中间代码生成评审材料

所有功能已实现

1.输出三地址码

（1）输出中间代码

【操作步骤】

进入gpl-master目录，打开终端

1. make

2. cd sample

3. ../gpl calc -tac

【测试用例】

calc.gpl

func.gpl

if.gpl

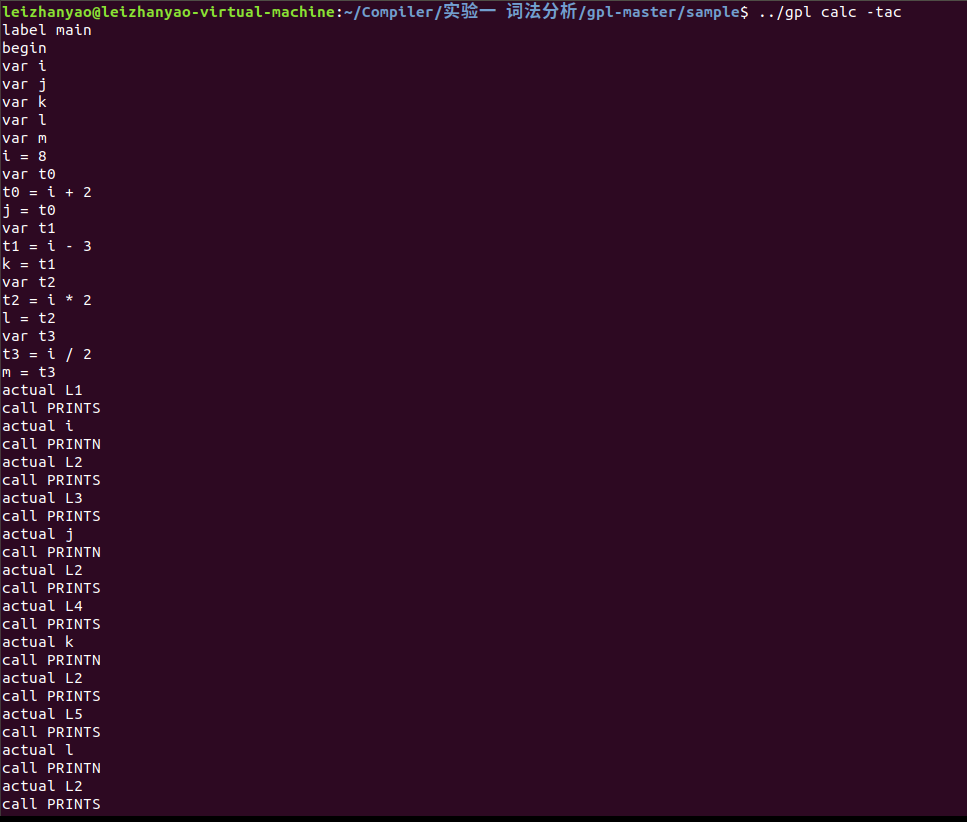
while.gpl

for.gpl

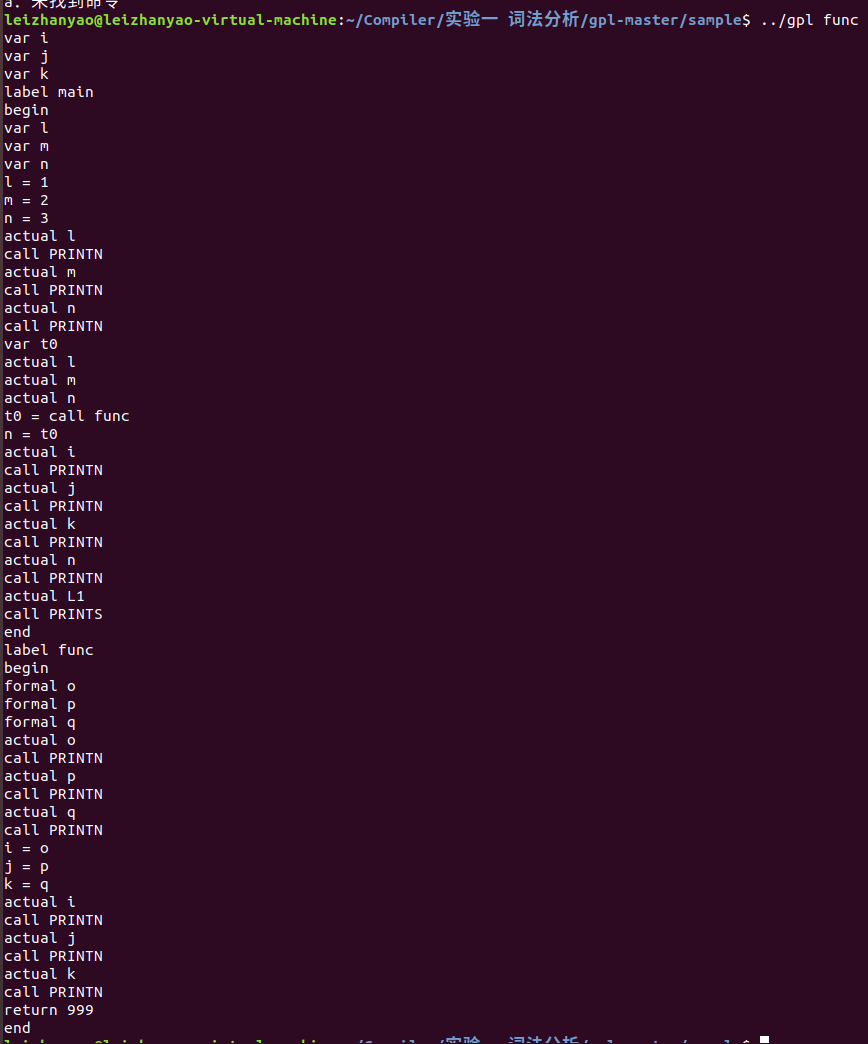
【结果】

输入上述命令后，会输出tac。以上测试用例的结果分别如下面的图所示：

