Compiling Project Report 201838039\_Dikshya Kafle



**Northwestern Polytechnical University**

**Student Number: 2018380039**

**Student Name: Dikshya Kafle**

**Exam Subject: Compiling principle**

**Class Teacher Name: Dr. Lin Yi**

1. Experimental requirements:

This experiment is to implement a mini-compiler and compile a subset of c0, which can generate intermediate code and final assembly language.

2. Experimental Design:

2.1 Grammar notes:

<addition operator> ::= +｜-

<Multiplication Operator> ::= \*｜/

<Relational Operator> ::= <｜<=｜>｜>=｜!=｜==

<letter> ::= ＿｜a｜. . . ｜z｜A｜. . . ｜Z

<number> ::= 0｜<non-zero number>

<Non-zero number> ::= １｜. . . ｜9

<Character> ::='<Addition operator>'｜'<Multiplication operator>'｜'<Letter>'｜'<Number>'

＜String＞ ::= "{ASCII character with decimal code 32,33,35-126}"

<Program> ::= [<Constant description>] [<Variable description>] {<Function definition with return value>|<Function definition without return value>}<Main function>

＜Constant description＞ ::= const＜constant definition＞;{ const＜constant definition＞;}

<constant definition> ::= int<identifier>=<integer>{,<identifier>=<integer>}| char<identifier>=<char>{,<identifier>=<char>}

＜Unsigned integer＞ ::= ＜Non-zero number＞ {＜Number＞}

<Integer> ::= [＋｜－] <Unsigned integer>｜0

<Identifier> ::= ＜Letter> {<Letter>｜<Number>}

<Declaration header> ::= int<identifier>|char<identifier>

＜Variable description＞ ::= ＜Variable definition＞;{＜Variable definition＞;}

<variable definition> ::= <type identifier>(<identifier>|<identifier>'['<unsigned integer>']'){,(<identifier>|<identifier>'['< Unsigned integer>']')}

<Type identifier> ::= int | char

＜Function definition with return value＞ ::= ＜Declaration header＞‘(’＜Parameter list＞‘)’ ‘{’＜Compound sentence＞‘}’

＜No return value function definition＞ ::= void＜identifier＞‘(’＜Parameter list＞‘)’‘{’＜Compound statement＞‘}’

<Compound sentence> ::= [<Constant description>] [<Variable description>] <Sentence list>

<Parameter table> ::= <Type identifier> <Identifier> {,<Type identifier> <Identifier>}|<Empty>

＜Main function＞ ::= void main‘(’‘)’ ‘{’＜Compound sentence＞‘}’

<Expression> ::= [＋｜－] <term> {<addition operator> <term>}

<term> ::= <factor> {<multiplication operator> <factor>}

<factor> ::= <identifier>｜<identifier>'['<expression>']'｜<integer>|<char>｜<function call statement with return value>|'('<expression> ')'

<statement> ::= <conditional statement>｜<loop statement>｜<case statement>|'{'<statement list>'}'｜<function call statement with return value>; |<function call statement without return value> ;｜<assignment sentence>;｜<read sentence>;｜<write sentence>;｜<empty>;｜<return sentence>;

<Assignment Statement> ::= <Identifier>=<Expression>|<Identifier>’[’<Expression>‘]’=<Expression>

<Conditional statement> ::= if ‘(’<condition>’)’ <statement>

<condition> ::= <expression> <relational operator> <expression>｜<expression> //The expression is 0 and the condition is false, otherwise it is true

<Loop statement> ::= do<statement> while ‘(’<condition>’)’

<constant> ::= <integer>|<character>

<Case statement> ::= switch ‘(’<Expression>‘)’ ‘{’<Case table> ‘}’

<Case table> ::= <Case clause> {<Case clause>}

<Case sub-statement> ::= case<constant>: <statement>

＜Function call statement with return value＞ ::= ＜identifier＞‘(’＜value parameter list＞‘)’

＜No return value function call statement＞ ::= ＜identifier＞‘(’＜value parameter list＞‘)’

＜Value parameter table＞ ::= ＜Expression＞{,＜Expression＞}｜＜Empty＞

<Sentence list> ::={<Sentence>}

<Reading sentence> ::= scanf ‘(’<identifier>{,<identifier>}’)’

＜write sentence＞ ::= printf‘(’＜string＞,＜expression＞‘)’|printf ‘(’＜string＞‘)’|printf ‘(’＜expression＞‘)’

<Return Statement> ::= return[‘(’<Expression>‘)’]

**2.2 Description of the main structure:**

**A single symbol uses the following structure for storage:**

struct one\_symbol {

char name[MAXNAME]; //Identifier name

int kind; /\*species

0：constant

1：variable

2：function

3：Function parameters

\*/

int type; /\*Types of

0：void((Only for functions with no return value)

1：int

2: char

\*/

int value; /\*Constant: value or ascii code

other：0

\*/

int size; /\*Function: the number of parameters

Array: the number of elements

Other: -1

\*/

int in\_address; //The offset relative to the first address of the AR (the first address of the array record

};

Based on this, the structure of the symbol table is as follows:

struct table {

struct one\_symbol List[MAXLIST]; //Symbol table

int list\_index; //index

int num\_of\_func; //The number of functions that have been alive

int index\_of\_func[MAXFUNC]; //Initialize all -1, store the table index of the function

};

The intermediate code or quaternion is stored in the following structure:

struct mid\_code {

int op;

char src1[100];

char src2[100];

char result[100];

int is\_effective;

};

In this structure

op: operator

src1: The identifier of operand 1 (may also be a temporary variable in the middle)

src2: The identifier of operand 2 (may also be a temporary variable in the middle)

result: the identifier of the result (may also be a temporary variable in the middle)

is\_effective: Is this intermediate code effective?

All intermediate codes are stored and indexed in the following structure

struct mid\_code\_list {

struct mid\_code MID\_CODE[MAX\_MID\_CODE];

int code\_num;

};

**2.3 Storage allocation plan**

Use static storage to store all strings and global variables that need to be printed, and use dynamic storage allocation to allocate local variables and temporary variables.

Corresponding to the mips assembly, that is, all strings and global variables that need to be printed are stored in the heap space, and declared in advance at the beginning of the mips assembly with .data. For local variables and temporary variables, they are placed in the running stack (that is, stored in the activity record of the corresponding function)

The runtime stack is a dynamic structure that stores activity records. When a function is called, a new activity record is created correspondingly to it (the activity record stores explicit parameters, implicit parameters, local data, etc.) . Update $fp to the top of the original activity record, and $sp always points to the next space on the top of the current stack.

More specifically, as follows:

$fp stores the top of the original activity record

$fp+4 stores the return address of the current function

$fp+8 stores the value parameters of the function in turn from here, internal local variables, local constants, and temporary variables

2.4四元式设计

所有具体的四元式含义如下：

1 CONST INT src1 a const int src1 = a;

整型常量src1的声明(a为一个整数)

2 CONST CHAR src1 a const char src1 = a;

字符型常量src1的声明(a为字符)

3 INT src1 int src1;

整型变量声明

4 CHAR src1 char src1;

字符型变量声明

5 ARRAY INT src1 NUM int src1[NUM];

整型数组声明(长度为NUM)

6 ARRAY CHAR src1 NUM char src1[NUM];

字符型数组声明(长度为NUM)

7 FUNC INT src1 int src1();

整型函数声明

8 FUNC CHAR src1 char src1();

字符型函数声明

9 FUNC VOID src1 void src1();

void型函数声明

10 PARA src2 src1 void func(int src1/char src1);

函数的参数声明

11 result = src1 result = src1(普通赋值语句)

12 result[src2] = src1 result[src2] = src1(将值赋值给数组元素)

13 reslut = src1[src2] reslut = src1[src2](数组元素赋值给普通变量语句)

14 CALL src1 (result) 调用名字为src1的有返回值调用函数函数

15 CALL src1 调用名字为src1的无返回值调用函数函数

16 result = src1 <= src2 当src1 <= src2时，result = 1。否则result = 0

17 result = src1 < src2 当src1 < src2时，result = 1。否则result = 0

18 result = src1 >= src2 当src1 >= src2时，result = 1。否则result = 0

19 result = src1 > src2 当src1 > src2时，result = 1。否则result = 0

20 result = src1== src2 当src1 == src2时，result = 1。否则result = 0

21 result = src1 != src2 当src1 != src2时，result = 1。否则result = 0

22 EQU src1 0 src2 当scr1和0相等时，跳到scr2对应的地址

23 NEQ src1 0 src2 当scr1和0不等时，跳到scr2对应的地址

24 SET src1 声明src1对应的label

25 RT 无返回值函数返回语句

26 RT src1 有返回值函数返回语句，返回src1

27 SCAN result 将读入的值存到result中

28 PRINT src1 src2 src1存放字符串,src2存放的是表达式值，打印出来。

src1或scr2可以有一个为空

29 result = src1 + src2 result = src1 + src2

30 result = src1 - src2 result = src1 - src2

31 result = src1 \* src2 result = src1 \* src2

32 result = src1 / src2 result = src1 / src2

33 GOTO src1 无条件跳转到src1处

34 END 程序结束

35 PUSH src1 将src1压进运行栈顶

36 GET src1[src2] result 取数组元素

3词法分析

词法分析部分，输入为待编译文件上的一个个字符，输出则是当前扫描得到的一个符号如标识符或者无符号整数，其symbol号和其缓存，由语法分析部分对其调用并利用其功能生成符号表。

其核心功能函数为insymbol函数，其实现是一连串的if-else if-else语句根据调用的get\_ch函数不断从conpiler\_file读取的单个字符进行判断，对生成的是哪一个类型的符号，是否生成完毕不断进行判断，返回的是symbol号，缓存字符串暂存在全局变量TOKEN中

具体的：第一步是对包括空格，换行等的预处理和是否结束即读到EOF进行判断

之后是对标识符的判断，以及然后是该标识符是不是关键字的判断

然后是对数字，无符号整数的判断

之后是+，-，\*，/,>,>=，（，[，{等一系列符号的判断

在这些判断过程中需要随时监测错误的出现，比如检测到的第一个字符是0，判断属于无符号整数类型，但是如果0后面直接跟了其他数字而不是结束，那么则报错

4语法分析

语法分析是编译过程极其重要的部分。整体代码分为两个部分，一部分是递归下降部分，基本上文法中每一个非符号的组成部分，都有其对应的递归函数最终拆解成各种基础的符号，其中program()函数是语法分析的入口，因为程序是文法中最高层次的一个概念。另外一个部分，则是辅助递归函数的函数。比如append\_to\_table函数用于扩展符号表，get\_func\_type函数，根据函数名检索符号表，返回函数类型。

语法分析在main函数中通过program函数开始

首先是const检测即常量定义，如果无，则进入变量声明部分，此时需要保存现场根据后续是否出现“（”判断是有返回值函数还是变量并恢复现场，进入相对应的递归程序。在检测到“;”后还需要判断已经该进入main函数了还是继续循环，进行下一个函数定义或者变量声明

这里主要有四个重要函数，分别是void\_func\_def(),val\_func\_def()，const\_dec()， var\_dec(),分别是无返回值函数定义，有返回值函数定义，常量声明，变量声明。以无返回值函数为例说明其情况。

首先读取函数名，，然后以此为据扩充符号表，并生成对应函数声明的中间代码，然后读取“(”，然后调用para\_list函数，值参数函数，对参数进行处理，如果确实存在参数则生成对应的10号中间代码，之后是各种括号是否完整匹配的判断，在{}内调用comp\_state()函数，因为函数内部是一个完整的作用域。

Main函数中首先是，它本身作为一个函数，记录进符号表里面，同时生成第9号中间代码，VOID函数声明，然后是对一系列括号的判断，之后调用函数comp\_state()作为作用域的主成分，其中可能包括常量声明（const\_dec()），变量声明(var\_dec())，一定包括语句列(state\_list()),然后语句列不断检测新的关键字，比如if,for等进入语句，进入语句部分（state()）,再在语句部分根据关键字进入不同的处理部分，如if\_state(),loop\_state(),如果当前symbol是一个普通的标识符的话，则，判断后续是“（”则是函数调用，是“=”/“[”则是赋值语句，同时包括scanf,print,return 等均在此行列处理。

在赋值语句中，如果当前符号是“=”，读取下一个符号，并进入表达式函数（expr()），之后判断可能出现的三种错误，即1.常量赋值错误 2.char被int赋值 3.数组整体赋值。之后将生成的now\_result值存入src1之后生成对应中间代码。如果当前符号是“[”，则在[]内部和=后接需要调用一次表达式函数，得到两个值即数组标签和对应值，生成对应的数组赋值的第12号中间代码，这里还需要判断“]”和“=”的存在，如果不存在则报错。但是如果标识符之后是”(”,则说明当前是函数调用，首先是恢复现场，之后通过 get\_func\_type(TOKEN)函数查找对应函数类型，是0则调用void\_fun\_call();，是1则调用val\_fun\_call();而这两个函数内容是在包括括号，函数类型等的检查后，生成调用函数的中间代码，无返回值函数是15号，有返回值函数是14号。

对应if\_state()语句，首先通过generate\_labe()语句生成自己的标签，每一个这样的state语句都会有生成自己的标签。然后在（）内调用condition()语句，先通过expr()生成第一个值，之后判断判断符号是哪一个，<=,<,>,>=,==,!=,然后通过expr函数计算第二项的值，并进行比对，这个比对的式子有自己对应的中间代码，对应编号16\_21.如果表达式内是多个表达式用||或者&&相连怎么处理？不处理这种情况，或者说直接报错，相应功能可以通过if-elseif-else的形式实现。Condition()语句之后,存储条件判断结果，并生成22号中间代码，即根据condition结果，跳转对应label。

对应的如果是读语句，先判断“（”，之后判断是否是标识符，如果不是标识符则报错，然后存取标识符的名字，从TOKEN读入var\_name,然后通过index\_in\_table查询该标识符的标签，然后并确定其类型是INT还是CHAR，以及是不是变量，如果不是则直接报错，如果是，则根据类型生成27号四元式。之后则是匹配括号，匹配失败也报错。至于其它的如写语句，然会语句，switch语句等基本与上类似，不在赘述。

5中间代码生成

当program函数运行完成时，整个中间代码的生成也在这个过程中完成了。主要是通过generate\_mid\_code函数，在对应的语句分析完成确定其成分后，即调用该函数生成合适的中间代码。其先判断中间代码表CODE\_LIST存储的中间代码数是否超过上限，如果没有则先记录下一个中间代码的标签，然后在对应标签存储通过传参带来的该条中间代码，并增加中间代码数的计数。

Program函数执行完了之后，会调用print\_mid\_code函数，其中会打开mid\_code.txt并且根据之前约定的四元式格式，调用存储在CODE\_LIST里的各个参数，写入完整的中间代码参数。

6最终代码生成

最终代码生成则是在print\_mid\_code函数之后调用generate\_all\_final\_code()函数实现，相关函数具以集成在其中了。首先是调用get\_new\_table函数，生成新的符号表，主要是更新临时变量等。之后打开final.asm文件，并先写入.data，之后调用data\_area\_generate（）函数，根据符号表把全局量，根据MIDCODE把字符串和中间变量写入.data部分，之后在.asm文件中写入

.text

move $fp $sp

j main

然后调用generate\_text()函数，生成.text剩下的部分，具体是使用for循环检索每一个中间代码，先判断是否有效，然后根据代码类型即op属性，进行不同的处理，比如函数声明会要求生成新的运行栈，算术表达式会要求使用generate\_label()函数生成标签以及load\_data函数与store\_data函数取存数据等等，最终根据相应情况将汇编代码写入生成文件中。

7实验结果展示

使用的待编译文件如下：

输出的中间结果如下：

INT ah

FUNC INT add

PARA INT x

$VAR0 = 3 \* x

RT $VAR0

FUNC VOID main

ARRAY INT a 10

INT i

i = 0

$VAR0 = i >= 0

EQUZ $VAR0 LABEL0

a[i] = i

SET LABEL0

END

END

展示语法分析过程的result.txt内容如下：

This is a variable defination!

