**《编译原理》**

**课程设计说明书**

****

**学 号 1652262**

**姓 名 涂远鹏**

**专 业 计算机科学与技术**

**授课老师 丁志军**

目录

[一、实验目的](#_Toc20352_WPSOffice_Level1) [3](#_Toc20352_WPSOffice_Level1)

[二、开发环境](#_Toc18522_WPSOffice_Level1) [3](#_Toc18522_WPSOffice_Level1)

[三、具体要求](#_Toc23879_WPSOffice_Level1) [3](#_Toc23879_WPSOffice_Level1)

[四、需求分析](#_Toc3253_WPSOffice_Level1) [3](#_Toc3253_WPSOffice_Level1)

[4.1程序任务输入及其范围](#_Toc18522_WPSOffice_Level2) [3](#_Toc18522_WPSOffice_Level2)

[4.2输出形式](#_Toc23879_WPSOffice_Level2) [4](#_Toc23879_WPSOffice_Level2)

[4.3程序功能](#_Toc3253_WPSOffice_Level2) [5](#_Toc3253_WPSOffice_Level2)

[4.4测试数据](#_Toc31740_WPSOffice_Level2) [6](#_Toc31740_WPSOffice_Level2)

[正确数据：](#_Toc18522_WPSOffice_Level3) [6](#_Toc18522_WPSOffice_Level3)

[4.5可行性分析](#_Toc7390_WPSOffice_Level2) [8](#_Toc7390_WPSOffice_Level2)

[五、概要设计](#_Toc31740_WPSOffice_Level1) [8](#_Toc31740_WPSOffice_Level1)

[5.1任务的分解](#_Toc30661_WPSOffice_Level2) [8](#_Toc30661_WPSOffice_Level2)

[5.2 数据类型定义](#_Toc20828_WPSOffice_Level2) [8](#_Toc20828_WPSOffice_Level2)

[5.3 主程序流程](#_Toc23780_WPSOffice_Level2) [10](#_Toc23780_WPSOffice_Level2)

[5.4模块间调用关系](#_Toc11224_WPSOffice_Level2) [11](#_Toc11224_WPSOffice_Level2)

[六、详细设计](#_Toc7390_WPSOffice_Level1) [12](#_Toc7390_WPSOffice_Level1)

[6.1 词法分析部分](#_Toc31495_WPSOffice_Level2) [12](#_Toc31495_WPSOffice_Level2)

[A. 词法规则](#_Toc23879_WPSOffice_Level3) [12](#_Toc23879_WPSOffice_Level3)

[B. 状态转换图](#_Toc3253_WPSOffice_Level3) [13](#_Toc3253_WPSOffice_Level3)

[C. 核心算法流程图](#_Toc31740_WPSOffice_Level3) [13](#_Toc31740_WPSOffice_Level3)

[D. 程序实现](#_Toc7390_WPSOffice_Level3) [14](#_Toc7390_WPSOffice_Level3)

[6.2语法分析](#_Toc26440_WPSOffice_Level2) [14](#_Toc26440_WPSOffice_Level2)

[6.2.1所用的语法如上面所述，这里不再赘述](#_Toc30661_WPSOffice_Level3) [14](#_Toc30661_WPSOffice_Level3)

[6.2.2 LR(1)算法实现详述](#_Toc20828_WPSOffice_Level3) [14](#_Toc20828_WPSOffice_Level3)

[6.2.3 数据结构](#_Toc23780_WPSOffice_Level3) [18](#_Toc23780_WPSOffice_Level3)

[6.2.4 设计思路](#_Toc11224_WPSOffice_Level3) [20](#_Toc11224_WPSOffice_Level3)

[6.2.5代码实现](#_Toc31495_WPSOffice_Level3) [22](#_Toc31495_WPSOffice_Level3)

[6.3生成中间代码和汇编代码](#_Toc11079_WPSOffice_Level2) [23](#_Toc11079_WPSOffice_Level2)

[6.3.1 四元式设计](#_Toc26440_WPSOffice_Level3) [24](#_Toc26440_WPSOffice_Level3)

[6.3.2 存储分配方案](#_Toc11079_WPSOffice_Level3) [24](#_Toc11079_WPSOffice_Level3)

[6.3.3 Mips指令语义](#_Toc17017_WPSOffice_Level3) [25](#_Toc17017_WPSOffice_Level3)

[6.3.4存储器动态管理](#_Toc6920_WPSOffice_Level3) [25](#_Toc6920_WPSOffice_Level3)

[6.3.5 四元式转汇编语言](#_Toc8153_WPSOffice_Level3) [26](#_Toc8153_WPSOffice_Level3)

[6.4代码实现](#_Toc17017_WPSOffice_Level2) [26](#_Toc17017_WPSOffice_Level2)

[6.4 中间代码优化](#_Toc6920_WPSOffice_Level2) [28](#_Toc6920_WPSOffice_Level2)

[七、调试分析](#_Toc30661_WPSOffice_Level1) [29](#_Toc30661_WPSOffice_Level1)

[1.测试结果](#_Toc8153_WPSOffice_Level2) [29](#_Toc8153_WPSOffice_Level2)

[2.时间复杂度分析](#_Toc27416_WPSOffice_Level2) [38](#_Toc27416_WPSOffice_Level2)

[3.遇到的问题及解决](#_Toc11446_WPSOffice_Level2) [38](#_Toc11446_WPSOffice_Level2)

[八、用户使用说明](#_Toc20828_WPSOffice_Level1) [39](#_Toc20828_WPSOffice_Level1)

[九、问题和解决的方法、心得体会](#_Toc23780_WPSOffice_Level1) [40](#_Toc23780_WPSOffice_Level1)

[十、参考资料](#_Toc11224_WPSOffice_Level1) [41](#_Toc11224_WPSOffice_Level1)

一、实验目的

1、掌握使用高级程序语言实现一个一遍完成的、简单语言的编译器的方法。

2、掌握简单的词法分析器、语法分析器、符号表管理、中间代码生成以及目标代码生成的实现方法。

3、掌握将生成代码写入文件的技术。

二、开发环境

编程语言:C++

使用工具: Qt Creator + mingw32编译器

操作系统:Windows 10 专业版

三、具体要求

使用高级程序语言实现一个类C语言的编译器，可以提供词法分析、语法分析、符号表管理、中间代码生成以及目标代码生成等功能。具体要求如下：

1. 使用高级程序语言作为实现语言，实现一个类C语言的编译器。编码实现编译器的组成部分。
2. 要求的类C编译器是个一遍的编译程序，词法分析程序作为子程序，需要的时候被语法分析程序调用。
3. 使用语法制导的翻译技术，在语法分析的同时生成中间代码，并保存到文件中。
4. 要求输入类C语言源程序，输出中间代码表示的程序;
5. 要求输入类C语言源程序，输出目标代码（可汇编执行）的程序。
6. 实现过程、函数调用的代码编译

其中（1）、（2）（3）（4）是必做内容，（5）（6）是选作内容。

四、需求分析

**4.1程序任务输入及其范围**

程序输入为类C语言程序，其要求为符合以下词法及文法规则：

**【词法规则】**

|  |
| --- |
| 关键字：int | void | if | else | while | return  标识符： 字母（字母|数字）\* （注：不与关键字相同）  数值：数字（数字）\*  赋值号： =  算符： + | - | \* | / | = | == | > | >= | < | <= | !=  界符： ；  分隔符： ，  注释号： /\* \*/ | //  左括号： （  右括号： ）  左大括号： {  右大括号： }  字母：| a |....| z | A |....| Z |  数字：0| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  结束符：# |

**【文法1\_包含过程调用】**

|  |
| --- |
| Program ::= <声明串>  <声明串> ::=<声明>{<声明>}  <声明> ::=int <ID> <声明类型> | void <ID> <函数声明>  <声明类型>::=<变量声明> | <函数声明> <变量声明> ::= ;  <函数声明> ::=’ (‘<形参>’) ‘<语句块>  <形参>::= <参数列表> | void  <参数列表> ::= <参数> {, <参数>}  <参数> ::= int <ID>  <语句块> ::= ‘{‘<内部声明> <语句串>’}’  <内部声明> ::= 空 | <内部变量声明>{; <内部变量声明>}  <内部变量声明>::=int <ID>  <语句串> ::= <语句>{ <语句> }  <语句> ::= <if语句> |< while语句> | <return语句> | <赋值语句>  <赋值语句> ::= <ID> =<表达式>;  <return语句> ::= return [ <表达式> ] **（注：[ ]中的项表示可选）**  <while语句> ::= while ‘( ‘<表达式> ‘)’ <语句块>  <if语句> ::= if ‘(‘<表达式>’)’ <语句块> [ else <语句块> ]**（注：[ ]中的项表示可选）**  <表达式>::=<加法表达式>{ relop  <加法表达式> }  **（注：relop-> <|<=|>|>=|==|!=）**  <加法表达式> ::= <项> {+ <项> | -<项>}  <项> ::= <因子> {\* <因子> | /<因子>}  <因子> ::=num | ‘(‘<表达式>’)’ |<ID> FTYPE  FTYPE ::= <call> | 空  <call> ::=’(’<实参列表> ’) ’  <实参> ::=<实参列表> | 空 <实参列表> ::= <表达式>{, <表达式>}  <ID>::=字母(字母|d数字)\* |