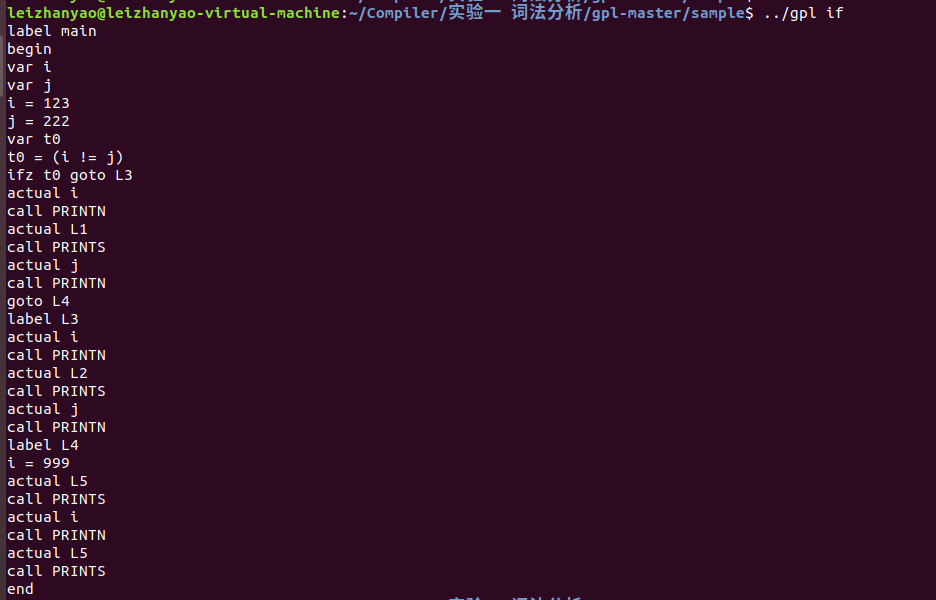
calc的：



func的：



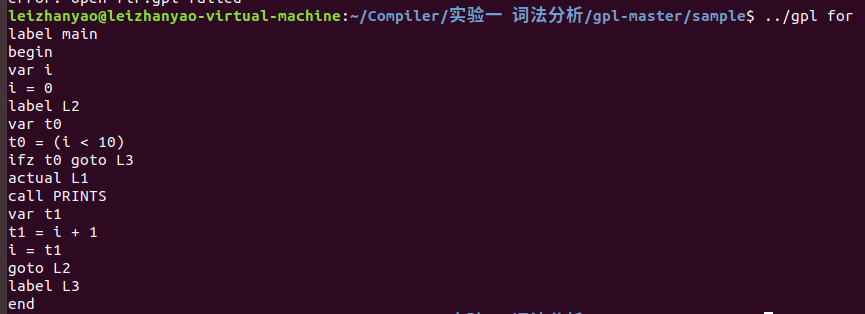
if的：



while的：



for的：



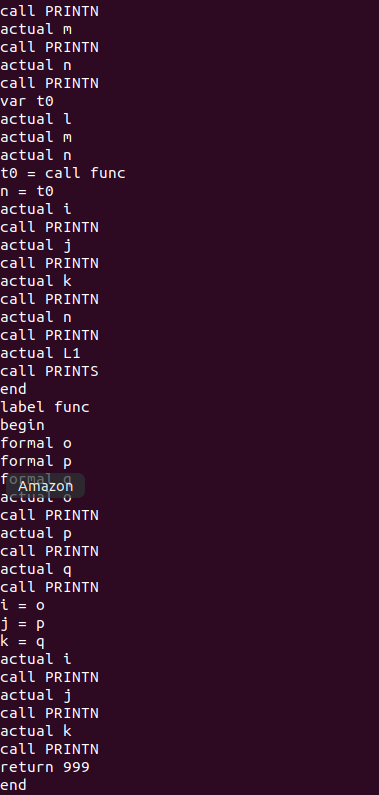
经过比对，发现中间代码正确，功能完成。

【分析】这个功能其实已经在老师给的代码中实现。我发现主要是在语义子程序里面实现的tac的创建和连接等。主要是在.y文件中的产生式后面的函数操作里面完成的。

（2）按照源代码中顺序说明三地址码中实参

【操作说明】对func.gpl使用 结果如下

【运行结果】



可见 对于源代码中的print func等函数的调用 其实参的传递都是按照源代码顺序的（如里面的actual o、actual p、actual q等）

说明结果正确，功能实现。

【代码简述】

主要是872~900行的do\_call函数。这个函数主要负责函数调用时的tac生成。原来的代码是反过来压栈的，我们只需要把它改成按源代码顺序的压栈即可。

（3）支持for语句

【测试用例】

for.gpl

main()

{

for(int i=0;i<10;i=i+1)