This is a defination head!

This is a parameter list!

This is an integer!

This is a factor!

This is a factor!

This is a term!

This is an expression!

This is a return statement!

This is a statement!

This is a statement list!

This is a composite statement!

This is a value function defination!

This is a variable defination!

This is a variable defination!

This is a variable declaration!

This is an integer!

This is a factor!

This is a term!

This is an expression!

This is an assignment statement!

This is a statement!

This is a factor!

This is a term!

This is an expression!

This is a relational operation!

This is an integer!

This is a factor!

This is a term!

This is an expression!

This is a condition!

This is a factor!

This is a term!

This is an expression!

This is a factor!

This is a term!

This is an expression!

This is an assignment statement!

This is a statement!

This is a statement list!

This is a statement!

This is an if statement!

This is a statement!

This is a return statement!

This is a statement!

This is a statement list!

This is a composite statement!

This is a main function defination!

This is a program!

finish compiling

最终生成的汇编语句如下：

.data

ah: .word 0

.text

move $fp $sp

j main

add:

lw $s7 0($sp)

subi $sp $sp 8

sw $s7 0($sp)

addi $sp $sp 12

move $s7 $fp

move $fp $sp

subi $fp $fp 4

sw $s7 0($fp)

sw $ra -4($fp)

subi $sp $sp 16

li $t0 3

lw $t1 -8($fp)

mul $t0 $t0 $t1

sw $t0 -12($fp)

lw $t0 -12($fp)

move $v0 $t0

lw $ra -4($fp)

addi $sp $fp 4

lw $fp 0($fp)

jr $ra

main:

move $s7 $fp

move $fp $sp

subi $fp $fp 4

sw $s7 0($fp)

sw $ra -4($fp)

subi $sp $sp 56

li $t0 0

sw $t0 -48($fp)

lw $t0 -48($fp)

li $t1 0

bge $t0 $t1 LABEL1

sw $zero -52($fp)

j LABEL2

LABEL1:

li $t2 1

sw $t2 -52($fp)

LABEL2:

lw $t0 -52($fp)

beq $t0 $zero LABEL0

lw $t0 -48($fp)

lw $t1 -48($fp)

li $s6 4

mul $s6 $s6 $t1

subi $s7 $fp 44

add $s7 $s7 $s6

sw $t0 0($s7)

LABEL0:

li $v0 10

syscall

li $v0 10

syscall

8实验代码

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include<ctype.h>

//宏定义

//gramma.c部分

//读入单词类型定义

#define CONSY 1 //const

#define INTSY 2 //int

#define CHARSY 3 //char

#define VOIDSY 4 //void

#define MAINSY 5 //main

#define DOSY 6 //do

#define WHISY 7 //while

#define SWISY 8 //switch

#define CASESY 9 //case

#define IFSY 10 //if

#define SCANSY 11 //scanf

#define PRISY 12 //printf

#define RTSY 13 //return 前13个是保留字

#define IDSY 14 //普通标识符

#define PLUSSY 15 //+

#define MINUSSY 16 //-

#define STARSY 17 //\*

#define DIVSY 18 //'/'

#define COMMASY 19 //,

#define COLONSY 20 //:

#define LPARSY 21 //(

#define RPARSY 22 //)

#define LBRASY 23 //[

#define RBRASY 24 //]

#define LBRASSY 25 //{

#define RBRASSY 26 //}

#define ASSIGNSY 27 //=

#define EQUSY 28 //==

#define GTRSY 29 //>

#define GTESY 30 //>=

#define LTSY 31 //<

#define LESY 32 //<=

#define NEQSY 33 // !=

#define DIGITSY 34 //无符号整数

#define CHAR2SY 35 //字符

#define STRINGSY 36 //字符串

#define SEMISY 37 //;

#define MAXLIST 1024

#define MAXNAME 50

#define MAXFUNC 100

//mid\_code.c部分

#define MAX\_MID\_CODE 1024

//optimizer.c部分

#define MAX\_FUNC\_BLOCK 128

#define MAX\_BASIC\_BLOCK 256

#define MAX\_NODE\_NUM 100

#define MAX\_NUM\_ONE\_NODE 10

//error.c部分

#define FILE\_OPEN\_FAIL 0 //未能打开待编译文件

//下面是词法分析中的报错

#define ZERO\_START\_ERROR 1 //非零的整数以0开头

#define INVALID\_CHAR\_ERROR 2 //字符中的char是非法的

#define NO\_MATCH\_SQ\_ERROR 3 //单引号不匹配

#define NO\_MATCH\_DQ\_ERROR 4 //双引号不匹配

#define INVALID\_CHARACTER\_ERROR 5 //文件中出现语法不接受的成分

#define OUT\_OF\_BOUND\_ERROR 6 //整型常量的值超过int

//下面是语法和语义分析中的报错

#define OUT\_OF\_TABLE\_ERROR 7 //符号表已满，无法添加新的标识符

#define PARA\_NUM\_ERROR 8 //引用函数时，值参和原函数定义的形参的数目不同

#define FUNC\_REDEF\_ERROR 9 //定义函数时，发现符号表已经有同名的函数

#define VAR\_REDEF\_ERROR 10 //在同层里已经有同名变量定义

#define ID\_ERROR 11 //在类型标识符后面不是的sym不是标识符

#define CONST\_DEF\_ASSIGN\_ERROR 12 //在常量定义中，应该是等号的位置不是等号

#define CONST\_DEF\_TYPE\_ERROR 13 //常量定义中，等号后面的sym的类型和常量定义的类型不一致或者sym不是整数或者字符

#define CONST\_DEC\_TYPE\_ERROR 14 //常量声明中，常量的类型标识符不是int或char

#define SEMI\_ERROR 15 //应该是分号的地方打错成别的sym类型

#define VAR\_DEF\_ARRAYINDEX\_ERROR 16 //变量定义中，数组的索引不是无符号整数

#define RPAR\_ERROR 17 //应该时右小括号的地方sym类型不正确

#define RBRA\_ERROR 18 //应该时右中括号的地方sym类型不正确

#define RBRAS\_ERROR 19 //应该时右大括号的地方sym类型不正确

#define VAR\_KIND\_ERROR 20 //读语句中的标识符不是变量

#define MAINSY\_ERROR 21 //应该是main保留字的地方sym类型错误

#define LPAR\_ERROR 22 //应该时左小括号的地方sym类型不正确

#define LBRAS\_ERROR 23 //应该时左大括号的地方sym类型不正确

#define AFTER\_OP\_DIGIT\_ERROR 24 //在整数中，+-号后面的不是无符号整数

#define FACTOR\_ERROR 25 //在factor处理中认为所处理的部分不是factor,即识别符识别的第一个sym不符合factor任意一个分支

#define STATE\_AFTER\_ID\_ERROR 26 //在语句成分分析中，遇到了标识符作为开始，如果后面不是=,(,[中的一种，则需要报错

#define WRONG\_RELA\_OP\_ERROR 27 //错误的关系运算符

#define WHILESY\_ERROR 28 //循环语句中，分析完do和state后，应该是while保留字，不是则为错误

#define READ\_ARRAY\_ERROR 29 //读语句中，只能将读入的值存到普通变量中

#define VOID\_RT\_ERROR 30 //返回语句中，void函数的return后面有值

#define VAL\_RT\_ERROR 31 //返回语句中，非void函数的return后面没有值

#define NOT\_VOID\_MAIN\_ERROR 32 //main函数定义时前面不是void

#define ID\_DEC\_ERROR 33 //标识符声明时，后面应该是逗号、左小括号、左中括号或者分号，除此以外的标识符声明为错误格式(新)

#define NO\_RT\_FUNDEF\_ERROR 34 //有返回值函数定义时没有返回语句

#define CASE\_ERROR 35 //应该是case的地方是其他的symbol

#define CASE\_CONST\_ERROR 36 //case后面应该是常量，不是则报错

#define COLON\_ERROR 37 //应该是冒号的地方是其他symbol

#define ASSIGN\_ERROR 38 //应该是=的地方是其他symbol

#define FUNC\_NO\_DEF\_ERROR 39 //调用的函数未定义或者参数、类型不对

#define ID\_NO\_DEF\_ERROR 40 //标识符未定义或者类型不匹配

//中间代码生成中的报错

#define OUT\_OF\_CODE\_ERROR 41 //中间代码结构体已满，不能再存放代码

//语法分析中新增加的问题

#define CONST\_ASSIGNMENT\_ERROR 42 //常量被赋值

#define ARRAY\_ASSIGNMENT\_ERROR 43 //数组名字被直接赋值

#define CHAR\_ASSIGNMENT\_ERROR 44 //char不能被int赋值

//优化中的报错，用于提示自己

#define OUT\_OF\_FUNC\_BLOCK\_ERROR 45 //函数块太多

#define OUT\_OF\_BASIC\_BLOCK\_ERROR 46 //基本快太多

#define OUT\_OF\_NODE\_NUM 47 //节点表或者导出表或者dag的节点数目太多

#define OUT\_OF\_ONE\_NODE\_VAR 48 //一个节点上的变量数过多

//变量定义

//gramm.c定义的

int symbol; //读入的单词类型号

//特殊设计的全局变量

//1.保存现场用的

int pre\_symbol; //暂时记录symbol

char pre\_ch; //暂时记录CHAR

int pre\_line\_index; //暂时记录index\_in\_line

char PRETOKEN[100]; //用于在特殊时刻暂时记录TOKEN的值

//2.填符号表用的全局变量

//(0)公用的

int address = 0; //相对于当前AR的偏移地址

char TEMP[100];

int array\_flag = 0; //标识符是不是数组

int var\_flag = 0;

int const\_flag = 0;

int para\_flag = 0;

int global\_flag = 0; //当前的变量或者常量是不是全局的

//(1)函数声明和调用中需要的

int type\_symbol; //在函数声明中记录函数的type

char func\_name[100]; //记录函数名称，供填表和参数表应用

int para\_num; //记录函数的参数个数，填表用

int rt\_flag; //检查有返回值函数是不是有return语句

int main\_flag = 0; //在处理RT语句时，如果是main，直接退出

int void\_flag = 0;

int val\_flag = 0;

//(2)变量声明中需要的

char name\_of\_id[100]; //用于记录标识符的名字(包括常量和变量的名字)

int var\_type; //变量定义中记录变量的类型 或者 参数的类型

int var\_array\_size; //变量定义中记录数组变量的大小

//(3)常量声明中需要的

int const\_type; //常量定义中记录常量的类型

int const\_value; //常量定义中记录常量的值

//(4)表达式等需要的

int rela\_op\_symbol; //关系运算符的symbol值

int factor\_type = 0; //记录当前factor的类型

char now\_result[100];

//final\_result.c

char now\_func\_name[100];

int new\_const\_flag = 0;

int new\_var\_flag = 0;

int new\_para\_flag = 0;

int new\_global\_flag = 0;

//wordtest.c

///全局常量说明

const char CON[] = "const";

const char INT[] = "int";

const char \_CHAR[] = "char";

const char VOID[] = "void";

const char MAIN[] = "main";

const char DO[] = "do";

const char WHILE[] = "while";

const char SWITCH[] = "switch";

const char CASE[] = "case";

const char IF[] = "if";

const char SCANF[] = "scanf";

const char PRINTF[] = "printf";

const char RETURN[] = "return";

const int imax = 2147483647; //整形所要求的最大值

//全局变量声明

char CHAR;

char TOKEN[100]; //词法分析中连接不同字符组成标识符或者定义字

char read\_line[512];

long trans\_num = 0;

char out\_symbol[50];

int line = 0;

int index\_in\_line = 0;

int line\_lenth = 0;

//mid\_code.c

//全局常量说明

char CODE\_EMPTY[] = ""; // 应当是常量，但是为了函数参数形式符合，所以删去了const

char CODE\_INT[] = "INT";

char CODE\_CHAR[] = "CHAR";

char CODE\_CONST[] = "CONST";

char CODE\_ARRAY[] = "ARRAY";

char CODE\_FUNC[] = "FUNC";

char CODE\_PARA[] = "PARA";

char CODE\_CALL[] = "CALL";

char CODE\_RT[] = "RT";

char CODE\_EQU[] = "EQU";

char CODE\_NEQ[] = "NEQ";

char CODE\_LABEL[] = "LABEL";

char CODE\_SCAN[] = "SCAN";

char CODE\_PRINT[] = "PRINT";

char CODE\_GOTO[] = "GOTO";

char CODE\_END[] = "END";

char CODE\_ZERO[] = "0";

int label\_num = 0;

int var\_num = 0;

int new\_var\_num = 0;