**4.2输出形式**

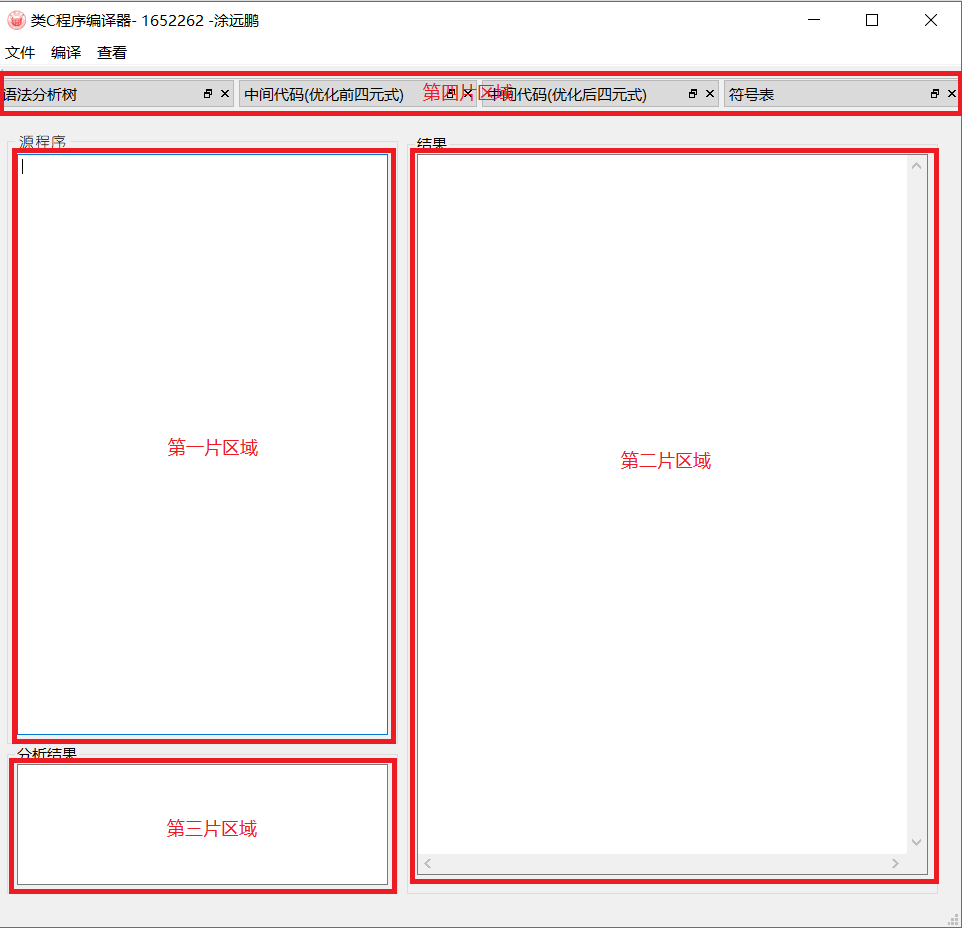
输出形式包含两种，屏幕输出和文件输出：

1、屏幕输出包含词法分析结果（符号表由tablewidget展示显示在主窗体中），语法分析结果（语法分析树由treewidget展示完整结构,在DockWidget浮动窗体中显示），中间代码生成结果（四元式表由tablewidget展示,在DockWidget浮动窗体中显示），优化后的中间代码生成结果（优化后四元式表由tablewidget展示，,在DockWidget浮动窗体中显示），目标代码生成结果（汇编代码由textBrowser文本框展示,在DockWidget浮动窗体中显示）。

2、输出到文件的格式包含三种，第一种为词法分析结果，第二为中间代码结果，第三个为目标代码结果，由于语法生成树无法输出到文件，所以只能在屏幕上以treewidget的形式展示。

**4.3程序功能**

程序主界面如下图所示：



程序界面分为四个区域。图中第一片区域是代码编辑区，打开或新建的代码在该区域进行查看和编辑；第二片区域是编译结果输出文本框，该区域可以显示词法分析、目标代码的生成结果；第三片区域是编译提示区，该区域可以显示编译的结果，在编译错误的情况下显示相关的错误信息。第四片区域为四个悬浮窗口，分别显示语法分析结果（语法分析树）、优化前的四元式、优化后的四元式、符号表。

通过按键对程序功能的描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 选项 | | 功能 |
| 文件 | 打开 | 打开已有的.c/.txt/.cpp文件 |
| 新建 | 新建源代码 |
| 退出 | 退出程序 |
| 保存 | 将源代码保存到当前路径 |
| 关闭 | 关闭打开的源代码 |
| 另存为 | 将源代码保存到另一指定路径 |
| 编译 | | 对源代码进行编译并生成相关文件 |
| 查看 | 词法分析 | 查看词法分析的结果 |
| 语法分析 | 查看语法分析的结果 |
| 优化前中间代码 | 查看生成的中间代码 |
| 目标代码 | 查看生成的目标代码 |
| 优化后中间代码 | 查看优化后的四元式表 |
|  | 符号表 | 查看LR生成的符号表 |

**4.4测试数据**

**正确数据：**

|  |
| --- |
| 类C语言程序实例（包含过程调用）：  **int a;**  **int b;**  **int program(int a,int b,int c)**  **{**  **int i;**  **int j;**  **i=0;**  **if(a>(b+c))**  **{**  **j=a+(b\*c+1);**  **}**  **else**  **{**  **j=a;**  **}**  **while(i<=100)**  **{**  **i=j\*2;**  **j=i;**  **}**  **return i;**  **}**  **int demo(int a)**  **{**  **a=a+2;**  **return a\*2;**  **}**  **void main(void)**  **{**  **int a;**  **int b;**  **int c;**  **a=3;**  **b=4;**  **c=2;**  **a=program(a,b,demo(c))**  **return;**  **}** |

**错误数据（错误原因：int i缺少‘；’结束符）：**

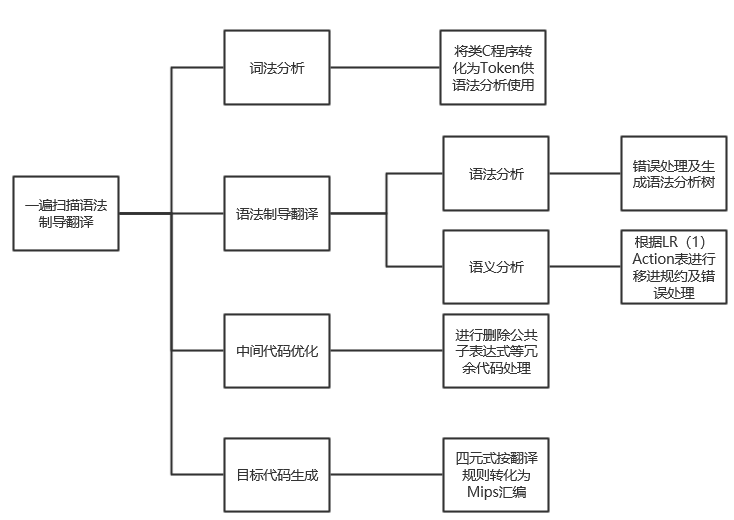
|  |
| --- |
| **int a;**  **int b;**  **int program(int a,int b,int c)**  **{**  **int i**  **int j;**  **i=0;**  **if(a>(b+c))**  **{**  **j=a+(b\*c+1);**  **}**  **else**  **{**  **j=a;**  **}**  **while(i<=100)**  **{**  **i=j\*2;**  **j=i;**  **}**  **return i;**  **}**  **int demo(int a)**  **{**  **a=a+2;**  **return a\*2;**  **}**  **void main(void)**  **{**  **int a;**  **int b;**  **int c;**  **a=3;**  **b=4;**  **c=2;**  **a=program(a,b,demo(c))**  **return;**  **}** |

**4.5可行性分析**

由于上学期编译原理已经使用LR（1）的分析方法完成了类C语言程序的词法分析、语法分析、语义分析，并且使用LR(1)分析方法进行编译器制作已有众多前辈制作成功，汇编代码生成方面，由四元式生成汇编代码可由第十一章目标代码生成中的理论实现所以认为技术方面是可行的。经济方面由于是本地开发的类C编译器，无需借助外部程序所以也认为是可行的。

