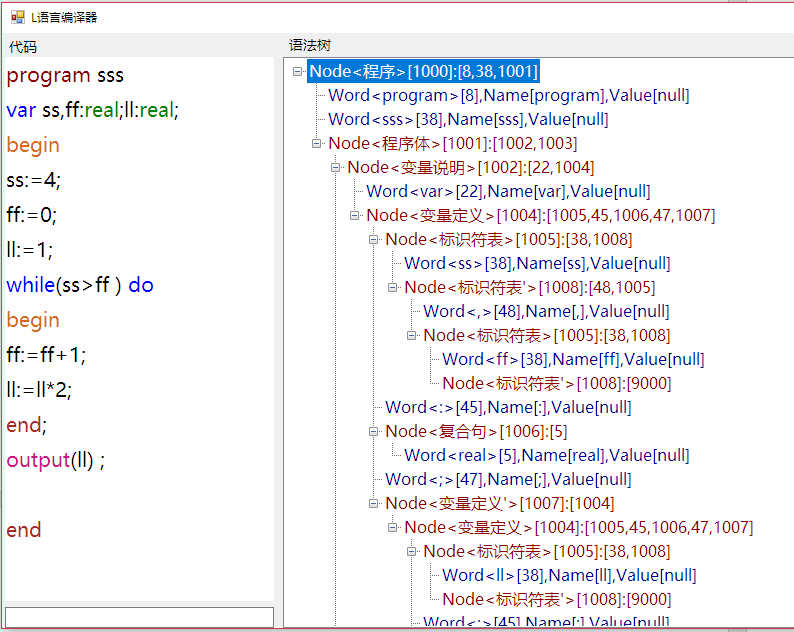
2．4 程序运行结果

图 2-5 程序运行结果

1. 语法制导翻译程序设计

用语法制导翻译方法，实现算术表达式、赋值语句和基本控制语句等的翻译。本语法制导翻译程序的输入是终结符号串（即单词符号串，以一个“#”结尾），如果输入符号串是句子,则按照其语义进行翻译,输出等价的四元式序列。

3．1 具体设计

在本次实验中，是使用的面向对象的方法，实现了类似与递归下降分析的面向过程的功能。由于其依赖于语法分析的结果，所以将相关的实现代码直接加入到了GrammarItem类中。

1. 加入语义的GrammarItem类

为了对树节点进行语义的分析，又因为是面向对象的设计模式，所以还需要将中间代码列表以及符号表作为参数，传入树节点的语义分析函数，而且为了使得程序相对的较为易读，语义分析的函数并未使用一样名字的函数进行递归调用，而是按照其语义对生成其中间代码的函数进行命名。

例如处理if语句的CondCommand函数其参数为符号表VarTable类的对象和一个中间代码列表MLList类的对象。在其中，先调用获取布尔表达式获取其变量地址的函数，其添加相关的中间代码，并且将保存布尔表达式出口变量的地址返回给调用函数。而后调用函数插入一条jz语句，参数为布尔表达式的出口变量，并且保存语句位置等待对其进行回填，然后处理相关的语句，在中间的执行语句结束后，获取中间代码位置，将其回填至jz语句中，若其为有else的if句，则中间执行语句完成后插入一条等待回填出口的j语句，再回填jz语句，待else块中的代码执行完成后，将当前中间语句位置回填j语句的跳转位置。

1. MLList类

其保存有一个中间语句列表，以及封装有向其中插入四元式的函数，以供调用者使用。以及一个用以获取下一条语句要插入位置的will be函数，以供回填时使用。

1. VarTable类

以该类以实现符号表的功能，其包装有一个变量对象列表，以及4个函数，分别为AddTemp、AddTemp（重载）、AddVar、GetVar函数，其主要的功能分别为为表达式添加一个临时变量用于保存中间结果并返回其地址，向其保存常量并返回地址、向其插入变量并返回地址、获取指定名称变量的地址。