//文件定义

//main.c

FILE\* compile\_file; //待编译的文件

FILE\* output\_file; //词法分析结果输出的文件

//final\_result.c

FILE\* final\_code;

//final\_result2.c

FILE\* final\_code2;

//mid\_code.c

FILE\* mid\_code\_file;

FILE\* mid\_code\_file2;

//结构体定义

//grammar.c

//符号表设计

struct one\_symbol {

char name[MAXNAME]; //标识符名称

int kind; /\*种类

0：常量

1：变量

2：函数

3：函数参数

\*/

int type; /\*类型

0：void(仅对于无返回值函数)

1：int

2: char

\*/

int value; /\*常量：值或ascii码

其他：0

\*/

int size; /\*函数：参数个数

数组：元素个数

其他：-1

\*/

int in\_address; //相对于所在AR首地址的偏移量(数组记录首地址)

};

struct table {

struct one\_symbol List[MAXLIST]; //符号表

int list\_index; //索引

int num\_of\_func; //已经生命的函数个数

int index\_of\_func[MAXFUNC]; //初始化全-1，存放函数的table索引

};

struct table Table;

struct table new\_Table;

struct table new\_Table2;

//mid\_code.c

//中间代码的结构体设计

struct mid\_code {

int op;

char src1[100];

char src2[100];

char result[100];

int is\_effective;

};

//这里面存储四元式的内容

struct mid\_code\_list {

struct mid\_code MID\_CODE[MAX\_MID\_CODE];

int code\_num;

};

struct mid\_code\_list CODE\_LIST;

//final\_result.c

//内部结构体设计，存放各个寄存器对应的变量名字 使用的部分全部注释了

//struct reg\_var {

// char reg\_name[20];

// char var\_name[100];

// int is\_var;//0: const 1: just var -1: no data

//};

//final\_result2.c

//内部结构体设计，存放各个寄存器对应的变量名字

struct s\_reg {

char var\_name[9][100];

int total\_var\_num;

};

struct s\_reg S\_REG;

//optimizer.c

//定义结构

struct func\_block {

int start\_num;

int basic\_blocks[MAX\_BASIC\_BLOCK];

int b\_block\_index;

};

struct func\_blocks {

struct func\_block FUNC\_LIST[MAX\_FUNC\_BLOCK];

int func\_index;

};

struct node {

char name[100];

int node\_num; //节点号码

};

struct node\_table {

struct node NODE\_LIST[MAX\_NODE\_NUM];

int var\_num;

};

struct node\_for\_dag {

int var\_num; //一个节点对应的变量有多少

int op;

int left\_son\_num;

int right\_son\_num;

char name[MAX\_NUM\_ONE\_NODE][100];

};

struct like\_dag {

struct node\_for\_dag NODE\_LIST[MAX\_NODE\_NUM];

int node\_num;

};

//全局变量

struct func\_blocks FUNC\_BLOCKS;

struct node\_table NODE\_TABLE;

struct node\_table EMPTY\_NODE\_TABLE;

struct like\_dag DAG;

//函数定义及实现

//先是定义

//error.c

void error(int error\_type, int line\_index);

//grammar.c

void initial\_table();

void append\_to\_table(char name[], int kind, int type, int value, int size, int in\_address);

void append\_para(int num);

int index\_in\_table(char name[], int kind);

int get\_func\_type(char name[]);

int get\_const\_value(char name[]);

void program();

void var\_dec();

void const\_dec();

void val\_func\_def();

void void\_func\_def();

void main\_func();

void const\_def();

void var\_def();

void dec\_head();

void para\_list();

void comp\_state();

void state\_list();

void state();

void if\_state();

void condition();

void rela\_op();

void loop\_state();

void switch\_state();

void situation\_list(char label[], char for\_compare[]);

void case\_state(char label[], char for\_compare[]);

void val\_fun\_call2();

void val\_fun\_call();

void void\_fun\_call();

void val\_para\_list();

void assign\_state();

void read\_state();

void write\_state();

void return\_state();

void expr();

void term();

void factor();

//wordtest.c

int insymbol();

//mid\_code.c函数定义

//中间代码的操作函数声明

void init\_code\_list();

void init\_var\_num();

void generate\_mid\_code(int op, char src1[], char scr2[], char result[]);

void generate\_label(char label[]);

void generate\_temp\_var(char var[]);

void generate\_new\_temp\_var(char var[]);

void print\_mid\_code();

void print\_one\_code(int number, FILE\* mid\_code\_file);

void init\_func\_list();

void add\_func\_block();

//final\_result.c函数声明：

//函数声明

void get\_new\_Table();

void data\_area\_generate();

void generate\_all\_final\_code();

void generate\_text();

int new\_index\_in\_tab(char name[]);

int get\_func\_para\_num(char name[]);

void load\_data(const char reg\_name[], char var\_name[]);

void store\_data(const char reg\_name[], char var\_name[]);

int get\_func\_total\_size(char name[]);

//之后是实现

//grammar.c

//关于符号表的操作

void initial\_table() {

Table.num\_of\_func = 0;

Table.list\_index = -1;

}

void append\_to\_table(char name[], int kind, int type, int value, int size, int in\_address) {

int index, func\_index;

if (Table.list\_index >= (MAXLIST - 1)) {

error(OUT\_OF\_TABLE\_ERROR, line);

fclose(compile\_file);

fclose(output\_file);

exit(-1);

return;

}

if (kind == 2) {

for (index = 0; index < Table.num\_of\_func; index++) {

func\_index = Table.index\_of\_func[index];

if (strcmp(Table.List[func\_index].name, name) == 0) { //函数名字重复

error(FUNC\_REDEF\_ERROR, line);

return;

}

}

Table.index\_of\_func[Table.num\_of\_func] = Table.list\_index + 1;

Table.num\_of\_func++;

}

else {

func\_index = Table.index\_of\_func[Table.num\_of\_func - 1];

for (index = func\_index; index <= Table.list\_index; index++) {

if (strcmp(Table.List[index].name, name) == 0) { //变量或常量同层有重名

error(VAR\_REDEF\_ERROR, line);

return;

}

}

}

Table.list\_index++;

strcpy(Table.List[Table.list\_index].name, name);

Table.List[Table.list\_index].kind = kind;

Table.List[Table.list\_index].type = type;

Table.List[Table.list\_index].size = size;

Table.List[Table.list\_index].value = value;

Table.List[Table.list\_index].in\_address = in\_address;

}

void append\_para(int num) {

int index;

index = Table.index\_of\_func[Table.num\_of\_func - 1];

Table.List[index].size = num;

}

int index\_in\_table(char name[], int kind) { //待改善

int i, index, first\_func\_index;

array\_flag = 0;

var\_flag = 0;

const\_flag = 0;

para\_flag = 0;

global\_flag = 0;

if (kind == 2) { //函数

for (i = 0; i < Table.num\_of\_func; i++) {

index = Table.index\_of\_func[i];

if (strcmp(name, Table.List[index].name) == 0)

break;

}

if (i == Table.num\_of\_func) {

error(FUNC\_NO\_DEF\_ERROR, line);

return -1;

}

else { //找到了同名函数

index = Table.index\_of\_func[i];

if (Table.List[index].size != para\_num) {

error(PARA\_NUM\_ERROR, line);

return index;

}

return index;

}

}

else { //普通标识符

index = Table.index\_of\_func[Table.num\_of\_func - 1];

for (; index <= Table.list\_index; index++) {

if (strcmp(Table.List[index].name, name) == 0) { //变量或常量同层有重名

break;

}

}

if (index > Table.list\_index) { //本层找不到，去全局变量找

first\_func\_index = Table.index\_of\_func[0];

for (index = 0; index < first\_func\_index; index++) {

if (strcmp(name, Table.List[index].name) == 0)

break;

}

if (index == first\_func\_index) {

error(ID\_NO\_DEF\_ERROR, line);

printf("wrong: %s\n", name);

return -1;

}

global\_flag = 1;

if (Table.List[index].kind == 1) {

if (Table.List[index].size != -1)

array\_flag = 1;

else

var\_flag = 1;

return Table.List[index].type;

}

else if (Table.List[index].kind == 0) {

const\_flag = 1;

return Table.List[index].type;

}

else if (Table.List[index].kind == 3) { //应该不会出现

para\_flag = 1;

return Table.List[index].type;

}

}

else { //本层找到

if (Table.List[index].kind == 1) {

if (Table.List[index].size != -1)

array\_flag = 1;

else

var\_flag = 1;

return Table.List[index].type;

}

else if (Table.List[index].kind == 0) {

const\_flag = 1;

return Table.List[index].type;

}

else if (Table.List[index].kind == 3) {

para\_flag = 1;

return Table.List[index].type;

}

}

}

return 0;

}

int get\_func\_type(char name[]) {

int i, index;

for (i = 0; i < Table.num\_of\_func; i++) {

index = Table.index\_of\_func[i];

if (strcmp(name, Table.List[index].name) == 0)

break;

}

if (i == Table.num\_of\_func) {

error(FUNC\_NO\_DEF\_ERROR, line);

return -1;

}

else { //找到了同名函数

return Table.List[index].type;

}

}

int get\_const\_value(char name[]) {

int index, first\_func\_index;

index = Table.index\_of\_func[Table.num\_of\_func - 1];

for (; index <= Table.list\_index; index++) {

if (strcmp(Table.List[index].name, name) == 0) { //变量或常量同层有重名

break;

}

}

if (index > Table.list\_index) { //本层找不到，去全局变量找

first\_func\_index = Table.index\_of\_func[0];

for (index = 0; index < first\_func\_index; index++) {

if (strcmp(name, Table.List[index].name) == 0)

break;

}

if (index == first\_func\_index) {

error(ID\_NO\_DEF\_ERROR, line);

printf("wrong: %s\n", name);

return -1;

}

return Table.List[index].value;

}

else { //本层找到

return Table.List[index].value;

}

}

//各部分的递归子程序

void program() { //程序

if (symbol == CONSY) { //读入const关键字

const\_dec();

}

while (symbol == INTSY || symbol == CHARSY) { //变量声明部分

pre\_ch = CHAR; //因为int或者char开头的可能是变量声明或者函数声明，那种情况需要恢复现场

pre\_line\_index = index\_in\_line;

pre\_symbol = symbol; //这里预存之前的信息

strcpy(PRETOKEN, TOKEN);

symbol = insymbol();

if (symbol != IDSY) {

error(ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != INTSY && symbol != CHARSY && symbol != VOIDSY);

continue;

}

symbol = insymbol(); //用来检测下一个单词，从而判别是变量还是函数(函数对应的是小括号)

if (symbol == LPARSY) { //检测到小括号，确定是函数声明

strcpy(TOKEN, PRETOKEN); //恢复现场，跳出循环，进入函数声明处理

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol;

break;

}

else if (symbol == SEMISY || symbol == COMMASY || symbol == LBRASY) { //确认是变量声明

strcpy(TOKEN, PRETOKEN); //恢复现场，进入变量定义模块

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol;

var\_def();

if (symbol != SEMISY) {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != INTSY && symbol != CHARSY && symbol != VOIDSY); //跳到下一个函数或者变量声明

continue;

}

else {

symbol = insymbol();

}

}

else {

//这里应该输出错误信息：标识符后面的字符错误(非法声明)，并skip到下一个int、char、void

error(ID\_DEC\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != INTSY && symbol != CHARSY && symbol != VOIDSY); //跳到下一个函数或者变量声明

continue;

}

}

while (symbol == INTSY || symbol == CHARSY || symbol == VOIDSY) { //函数声明部分

if (symbol == INTSY || symbol == CHARSY) {

val\_func\_def();

}

else {

pre\_ch = CHAR; //记录现场

pre\_line\_index = index\_in\_line;

pre\_symbol = symbol;

strcpy(PRETOKEN, TOKEN);

symbol = insymbol();

if (symbol == MAINSY) {

strcpy(TOKEN, PRETOKEN); //恢复现场，跳出进入主函数定义模块

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol;

break;

}

else {

strcpy(TOKEN, PRETOKEN); //恢复现场，进入无返回值函数定义模块(此处可优化？)

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol;

void\_func\_def();

}

}

}

main\_func();//之前已经完成了必要的变量声明以及函数声明

fprintf(output\_file, "This is a program!\n");

//printf("This is a program!\n");

}

void var\_dec() {

while (symbol == INTSY || symbol == CHARSY) {

var\_def();

if (symbol != SEMISY) {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != INTSY && symbol != CHARSY /\*&& symbol != VOIDSY\*/ && symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != RTSY); //跳到下一句变量声明或者语句列

}

else {

symbol = insymbol();

}

}

fprintf(output\_file, "This is a variable declaration!\n");

//printf("This is a variable declaration!\n");

}

void const\_dec() {

while (symbol == CONSY) {

symbol = insymbol(); //再读入下一个字符

const\_def();

if (symbol != SEMISY) { //定义完成之后必须有分号

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != CONSY && symbol != INTSY && symbol != CHARSY && symbol != VOIDSY && symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != RTSY); //跳到下一句常量声明或者变量声明或语句列

}

else {

symbol = insymbol();

}

}

fprintf(output\_file, "This is a constant declaration!\n");

//printf("This is a constant declaration!\n");

}

void val\_func\_def() {

type\_symbol = symbol;

rt\_flag = 0;

dec\_head();

if (symbol == LPARSY) {

val\_flag = 1;

void\_flag = 0;

if (type\_symbol == CHARSY) {

//将函数名称+类型填表char

generate\_mid\_code(8, func\_name, CODE\_EMPTY, CODE\_EMPTY);

add\_func\_block();

}

else {

//将函数名称+类型填表int

generate\_mid\_code(7, func\_name, CODE\_EMPTY, CODE\_EMPTY);

add\_func\_block();

}

symbol = insymbol();

para\_list();

if (symbol == RPARSY) {

symbol = insymbol();

if (symbol == LBRASSY) {

symbol = insymbol();

comp\_state();

if (symbol == RBRASSY) {

symbol = insymbol();

if (rt\_flag == 0) {

printf("%s\n", func\_name);

error(NO\_RT\_FUNDEF\_ERROR, line);