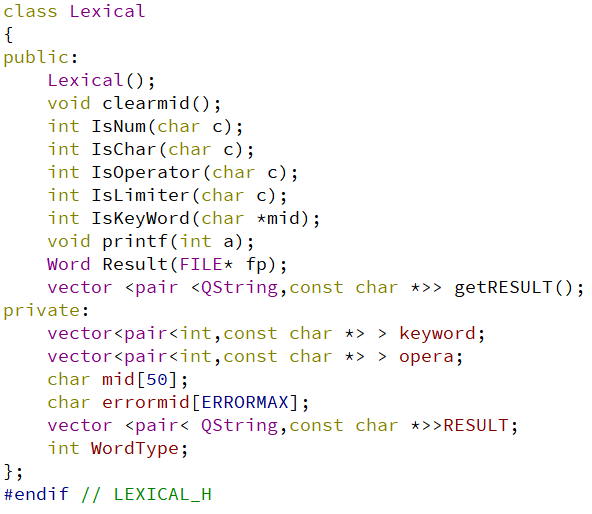
五、概要设计

5.1任务的分解



5.2 数据类型定义

词法分析类：



语法分析类包含语法分析函数及变量、语义分析函数及变量：





目标代码生成类：

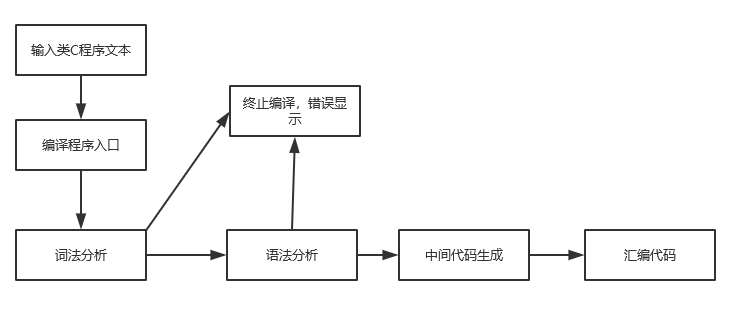


5.3 主程序流程

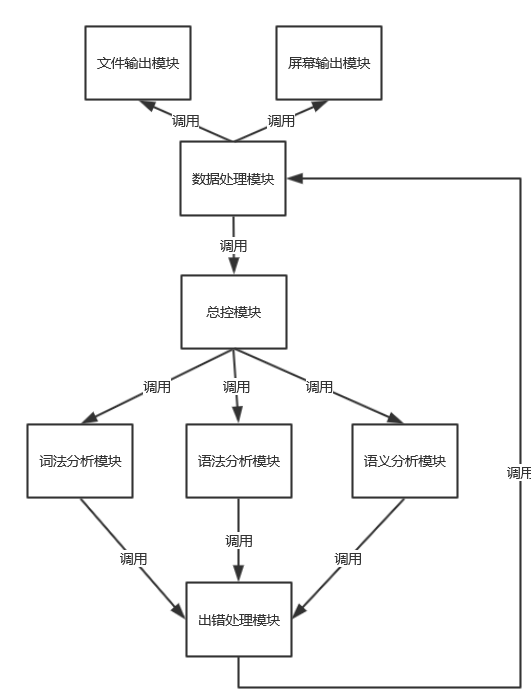
类C语言程序编译器的程序流程图大致如下所示



整体框架如下：



5.4模块间调用关系



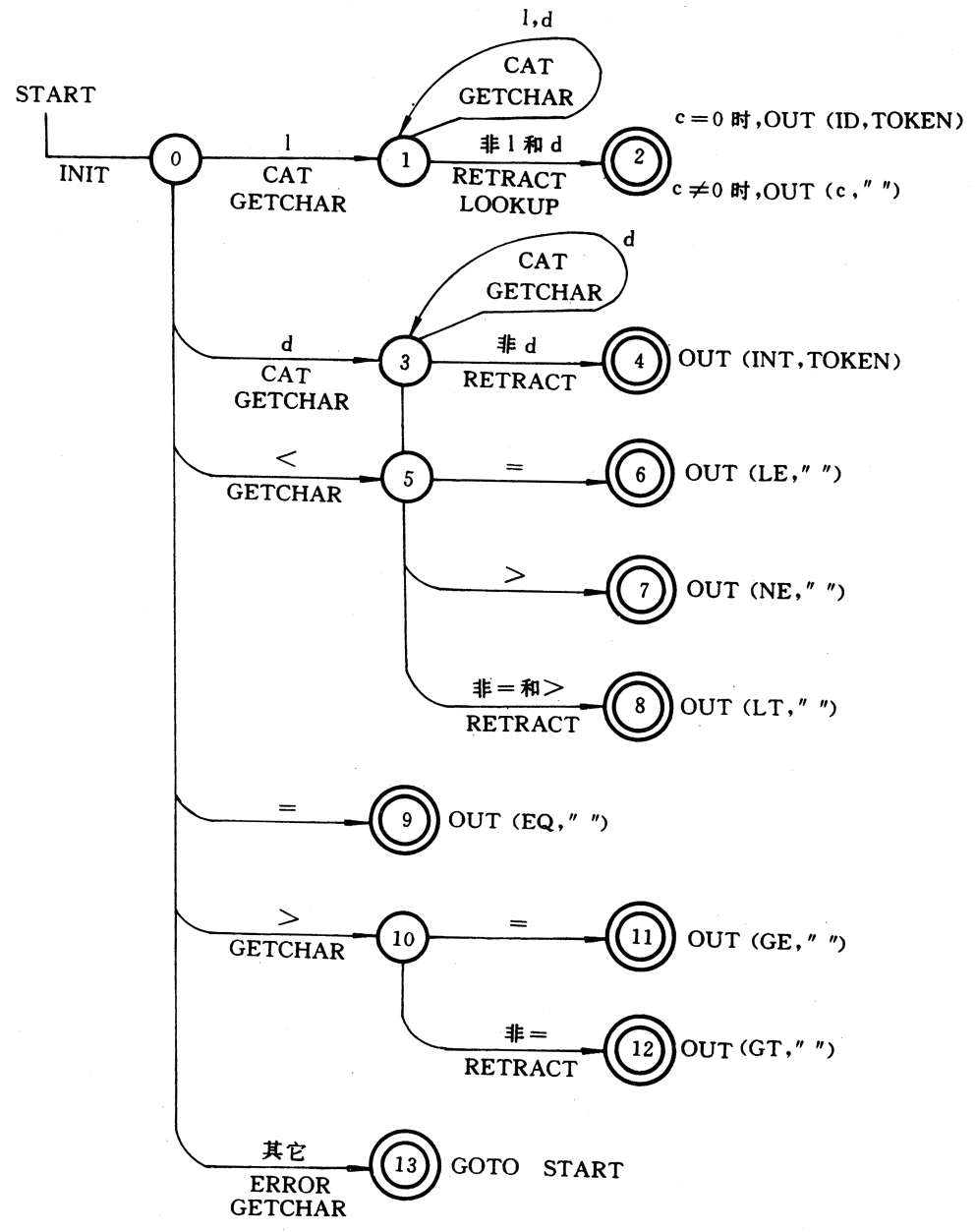
六、详细设计

### **6.1 词法分析部分**

1. 词法规则

|  |  |
| --- | --- |
| 词项 | 编码 |
| int | 1 |
| void | 2 |
| if | 3 |
| else | 4 |
| while | 5 |
| return | 6 |
| ID | 7 |
| NUM | 8 |
| = | 9 |
| + | 10 |
| - | 11 |
| \* | 12 |
| / | 13 |
| repo | 14 |
| ; | 15 |
| , | 16 |
| ( | 17 |
| ) | 18 |
| { | 19 |
| } | 20 |
| # | -1 |
| NONE | 0 |

1. 状态转换图

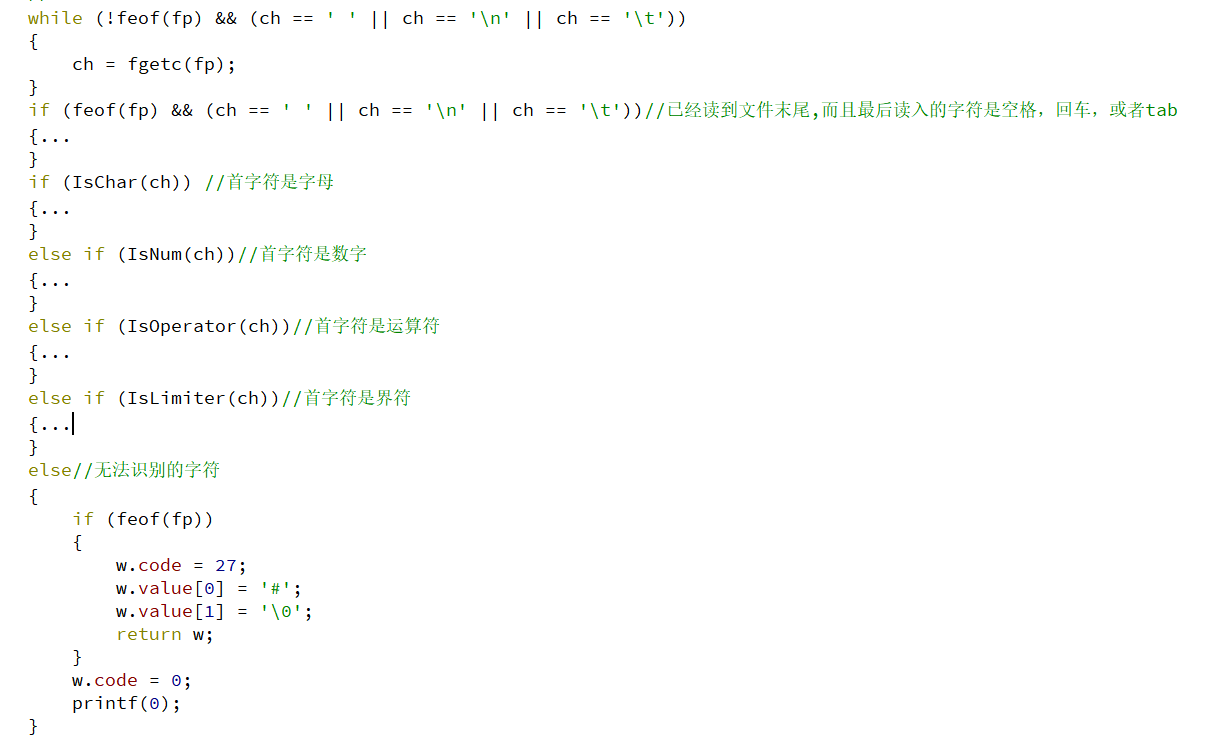


1. 核心算法流程图



1. 程序实现

由于代码较多，所以此处只显示大体框架：

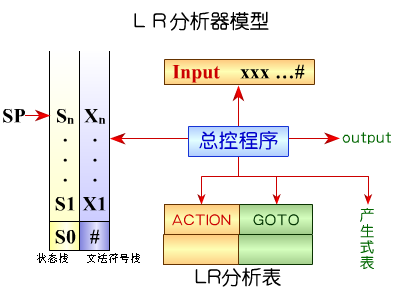


### **6.2语法分析**

### 6.2.1所用的语法如上面所述，这里不再赘述

6.2.2 LR(1)算法实现详述

基本原理如下：

****

6.2.2.1求First集算法

|  |
| --- |
| 连续使用下面的规则，直到每个集合FIRST不再增大为止：  （1）若X属于VT，则FIRST(X)={X}。  （2）若X属于VN，且有产生式X→a…，则把A加入到FIRST(X)中；若X→ ξ也是一条产生式，则把ξ也加入到FIRST中。 |

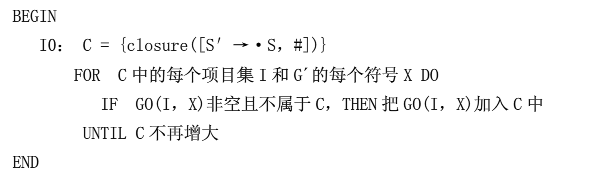
6.2.2.2构造CLOSURE(I)算法

|  |
| --- |
| （1）I中的所有项目都属于CLOSURE(I)；  （2）若项目[A→a▪Bβ,a]属于CLOSURE(I)，B→ξ是文法的一个产生式，则对于任何b∈FIRST(βa)，如果[B→▪ξ,b]原来不在CLOSURE(I)中，则把它加入CLOSURE(I)；  （3）重复执行步骤（2），直到CLOSURE(I)不再增大为止 |

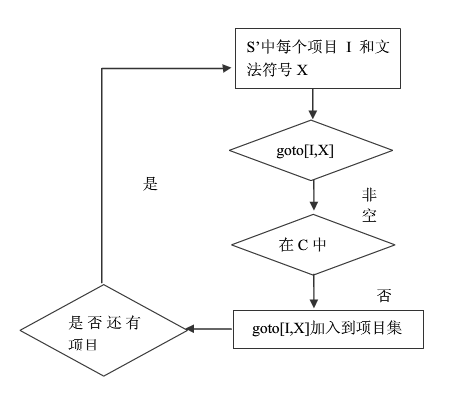
6.2.2.3 LR(1)项目集族C构造步骤：

|  |
| --- |
| 1、建立有限状态自动机DFA、哈希表H、项目集队列P，放入初始项目集I0，存入哈希表中并作为DFA的初始状态。  2、取出队首元素I，对于I的每个项目X，求I’=GO(I,X)，若I’不在哈希表中，则将其加入P和H中，并添加为DFA新的状态。为DFA添加一条边(I,X,I’)。  3、循环此操作直到P为空为止，DFA即代表了文法G的LR(1)项目集族。 |