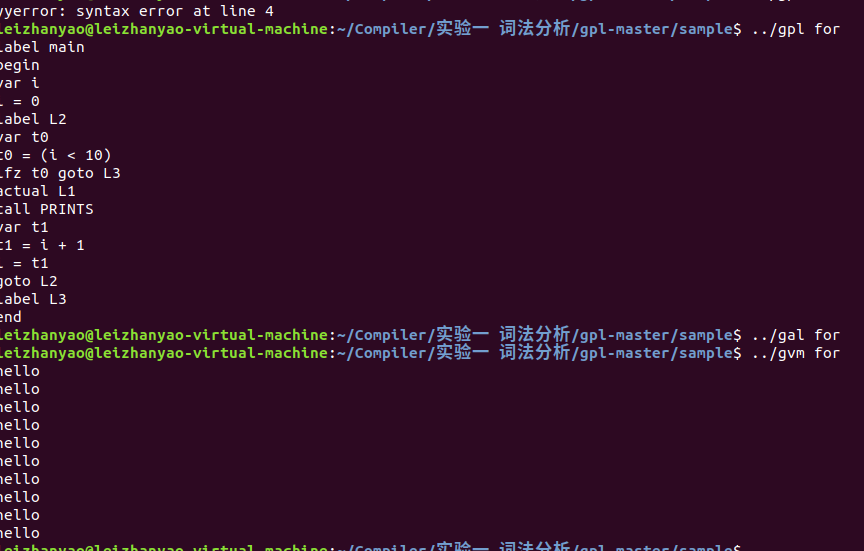
{

print("hello\n");

}

}

【结果】



如图所示 可以生成正确tac 并且可以成功执行 说明for语句支持成功

【代码简述】

其实很类似。直接对比老师给出其他的while if等语句的支持的代码，照葫芦画瓢就可以。主要是三点，一是在前面定义token for，二是修改中间的产生式，使其支持for语句，最后是添加相关处理的函数。如do\_for（见代码的983~991行）

（4）支持变量（包括局部变量、全局变量）的初始化操作。

【测试用例】

init.gpl

int a=1;

main()

{

int b=2;

print(a,"\n",b,"\n");

}

【结果】



从结果可见，已经支持了初始化操作。（老师原本的代码没有支持初始化）

【代码简述】

利用已有的非终结符和产生式，在原有的产生式里添加支持初始化的代码即可。

主要是这里：

declaration : INT variable\_list ';'

{

$$=$2;

}

|

INT IDENTIFIER '=' expression ';'

{

TAC \*var=declare\_var($2);

$$=join\_tac(var,do\_assign(getvar($2),$4));

}

把declaration加一个产生式就好了，没有加其他的非终结符。

2.出错处理

【测试用例】

error1.gpl

main()

{

int a;

a == 3;

}

【运行结果】



【代码简述】

这里明显是a=3写成了a==3。这句话本身没有什么语法问题，也是一个表达式，但是在语义上有问题。代码主要是在yylex里面加的，里面有些本来识别到错误，但是没有出错提示，而是直接退出系统的，在前面加一句报错即可。

3.问题一：你的三地址码中是如何表示变量的初始化操作的？

如图：



可以看到，这里的初始化的变量和一般的先定义，再赋值的变量的三地址码一样，都是类似于“var a a = 1”这种类型的两句。其实对于初始化变量的处理，只需要把他看作是一句定义加一句赋值语句就好，然后对于tac的处理就是先处理定义语句，再处理赋值语句。

3.问题二：是否提供了足够的测试用例（至少5个）证明你的程序实现了相关功能。

有足够的测试用例

在TAC生成中有calc.gpl、func.gpl、if.gpl、while.gpl、for.gpl共5个测试用例

在说明实参中有func.gpl一个测试用例

在for语句支持中有for.gpl一个测试用例

在出错处理中有error1.gpl一个测试用例

所有测试用例和相应生成的其他文件都放在评审材料的“测试用例文件夹中”

4.问题三：是否能够详细说明你的代码中最具特色或个性化的功能的实现方法。

我认为最具特色的功能就是在初始化变量的支持那里。起初我以为这个功能挺难实现的，但是没想到只加了几行就实现了。

declaration : INT variable\_list ';'

{

$$=$2;

}

|

INT IDENTIFIER '=' expression ';'

{

TAC \*var=declare\_var($2);

$$=join\_tac(var,do\_assign(getvar($2),$4));

}

在declaration这个终结符对应的产生式中，他前面的第一个产生式就是传统的不带初始化的定义的产生式。而后面那个产生式是我加的。首先按照C语言中初始化的模式写出产生式。然后在语义子程序中加入定义变量和赋值的TAC操作即可。