}

//生成四元式：函数的结束语句?

rt\_flag = 0;

fprintf(output\_file, "This is a value function defination!\n");

//printf("This is a value function defination!\n");

return;

}

else {

error(RBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != VOIDSY && symbol != INTSY && symbol != CHARSY);

return;

}

}

else {

error(LBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

else {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

else {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

void void\_func\_def() {

type\_symbol = VOIDSY;

symbol = insymbol();

if (symbol == IDSY) {

strcpy(func\_name, TOKEN);

//将函数名称+类型填表void

void\_flag = 1;

val\_flag = 0;

generate\_mid\_code(9, func\_name, CODE\_EMPTY, CODE\_EMPTY);

add\_func\_block();

address = 0;

append\_to\_table(func\_name, 2, 0, 0, 0, address);

symbol = insymbol();

if (symbol == LPARSY) {

symbol = insymbol();

para\_list();

if (symbol == RPARSY) {

symbol = insymbol();

if (symbol == LBRASSY) {

symbol = insymbol();

comp\_state();

if (symbol == RBRASSY) {

symbol = insymbol();

//生成四元式：函数的结束语句

generate\_mid\_code(25, CODE\_EMPTY, CODE\_EMPTY, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

fprintf(output\_file, "This is a void function defination!\n");

//printf("This is a void function defination!\n");

return;

}

else {

error(RBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != VOIDSY && symbol != INTSY && symbol != CHARSY);

return;

}

}

else {

error(LBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

else {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

else {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

else {

error(ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RBRASSY);

symbol = insymbol();

return;

}

}

void main\_func() { //且看主函数里面的操作

if (symbol != VOIDSY) { //这里的主函数只有VOID类型，不能接受其他的可能性

error(NOT\_VOID\_MAIN\_ERROR, line);

return; //跳的多么？

}

symbol = insymbol();

if (symbol == MAINSY) {

//填表+生成四元式

strcpy(func\_name, TOKEN);

val\_flag = 0;

void\_flag = 1;

main\_flag = 1;

generate\_mid\_code(9, func\_name, CODE\_EMPTY, CODE\_EMPTY); ///这里调用了生成中间代码的函数

add\_func\_block();

address = 0;

append\_to\_table(func\_name, 2, 0, 0, 0, address);

symbol = insymbol();

if (symbol != LPARSY) { // （

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != LBRASSY); //[

}

else {

symbol = insymbol(); //（） 里面的数据

}

if (symbol != RPARSY) { //要求main函数里面不能有任何的参数

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != LBRASSY);

}

else {

symbol = insymbol();

}

if (symbol != LBRASSY) { //之后必须直接大括号{

error(LBRAS\_ERROR, line);

return;

}

symbol = insymbol();

comp\_state(); //这个函数是用来处理主函数里面的各项信息，但是只出现了一次

if (symbol == RBRASSY) {

//生成四元式：函数的结束语句

generate\_mid\_code(34, CODE\_EMPTY, CODE\_EMPTY, CODE\_EMPTY);

fprintf(output\_file, "This is a main function defination!\n");

//sprintf("This is a main function defination!\n");

return;

}

else {

error(RBRAS\_ERROR, line);

return;

}

}

else {

error(MAINSY\_ERROR, line);

return; //跳的多么？

}

}

void const\_def() { //常量定义部分

int sign = 1; //1:+ 0:-

if (symbol == INTSY || symbol == CHARSY) { //确定是int还是char

const\_type = symbol;

do {

symbol = insymbol();

if (symbol == IDSY) { //这是普通标识符

strcpy(name\_of\_id, TOKEN);

symbol = insymbol();

if (symbol == ASSIGNSY) { //这是=

symbol = insymbol();

if (symbol == PLUSSY || symbol == MINUSSY) {// 这是+/-

sign = (symbol == PLUSSY) ? 1 : 0; //确定符号

symbol = insymbol();

if (symbol == DIGITSY || const\_type == INTSY) { //之后是数字或者int？？

if (trans\_num == 0) { //这里是报错了，但是如果输入数字就是单纯的0怎么办？？？

error(AFTER\_OP\_DIGIT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY); //,和;

continue;

}

if (sign == 1)

const\_value = trans\_num;

else

const\_value = 0 - trans\_num;

address = address + 1;

append\_to\_table(name\_of\_id, 0, 1, const\_value, -1, address);

sprintf(TEMP, "%d", const\_value);

generate\_mid\_code(1, name\_of\_id, TEMP, CODE\_EMPTY);

//填表，生成四元式

symbol = insymbol();

fprintf(output\_file, "This is an integer!\n");

//printf("This is an integer!\n");

}

else {

error(CONST\_DEF\_TYPE\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else if (symbol == DIGITSY && const\_type == INTSY) {

sign = 1;

const\_value = trans\_num;

//填表，生成四元式

address = address + 1;

append\_to\_table(name\_of\_id, 0, 1, const\_value, -1, address);

sprintf(TEMP, "%d", const\_value);

generate\_mid\_code(1, name\_of\_id, TEMP, CODE\_EMPTY);

symbol = insymbol();

fprintf(output\_file, "This is an integer!\n");

//printf("This is an integer!\n");

}

else if (symbol == CHAR2SY && const\_type == CHARSY) {

sign = 1;

const\_value = TOKEN[1];

//填表，生成四元式

address = address + 1;

append\_to\_table(name\_of\_id, 0, 2, const\_value, -1, address);

sprintf(TEMP, "%d", const\_value);

generate\_mid\_code(2, name\_of\_id, TEMP, CODE\_EMPTY);

symbol = insymbol();

fprintf(output\_file, "This is a char!\n");

//printf("This is a char!\n");

}

else {

error(CONST\_DEF\_TYPE\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else {

error(CONST\_DEF\_ASSIGN\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else {

error(ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue; //?

}

} while (symbol == COMMASY);

fprintf(output\_file, "This is a constant defination!\n");

//printf("This is a constant defination!\n");

return;

}

else {

error(CONST\_DEC\_TYPE\_ERROR, line); //const之后 ，必须接int或者char不然报错

do {

symbol = insymbol();

} while (symbol != SEMISY);

return;

}

}

void var\_def() {

int real\_type;

var\_type = symbol;

if (var\_type == INTSY) {

real\_type = 1;

}

else {

real\_type = 2;

}

do {

symbol = insymbol();

if (symbol == IDSY) {

strcpy(name\_of\_id, TOKEN);

symbol = insymbol();

var\_array\_size = 0;

if (symbol == COMMASY || symbol == SEMISY) {

//是普通变量，该变量填表

address = address + 1;

append\_to\_table(name\_of\_id, 1, real\_type, 0, -1, address);

if (real\_type == 1) {

generate\_mid\_code(3, name\_of\_id, CODE\_EMPTY, CODE\_EMPTY);

}

else {

generate\_mid\_code(4, name\_of\_id, CODE\_EMPTY, CODE\_EMPTY);

}

continue;

}

else if (symbol == LBRASY) { //数组

symbol = insymbol();

if (symbol == DIGITSY) {

var\_array\_size = trans\_num;

if (var\_array\_size != 0) {

symbol = insymbol();

if (symbol == RBRASY) {

//把数组的名字、大小、类型等填表

address = address + var\_array\_size;

append\_to\_table(name\_of\_id, 1, real\_type, 0, var\_array\_size, address);

sprintf(TEMP, "%d", var\_array\_size);

if (real\_type == 1) {

generate\_mid\_code(5, name\_of\_id, TEMP, CODE\_EMPTY);

}

else {

generate\_mid\_code(6, name\_of\_id, TEMP, CODE\_EMPTY);

}

symbol = insymbol();

}

else {

error(RBRA\_ERROR, line); //0不是无符号整数

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else {

error(VAR\_DEF\_ARRAYINDEX\_ERROR, line); //0不是无符号整数

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else {

error(VAR\_DEF\_ARRAYINDEX\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else {

error(ID\_DEC\_ERROR, line); //标识符格式错误

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

}

else {

error(ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != COMMASY && symbol != SEMISY);

continue;

}

} while (symbol == COMMASY);

fprintf(output\_file, "This is a variable defination!\n");

//printf("This is a variable defination!\n");

}

void dec\_head() {

int real\_type;

if (type\_symbol == INTSY) {

real\_type = 1;

}

else {

real\_type = 2;

}

symbol = insymbol();

if (symbol == IDSY) {

strcpy(func\_name, TOKEN);

address = 0;

append\_to\_table(func\_name, 2, real\_type, 0, 0, address);

//将函数的类型+名称填表

symbol = insymbol();

fprintf(output\_file, "This is a defination head!\n");

//printf("This is a defination head!\n");

return;

}

else {

error(ID\_ERROR, line);

return;

}

}

void para\_list() {

int real\_type;

para\_num = 0;

do {

if (symbol == COMMASY) {

symbol = insymbol();

}

if (symbol == INTSY || symbol == CHARSY) {

var\_type = symbol;

if (var\_type == INTSY) {

real\_type = 1;

}

else {

real\_type = 2;

}

symbol = insymbol();

if (symbol == IDSY) {

para\_num++;

strcpy(name\_of\_id, TOKEN);

address = address + 1;

append\_to\_table(name\_of\_id, 3, real\_type, 0, -1, address);

if (real\_type == 1) {

generate\_mid\_code(10, name\_of\_id, CODE\_INT, CODE\_EMPTY);

}

else {

generate\_mid\_code(10, name\_of\_id, CODE\_CHAR, CODE\_EMPTY);

}

//参数填表(类型在var\_type中)

symbol = insymbol();

}

else {

error(ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RPARSY);

return;

}

}

} while (symbol == COMMASY);

append\_para(para\_num);

//把函数的参数总体个数填表

fprintf(output\_file, "This is a parameter list!\n");

//printf("This is a parameter list!\n");

}

void comp\_state() { //那么能分开写吗“倘若处理语句在之后？

if (symbol == CONSY) { //考虑了常量定义及变量定义

const\_dec();

}

if (symbol == INTSY || symbol == CHARSY) {

var\_dec();

}

state\_list();

fprintf(output\_file, "This is a composite statement!\n");

//printf("This is a composite statement!\n");

}

void state\_list() { //这个语句会处理所有可能的语句并持续处理，if,do,switch,{,标识符，scanf，printf,;,return

while (symbol == IFSY || symbol == DOSY || symbol == SWISY || symbol == LBRASSY || symbol == IDSY || symbol == SCANSY || symbol == PRISY || symbol == SEMISY || symbol == RTSY) {

state();

}

fprintf(output\_file, "This is a statement list!\n");//各种状态语句

//printf("This is a statement list!\n");

}

void state() {

int func\_type;

init\_var\_num();

if (symbol == IFSY) {

if\_state();

}

else if (symbol == DOSY) {

loop\_state();

}

else if (symbol == SWISY) {

switch\_state();

}

else if (symbol == LBRASSY) { //大括号里面肯定接的是其他的语句块

symbol = insymbol();

state\_list();

if (symbol == RBRASSY) {

symbol = insymbol();

}

else {

error(RBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else if (symbol == IDSY) {

//函数或者赋值

pre\_ch = CHAR; //记录现场

pre\_line\_index = index\_in\_line;

pre\_symbol = symbol;

strcpy(PRETOKEN, TOKEN);

symbol = insymbol();

if (symbol == LPARSY) {

//函数调用

strcpy(TOKEN, PRETOKEN); //恢复现场，进入函数调用处理

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol; //先一并处理了

func\_type = get\_func\_type(TOKEN);

//查表，确认是哪种函数，暂时都写成这样

if (func\_type == 0) {

void\_fun\_call();

}

else if (func\_type == 1 || func\_type == 2) {

val\_fun\_call();

}

else {

error(FUNC\_NO\_DEF\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

if (symbol == SEMISY) {

symbol = insymbol();

fprintf(output\_file, "This is a statement!\n");

//printf("This is a statement!\n");

return;

}

else {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else if (symbol == ASSIGNSY || symbol == LBRASY) {

strcpy(TOKEN, PRETOKEN); //恢复现场，进入赋值语句处理

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol;

assign\_state();

if (symbol == SEMISY) {

symbol = insymbol();

fprintf(output\_file, "This is a statement!\n");

//printf("This is a statement!\n");

return;

}

else {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(STATE\_AFTER\_ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else if (symbol == SCANSY) {

read\_state();

if (symbol == SEMISY) {

symbol = insymbol();

}

else {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else if (symbol == PRISY) {

write\_state();

if (symbol == SEMISY) {

symbol = insymbol();

}

else {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else if (symbol == RTSY) {

return\_state();

if (symbol == SEMISY) {

symbol = insymbol();

}

else {

error(SEMI\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else if (symbol == SEMISY) {

symbol = insymbol();

}

else {

return; //暂时跳过

}

fprintf(output\_file, "This is a statement!\n");

//printf("This is a statement!\n");

}

void if\_state() {

char label[100];

char condition\_result[100];

generate\_label(label);

symbol = insymbol();

if (symbol == LPARSY) {

symbol = insymbol();

condition();

strcpy(condition\_result, now\_result);

generate\_mid\_code(22, condition\_result, label, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

if (symbol == RPARSY) {

symbol = insymbol();

state();

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

generate\_mid\_code(24, label, CODE\_EMPTY, CODE\_EMPTY);

fprintf(output\_file, "This is an if statement!\n");

//printf("This is an if statement!\n");

return;

}

else {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

void condition() {

char num1[100];

char num2[100];

strcpy(num1, CODE\_EMPTY);

strcpy(num2, CODE\_EMPTY);

expr();

strcpy(num1, now\_result);

if (symbol == EQUSY || symbol == LTSY || symbol == LESY || symbol == GTRSY || symbol == GTESY || symbol == NEQSY) { //这里的判断保险么？？？？？？？？？？？？

rela\_op();

expr();

strcpy(num2, now\_result);

generate\_temp\_var(now\_result);

switch (rela\_op\_symbol) {

case EQUSY: {

generate\_mid\_code(20, num1, num2, now\_result);

break;

}

case LTSY: {

generate\_mid\_code(17, num1, num2, now\_result);

break;

}

case LESY: {

generate\_mid\_code(16, num1, num2, now\_result);

break;

}

case GTRSY: {

generate\_mid\_code(19, num1, num2, now\_result);

break;