伪代码：



流程图：



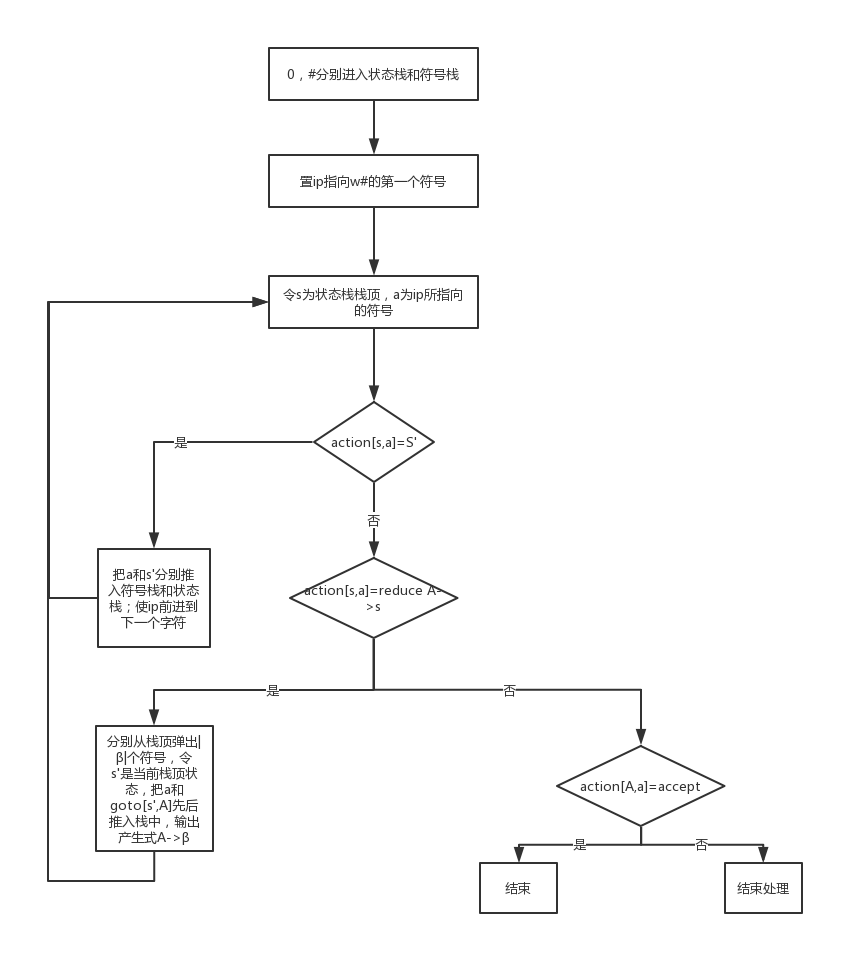
6.2.2.4 GO(I,X)算法计算算法

|  |
| --- |
| GO(I,X)=CLOSURE(J)其中J={任何形如[A→aX.Β,a]的项目[A→a.X.Β,a]属于I} |

6.2.2.5 LR(1)预测分析表生成算法

|  |
| --- |
| 1、若项目[A→▪a, b]属于Ik且GO(Ik,a)＝Ij，a为终结符，则置ACTION[k, a]为“sj”。  2、若项目[A→▪a]属于Ik，则置ACTION[k,a]为“rj”；其中假定A→为文法G的第j个产生式。  3、若项目[S→S▪,#]属于Ik，则置ACTION[k,#]为“acc”。  4、若GO(Ik，A)＝Ij，则置GOTO[k, A]=j。  5、分析表中凡不能用规则1至4填入信息的空白栏均填上“出错标志”(即将表中空白格均置为“报错标志”)。  在实现GO(I,X)时，记录下状态的转化。得到分析表中的移进部分。然后再扫描所有的项目集，找到其中包含归约项目的哪些项目集，根据其中项目，得到分析表中那些规约的部分。 |

6.2.2.6 LR（1）驱动程序流程图：





6.2.3 数据结构

|  |  |  |
| --- | --- | --- |
| yfq类 | | |
| 变量名 | 类型 | 作用 |
| end\_flag | int | 程序结束标志符 |
| read\_main | int | 是否已读过main函数 |
| num\_zdkh | int | 左大括号数量，用于判断是否为全局变量 |
| return\_flag | int | 是否为返回语句 |
| return\_words[100] | char | 返回字数组 |
| c | char | 临时读入变量 |
| bl\_name[100] | char | 临时记录变量名 |
| hs\_name[100] | char | 临时记录函数名 |
| len\_hs | int | 从函数名到左括号的长度 |
| avaliable\_T | int | Ti,i=avaliable\_T |
| avaliable\_L | int | Li,i=avaliable\_L |
| var\_name[10][100] | char | 记录参数名 |
| kh[100] | int | 记录读到的右括号 |
| len\_kh | int | 从函数到右括号长度 |
| headwords[100] | char | 句首字符 |
| is\_hs\_head | int | 是否函数居于句首标志 |
| is\_hs\_dy | int | 是否为函数定义 |
| len\_hs | int | 函数信息表长度 |
| break\_hs | int | 在函数名数组中搜索给定函数名时断开的数 |
| hs[500] | hs\_info | 函数信息表数组 |
| len\_global\_bl | int | 全局变量长度 |
| global\_bl[500] | bl\_info | 全局变量名数组 |
| len\_local\_bl | int | 局部变量长度 |
| local\_bl[500] | bl\_info | 局部变量名数组 |
| fin | ifstream | 类C语言程序文件句柄 |
| fin1 | ifstream | 文法文件句柄 |
| fout1 | ofstream | 中间代码文件句柄 |
| fout2 | ofstream | 汇编代码文件句柄 |
| 函数名 | 类型 | 作用 |
| zs\_handle() | bool | 判断是否为有效注释 && 去除注释 |
| u\_handle() | bool | 因子处理函数 |
| s\_handle() | bool | 加法表达式处理函数 |
| exist\_bl(char name[],int mode) | bool | 判断变量名是否存在,mode=1: 局部变量 mode = 0: 全局变量 |
| exist\_hs(char name[],int var\_num) | bool | 判断函数名是否存在 && 参数 数目是否正确 |
| insert\_bl(char name[],int mode) | void | 将定义的变量插入 global\_bl[]， local\_bl[] ，mode 0->全局变量 1->局部变量 |
| insert\_hs(char name[],int var\_num) | void | 将定义的函数插入 hs[]，var\_num：参数 |
| delete\_bl() | void | 读到'}'删除该区间读到的变量，'{'开始（并生成中间代码） |
| right\_num(char c[],int len\_c) | bool | 判断是否为合法数字 |
| fh\_rank(char fh) | int | 计算符号等级 |
| r\_handle(char c[],int len\_c,int mode) | bool | 表达式处理函数 |
| bl\_handle | bool | 句首为变量的处理函数 |
| void\_handle | bool | 句首为 void的处理函数 |
| int\_handle | bool | 句首为 int 的处理函数 |
| get\_kind(char c[],int len\_c) | int | 合法变量名，合法数字 |
| error\_rep() | void | 报错及相关处理函数 |
| readhead() | int | 读句首 |
| analyze() | void | 整体分析函数 |

|  |
| --- |
| struct bi\_info{//记录变量结构体  char bl\_name[20];  int pos;  int num;  }; |

|  |
| --- |
| struct hs\_info{ //记录函数的数据结构  char hs\_name[20];  int f\_return;  char w\_return[20];  int var\_num;  }; |

全局变量：

|  |
| --- |
| int available\_reg[32]; //有效寄存器号数组  int max\_num; //reg1 恒定值为 0，全局变量从$31向前分配寄存器， max\_num为全局变量使用的最小的寄存器 |

6.2.4 设计思路

6.2.4.1总体设计架构

本程序使用了 LR1 的设计思想，并做了较大的改动。 我最初的想法是严格按照 LR1 算法的思想，以 program 为 S’，并根 据状态转换图画出 action表。但是，在实现的过程中，发现这样的问 题很大。首先，类 c 语法有 26 个产生式，要准确画出全部的状态转换图是十分困难的。其次，这种方式也是十分不必要的。比如，当程序读到了 int，那么接下来的步骤我们可想而知是继续读，反映到 LR1 算法中，就是当前状态i 与当前待输入字符，对应的为 Sj。而若读到 了’;’，则对应的一定为 rj。所以当读到’;’、’}’等就一定是指向规约操 作，而读到其他的就会继续读入。所以，该程序以 LR1 算法思想为指 导，并根据对 c 语言语法的理解。

6.2.4.2 变量和函数声明的处理

Void 关键字一定是函数声明。Void\_handle()继续读入 void 后序 部分，直到’{‘，若 exist\_hs()==yes||exist\_hs()==error 说明已有 该函数名，或函数名/参数名 命名错误。会报错，因为该编译器不支 持函数重载。不然，将函数名和参数数目写入数组Hs\_info hs[]。无 参函数形式为 函数名( )，不是（void）。函数是否需要返回标志 f\_return=0;（只能根据定义时是 int 还是void 来判断，意味着如果 void 函数 有返回值或int 函数无返回值这种错误是没办法发现的）。

Int 关键字可能是函数定义，也可能是变量定义，其中变量定义还分为局部变量定义和全局变量定义。先判断是函数定义还是变量定义，从 int 后第一个非’ ‘的字符开始读到)或者；。然后判断方法及 操作同 void。如果是变量，根据f\_inside(初始化为0，每读到一个’{‘， f\_inside++；读到一个’}’，f\_inside--)是否为 0 判断此时是否为全局 还是局部变量。如果是全局变量，并且exist\_bl(变量名，0) == no 则插入全局变量数组；如果是局部变量，并且 exist\_bl(变量名， 1)== no，则插入局部变量数组。尤其是，当读到了’{‘，也需要插入到局部变量数组。

变量声明的过程不会有寄存器的分配，只有在使用变量的过程中才会有寄存器分配。

当给全局变量赋值时，会将该值存储在内存中。而当使用该变量时（等号右边），则会为该变量分配存寄存器。从$31开始依次向前 分配寄存器。其 pos置 2。当为局部变量赋值时，会将该变量的值存储在内存中，当使用该变量的时候，会从内存中取出该值分配给寄存器。寄存器分配过程：从 available\_reg 中取最小的为 1 的值 j，并把 $j分配给该变量，同时，该变量的 pos 置 2。

|  |
| --- |
| for(int j=1;j<max\_reg;j++){  if(avaliable\_reg[j]==1)  break;  } |

当读到了’}’，则会在局部变量栈中持续退栈，直至第一个’{‘也退栈，该过程即为局部变量的释放。

6.2.4.3 函数调用处理

函数调用若有参数，则需要先把参数入栈（将实参的 pos、num赋予形参），类似于传地址的参数传递方式。如果函数有返回值，则先传值，再return；否则，直接 return。

6.2.4.4 表达式expression处理

对有小括号的情况处理的不好，经常会程序中断，没有解决。将整个表达式 如：if( r )，则 r 为表达式 读入并传递给 r\_handle()， r\_handle()根据<,>,=进行拆分，如 s1<s2，则将 s1,s2 分别传递给 s\_handle()。r\_handle()扫描，确定\* /(rank = 2)和 + -(rank = 1)的 数目 num1,num2，将栈和 num1，num2 传递给 s\_handle().然后 s\_handle()利用栈和递归函数的方法最终得到一个空栈：

如有 s1 = a+b\*c+d 则初始的符号栈为 a+b\*c+d, 读到\*，则将b\*c退栈，Ti入栈，num2--(表示rank=2的数目)， 并将栈 a+Ti+b ,num1,num2 传递给下一个调用的 s\_handle()。

6.2.4.5 if,while条件语句处理

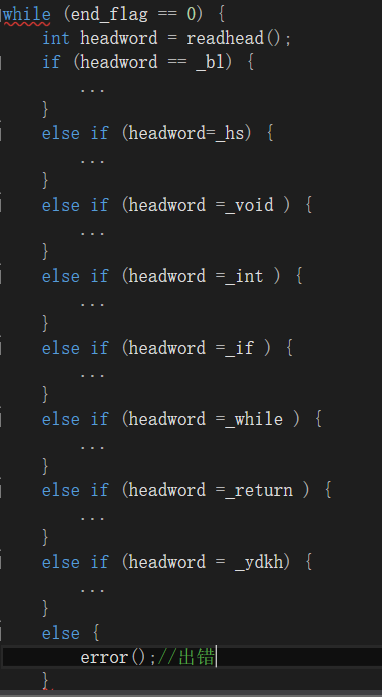
if 和 while 翻译成中间代码的形式分别为：

|  |  |
| --- | --- |
| if | L0:(jx,a,b,L2)  L1:(j,-,-,$)  L2:(-,-,-,-)  ......  Li: |
| while | L0:(jx,a,b,L2)  L1:(j,-,-,$)  L2:(-,-,-,-)  ......  Li-1:(j,-,-,L0)  Li: |