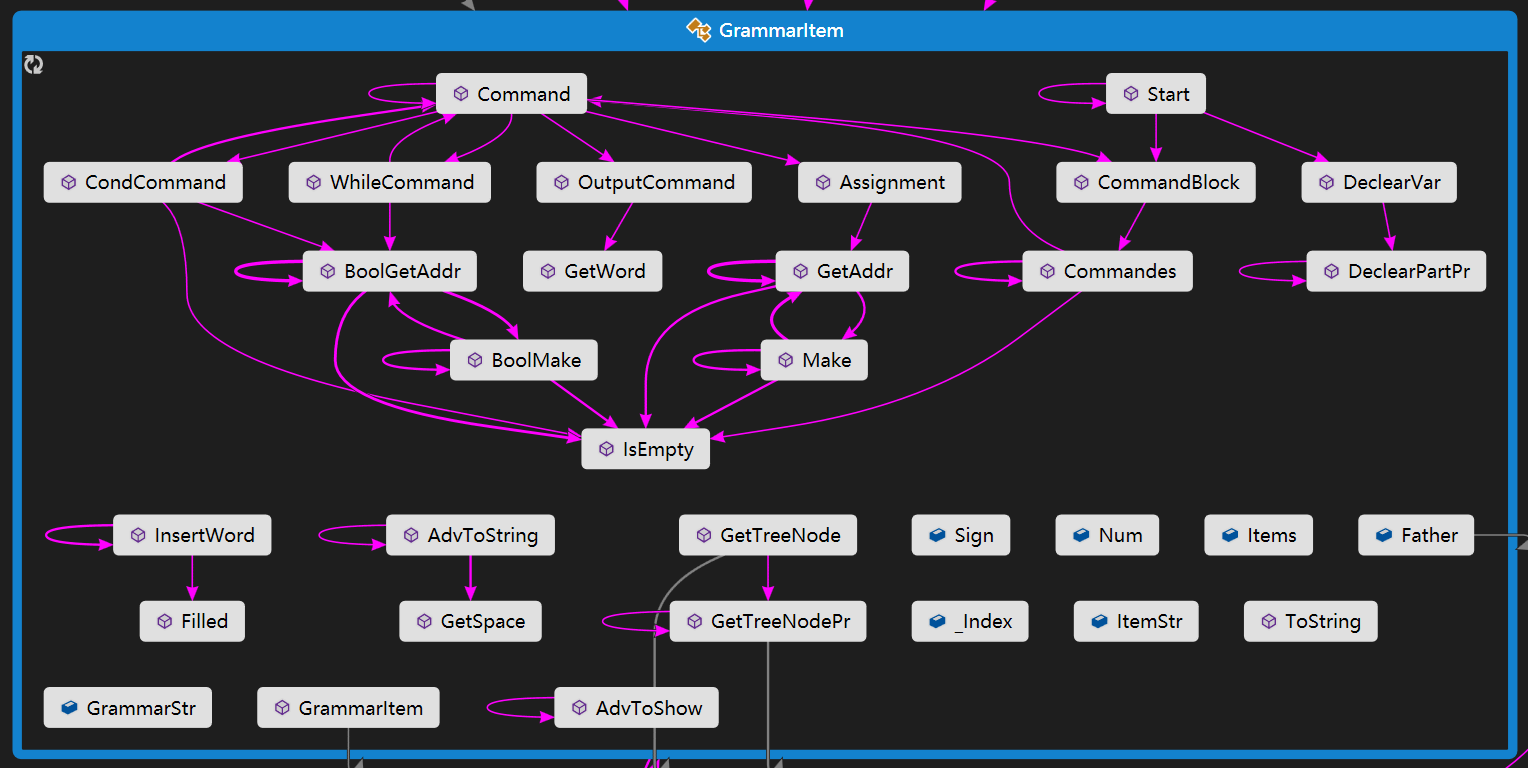