}

case GTESY: {

generate\_mid\_code(18, num1, num2, now\_result);

break;

}

case NEQSY: {

generate\_mid\_code(21, num1, num2, now\_result);

break;

}

}

}

else {

generate\_temp\_var(now\_result);

generate\_mid\_code(21, num1, CODE\_ZERO, now\_result);

}

//address = address+1;

//append\_to\_table(now\_result,1,1,0,-1,address); 233

fprintf(output\_file, "This is a condition!\n");

//printf("This is a condition!\n");

}

void rela\_op() {

if (symbol == EQUSY || symbol == GTRSY || symbol == GTESY || symbol == LTSY || symbol == LESY || symbol == NEQSY) {

rela\_op\_symbol = symbol;

symbol = insymbol();

}

else {

error(WRONG\_RELA\_OP\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

fprintf(output\_file, "This is a relational operation!\n");

//printf("This is a relational operation!\n");

}

void loop\_state() {

char label1[100];

generate\_label(label1);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

generate\_mid\_code(24, label1, CODE\_EMPTY, CODE\_EMPTY);

symbol = insymbol();

state();

if (symbol == WHISY) {

symbol = insymbol();

if (symbol == LPARSY) {

symbol = insymbol();

condition();

if (symbol == RPARSY) {

symbol = insymbol();

generate\_mid\_code(23, now\_result, label1, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

fprintf(output\_file, "This is a loop statement!\n");

//printf("This is a loop statement!\n");

return;

}

else {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(WHILESY\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

void switch\_state() {

char label[100];

char for\_compare[100]; //expr算出来的结果

generate\_label(label);

symbol = insymbol();

if (symbol == LPARSY) {

symbol = insymbol();

expr();

strcpy(for\_compare, now\_result);

if (symbol == RPARSY) {

symbol = insymbol();

if (symbol == LBRASSY) {

symbol = insymbol();

situation\_list(label, for\_compare);

if (symbol == RBRASSY) {

symbol = insymbol();

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

generate\_mid\_code(24, label, CODE\_EMPTY, CODE\_EMPTY);

fprintf(output\_file, "This is a switch statement!\n");

//printf("This is a switch statement!\n");

return;

}

else {

error(RBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(LBRAS\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

else {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

void situation\_list(char label[], char for\_compare[]) {

do {

case\_state(label, for\_compare);

} while (symbol == CASESY);

fprintf(output\_file, "This is a situation list!\n");

//printf("This is a situation list!\n");

}

void case\_state(char label[], char for\_compare[]) {

char compare\_result[100];

char my\_label[100];

//generate\_temp\_var(compare\_result);

generate\_new\_temp\_var(compare\_result);

//address = address+1;

//append\_to\_table(compare\_result,1,1,0,-1,address);

generate\_label(my\_label);

if (symbol == CASESY) {

symbol = insymbol();

if (symbol == CHAR2SY) {

const\_value = TOKEN[1];

sprintf(TEMP, "%d", const\_value);

generate\_mid\_code(20, for\_compare, TEMP, compare\_result);

generate\_mid\_code(22, compare\_result, my\_label, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

fprintf(output\_file, "This is a char!\n");

//printf("This is a char!\n");

fprintf(output\_file, "This is a constant!\n");

//printf("This is a constant!\n");

}

else if (symbol == PLUSSY || symbol == MINUSSY) {

symbol = insymbol();

if (symbol != DIGITSY) {

error(AFTER\_OP\_DIGIT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

if (trans\_num == 0) {

error(AFTER\_OP\_DIGIT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

generate\_mid\_code(20, for\_compare, TOKEN, compare\_result);

generate\_mid\_code(22, compare\_result, my\_label, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

fprintf(output\_file, "This is an integer!\n");

//printf("This is an integer!\n");

fprintf(output\_file, "This is a constant!\n");

//printf("This is a constant!\n");

}

else if (symbol == DIGITSY) {

generate\_mid\_code(20, for\_compare, TOKEN, compare\_result);

generate\_mid\_code(22, compare\_result, my\_label, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

fprintf(output\_file, "This is an integer!\n");

//printf("This is an integer!\n");

fprintf(output\_file, "This is a constant!\n");

//printf("This is a constant!\n");

}

else {

error(CASE\_CONST\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

symbol = insymbol();

if (symbol != COLONSY) {

error(COLON\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

symbol = insymbol();

state();

generate\_mid\_code(33, label, CODE\_EMPTY, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

generate\_mid\_code(24, my\_label, CODE\_EMPTY, CODE\_EMPTY);

fprintf(output\_file, "This is a case statement!\n");

//printf("This is a case statement!\n");

return;

}

else {

error(CASE\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

}

void val\_fun\_call2() {

int type;

char my\_func\_name[100];

strcpy(my\_func\_name, TOKEN); //这个保险么？？？？？func\_name 应该不会贝更改？

symbol = insymbol(); //假设标识符和(已经检查过

symbol = insymbol();

val\_para\_list();

type = get\_func\_type(my\_func\_name);

if (type != 1 && type != 2) {

error(FUNC\_NO\_DEF\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RPARSY);

symbol = insymbol();

return;

}

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

generate\_temp\_var(now\_result);

//address = address+1;

//append\_to\_table(now\_result,1,1,0,-1,address);

generate\_mid\_code(14, my\_func\_name, CODE\_EMPTY, now\_result);

if (factor\_type == 0) {

if (type == 1) {

factor\_type = 1;

}

else {

factor\_type = 2;

}

}

symbol = insymbol();

fprintf(output\_file, "This is a value function call!\n");

//printf("This is a value function call!\n");

}

void val\_fun\_call() {

int type;

char my\_func\_name[100];

strcpy(my\_func\_name, TOKEN); //这个保险么？？？？？func\_name 应该不会贝更改？

symbol = insymbol(); //假设标识符和(已经检查过

symbol = insymbol();

val\_para\_list();

type = get\_func\_type(my\_func\_name);

if (type != 1 && type != 2) {

error(FUNC\_NO\_DEF\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RPARSY);

symbol = insymbol();

return;

}

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

generate\_mid\_code(14, my\_func\_name, CODE\_EMPTY, CODE\_EMPTY);

symbol = insymbol();

fprintf(output\_file, "This is a value function call!\n");

//printf("This is a value function call!\n");

}

void void\_fun\_call() {

int type;

char my\_func\_name[100];

strcpy(my\_func\_name, TOKEN); //这个保险么？？？？？func\_name 应该不会贝更改？

symbol = insymbol(); //假设标识符和(已经检查过

symbol = insymbol();

val\_para\_list();

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

type = get\_func\_type(my\_func\_name);

if (type != 0) {

error(FUNC\_NO\_DEF\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != RPARSY);

symbol = insymbol();

return;

}

symbol = insymbol();

generate\_mid\_code(15, my\_func\_name, CODE\_EMPTY, CODE\_EMPTY);

fprintf(output\_file, "This is a void function call!\n");

//printf("This is a void function call!\n");

}

void val\_para\_list() {

para\_num = 0;

do {

if (symbol == COMMASY) {

symbol = insymbol();

}

if (symbol == PLUSSY || symbol == MINUSSY || symbol == IDSY || symbol == CHAR2SY || symbol == DIGITSY || symbol == LPARSY) {

expr();

generate\_mid\_code(35, now\_result, CODE\_EMPTY, CODE\_EMPTY);

para\_num++;

}

} while (symbol == COMMASY);

fprintf(output\_file, "This is a value parameter list!\n");

//printf("This is a value parameter list!\n");

return;

}

void assign\_state() {

char src1[100];

char src2[100];

char result[100];

int nothing;

//char temp[50];

strcpy(result, TOKEN);

symbol = insymbol();

if (symbol == ASSIGNSY) {

symbol = insymbol();

expr();

nothing = index\_in\_table(result, 0); //这里要判断：1.常量赋值错误 2.char被int赋值 3.数组整体赋值

if (const\_flag) {

error(CONST\_ASSIGNMENT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

if (array\_flag) {

error(ARRAY\_ASSIGNMENT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

if (nothing == 2 && factor\_type == 1) {

error(CHAR\_ASSIGNMENT\_ERROR, line);

}

strcpy(src1, now\_result);

generate\_mid\_code(11, src1, CODE\_EMPTY, result);

fprintf(output\_file, "This is an assignment statement!\n");

//printf("This is an assignment statement!\n");

return;

}

else { //数组元素赋值

symbol = insymbol();

expr();

strcpy(src2, now\_result);

if (symbol != RBRASY) {

error(RBRA\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

symbol = insymbol();

if (symbol != ASSIGNSY) {

error(ASSIGN\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

symbol = insymbol();

expr();

strcpy(src1, now\_result);

generate\_mid\_code(12, src1, src2, result);

fprintf(output\_file, "This is an assignment statement!\n");

//printf("This is an assignment statement!\n");

return;

}

}

void read\_state() {

char var\_name[100];

int get\_index;

symbol = insymbol();

if (symbol != LPARSY) {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

do {

symbol = insymbol();

if (symbol != IDSY) {

error(ID\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

strcpy(var\_name, TOKEN);

get\_index = index\_in\_table(var\_name, 0);

factor\_type = get\_index;

if (var\_flag == 0) {

error(READ\_ARRAY\_ERROR, line);

}

if (factor\_type == 1) {

generate\_mid\_code(27, CODE\_INT, CODE\_EMPTY, var\_name);

}

else {

generate\_mid\_code(27, CODE\_CHAR, CODE\_EMPTY, var\_name);

}

symbol = insymbol();

} while (symbol == COMMASY);

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

symbol = insymbol();

fprintf(output\_file, "This is a read statement!\n");

//printf("This is a read statement!\n");

}

void write\_state() {

char src1[100];

char src2[100];

strcpy(src1, CODE\_EMPTY);

strcpy(src2, CODE\_EMPTY);

symbol = insymbol();

if (symbol != LPARSY) {

error(LPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

symbol = insymbol();

if (symbol == STRINGSY) {

strcpy(src1, TOKEN);

fprintf(output\_file, "This is a string!\n");

//printf("This is a string!\n");

symbol = insymbol();

if (symbol == COMMASY) {

symbol = insymbol();

expr();

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

strcpy(src2, now\_result);

symbol = insymbol();

if (factor\_type == 1) {

generate\_mid\_code(28, src1, src2, CODE\_INT);

}

else {

generate\_mid\_code(28, src1, src2, CODE\_CHAR);

}

fprintf(output\_file, "This is a write statement!\n");

//printf("This is a write statement!\n");

return;

}

else if (symbol == RPARSY) {

symbol = insymbol();

generate\_mid\_code(28, src1, src2, CODE\_EMPTY);

fprintf(output\_file, "This is a write statement!\n");

//printf("This is a write statement!\n");

return;

}

else {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

//暂时没有else

}

else {

expr();

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

strcpy(src2, now\_result);

if (factor\_type == 1) {

generate\_mid\_code(28, src1, src2, CODE\_INT);

}

else {

generate\_mid\_code(28, src1, src2, CODE\_CHAR);

}

symbol = insymbol();

fprintf(output\_file, "This is a write statement!\n");

//printf("This is a write statement!\n");

return;

}

}

void return\_state() {

char src[100];

symbol = insymbol();

if (symbol == LPARSY) {

if (void\_flag == 1) {

error(VOID\_RT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY);

return;

}

rt\_flag = 1;

symbol = insymbol();

expr();

strcpy(src, now\_result);

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

generate\_mid\_code(26, src, CODE\_EMPTY, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

symbol = insymbol();

fprintf(output\_file, "This is a return statement!\n");

//printf("This is a return statement!\n");

return;

}

else {

if (val\_flag == 1) {

error(VAL\_RT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != IFSY && symbol != DOSY && symbol != SWISY && symbol != LBRASSY && symbol != IDSY && symbol != SCANSY && symbol != PRISY && symbol != SEMISY && symbol != RTSY);

return;

}

if (main\_flag) {

generate\_mid\_code(34, CODE\_EMPTY, CODE\_EMPTY, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

}

else {

generate\_mid\_code(25, CODE\_EMPTY, CODE\_EMPTY, CODE\_EMPTY);

//generate\_mid\_code(37,CODE\_EMPTY,CODE\_EMPTY,CODE\_EMPTY);

}

fprintf(output\_file, "This is a return statement!\n");

//printf("This is a return statement!\n");

return;

}

}

void expr() {

factor\_type = 0;

char src1[100];

char src2[100];

char result[100];

if (symbol == PLUSSY || symbol == MINUSSY) {

factor\_type = 1;

if (symbol == PLUSSY) {

symbol = insymbol();

term();

strcpy(result, now\_result);

}

else {

symbol = insymbol();

term();

strcpy(src2, now\_result);

generate\_temp\_var(result);

//address = address+1;

//append\_to\_table(result,1,1,0,-1,address);

generate\_mid\_code(30, CODE\_ZERO, src2, result);

}

while (symbol == PLUSSY || symbol == MINUSSY) {

factor\_type = 1;

strcpy(src1, result);

generate\_temp\_var(result);

//address = address+1;

//append\_to\_table(result,1,1,0,-1,address);

if (symbol == PLUSSY) { //默认用的是同一个临时变量存储的

symbol = insymbol();

term();

strcpy(src2, now\_result);

generate\_mid\_code(29, src1, src2, result);

}

else {

symbol = insymbol();

term();

strcpy(src2, now\_result);

generate\_mid\_code(30, src1, src2, result);

}

}

}

else {

term();

strcpy(result, now\_result);

while (symbol == PLUSSY || symbol == MINUSSY) {

factor\_type = 1;

strcpy(src1, result);

generate\_temp\_var(result);

//address = address+1;

//append\_to\_table(result,1,1,0,-1,address);

if (symbol == PLUSSY) { //默认用的是同一个临时变量存储的

symbol = insymbol();

term();

strcpy(src2, now\_result);

generate\_mid\_code(29, src1, src2, result);

}

else {

symbol = insymbol();

term();

strcpy(src2, now\_result);

generate\_mid\_code(30, src1, src2, result);

}

}

}

strcpy(now\_result, result);

fprintf(output\_file, "This is an expression!\n");

//printf("This is an expression!\n");

}

void term() {

char src1[100];

char src2[100];

char result[100];

factor();

strcpy(result, now\_result);

while (symbol == STARSY || symbol == DIVSY) {

strcpy(src1, result);

generate\_temp\_var(result);

