**编译系统实验报告**

**实验3 – 中间代码生成**

|  |
| --- |
| 程序实现的功能及如何实现： |
| 程序主要实现的功能是中间代码生成，具体如下：   1. 程序需要将符合实验指导书上7个假设的C--一源代码翻译为中间代码。 2. 程序的输入是一个包含C--源代码的文本文件，程序需要能够接收一个输入，另外需要文件名和一个输出文件名（如out.ir）作为参数。 3. 实验三要求程序将运行结果输出到文件。输出文件要求每行一条中间代码。 4. 如果输人文件包含多个函数定义，则需要通过FUNCTION语句将这些函数隔开。 5. 注意：本次实验只完成了老师所要求的基本功能（即完整实验指导书上的前两个例子），对于实验指导书上后两个例子中的结构体与高维数组并不作支持。   关于以上功能，具体的实现方法及程序文件简要介绍如下：   1. 关于每个程序文件的功能，简要介绍如下：   lexical.l：该文件中保存着符号对应的正则表达式，以及根据符号表中的每一个符号构建语法树的步骤。  node.h：该文件中以结构体表示语法树的每一个节点。  main.c：整个中间代码生成程序的执行文件。  semantic.h：保存着语义分析的函数声明以及重要的结构体定义。  semantic.c：语义分析的核心功能代码，执行遍历语法树，检索错误的功能。  syntax.y：保存着语法树的构建过程对应的代码。  interim.h：中间代码生成部分的头文件，保存着重要的结构体（如操作数，中间代码等）的定义和interim.c中用到的函数声明。  interim.c： 中间代码生成的核心文件，包括了主要函数的实现。  其中，只有interim.c和interim.h是本次实验完成的有关中间代码生成的文件，其余文件均是lab1与lab2所完成的。   1. 关键的结构体定义 2. 操作数结构体Operand\_   节点保存的信息有：枚举类型kind（表示操作数属于变量，地址，常量，或者其他类型），int， char，type。如果操作数为数量，则需要通过int类型保存其值。   1. 代表中间代码的结构体InterCode\_   节点保存的信息有：枚举类型kind（同样保存中间代码的类型），同时用联合体保存不同类型中间代码所需要的额外数据。   1. 整体程序执行流程   程序整体执行流程可以参照main.c。首先通过yyparse()函数执行lab1和lab2中构建语法树，遍历语法树构建符号表、遍历语法树检查语句类型，遇到错误的类型则输出对应的错误的过程。而在中间代码生成部分，对每一个语法单元设置相应 translate 函数，再根据产生式调用下层 translate 函数。根据讲义以及上述自行翻译的语句生成对应的中间代码，最后回到Program 中后进行中间代码的打印。 |
| 如何编译程序： |
| 最终的代码文件如图所示：  testfile文件夹中保存着测试文件，对应着指导书中的前两个文件。  parser 是gcc编译后得到的可执行文件。  其余文件是中间过程生成的文件。  执行以下几条命令以编译程序：  flex -o ./lex.yy.c ./lexical.l  bison -o ./syntax.tab.c -d -v ./syntax.y  gcc -c ./syntax.tab.c -o ./syntax.tab.o  gcc -std=c99 -c -o interim.o interim.c  gcc -std=c99 -c -o main.o main.c  gcc -std=c99 -c -o semantic.o semantic.c  gcc -o parser ./interim.o ./main.o ./semantic.o ./syntax.tab.o -lfl -ly  在本实验中，我将以上命令写入脚本，在Linux下使用时，切入Lab3文件夹，直接执行 run.sh 便可完成程序编译。  完成之后，输入 ./parser ./test/testfile.cmm 便可以用程序对进行中间代码生成。 |
| 程序执行结果： |
| 对于实验指导书中的例子1与例子2，运行结果及生成的outx.ir如图所示：  out1.ir： our2.ir： |