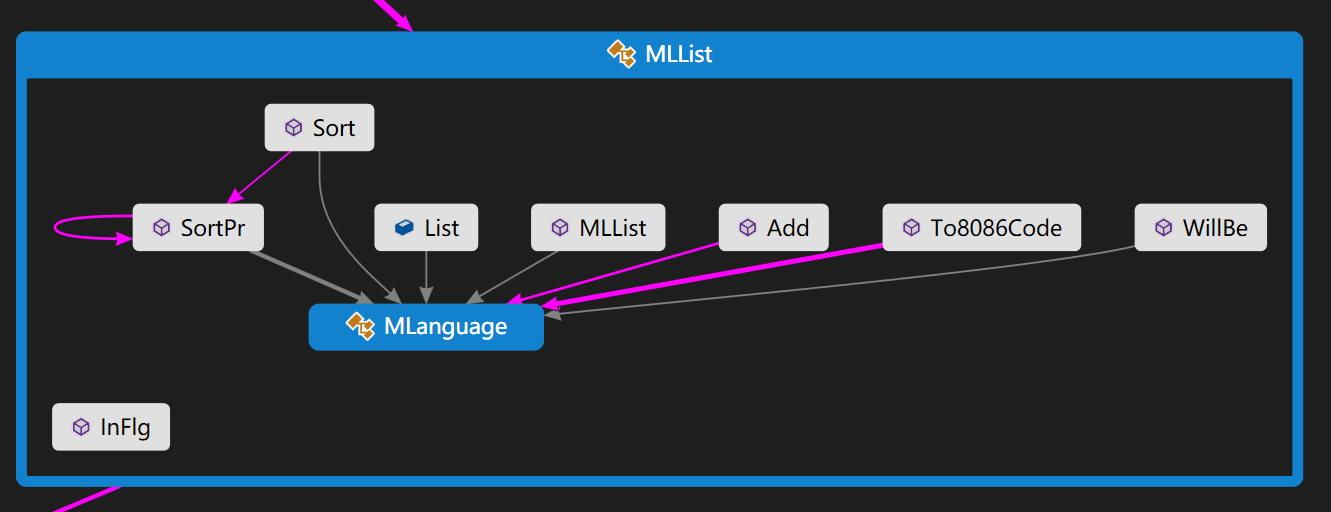
3．2 相关对象设计