//address = address+1;

//append\_to\_table(result,1,1,0,-1,address);

if (symbol == STARSY) {

symbol = insymbol();

factor();

strcpy(src2, now\_result);

generate\_mid\_code(31, src1, src2, result);

}

else {

symbol = insymbol();

factor();

strcpy(src2, now\_result);

generate\_mid\_code(32, src1, src2, result);

}

}

strcpy(now\_result, result);

fprintf(output\_file, "This is a term!\n");

//printf("This is a term!\n");

}

void factor() {

char result[100];

char id\_record[100];

int temp\_type;

int sign;

strcpy(result, CODE\_EMPTY);

if (symbol == IDSY) {

strcpy(id\_record, TOKEN);

//保存现场？

pre\_ch = CHAR; //保护现场

pre\_line\_index = index\_in\_line;

pre\_symbol = symbol;

strcpy(PRETOKEN, TOKEN);

symbol = insymbol();

if (symbol == LBRASY) {

symbol = insymbol();

temp\_type = factor\_type;

expr();

factor\_type = temp\_type;

if (symbol != RBRASY) {

error(RBRA\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY && symbol != ASSIGNSY && symbol != PLUSSY && symbol != MINUSSY && symbol != STARSY && symbol != DIVSY && symbol != RPARSY && symbol != COMMASY && symbol != LTSY && symbol != LESY && symbol != GTRSY && symbol != GTESY && symbol != EQUSY && symbol != NEQSY);

return;

}

//这个地方应该是要检查变量是否存在的

temp\_type = index\_in\_table(id\_record, 0);

if (array\_flag == 0) {

error(ID\_NO\_DEF\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY && symbol != ASSIGNSY && symbol != PLUSSY && symbol != MINUSSY && symbol != STARSY && symbol != DIVSY && symbol != RPARSY && symbol != COMMASY && symbol != LTSY && symbol != LESY && symbol != GTRSY && symbol != GTESY && symbol != EQUSY && symbol != NEQSY);

return;

}

if (factor\_type == 0) {

factor\_type = temp\_type;

}

strcpy(result, now\_result);

generate\_temp\_var(now\_result);

//address = address+1;

//append\_to\_table(now\_result,1,1,0,-1,address);

generate\_mid\_code(36, id\_record, result, now\_result);

symbol = insymbol();

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

else if (symbol == LPARSY) {

strcpy(TOKEN, PRETOKEN); //恢复现场

CHAR = pre\_ch;

index\_in\_line = pre\_line\_index;

symbol = pre\_symbol;

val\_fun\_call2();

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

else {

if (factor\_type == 0) {

factor\_type = index\_in\_table(id\_record, 0);

}

temp\_type = index\_in\_table(id\_record, 0);

if (const\_flag == 1) {

temp\_type = get\_const\_value(id\_record);

sprintf(now\_result, "%d", temp\_type);

const\_flag = 0;

}

else {

strcpy(now\_result, id\_record);

}

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

}

else if (symbol == CHAR2SY) {

if (factor\_type == 0) {

factor\_type = 2;

}

const\_value = TOKEN[1];

sprintf(now\_result, "%d", const\_value);

symbol = insymbol();

fprintf(output\_file, "This is a char!\n");

//printf("This is a char!\n");

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

else if (symbol == PLUSSY || symbol == MINUSSY) {

if (symbol == PLUSSY) {

sign = 1;

}

else {

sign = -1;

}

symbol = insymbol();

if (symbol != DIGITSY) {

error(AFTER\_OP\_DIGIT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY && symbol != ASSIGNSY && symbol != PLUSSY && symbol != MINUSSY && symbol != STARSY && symbol != DIVSY && symbol != RPARSY && symbol != COMMASY && symbol != LTSY && symbol != LESY && symbol != GTRSY && symbol != GTESY && symbol != EQUSY && symbol != NEQSY);

return;

}

if (trans\_num == 0) {

error(AFTER\_OP\_DIGIT\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY && symbol != ASSIGNSY && symbol != PLUSSY && symbol != MINUSSY && symbol != STARSY && symbol != DIVSY && symbol != RPARSY && symbol != COMMASY && symbol != LTSY && symbol != LESY && symbol != GTRSY && symbol != GTESY && symbol != EQUSY && symbol != NEQSY);

return;

}

trans\_num = sign \* trans\_num;

sprintf(now\_result, "%ld", trans\_num);

factor\_type = 1;

symbol = insymbol();

fprintf(output\_file, "This is an integer!\n");

//printf("This is an integer!\n");

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

else if (symbol == DIGITSY) {

sprintf(now\_result, "%ld", trans\_num);

factor\_type = 1;

symbol = insymbol();

fprintf(output\_file, "This is an integer!\n");

//printf("This is an integer!\n");

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

else if (symbol == LPARSY) {

symbol = insymbol();

expr();

if (symbol != RPARSY) {

error(RPAR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY && symbol != ASSIGNSY && symbol != PLUSSY && symbol != MINUSSY && symbol != STARSY && symbol != DIVSY && symbol != RPARSY && symbol != COMMASY && symbol != LTSY && symbol != LESY && symbol != GTRSY && symbol != GTESY && symbol != EQUSY && symbol != NEQSY);

return;

}

symbol = insymbol();

fprintf(output\_file, "This is a factor!\n");

//printf("This is a factor!\n");

return;

}

else {

error(FACTOR\_ERROR, line);

do {

symbol = insymbol();

} while (symbol != SEMISY && symbol != ASSIGNSY && symbol != PLUSSY && symbol != MINUSSY && symbol != STARSY && symbol != DIVSY && symbol != RPARSY && symbol != COMMASY && symbol != LTSY && symbol != LESY && symbol != GTRSY && symbol != GTESY && symbol != EQUSY && symbol != NEQSY);

return;

}

}

//error.c 唯一一个函数，根据输入的结果，打印出错误类型的提示

void error(int error\_type, int line\_index) {

switch (error\_type) {

case 1: {

fprintf(output\_file, "line %d : non-zero integer shouldn't start with 0'\n", line\_index);

printf("line %d : non-zero integer shouldn't start with 0'\n", line\_index);

break;

}

case 2: {

fprintf(output\_file, "line %d : invalid char being declared\n", line\_index);

printf("line %d : invalid char being declared\n", line\_index);

break;

}

case 3: {

fprintf(output\_file, "line %d : single quatation don't match\n", line\_index);

printf("line %d : single quatation don't match\n", line\_index);

break;

}

case 4: {

fprintf(output\_file, "line %d : double quatation don't match\n", line\_index);

printf("line %d : double quatation don't match\n", line\_index);

break;

}

case 5: {

fprintf(output\_file, "line %d : invalid char appears\n", line\_index);

printf("line %d : invalid char appears\n", line\_index);

break;

}

case 6: {

fprintf(output\_file, "line %d : integer out of the range\n", line\_index);

printf("line %d : integer out of the range\n", line\_index);

break;

}

case 7: {

fprintf(output\_file, "line %d : table overflow!\n", line\_index);

printf("line %d : table overflow!\n", line\_index);

break;

}

case 8: {

fprintf(output\_file, "line %d: the parameter number doesn't match!\n", line\_index);

printf("line %d: the parameter number doesn't match!\n", line\_index);

break;

}

case 9: {

fprintf(output\_file, "line %d: this function name already exists!\n", line\_index);

printf("line %d: this function name already exists!\n", line\_index);

break;

}

case 10: {

fprintf(output\_file, "line %d: this variable name already exists!\n", line\_index);

printf("line %d: this variable name already exists!\n", line\_index);

break;

}

case 11: {

fprintf(output\_file, "line %d: this place should be an Identifier after type identifier!\n", line\_index);

printf("line %d: this place should be an Identifier after type identifier!\n", line\_index);

break;

}

case 12: {

fprintf(output\_file, "line %d: this place should be an equal in constant defination!\n", line\_index);

printf("line %d: this place should be an equal in constant defination!\n", line\_index);

break;

}

case 13: {

fprintf(output\_file, "line %d: the type doesn't match in cosntant defination!\n", line\_index);

printf("line %d: the type doesn't match in cosntant defination!\n", line\_index);

break;

}

case 14: {

fprintf(output\_file, "line %d: the type identifier is wrong in cosntant defination!\n", line\_index);

printf("line %d: the type identifier is wrong in cosntant defination!\n", line\_index);

break;

}

case 15: {

fprintf(output\_file, "line %d: this place should be a semicolon!\n", line\_index);

printf("line %d: this place should be a semicolon!\n", line\_index);

break;

}

case 16: {

fprintf(output\_file, "line %d: the index of array is not unsigned integer in variable defination!\n", line\_index);

printf("line %d: the index of array is not unsigned integer in variable defination!\n", line\_index);

break;

}

case 17: {

fprintf(output\_file, "line %d: this place should be a right parenthese!\n", line\_index);

printf("line %d: this place should be a right parenthese!\n", line\_index);

break;

}

case 18: {

fprintf(output\_file, "line %d: this place should be a right bracket!\n", line\_index);

printf("line %d: this place should be a right bracket!\n", line\_index);

break;

}

case 19: {

fprintf(output\_file, "line %d: this place should be a right brace!\n", line\_index);

printf("line %d: this place should be a right brace!\n", line\_index);

break;

}

case 20: {

fprintf(output\_file, "line %d: the identifier is not a variable in read statement!\n", line\_index);

printf("line %d: the identifier is not a variable in read statement!\n", line\_index);

break;

}

case 21: {

fprintf(output\_file, "line %d: this place should be the key word main!\n", line\_index);

printf("line %d: this place should be the key word main!\n", line\_index);

break;

}

case 22: {

fprintf(output\_file, "line %d: this place should be a left parenthese!\n", line\_index);

printf("line %d: this place should be a left parenthese!\n", line\_index);

break;

}

case 23: {

fprintf(output\_file, "line %d: this place should be a left brace!\n", line\_index);

printf("line %d: this place should be a left brace!\n", line\_index);

break;

}

case 24: {

fprintf(output\_file, "line %d: there should be an unsigned integer after + or -!\n", line\_index);

printf("line %d: there should be an unsigned integer after + or -!\n", line\_index);

break;

}

case 25: {

fprintf(output\_file, "line %d: there should be a factor here!\n", line\_index);

printf("line %d: there should be a factor here!\n", line\_index);

break;

}

case 26: {

fprintf(output\_file, "line %d: there should be a left parenthese or a left bracket or a left brace after identifier!\n", line\_index);

printf("line %d: there should be a left parenthese or a left bracket or a left brace after identifier!\n", line\_index);

break;

}

case 27: {

fprintf(output\_file, "line %d: wrong relational operater!\n", line\_index);

printf("line %d: wrong relational operater!\n", line\_index);

break;

}

case 28: {

fprintf(output\_file, "line %d: this place should be the key word while!\n", line\_index);

printf("line %d: this place should be the key word while!\n", line\_index);

break;

}

case 29: {

fprintf(output\_file, "line %d: array or const is not allowed to appear in read statement!\n", line\_index);

printf("line %d: array or const is not allowed to appear in read statement!\n", line\_index);

break;

}

case 30: {

fprintf(output\_file, "line %d: in void function, should not return value!\n", line\_index);

printf("line %d: in void function, should not return value!\n", line\_index);

break;

}

case 31: {

fprintf(output\_file, "line %d: in int or char function, should return value!\n", line\_index);

printf("line %d: in int or char function, should return value!\n", line\_index);

break;

}

case 32: {

fprintf(output\_file, "line %d: there should be void before the key word main!\n", line\_index);

printf("line %d: there should be void before the key word main!\n", line\_index);

break;

}

case 33: {

fprintf(output\_file, "line %d: mistake after identifier in identifier declararion!\n", line\_index);

printf("line %d: mistake after identifier in identifier declararion!\n", line\_index);

break;

}

case 34: {

fprintf(output\_file, "line %d: in int or char function, should have return statement!\n", line\_index);

printf("line %d: in int or char function, should have return statement!\n", line\_index);

break;

}

case 35: {

fprintf(output\_file, "line %d: this place should be the key word case!\n", line\_index);

printf("line %d: this place should be the key word case!\n", line\_index);

break;

}

case 36: {

fprintf(output\_file, "line %d: this place should be a constant after case!\n", line\_index);

printf("line %d: this place should be a constant after case!\n", line\_index);

break;

}

case 37: {

fprintf(output\_file, "line %d: this place should be a colon!\n", line\_index);

printf("line %d: this place should be a colon!\n", line\_index);

break;

}

case 38: {

fprintf(output\_file, "line %d: this place should be an equal!\n", line\_index);

printf("line %d: this place should be an equal!\n", line\_index);

break;

}

case 39: {

fprintf(output\_file, "line %d: undefined function call or the num of value para is not equal to defined para or the type is wrong!\n", line\_index);

printf("line %d: undefined function call or the num of value para is not equal to defined para or the type is wrong!\n", line\_index);

break;

}

case 40: {

fprintf(output\_file, "line %d: undefined identifier or wrong type!\n", line\_index);

printf("line %d: undefined identifier or wrong type!\n", line\_index);

break;

}

case 41: {

fprintf(output\_file, "line %d: mid\_code out of index!\n", line\_index);

printf("line %d: mid\_code out of index!\n", line\_index);

break;

}

case 42: {

fprintf(output\_file, "line %d: constant can't be assigned!\n", line\_index);

printf("line %d: constant can't be assigned!\n", line\_index);

break;

}

case 43: {

fprintf(output\_file, "line %d: array's name can't be assigned!\n", line\_index);

printf("line %d: array's name can't be assigned!\n", line\_index);

break;

}

case 44: {

fprintf(output\_file, "line %d: char shouldn't be assigned by int!(warning)\n", line\_index);

printf("line %d: char shouldn't be assigned by int!(warning)\n", line\_index);

break;

}

case 45: {

fprintf(output\_file, "line %d: function block out of range(optimizer)\n", line\_index);

printf("line %d: function block out of range(optimizer)\n", line\_index);

break;

}

case 46: {

fprintf(output\_file, "line %d: basic block out of range(optimizer)\n", line\_index);

printf("line %d: basic block out of range(optimizer)\n", line\_index);

break;