$表示待定的地址，x 表示相应的操作符。因为无法在第一遍扫描即 得知$的值，所以，要先把字符串”L0”、 ”:”、 ”(“、 ”jx”、 ”,”、 ”a”...”$”、 ”:” 读到一个二维字符数组 s[500][20]。然后，当$:确定后，再把 s[][] 写入中间代码文件，其中$用最后的地址(Li)来替代。

6.2.4.6结束语句处理

读到 main 函数后， read\_main=1;f\_kh 初始化为 0，每读到一个”{“， f\_kh++，读到一个”}”f\_kh--。若 read\_main = 1 && f\_kh == 0，程序结束。大题程序框架如下：



6.2.5代码实现

由于代码量较大，所以此处以变量定义和函数定义的语法分析为例展示：



### **6.3生成中间代码和汇编代码**

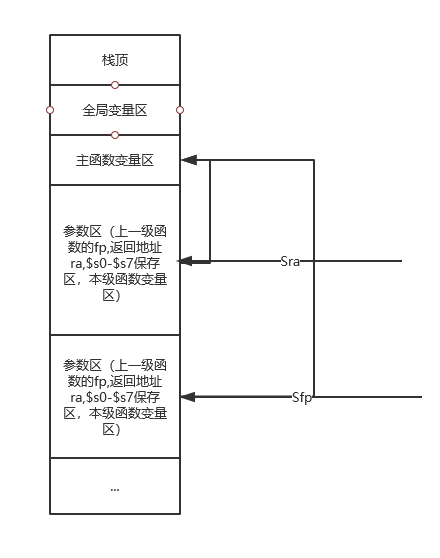
### 由于使用的是语法制导，所以在上面语法检测分析的同时，中间代码和目标代码已生成：

|  |
| --- |
| 扩充分析栈：增设语义栈  1.在语法分析的基础上，增加一个语义栈，用于存放分析栈中文法符号所对应的属性值，栈内元素为语义结点。   1. 结点类是S属性文法的表示，判别每次语法分析所使用的产生式，实现不同的语义动作，每当规约到特定非终结符时，即可产生中间代码。   扩充LR分析器功能：当执行规约产生式动作时，也执行产生式对应的语义动作。并且由于是归约时执行语义动作，限制语义动作仅能放在产生式右部的最右边。  符号表和函数表：每次规约识别出一个新的标识符，都会将其加入符号表中，符号的信息包括标识符、中间变量名、类型、占用空间、内存偏移量、作用的函数等。而当规约到函数定义的时候，则将函数名、形参列表、代号加入函数表，如下图所示：    中间代码生成：将语句分为赋值语句、算术运算语句、函数调用语句、循环语句、选择语句、跳出语句、函数定义语句分别处理。 |

6.3.1 四元式设计

|  |  |
| --- | --- |
| 四元式 | 含义 |
| (=,2,,temp) | temp=2; |
| (call,f,,a) | a=f(); |
| (call,f,,) | f(); |
| (<=..,a,b,) | a<=b; |
| (jne,,,label) | if not satisfy(==false) then jump |
| (jmp,,,label) | jump to label |
| ret | return |
| Par a | f(int a,---) |

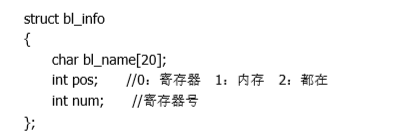
6.3.2 存储分配方案



6.3.3 Mips指令语义

|  |  |  |
| --- | --- | --- |
| 含义 | mips指令 | 表达式 |
| 取立即数 | li $s1,100 | $s1=100 |
| 加 | add $s1,$s2,$s3 | $s3=$s1+$s2 |
| 立即数加 | addi $s1,$s2,100 | $s3=$s1+100 |
| 立即数减 | subi $s1,$s2,100 | $s3=$s1-100 |
| 减 | sub $s1,$s2,$s3 | $s3=$s1-$s2 |
| 乘 | mult $s2,$s3 | Hi,Lo=$s2\*$s3 |
| 除 | div $s2,$s3 | Lo=$s2/$s3  Hi=$s2 Mod $s3 |
| 取字 | lw $s1,100($s1,100($s2)) | $s1=memory[$s1=memory[$s2+100] |
| 存字 | sw $s1,100($s1,100($s2)) | memory[$s2+100]=$s2+100]=$s1 |
| Beq | beq $s1,$s2,100 | if($s1==$s2) goto PC+4+400 |
| Bne | bne $s1,$s2,100 | if($s1!=$s2)goto PC+4+400 |
| Slt | slt $s1,$s2,$s3 | if($s2<$s3)$s1=1;  else $s1=0; |
| j | j 2000 | Goto 8000; |
| jr | ja $ra | Goto $ra; |
| jal | jal 2500 | $ra=PC+4;  Goto 10000; |

6.3.4存储器动态管理



当给全局变量赋值时，会将该值存储在内存中。而当使用该变量时（等号右边），则会为该变量分配存寄存器。从$31开始依次向前 分配寄存器。其 pos置 2。当为局部变量赋值时，会将该变量的值存储在内存中，当使用该变量的时候，会从内存中取出该值分配给寄存器。寄存器分配过程：从 available\_reg 中取最小的为 1 的值 j，并把 $j分配给该变量，同时，该变量的 pos 置 2。

当读到了’}’，则会在局部变量栈中持续退栈，直至第一个’{‘也退栈，该过程即为局部变量的释放。

6.3.5 四元式转汇编语言

对于四元式(op,num1,num2,result):

**当 op===：**

若 num1 为数，则操作为 li $t0 2 sw $t0 -24($fp) 若 num2 为变量，则先把 num2 移动到寄存器，再进行赋值操作： sw $t0 -8($fp) li $t0 2 sw $t0 -24($fp)

**当 op == + - \* / ：**

1、若 num1、num2 为数，则只需要再分配一个寄存器，将 num1 的值存储进寄存器，op $1 num2

2、若 num1、num2 一个为数，一个为变量，则为变量分配寄存器，Op $1 num2，并置 bl.pos = 0，表示该变量的值只存在于内存中。记录新变量的属性。

3、若 num1、num2 都为变量，则为这两个变量都分配寄存器，然后，Op $1 $2 同时置 bl1.pos = 0，记录新变量的属性。

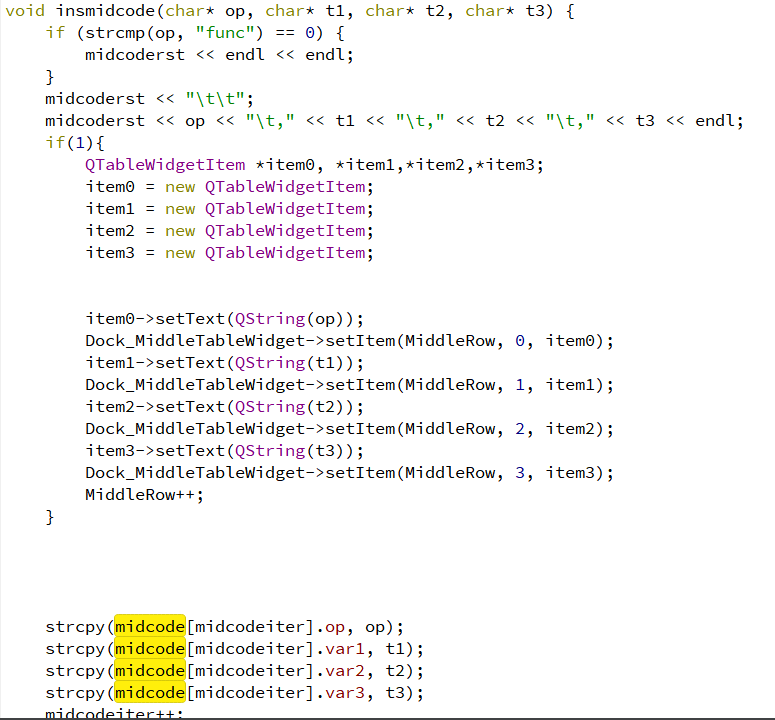
**当op为call，ret，if、while，jne：** 对应bne, j、jnz、jne、jg、jge、jl、jle 等跳转汇编指令。

**当op为int,para等变量定义语句**： 对应li,sw等存储相关汇编指令。

**当op为func等函数定义语句：** 对应生成label标签汇编指令。

6.4代码实现

生成中间代码(部分)：

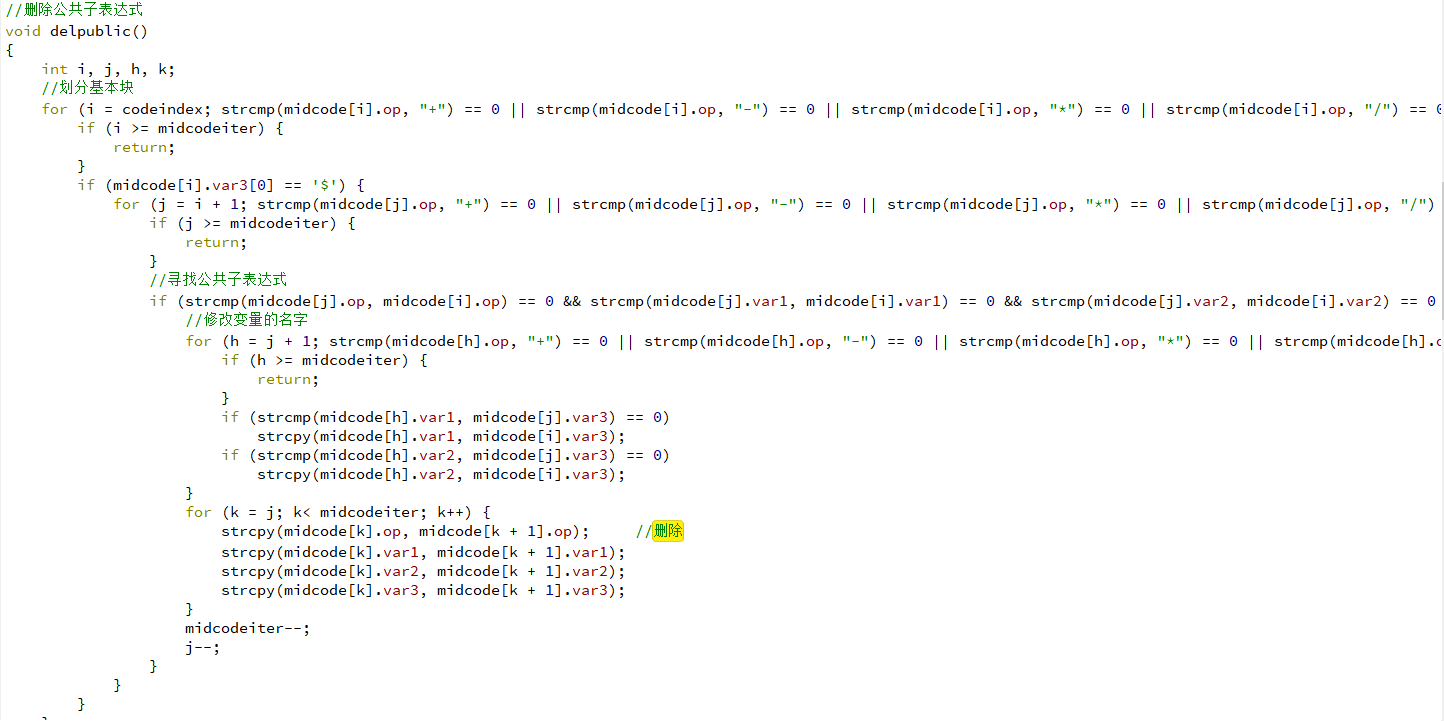
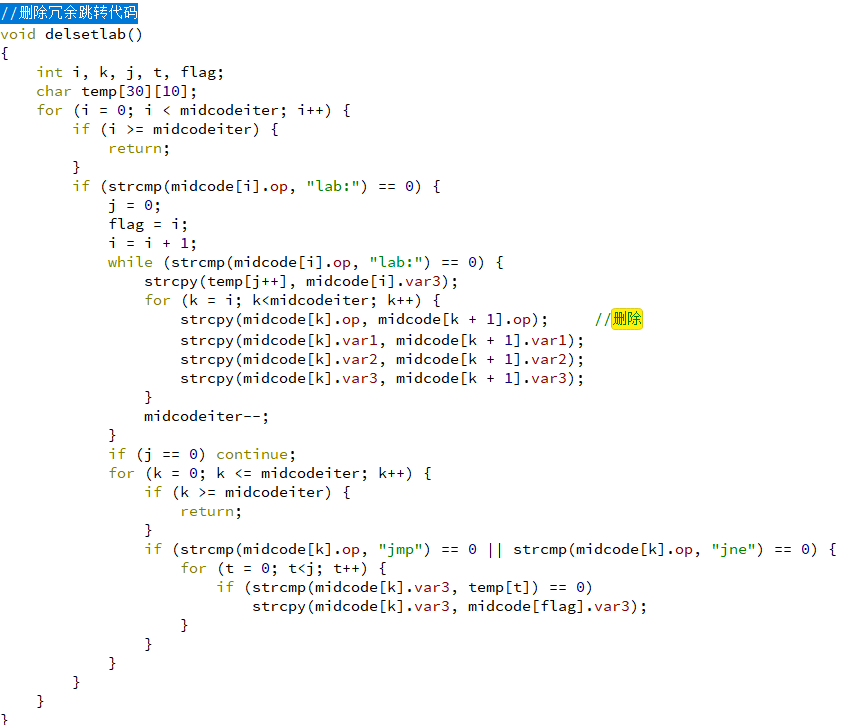