图3-1 有语义分析的GrammarItem类（上半部分为语义分析相关函数）



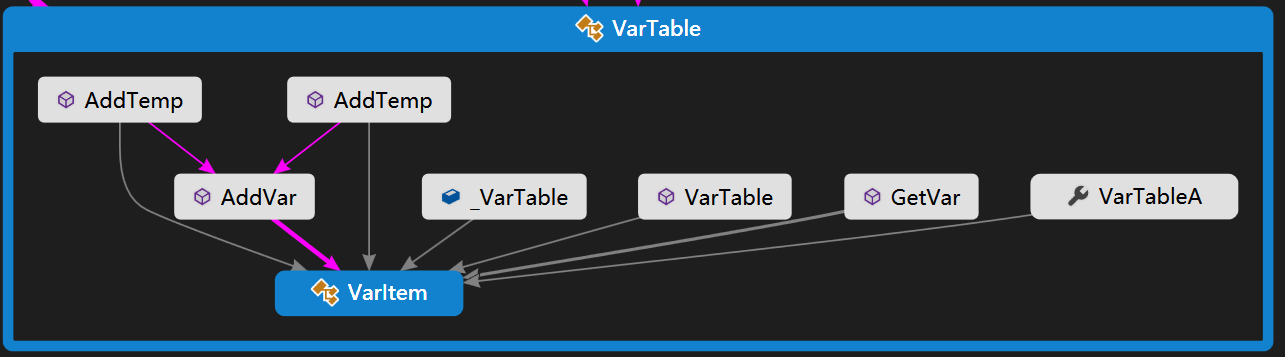
图 3-2 MLList类代码图

图 3-3 VarTable类代码图

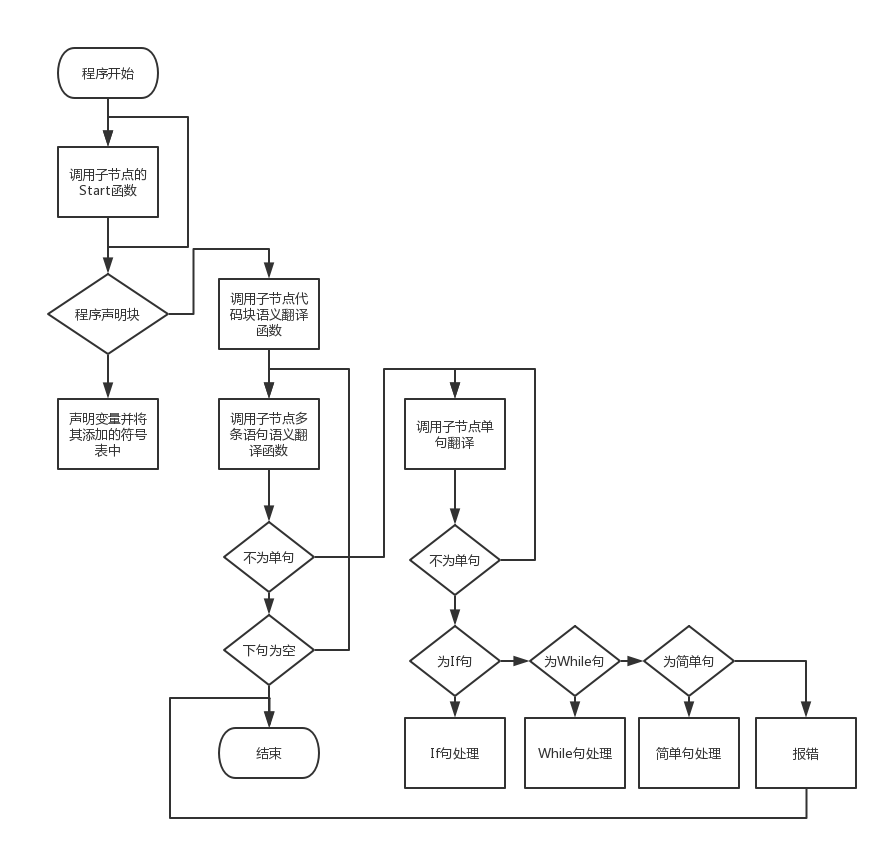
3．4 程序流程图

图 3-4 语法树处理流程

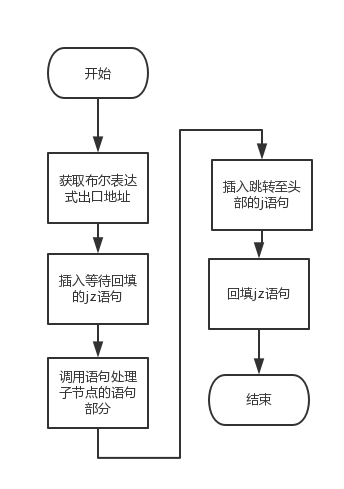
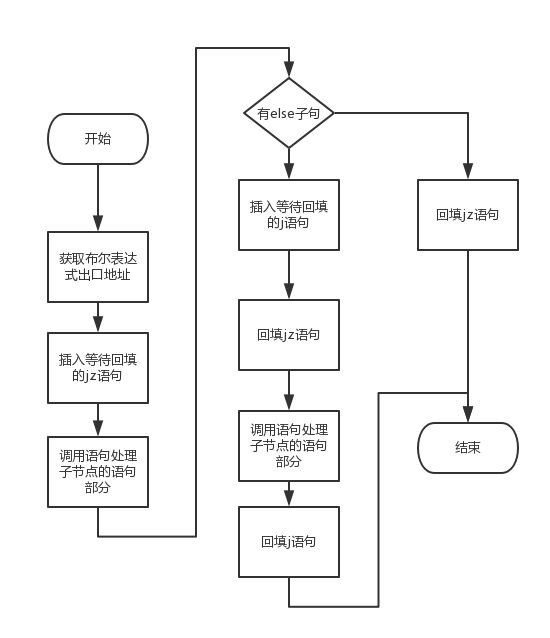
图 3-4 if句处理流程