}

case 47: {

fprintf(output\_file, "line %d: num of node out of range(optimizer)\n", line\_index);

printf("line %d: num of node out of range(optimizer)\n", line\_index);

break;

}

case 48: {

fprintf(output\_file, "line %d: num of var one node out of range(optimizer)\n", line\_index);

printf("line %d: num of var one node out of range(optimizer)\n", line\_index);

break;

}

}

//fclose(output\_file);

return;

}

//wordtest.c

int get\_ch() {

if (index\_in\_line == line\_lenth) {

//index\_in\_line初始为0，line\_lenth初始为0，此处意为一行已经读到尽头

line++;

if (fgets(read\_line, 512, compile\_file) != NULL) { //compile\_file中读取一行或512个字符，read\_line作为缓冲数组

line\_lenth = strlen(read\_line); //读取这一行的长度

index\_in\_line = 0; //需读位置从0开始

return('\n'); //相当于都到了行末尾的回车

}

else

return(EOF);

}

else {

index\_in\_line += 1;

return(read\_line[index\_in\_line - 1]);//返回一个字节

}

}

void translate(int sym\_num) {

switch (sym\_num) {

case CONSY: {

strcpy(out\_symbol, "CONSY");

break;

}

case INTSY: {

strcpy(out\_symbol, "INTSY");

break;

}

case CHARSY: {

strcpy(out\_symbol, "CHARSY");

break;

}

case VOIDSY: {

strcpy(out\_symbol, "VOIDSY");

break;

}

case MAINSY: {

strcpy(out\_symbol, "MAINSY");

break;

}

case DOSY: {

strcpy(out\_symbol, "DOSY");

break;

}

case WHISY: {

strcpy(out\_symbol, "WHISY");

break;

}

case SWISY: {

strcpy(out\_symbol, "SWISY");

break;

}

case CASESY: {

strcpy(out\_symbol, "CASESY");

break;

}

case IFSY: {

strcpy(out\_symbol, "IFSY");

break;

}

case SCANSY: {

strcpy(out\_symbol, "SCANSY");

break;

}

case PRISY: {

strcpy(out\_symbol, "PRISY");

break;

}

case RTSY: {

strcpy(out\_symbol, "RTSY");

break;

}

case IDSY: {

strcpy(out\_symbol, "IDSY");

break;

}

case PLUSSY: {

strcpy(out\_symbol, "PLUSSY");

break;

}

case MINUSSY: {

strcpy(out\_symbol, "MINUSSY");

break;

}

case STARSY: {

strcpy(out\_symbol, "STARSY");

break;

}

case DIVSY: {

strcpy(out\_symbol, "DIVSY");

break;

}

case COMMASY: {

strcpy(out\_symbol, "COMMASY");

break;

}

case COLONSY: {

strcpy(out\_symbol, "COLONSY");

break;

}

case LPARSY: {

strcpy(out\_symbol, "LPARSY");

break;

}

case RPARSY: {

strcpy(out\_symbol, "RPARSY");

break;

}

case LBRASY: {

strcpy(out\_symbol, "LBRASY");

break;

}

case RBRASY: {

strcpy(out\_symbol, "RBRASY");

break;

}

case LBRASSY: {

strcpy(out\_symbol, "LBRASSY");

break;

}

case RBRASSY: {

strcpy(out\_symbol, "RBRASSY");

break;

}

case ASSIGNSY: {

strcpy(out\_symbol, "ASSIGNSY");

break;

}

case EQUSY: {

strcpy(out\_symbol, "EQUSY");

break;

}

case GTRSY: {

strcpy(out\_symbol, "GTRSY");

break;

}

case GTESY: {

strcpy(out\_symbol, "GTESY");

break;

}

case LTSY: {

strcpy(out\_symbol, "LTSY");

break;

}

case LESY: {

strcpy(out\_symbol, "LESY");

break;

}

case NEQSY: {

strcpy(out\_symbol, "NEQSY");

break;

}

case DIGITSY: {

strcpy(out\_symbol, "DIGITSY");

break;

}

case CHAR2SY: {

strcpy(out\_symbol, "CHAR2SY");

break;

}

case STRINGSY: {

strcpy(out\_symbol, "STRINGSY");

break;

}

case SEMISY: {

strcpy(out\_symbol, "SEMISY");

break;

}

}

}

void reset\_token() { //为下一次读入单词做准备

int i;

for (i = 0; i < 100; i++)

TOKEN[i] = '\0';

return;

}

int trans\_to\_num() { //将数字字符串转化为数字

int len, i;

long long result;

len = strlen(TOKEN);

result = 0;

for (i = 0; i < len; i++) {

result = result \* 10 + TOKEN[i] - '0';

if (result > imax) {

error(OUT\_OF\_BOUND\_ERROR, line);

return 0;

}

}

return result;

}

void cat(char c) {

int i = 0;

while (TOKEN[i] != '\0') {

i++;

if (i == 99) {

printf("too long words\n");

return;

}

}

TOKEN[i] = c;

}

//连接一个新字符进入TOKEN数组中并且判定是否越界

int is\_letter() {

return (isalpha(CHAR) || (CHAR == '\_'));

}

//isalpha()函数是C语言内置函数判断是不是字母字符

int is\_digit() {

return isdigit(CHAR);

}

//检测是否十进制数字

int is\_colon() {

if (CHAR == ':')

return 1;

return 0;

}

//检测是否':'

int is\_semi() {

if (CHAR == ';')

return 1;

return 0;

}

//检测是否分号

int is\_comma() {

if (CHAR == ',')

return 1;

return 0;

}

//检测逗号

int is\_plus() {

if (CHAR == '+')

return 1;

return 0;

}

int is\_minus() {

if (CHAR == '-')

return 1;

return 0;

}

int is\_star() {

if (CHAR == '\*')

return 1;

return 0;

}

int is\_div() {

if (CHAR == '/')

return 1;

return 0;

}

int is\_squo() {

if (CHAR == '\'')

return 1;

return 0;

}

//检测是否'\'反斜杠

int is\_dquo() {

if (CHAR == '\"')

return 1;

return 0;

}

//检测是否'\"'双引号字符

int is\_lparent() {

if (CHAR == '\(')

return 1;

return 0;

}

//左小括号

int is\_rparent() {

if (CHAR == ')')

return 1;

return 0;

}

//右小括号

int is\_lbracket() {

if (CHAR == '[')

return 1;

return 0;

}

//左中括号

int is\_rbracket() {

if (CHAR == ']')

return 1;

return 0;

}

//右中括号

int is\_lbrace() {

if (CHAR == '{')

return 1;

return 0;

}

//左大括号

int is\_rbrace() {

if (CHAR == '}')

return 1;

return 0;

}

//右大括号

int is\_gtr() {

if (CHAR == '>')

return 1;

return 0;

}

//大于

int is\_lt() {

if (CHAR == '<')

return 1;

return 0;

}

//小于

int is\_equ() {

if (CHAR == '=')

return 1;

return 0;

}

//等于

int is\_exc() {

if (CHAR == '!')

return 1;

return 0;

}

//感叹

int is\_zero() {

if (CHAR == '0')

return 1;

return 0;

}

//判0

int is\_valid\_character() {

if (CHAR == 32)

return 1;

else if (CHAR == 33)

return 1;

else if ((CHAR >= 35) && (CHAR <= 126))

return 1;

else

return 0;

}//可显示字符

int is\_key() {

if (strcmp(TOKEN, CON) == 0)

return CONSY;

else if (strcmp(TOKEN, INT) == 0)

return INTSY;

else if (strcmp(TOKEN, \_CHAR) == 0)

return CHARSY;

else if (strcmp(TOKEN, VOID) == 0)

return VOIDSY;

else if (strcmp(TOKEN, MAIN) == 0)

return MAINSY;

else if (strcmp(TOKEN, DO) == 0)

return DOSY;

else if (strcmp(TOKEN, WHILE) == 0)

return WHISY;

else if (strcmp(TOKEN, SWITCH) == 0)

return SWISY;

else if (strcmp(TOKEN, CASE) == 0)

return CASESY;

else if (strcmp(TOKEN, IF) == 0)

return IFSY;

else if (strcmp(TOKEN, SCANF) == 0)

return SCANSY;

else if (strcmp(TOKEN, PRINTF) == 0)

return PRISY;

else if (strcmp(TOKEN, RETURN) == 0)

return RTSY;

else

return 0;

}

//判断是定义字还是普通标识符(返回0)

//判断是否标识符

int insymbol() {

int symbol;

int zero\_flag = 0;

reset\_token(); //token数组全置空

while ((CHAR == ' ') || (CHAR == '\t') || (CHAR == '\n')) { //相当于预处理，空格换行TAB跳过

CHAR = get\_ch();

if (CHAR == EOF) //文件末尾符号为EOF

{

printf("end of the file\n");

fprintf(output\_file, "end of the file\n"); //这句的输出说明了已经实现了全文编译

//fclose(compile\_file);

//fclose(output\_file);

//print\_mid\_code();

exit(0);

return -2;

}

}

if (is\_letter()) {

do {

if (isupper(CHAR)) //判断是否大写字符

cat(tolower(CHAR));//转换成小写字符并且写入进token数组中

else

cat(CHAR);

CHAR = get\_ch(); //然后输入新的字符

} while (is\_letter() || is\_digit()); //只有还是字母或者\_或者数字，接可以继续连接

symbol = is\_key();

if (symbol == 0)

symbol = IDSY;

}

else if (is\_digit()) {

if (is\_zero()) {

zero\_flag = 1;

}

do {

cat(CHAR);

CHAR = get\_ch();

} while (is\_digit()); //所以0后面不能直接跟其他的数字

symbol = DIGITSY;

trans\_num = trans\_to\_num(); //这里是返回具体的数字而且不应该是0

if ((zero\_flag == 1) && (trans\_num != 0)) {

error(ZERO\_START\_ERROR, line);

}

}

else if (is\_plus()) {

cat(CHAR);

CHAR = get\_ch();

symbol = PLUSSY;

}

else if (is\_minus()) {

cat(CHAR);

CHAR = get\_ch();

symbol = MINUSSY;

}

else if (is\_star()) {

cat(CHAR);

CHAR = get\_ch();

symbol = STARSY;

}

else if (is\_div()) {

cat(CHAR);

CHAR = get\_ch();

symbol = DIVSY;

}

else if (is\_gtr()) {

cat(CHAR);

CHAR = get\_ch();

if (is\_equ()) {

cat(CHAR);

symbol = GTESY;

CHAR = get\_ch();

}

else

symbol = GTRSY;

}

else if (is\_lt()) {

cat(CHAR);

CHAR = get\_ch();

if (is\_equ()) {

cat(CHAR);

symbol = LESY;

CHAR = get\_ch();

}

else

symbol = LTSY;

}

else if (is\_equ()) {

cat(CHAR);

CHAR = get\_ch();

if (is\_equ()) {

cat(CHAR);

symbol = EQUSY;

CHAR = get\_ch();

}

else

symbol = ASSIGNSY;

}

else if (is\_exc()) {

cat(CHAR);

CHAR = get\_ch();

if (is\_equ()) {

cat(CHAR);

symbol = NEQSY;

CHAR = get\_ch();

}

else {

error(INVALID\_CHARACTER\_ERROR, line);

symbol = -1; //这里怎么处理？

}

}

else if (is\_lparent()) {

cat(CHAR);

CHAR = get\_ch();

symbol = LPARSY;

}

else if (is\_rparent()) {

cat(CHAR);

CHAR = get\_ch();

symbol = RPARSY;

}

else if (is\_lbracket()) {

cat(CHAR);

CHAR = get\_ch();

symbol = LBRASY;

}

else if (is\_rbracket()) {

cat(CHAR);

CHAR = get\_ch();

symbol = RBRASY;

}

else if (is\_lbrace()) {

cat(CHAR);

CHAR = get\_ch();

symbol = LBRASSY;

}

else if (is\_rbrace()) {

cat(CHAR);

CHAR = get\_ch();

symbol = RBRASSY;

}

else if (is\_comma()) {

cat(CHAR);

CHAR = get\_ch();

symbol = COMMASY;

}

else if (is\_colon()) {

cat(CHAR);

CHAR = get\_ch();

symbol = COLONSY;

}

else if (is\_semi()) {

cat(CHAR);

CHAR = get\_ch();

symbol = SEMISY;

}

else if (is\_squo()) {

cat(CHAR);

CHAR = get\_ch();

if (is\_plus() || is\_minus() || is\_star() || is\_div() || is\_letter() || is\_digit()) {

cat(CHAR);

}

else {

error(INVALID\_CHAR\_ERROR, line);

}

CHAR = get\_ch();

if (is\_squo()) {

cat(CHAR);

CHAR = get\_ch();

symbol = CHAR2SY;

}

else {

error(NO\_MATCH\_SQ\_ERROR, line);

CHAR = get\_ch();

symbol = -1;

}

}

else if (is\_dquo()) {

cat(CHAR);

CHAR = get\_ch();

while (is\_valid\_character()) {

if (CHAR == '\\')

cat(CHAR);

cat(CHAR);

CHAR = get\_ch();

}

if (is\_dquo()) {

cat(CHAR);

CHAR = get\_ch();

symbol = STRINGSY;

}

else {

error(NO\_MATCH\_DQ\_ERROR, line);

symbol = STRINGSY;

}

}

else {

symbol = -1;

error(INVALID\_CHARACTER\_ERROR, line);

}

return symbol;

}

//mid\_code.c

//函数具体实现

void init\_code\_list() {

CODE\_LIST.code\_num = 0;

}

void init\_var\_num() {

var\_num = 0;

}

void generate\_mid\_code(int op, char src1[], char src2[], char result[]) { //这里主要是体现一个填值的作用

int index;

if (CODE\_LIST.code\_num >= MAX\_MID\_CODE) { //中间代码量太大则报错

error(OUT\_OF\_CODE\_ERROR, line);

fclose(compile\_file);

fclose(output\_file);

exit(-1);

return;

}

index = CODE\_LIST.code\_num;

CODE\_LIST.MID\_CODE[index].op = op;

strcpy(CODE\_LIST.MID\_CODE[index].src1, src1);

strcpy(CODE\_LIST.MID\_CODE[index].src2, src2);

strcpy(CODE\_LIST.MID\_CODE[index].result, result);