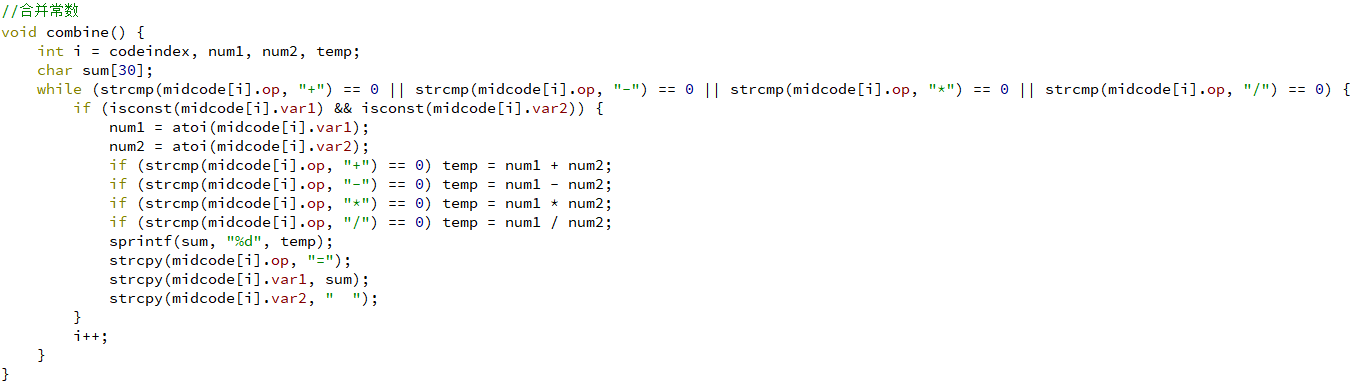
汇编代码生成：



### **6.4 中间代码优化**

此处仅做了三项优化，分别是删除无用跳转中间代码和删除公共子表达式、合并常数变量，具体代码如下：



七、调试分析

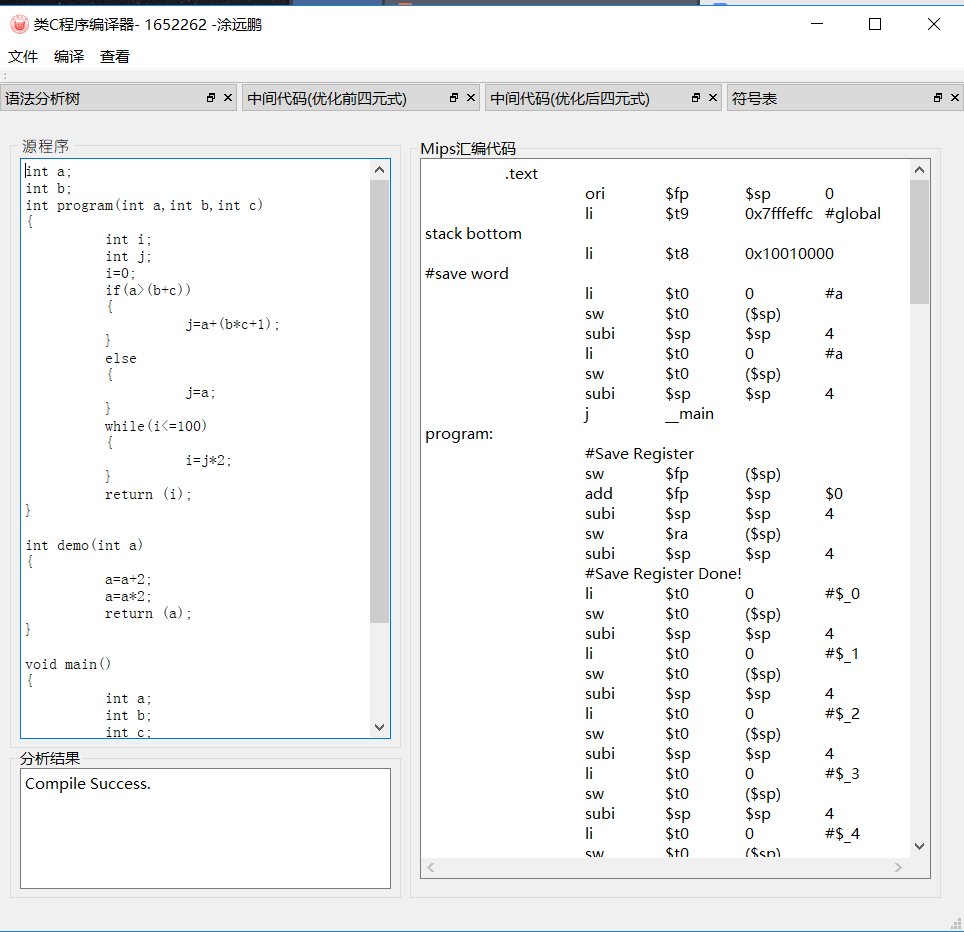
**1.测试结果**

正确数据测试：

类C语言源程序如下：

|  |
| --- |
| int a;  int b;  int program(int a,int b,int c)  {  int i;  int j;  i=0;  if(a>(b+c))  {  j=a+(b\*c+1);  }  else  {  j=a;  }  while(i<=100)  {  i=j\*2;  }  return (i);  }  int demo(int a)  {  a=a+2;  a=a\*2;  return (a);  }  void main()  {  int a;  int b;  int c;  a=3;  b=4;  c=2;  a=program(a,b,demo(c));  return;  } |

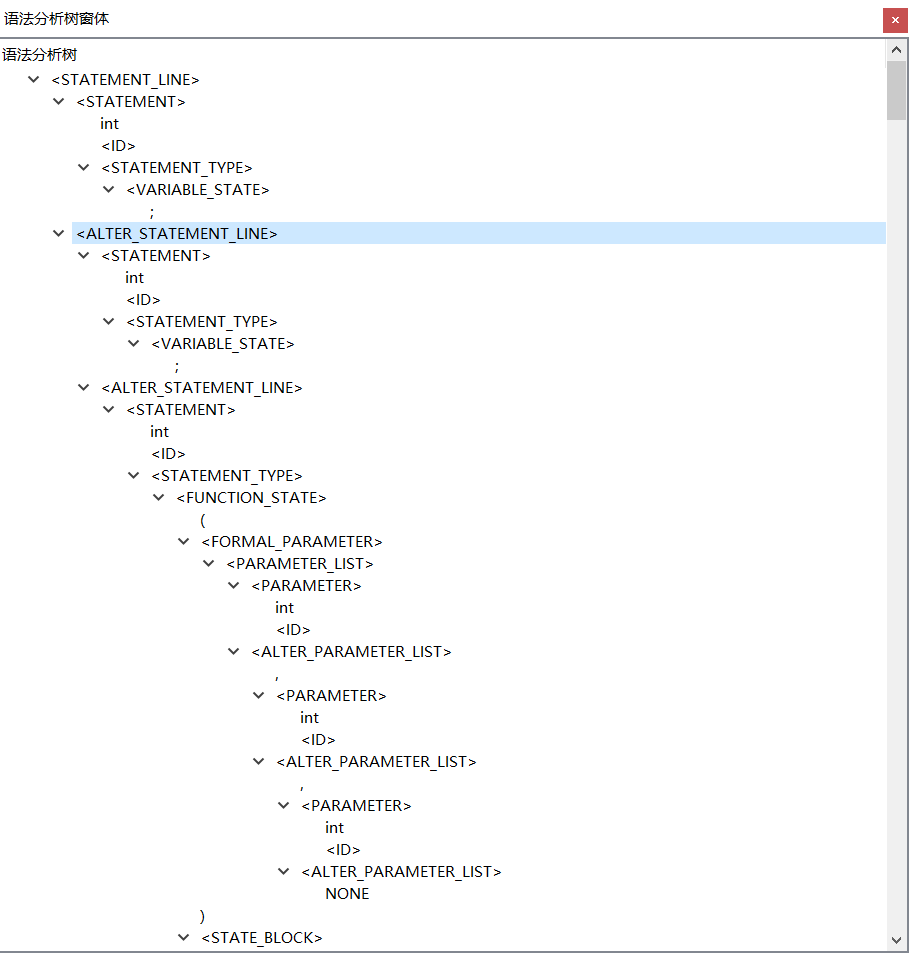
输入正确的类C源文件，输出显示“分析成功”，可以显示四种分析结果，并且在Windows底下Mars4\_5.jar运行，成功运行：