图 3-5 while句处理流程

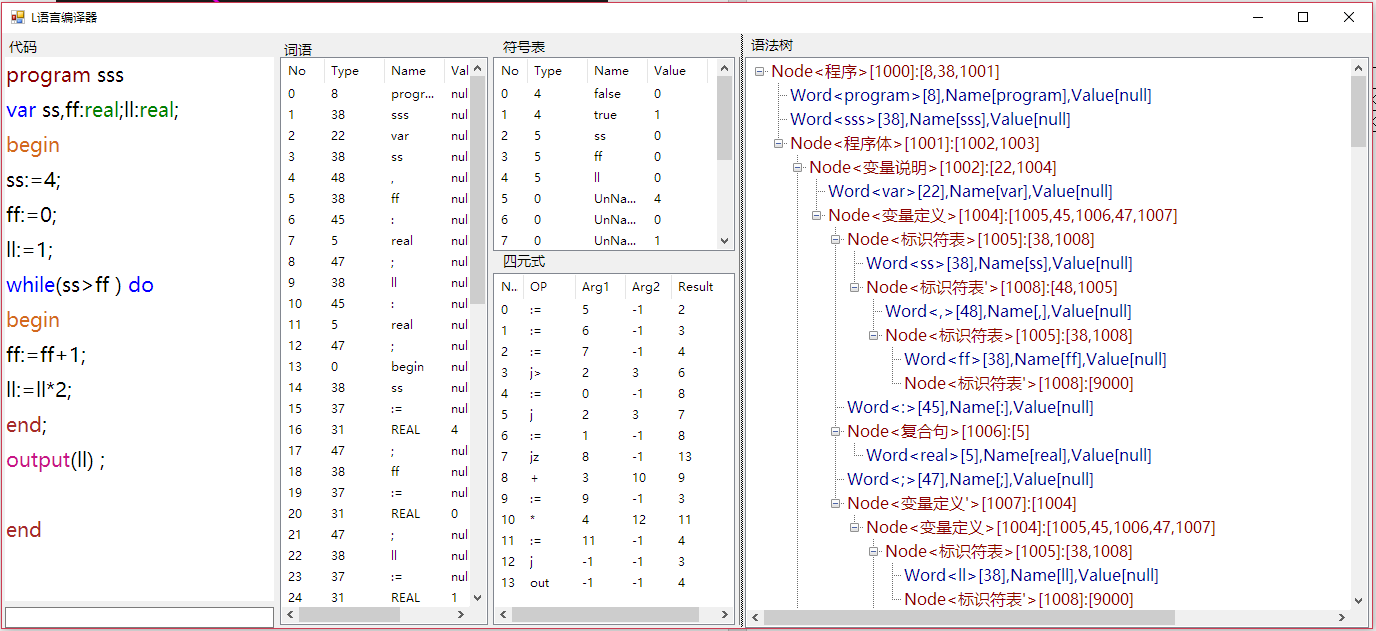
3．4 程序运行结果

图 3-4 程序运行结果

1. 目标代码生成器设计

将程序设计语言的中间代码程序翻译为目标代码程序，其输入是四元式序列，输出是一个汇编代码文件。

4．1 具体设计

在本次实验中，由于一般四元式与汇编代码可以一一对应，所以将翻译的代码加入到四元式MLanguage类中，以供MLList类调用。而在实际编程中，使用MLList类对MLLanguage列表进行封装。

但是在实际的应用过程中发现了很多问题，其中最大的问题就是跳转语句的相关问题。首先是跳转位置的设置，因为为了使其跳转到指定位置，所以使用了标签来标记跳转的位置。但是导致了如果由后跳转至前头，就无法确定标签，而对其扫描两次有比较耗费资源，于是在翻译时将每一句前加一个标签，使其与四元式位置一一对应，就可以解决标签的问题，但是导致了其中每一行中都有标签。而且因为有跳转语句的问题，导致很多地方都不可能进行优化，尤其是在跳转位置的附近，因为不运行程序是几乎不可能确认跳转前的寄存器状态。

