目录

[编译原理实验3报告--石卓凡120L021011 1](#_Toc134886609)

[a） 程序实现了哪些功能？简要说明如何实现这些功能。 1](#_Toc134886610)

[设置全局变量表，全局标签编号 1](#_Toc134886611)

[对if/while的处理 1](#_Toc134886612)

[对int一维数组的处理 2](#_Toc134886613)

[对函数的处理 3](#_Toc134886614)

[对表达式的编译（部分） 3](#_Toc134886615)

[b）程序应该如何被编译？ 3](#_Toc134886616)

# 编译原理实验3报告--石卓凡120L021011

## 程序实现了哪些功能？简要说明如何实现这些功能。

1. 将C−−源代码翻译为中间代码

### 设置全局变量表，全局标签编号

通过全局变量表，在每定义一个变量或者声明一个参数时候，利用全局变量编号globalVarId记录并分配唯一的变量编号，再并记录在全局变量表variables

维护全局标签编号globalLabelId用于后续分配唯一的标签编号

与下文的两条函数配合使用

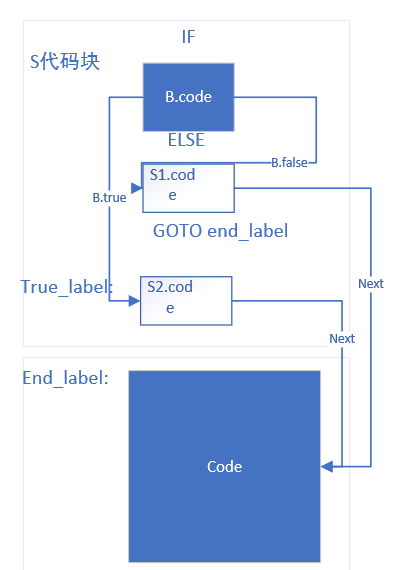
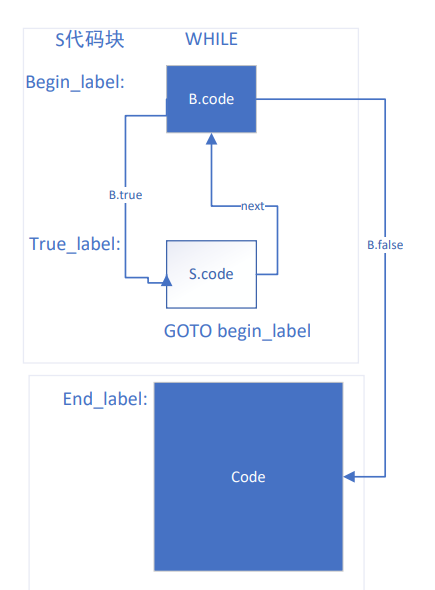
#### char\* getTranslate(int var,int k)

根据var下标,返回对应的字符串内容,比如说var=3代表着常数2,返回为#2;变量x应该返回t1这种东西,即临时变量或者常量的编号翻译为字符串

#### int getVariableByName(char\* name)

根据variableNames在变量表里查variables中的变量编号

### 对if/while的处理

If与while处理示意图

**IF的处理，void handleIfElse(Node \*tree);**

1. If B为真 GOTO true\_label:
   1. S1语句(若有else，则把这一句迭代分析语句)
2. GOTO end\_label
3. True\_label:
   1. S2语句
4. End\_label:
   1. 下一段语句

**WHILE的处理，void handleWhile(Node \*tree);**

1. Begin\_babel:
   1. If B为真 GOTO true\_label:
2. GOTO end\_label
3. True\_label:
   1. S语句
   2. Goto begin\_label
4. End\_label:
   1. 下一段语句

### 对int一维数组的处理

int handleIntArray(Node \*tree, int isAarrayOnLeft)：

以a[0] = 0为例，通过tree->child->child取出对应的数组名字a,

通过数组名字a,获取数组a[]基地址,再通过tree->child->next获取a[0]中的0

计算出偏移地址与实际地址。

根据节点的isArrayOnleft属性，来判断数组元素在左侧或是右侧，依次来判断是否需要取值

### 对函数的处理

int handleFunc(const char \*funcname, const int \*arglist, int ptr)

处理函数调用,处理read与write以及其他函数

1. 首先特殊处理read与write
   1. 如果发现funcname == read或者write，直接根据函数特性继续进行输出
2. 随后在调用一个函数之前，先通过arg传入所有参数，
3. 然后通过使用CALL调用该函数并存储返回值

### 对表达式的编译（部分）

int CalcExp(Node\* tree,int isAarrayOnLeft)，对于非条件的Exp生成计算代码并返回临时变量在符号表中的编号，而isAarrayOnLeft=1则表示数组，返回的是数组元素地址

1. EXP->ID:
   1. 对扫描到的ID，比如a，直接通过全局变量表，返回ti的i
2. EXT->LP EXP RP
   1. 递归处理中间的EXP,CalcEXP(tree->child->next,0)
3. EXP->ID LP ARGS RP
   1. 这里需要处理函数
   2. 先从tree->child取出functionName，再tree->child->next->next拿出参数args
   3. 根据Args → Exp COMMA Args | Exp ,递归处理其中的所有参数args，直到结束
   4. 调用函数处理handleFunc(funcname, arglist, ptr)
4. Exp->MINUS Exp
   1. 通过handleNegativeNum(tree)处理负数，简单的根据-1翻译为ti = #0 - 1
5. Exp->Exp STAR Exp; Exp->Exp DIV Exp; Exp->Exp PLUS Exp; Exp->Exp MINUS Exp
   1. 通过tree->subtype来获取操作符（\*,/,+）
   2. 对第一个和第二个操作数进行CalcEXP()；来进行处理
   3. 输出本次表达式"t%d := %s %c %s\n"

## b）程序应该如何被编译？

**编译命令为**

./make.sh

**运行测试样例命令**

1. chmod +x parser
2. ./parser 1.cmm result1.ir（以1.cmm输出到result1.ir为例）