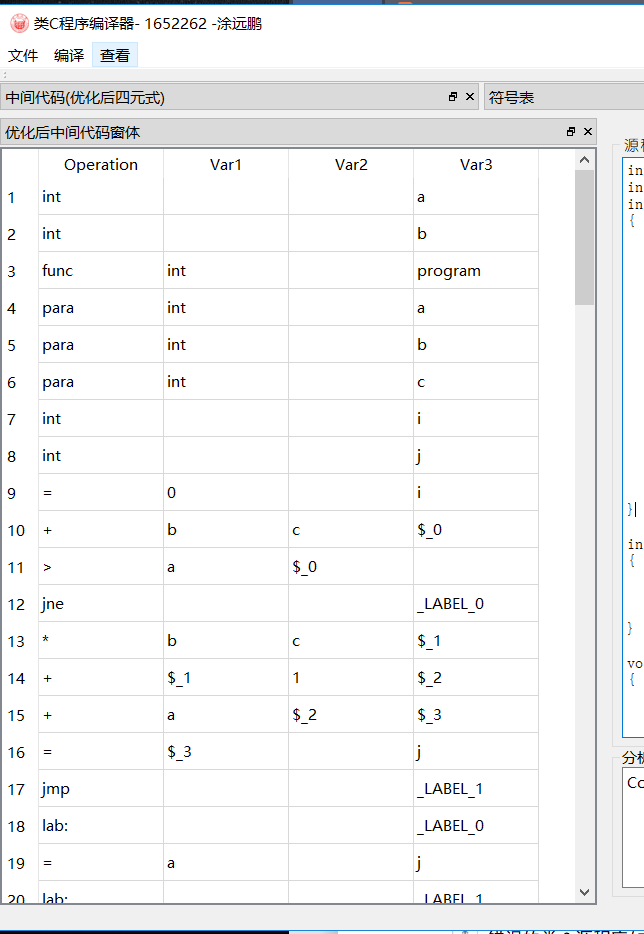
生成的词法分析表：



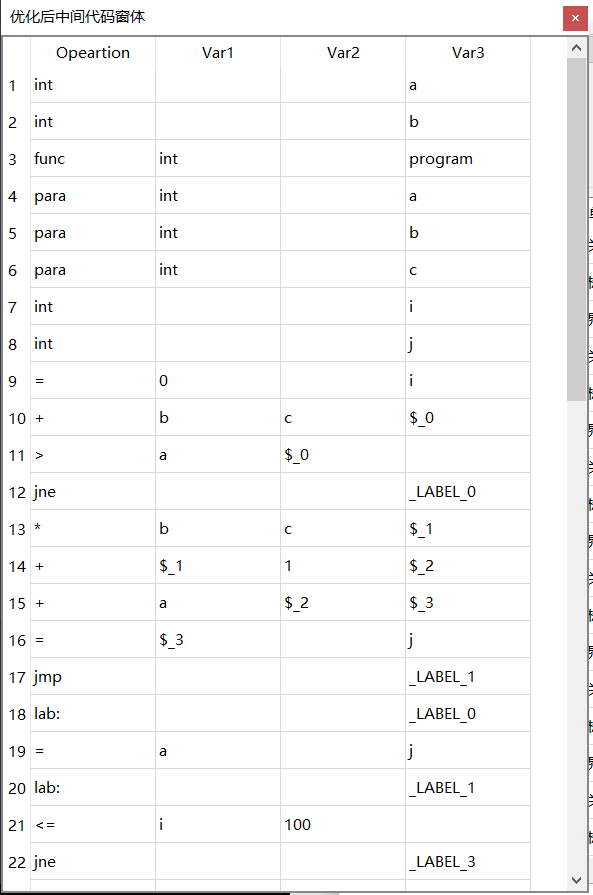
生成的语法分析树：



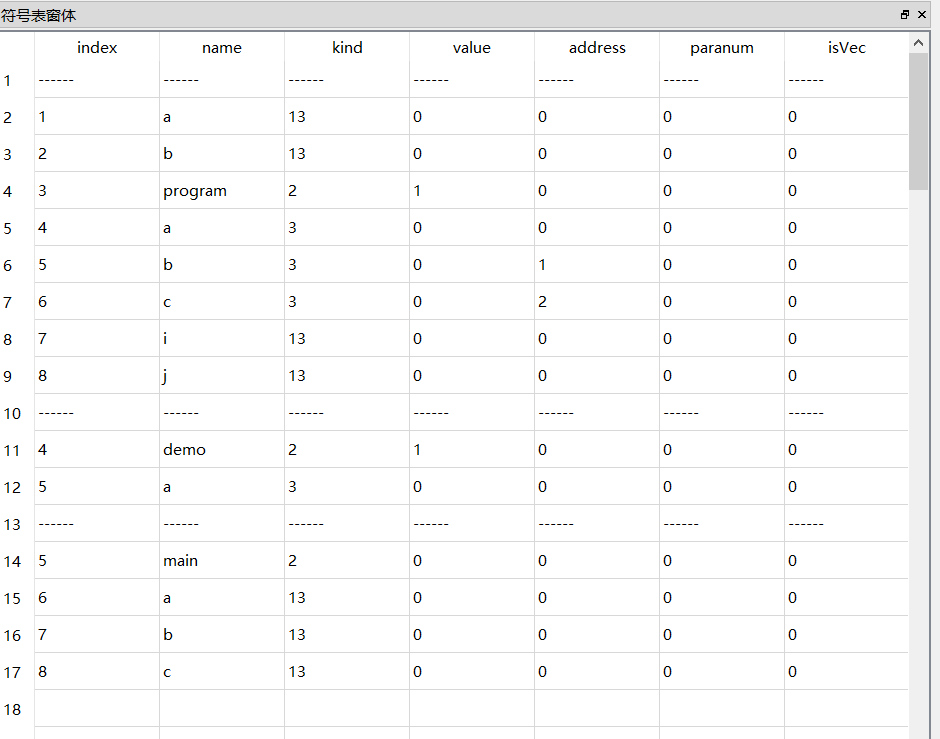
生成的优化前的中间代码（四元式表）：



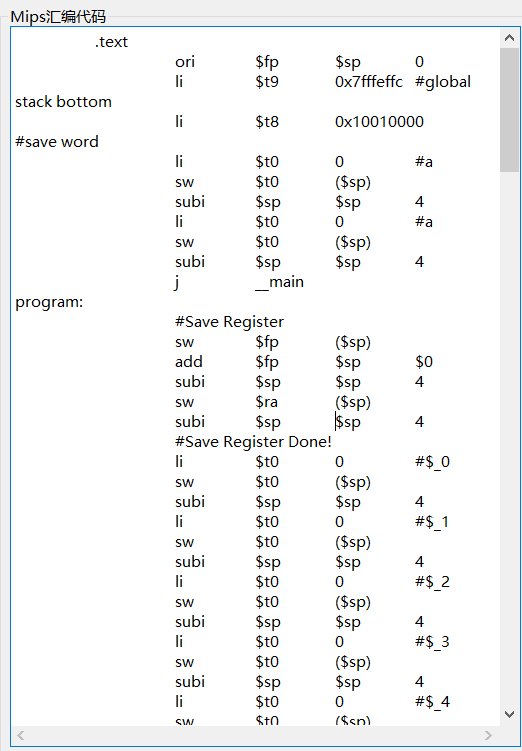
生成的优化后的中间代码（四元式表）：



LR(1)的符号表：

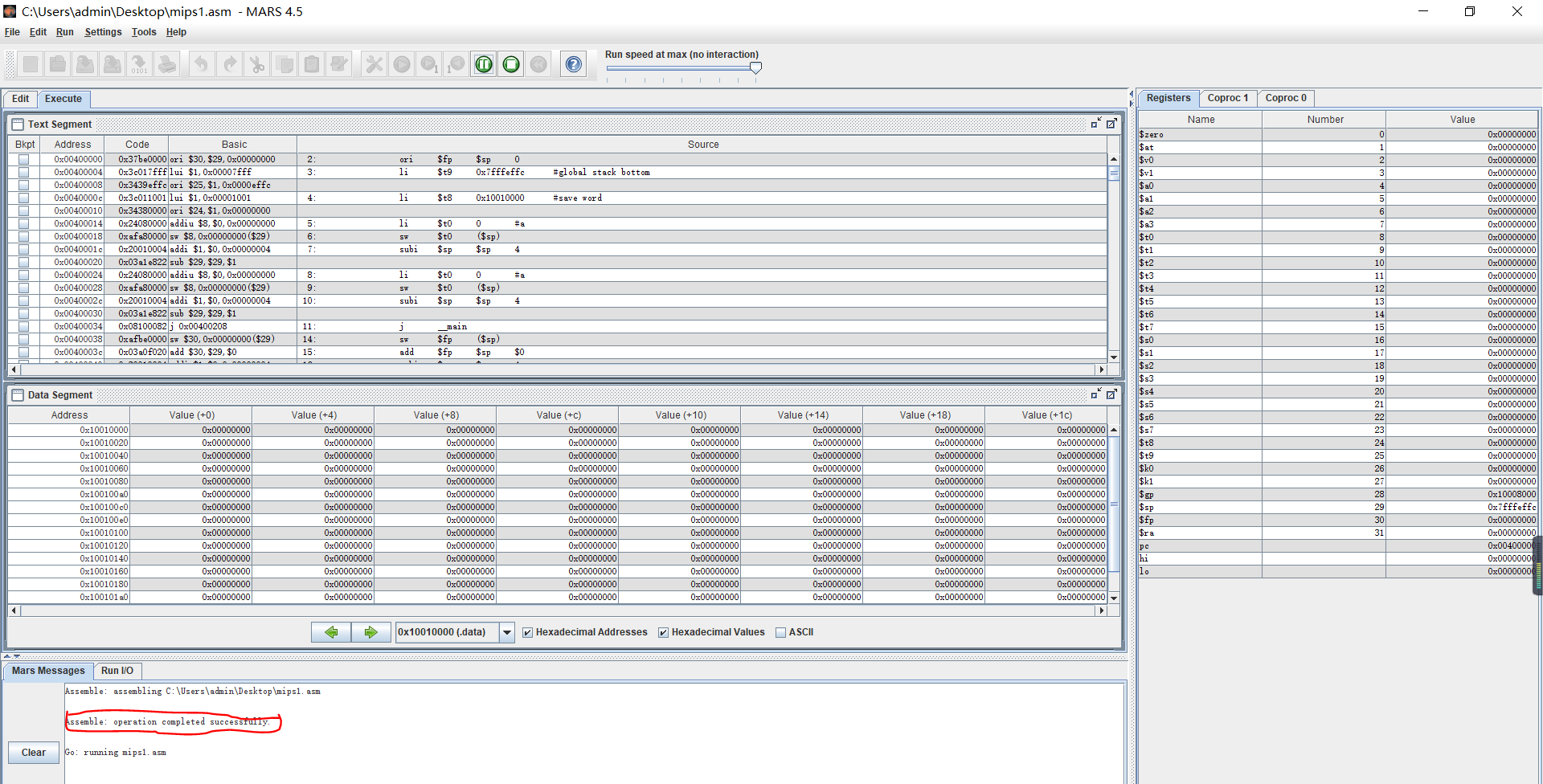


生成的汇编代码(部分)：

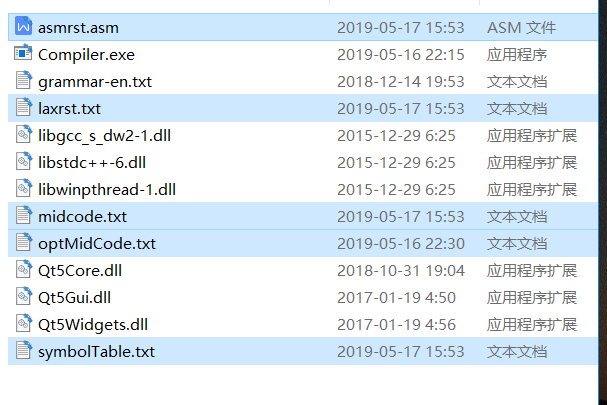


|  |
| --- |
| .text  ori $fp $sp 0  li $t9 0x7fffeffc #global stack bottom  li $t8 0x10010000 #save word  li $t0 0 #a  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #a  sw $t0 ($sp)  subi $sp $sp 4  j \_\_main  program:  #Save Register  sw $fp ($sp)  add $fp $sp $0  subi $sp $sp 4  sw $ra ($sp)  subi $sp $sp 4  #Save Register Done!  li $t0 0 #$\_0  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #$\_1  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #$\_2  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #$\_3  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #$\_4  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #program  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0 #program  sw $t0 ($sp)  subi $sp $sp 4  li $t0 0  sw $t0 -28($fp)  lw $t0 8($fp)  lw $t1 4($fp)  add $t0 $t0 $t1  sw $t0 -8($fp)  lw $t0 12($fp)  lw $t1 -8($fp)  slt $t0 $t1 $t0  \_LABEL\_1:  syscall |