其次就是在实现not转为汇编代码的问题，在本实验中以1带表布尔的true而0代表布尔false，开始将not翻译为汇编的not，导致了二者相对的变为了被识别为true的值，因为汇编中not为按位取反，导致二者均不为0，而后将汇编的not换为xor reg,01h便很好的解决了布尔运算的问题。

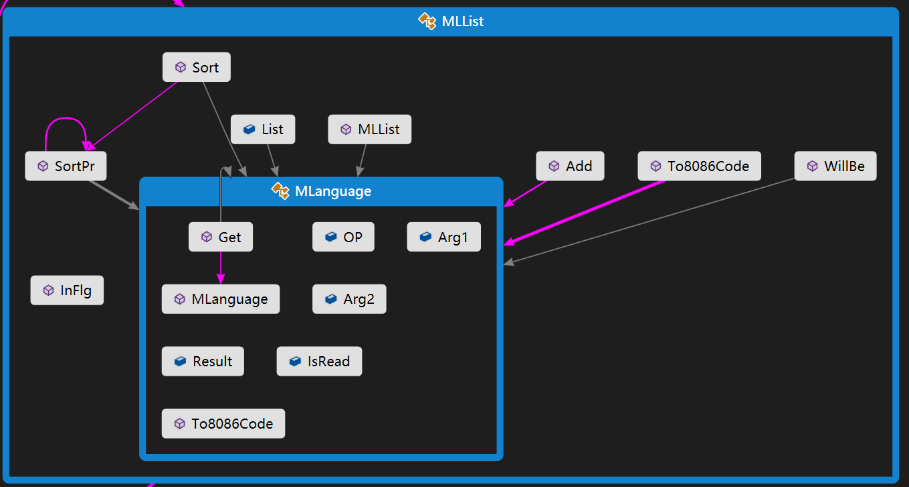
4．2 相关对象设计

图 4-1 有代码生成功能的MLList的代码图

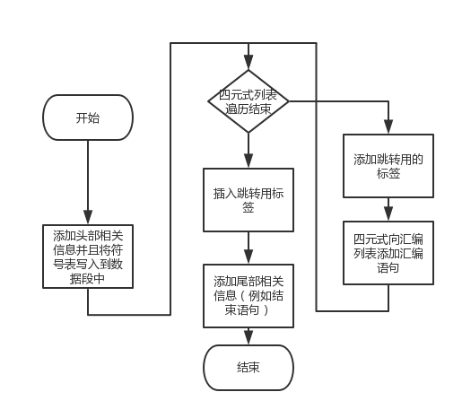
4．3 程序流程图

图4-2 汇编代码生成流程图

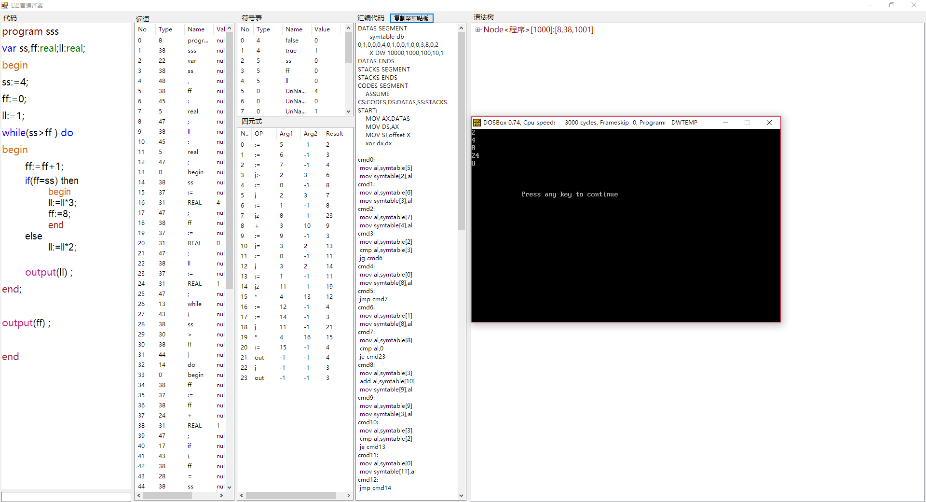
4．4 程序运行图

图 4-3 实际运行结果(汇编的结果为2 4 8 24 8)

**三、课程设计总结或结论**

1. 在本次实验中，通过对汇编的相关技术的学习，使我学习到了一些特殊问题的解决方法，比如说通过对词法分析器的编程，学习到了词法分析器其对词进行处理，以及其对特定词语的辨别，而且学习到了这种算法后，该算法不仅可以在辨别词语的情况中使用，还可以在利用识别网页内容中使用，虽然正则表达式也可以完成相同的功能，但是使用状态机的方法更为灵活。而且在通过对词法分析器的编程，也了解到了正则表达式的工作原理，在以后使用正则表达式时也能知道其是如何辨别内容的。
2. 而对语法分析的编程，更是学习到了语言其可以作为一颗树来建立，使我学习到了解决原来无法解决的问题的解决方案，而且在学习到该技术，不仅可以在程序语言中使用，而且经过一定的扩展，还可以在自然语言中使用。而通过对语义分析学习更是了解了语句其实际语义的翻译方式。而且通过编译，也在一定程度上的了解了高级语言的具体实现的方法，进一步的了解了高级语言编程的本质。

**四、参考文献**

[1] 鲁斌等. 编译技术基础教程. 北京：清华大学出版社，第一版. 2011，11

**附录（设计流程图、程序、表格、数据等）**

static class GrammarTable

{

public static string[] GrammarRule = {

"8:1000>8,38,1001",//〈程序〉→ program〈标识符〉〈程序体 1001〉.

"22:1001>1002,1003",//〈程序体1001〉→〈变量说明1002〉〈复合句1003〉

"0:1001>1002,1003",

...

"36:1208>36",

};

//获取指定终结符号和词语号的语法树节点

public static GrammarItem GetGrammarItem(int GrammarIndex, int WordType,GrammarItem father)

{

for(int i = 0; i < GrammarRule.Length; i++)

{

if (GrammarRule[i].Split('>')[0].Split(':')[1] == GrammarIndex.ToString() &&

GrammarRule[i].Split('>')[0].Split(':')[0] == WordType.ToString())

{

string[] stres = GrammarRule[i].Split('>')[1].Split('|')[0].Split(',');

int[] tar = new int[stres.Length];

object[] items = new object[stres.Length];

for(int j=0;j<stres.Length;j++)

{

tar[j] = int.Parse(stres[j]);

}

GrammarItem gi = new GrammarItem(father);

gi.Sign = tar;

gi.Items = items;

gi.Num = GrammarIndex;

gi.ItemStr = GrammarRule[i].Split('>')[1];

gi.GrammarStr = GrammarRule[i].Split(':')[1];

return gi;

}

}

GrammarItem gie = new GrammarItem(father);

if (GrammarIndex == 9000)

{

gie.Num = 9000;

gie.\_Index = 1;

gie.Sign = null;

return gie;

}

return null;

}

}

}

//向树中插入词语调用函数 counter用于记录插入词语的数量，以供报错时使用

public void InsertWord(Word wd,ref int counter)

{

if (Sign[\_Index] < 1000)

{

if (Sign[\_Index] == wd.Type)

{

Items[\_Index] = wd;

\_Index++;

counter++;

}

else

{

throw new Exception("语法错误:"+(counter-1).ToString());

}

}

else

{

if (Items[\_Index] == null)

{

GrammarItem temp\_gi = GrammarTable.GetGrammarItem(Sign[\_Index], wd.Type, this);

if (temp\_gi == null)

{

throw new Exception("语法错误:" + (counter - 1).ToString());

}

else

{

Items[\_Index] = temp\_gi;

}

}

GrammarItem git = ((GrammarItem)Items[\_Index]);

if (git.Filled())

{

//MessageBox.Show(wd.ToString());

//MessageBox.Show(this.ToString());

\_Index++;

if (this.Father == null)

{

throw new Exception("错误的结尾");

}

this.Father.InsertWord(wd,ref counter);

// this.InsertWord(wd);

}

else

{

git.InsertWord(wd,ref counter);

}

}

}

。。。