测试生成的汇编代码是否正确,显示Assemble operation completed successfully：

结果也与原类C程序实际执行结果一致。

同时词法分析、优化前后的中间代码、汇编代码、符号表都输出到文件，文件名分别为：laxrst.txt，midcode.txt,optMidCode.txt,asmrst.asm,symbolTable.txt:

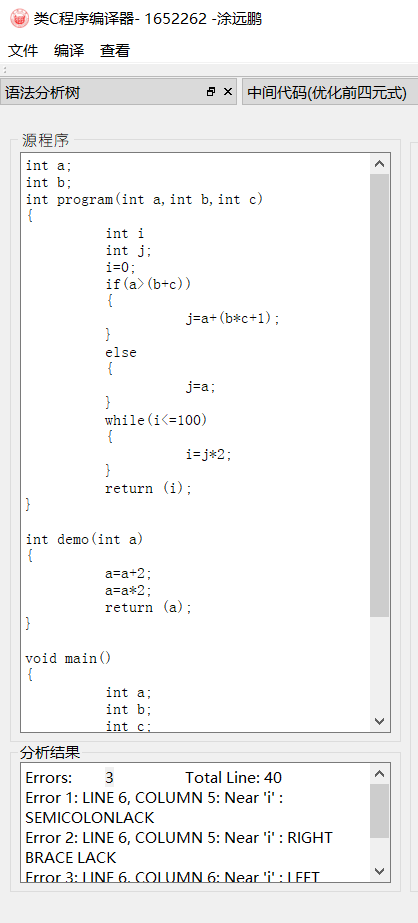


错误数据测试：

错误的类C源程序如下：

|  |
| --- |
| int a;  int b;  int program(int a,int b,int c)  {  int i  int j;  i=0;  if(a>(b+c))  {  j=a+(b\*c+1);  }  else  {  j=a;  }  while(i<=100)  {  i=j\*2;  }  return (i);  }  int demo(int a)  {  a=a+2;  a=a\*2;  return (a);  }  void main()  {  int a;  int b;  int c;  a=3;  b=4;  c=2;  a=program(a,b,demo(c));  return;  } |

输入错误的类C源文件，输出显示错误信息包含错误行数，以及分析所得的错误原因：



|  |
| --- |
| Errors: 3 Total Line: 40  Error 1: LINE 6, COLUMN 5: Near 'i' : SEMICOLONLACK//i的分号确实  Error 2: LINE 6, COLUMN 5: Near 'i' : RIGHT BRACE LACK//右侧大括号缺失  Error 3: LINE 6, COLUMN 6: Near 'j' : LEFT PARENT LACK//j左侧没有父函数 |

**2.时间复杂度分析**

由于在生成汇编码时涉及到while的两重循环及递归函数的使用，所以程序的时间复杂度较高为O(n\*n)级别。

**3.遇到的问题及解决**

问题：本程序使用了 LR1 的设计思想，并做了较大的改动。 我最初的想法是严格按照 LR1 算法的思想，以 program 为 S’，并根 据状态转换图画出 action表。

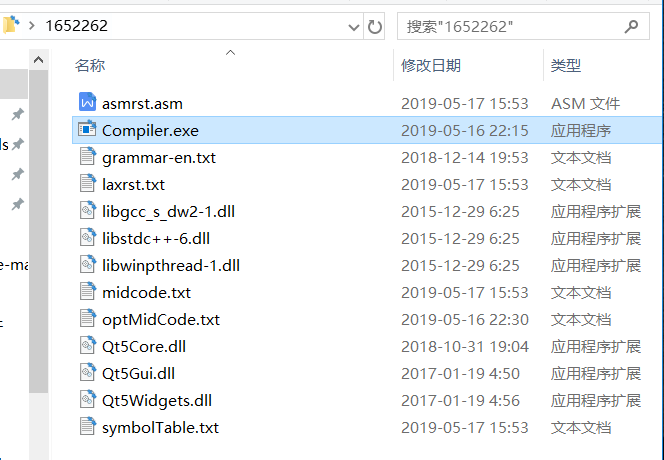
但是在实现的过程中，发现这样的问题很大。类 c 语法有 26 个产生式，要准确画出全部的状态转换图是十分困难的。

解决：对于上述问题，可以通过以下显而易见的规则进行优化。比如当程序读到了 int，那么接下来的步骤我们可想而知是继续读，反映到 LR1 算法中，就是当前状态i 与当前待输入字符，对应的为 Sj。而若读到 了’;’，则对应的一定为 rj。所以当读到’;’、’}’等就一定是指向规约操作，而读到其他的就会继续读入。所以我在构建action表时便按照上述的规则进行优化，对于后续可以预知到动作的表项填入对应的移进或规约操作，而不需要根据状态转换图来填表，从而进一步降低移进规约冲突出现的可能。

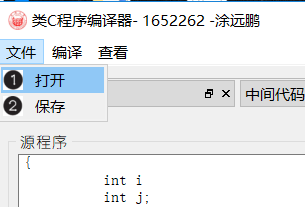
八、用户使用说明

使用步骤如下所示：

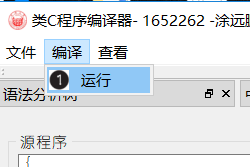
1.首先将压缩包中的1652262文件夹复制到桌面，然后点击1652262文件夹中的Compiler.exe



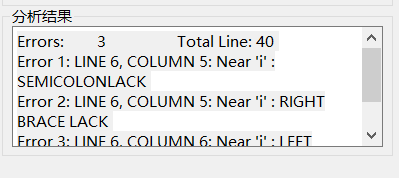
2.选择运行的Compiler.exe中导航栏的“文件”选项，选择“打开”，选择要编译的类C源程序，选择的类C程序会显示在左侧的textBrowser中：



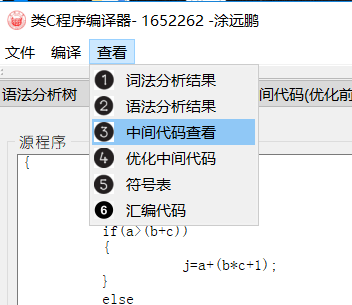
3.点击导航栏中编译中的“运行”，即可对类C程序进行编译：



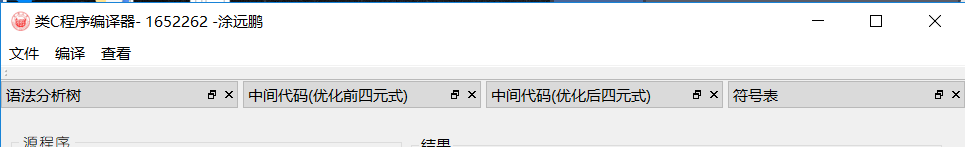
4.如果编译成功将在下方的文本框中显示“Compile Success.”，如果失败则会显示编译错误信息，显示在第几行第几列出现了什么错误



5.点击导航栏中的“查看”，选择其中的词法分析结果、语法分析结果、中间代码查看、优化中间代码、汇编代码、符号表，即可查看词法分析表、语法分析树、四元式表、优化后四元式表、汇编代码、符号表，并且会同时输出到exe所在文件夹下的txt文件中：



6.上述六个选项中，其中语法分析结果、中间代码查看、优化中间代码、符号表会显示在四个悬浮窗口中：



7.由于程序在开发时使用了比较多的全局变量，所以每次编译过后需要重新启动exe才能再次进行编译......

九、问题和解决的方法、心得体会

持续一个月的编译原理课程设计，让我学到了很多东西。首先是加深了对编译原理课程的理解，对词法分析、语法分析、语义分析有了更进一步的掌握，其次是编程能力的提高，这样的代码量，一年也只有几次能够遇到。有了前面课程设计的经验，这一次的编译原理课程设计，我更注重数据结构，数据结构的好坏直接决定了代码的复杂度，代码的量，有一些可以用链表做的东西我尽量避免用链表，换用一个数组和一个表示长度的整型变量的数据结构来表示，因为链表虽然方便灵活，但是其很难调试，遇到错误常常会使程序直接崩溃。

在课程设计的过程中，我遇到了极大的困难。总共历时一个月，才完成了一个基本的编译器。通过这次课程设计，让我对编译器的工作原理有了较深的认识。同时也发现了自己程序功力的不足。该编译器还是有挺多不是很满意的地方。比如：由于设置了太多全局变量导致程序的整体结构比较混乱，并且每次打开只能执行一次，然后需要重新打开才能再次编译。

主要因为基础功能的实现耗费了大量的时间，没有空余来完成原定的特性，可以说一个支持众多语法特性的编译器不能一蹴而就。

在语义分析的过程中，最大的难点是如何实现一遍扫描。MIPS汇编代码最大的难题是函数调用的翻译，在前面已经说过解决的思路了，就不再多言。通过此次编译原理课程设计我非常深入地理解了从形式语言到汇编代码的编译过程，获得了很多知识。

十、参考资料

[1]编译原理[M]. 机械工业出版社 , (美) 阿霍 (Alfred, 2009

[2]编译器构造[M]. 机械工业出版社, (美)CharlesN.Fischer,(美)RichardJ.LeBlanc,Jr.著, 2005

[3]C程序设计语言[M]. 机械工业出版社 , (美)BrianW.Kernighan,(美)DennisM.Ritchie著, 2004

[4]数据结构[M]. 清华大学出版社 , 严蔚敏,吴伟民编著, 2002

[5]编译原理和技术[M]. 中国科学技术大学出版社 , 陈意云编著, 1997