CODE\_LIST.MID\_CODE[index].is\_effective = 1;

CODE\_LIST.code\_num++;

}

void generate\_label(char label[]) {

sprintf(label, "LABEL%d", label\_num);

label\_num++;

return;

}

void generate\_temp\_var(char var[]) {

sprintf(var, "$VAR%d", var\_num);

var\_num++;

return;

}

void generate\_new\_temp\_var(char var[]) {

sprintf(var, "$S%d", new\_var\_num);

new\_var\_num++;

return;

}

void print\_mid\_code() {

int i;

mid\_code\_file = fopen("mid\_code.txt", "w");

for (i = 0; i < CODE\_LIST.code\_num; i++) {

if (CODE\_LIST.MID\_CODE[i].is\_effective == 1) {

print\_one\_code(i, mid\_code\_file);

}

}

fclose(mid\_code\_file);

}

void print\_one\_code(int number, FILE\* mid\_code\_file) {

int op\_type;

op\_type = CODE\_LIST.MID\_CODE[number].op;

switch (op\_type) {

case 1: {

fprintf(mid\_code\_file, "CONST INT %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 2: {

fprintf(mid\_code\_file, "CONST CHAR %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 3: {

fprintf(mid\_code\_file, "INT %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 4: {

fprintf(mid\_code\_file, "CHAR %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 5: {

fprintf(mid\_code\_file, "ARRAY INT %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 6: {

fprintf(mid\_code\_file, "ARRAY CHAR %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 7: {

fprintf(mid\_code\_file, "FUNC INT %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 8: {

fprintf(mid\_code\_file, "FUNC CHAR %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 9: {

fprintf(mid\_code\_file, "FUNC VOID %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 10: {

fprintf(mid\_code\_file, "PARA %s %s\n", CODE\_LIST.MID\_CODE[number].src2, CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 11: {

fprintf(mid\_code\_file, "%s = %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 12: {

fprintf(mid\_code\_file, "%s[%s] = %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src2, CODE\_LIST.MID\_CODE[number].src1);

break;

}

/\*case 13:{//never use

fprintf(mid\_code\_file,"SAVE\n");

break;

}\*/

case 14: {

if (strcmp(CODE\_EMPTY, CODE\_LIST.MID\_CODE[number].result) == 0) {

fprintf(mid\_code\_file, "CALL VALUE %s\n", CODE\_LIST.MID\_CODE[number].src1);

}

else {

fprintf(mid\_code\_file, "CALL VALUE %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].result);

}

break;

}

case 15: {

fprintf(mid\_code\_file, "CALL VOID %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 16: {

fprintf(mid\_code\_file, "%s = %s <= %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 17: {

fprintf(mid\_code\_file, "%s = %s < %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 18: {

fprintf(mid\_code\_file, "%s = %s >= %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 19: {

fprintf(mid\_code\_file, "%s = %s > %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 20: {

fprintf(mid\_code\_file, "%s = %s == %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 21: {

fprintf(mid\_code\_file, "%s = %s != %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 22: {

fprintf(mid\_code\_file, "EQUZ %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 23: {

fprintf(mid\_code\_file, "NEQZ %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 24: {

fprintf(mid\_code\_file, "SET %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 25: {

fprintf(mid\_code\_file, "RT\n");

break;

}

case 26: {

fprintf(mid\_code\_file, "RT %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 27: {

fprintf(mid\_code\_file, "SCAN %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].result);

break;

}

case 28: {

if (strcmp(CODE\_EMPTY, CODE\_LIST.MID\_CODE[number].src1) == 0) {

fprintf(mid\_code\_file, "PRINT %s %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src2);

}

else if (strcmp(CODE\_EMPTY, CODE\_LIST.MID\_CODE[number].src2) == 0) {

fprintf(mid\_code\_file, "PRINT %s\n", CODE\_LIST.MID\_CODE[number].src1);

}

else {

fprintf(mid\_code\_file, "PRINT %s %s %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src2);

}

break;

}

case 29: {

fprintf(mid\_code\_file, "%s = %s + %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 30: {

fprintf(mid\_code\_file, "%s = %s - %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 31: {

fprintf(mid\_code\_file, "%s = %s \* %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 32: {

fprintf(mid\_code\_file, "%s = %s / %s\n", CODE\_LIST.MID\_CODE[number].result, CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2);

break;

}

case 33: {

fprintf(mid\_code\_file, "GOTO %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 34: {

fprintf(mid\_code\_file, "END\n");

break;

}

case 35: {

fprintf(mid\_code\_file, "PUSH %s\n", CODE\_LIST.MID\_CODE[number].src1);

break;

}

case 36: {

fprintf(mid\_code\_file, "GET %s[%s] %s\n", CODE\_LIST.MID\_CODE[number].src1, CODE\_LIST.MID\_CODE[number].src2, CODE\_LIST.MID\_CODE[number].result);

break;

}

case 37: {

fprintf(mid\_code\_file, "NOP\n");

break;

}

}

}

void init\_func\_list() {

FUNC\_BLOCKS.func\_index = -1;

}

void add\_func\_block() {

if (FUNC\_BLOCKS.func\_index == MAX\_FUNC\_BLOCK - 1) {

error(OUT\_OF\_FUNC\_BLOCK\_ERROR, line);

return;

}

FUNC\_BLOCKS.func\_index++;

FUNC\_BLOCKS.FUNC\_LIST[FUNC\_BLOCKS.func\_index].start\_num = CODE\_LIST.code\_num - 1;

FUNC\_BLOCKS.FUNC\_LIST[FUNC\_BLOCKS.func\_index].b\_block\_index = -1;

//printf("func: %d\n",FUNC\_BLOCKS.FUNC\_LIST[FUNC\_BLOCKS.func\_index].start\_num);

}

//final\_result.c 函数实现：

//函数实现

void get\_new\_Table() {

int index, f\_index, i;

new\_Table.num\_of\_func = 0;

new\_Table.list\_index = -1;

for (index = 0; index < Table.index\_of\_func[0]; index++) {

new\_Table.list\_index++;

new\_Table.List[new\_Table.list\_index].in\_address = Table.List[index].in\_address;

new\_Table.List[new\_Table.list\_index].kind = Table.List[index].kind;

new\_Table.List[new\_Table.list\_index].size = Table.List[index].size;

new\_Table.List[new\_Table.list\_index].type = Table.List[index].type;

new\_Table.List[new\_Table.list\_index].value = Table.List[index].value;

strcpy(new\_Table.List[new\_Table.list\_index].name, Table.List[index].name);

}

for (f\_index = 0; f\_index < Table.num\_of\_func; f\_index++) {

new\_Table.index\_of\_func[new\_Table.num\_of\_func] = new\_Table.list\_index + 1;

if (f\_index != Table.num\_of\_func - 1) {

for (index = Table.index\_of\_func[f\_index]; index < Table.index\_of\_func[f\_index + 1]; index++) {

if (Table.List[index].name[0] != '$') {

new\_Table.list\_index++;

new\_Table.List[new\_Table.list\_index].in\_address = Table.List[index].in\_address;

new\_Table.List[new\_Table.list\_index].kind = Table.List[index].kind;

new\_Table.List[new\_Table.list\_index].size = Table.List[index].size;

new\_Table.List[new\_Table.list\_index].type = Table.List[index].type;

new\_Table.List[new\_Table.list\_index].value = Table.List[index].value;

strcpy(new\_Table.List[new\_Table.list\_index].name, Table.List[index].name);

}

}

//接下来添加修改后代码的临时变量

for (index = FUNC\_BLOCKS.FUNC\_LIST[f\_index].start\_num; index < FUNC\_BLOCKS.FUNC\_LIST[f\_index + 1].start\_num; index++) {

if (CODE\_LIST.MID\_CODE[index].is\_effective == 0)

continue;

if (strcmp(CODE\_LIST.MID\_CODE[index].result, CODE\_EMPTY) == 0)

continue;

if (CODE\_LIST.MID\_CODE[index].result[0] == '$') {

for (i = new\_Table.index\_of\_func[new\_Table.num\_of\_func]; i <= new\_Table.list\_index; i++) {

if (strcmp(CODE\_LIST.MID\_CODE[index].result, new\_Table.List[i].name) == 0)

break;

}

if (i > new\_Table.list\_index) {

new\_Table.list\_index++;

new\_Table.List[new\_Table.list\_index].in\_address = new\_Table.List[new\_Table.list\_index - 1].in\_address + 1;

new\_Table.List[new\_Table.list\_index].kind = 1;

new\_Table.List[new\_Table.list\_index].size = -1;

new\_Table.List[new\_Table.list\_index].type = 1;

new\_Table.List[new\_Table.list\_index].value = 0;

strcpy(new\_Table.List[new\_Table.list\_index].name, CODE\_LIST.MID\_CODE[index].result);

}

}

}

}

else {

for (index = Table.index\_of\_func[f\_index]; index <= Table.list\_index; index++) {

if (Table.List[index].name[0] != '$') {

new\_Table.list\_index++;

new\_Table.List[new\_Table.list\_index].in\_address = Table.List[index].in\_address;

new\_Table.List[new\_Table.list\_index].kind = Table.List[index].kind;

new\_Table.List[new\_Table.list\_index].size = Table.List[index].size;

new\_Table.List[new\_Table.list\_index].type = Table.List[index].type;

new\_Table.List[new\_Table.list\_index].value = Table.List[index].value;

strcpy(new\_Table.List[new\_Table.list\_index].name, Table.List[index].name);

}

}

//接下来添加修改后代码的临时变量

for (index = FUNC\_BLOCKS.FUNC\_LIST[f\_index].start\_num; index < CODE\_LIST.code\_num; index++) {

if (CODE\_LIST.MID\_CODE[index].is\_effective == 0)

continue;

if (strcmp(CODE\_LIST.MID\_CODE[index].result, CODE\_EMPTY) == 0)

continue;

if (CODE\_LIST.MID\_CODE[index].result[0] == '$') {

for (i = new\_Table.index\_of\_func[new\_Table.num\_of\_func]; i <= new\_Table.list\_index; i++) {

if (strcmp(CODE\_LIST.MID\_CODE[index].result, new\_Table.List[i].name) == 0)

break;

}

if (i > new\_Table.list\_index) {

new\_Table.list\_index++;

new\_Table.List[new\_Table.list\_index].in\_address = new\_Table.List[new\_Table.list\_index - 1].in\_address + 1;

new\_Table.List[new\_Table.list\_index].kind = 1;

new\_Table.List[new\_Table.list\_index].size = -1;

new\_Table.List[new\_Table.list\_index].type = 1;

new\_Table.List[new\_Table.list\_index].value = 0;

strcpy(new\_Table.List[new\_Table.list\_index].name, CODE\_LIST.MID\_CODE[index].result);

}

}

}

}

new\_Table.num\_of\_func++;

}

}

void load\_data(const char reg\_name[], char var\_name[]) {

int address;

if (islower(var\_name[0]) || var\_name[0] == '\_' || var\_name[0] == '$') {//要取的是变量或者常量

address = new\_index\_in\_tab(var\_name);

if (new\_global\_flag) {

if (new\_const\_flag) {

fprintf(final\_code, "li %s %d\n", reg\_name, address);

}

else if (new\_var\_flag) {

fprintf(final\_code, "la $s7 %s\n", var\_name);

fprintf(final\_code, "lw %s 0($s7)\n", reg\_name);

}

}

else {

if (new\_const\_flag) {

fprintf(final\_code, "li %s %d\n", reg\_name, address);

}

else if (new\_var\_flag) {

fprintf(final\_code, "lw %s %d($fp)\n", reg\_name, -(address + 1) \* 4);

}

else if (new\_para\_flag) {

fprintf(final\_code, "lw %s %d($fp)\n", reg\_name, -(address + 1) \* 4);

}

}

}

else { //载入的是数字

fprintf(final\_code, "li %s %s\n", reg\_name, var\_name);

}

}

void store\_data(const char reg\_name[], char var\_name[]) {

address = new\_index\_in\_tab(var\_name);

if (new\_global\_flag) { //全局变量，应该直接存回.data对应的名字中

fprintf(final\_code, "la $s7 %s\n", var\_name);

fprintf(final\_code, "sw %s 0($s7)\n", reg\_name);

}

else {

fprintf(final\_code, "sw %s %d($fp)\n", reg\_name, -(address + 1) \* 4);

}

}

//根据符号表把全局量，根据MIDCODE把字符串和中间变量写入.data

void data\_area\_generate() {

int index, first\_func\_index;

first\_func\_index = new\_Table.index\_of\_func[0];

for (index = 0; index < first\_func\_index; index++) {

switch (new\_Table.List[index].kind) {

case 0: { //const

fprintf(final\_code, "%s: .word %d\n", new\_Table.List[index].name, new\_Table.List[index].value);

break;

}

case 1: { //var

if (new\_Table.List[index].size != -1) {

fprintf(final\_code, "%s: .space %d\n", new\_Table.List[index].name, new\_Table.List[index].size \* 4);

}

else {

fprintf(final\_code, "%s: .word 0\n", new\_Table.List[index].name);

}

}

}

}

for (index = 0; index < CODE\_LIST.code\_num; index++) {

if ((CODE\_LIST.MID\_CODE[index].op == 28) && (strcmp(CODE\_LIST.MID\_CODE[index].src1, CODE\_EMPTY) != 0))

fprintf(final\_code, "S%d: .asciiz %s\n", index, CODE\_LIST.MID\_CODE[index].src1);

}

}

//生成.text段

void generate\_text() {

int index, i;

int address, size, para\_num;

char label1[50];

char label2[50];

for (index = 0; index < CODE\_LIST.code\_num; index++) {

if (CODE\_LIST.MID\_CODE[index].is\_effective == 0)

continue;

switch (CODE\_LIST.MID\_CODE[index].op) {

case 7:

case 8:

case 9:

{

//init\_reg\_var\_list();

strcpy(now\_func\_name, CODE\_LIST.MID\_CODE[index].src1);

//形成新的运行栈

size = get\_func\_total\_size(CODE\_LIST.MID\_CODE[index].src1);

size = size + 2;

//size一共对应的地方是所有局部变量常量+返回地址、返回上一个AR的fp地址

fprintf(final\_code, "%s:\n", CODE\_LIST.MID\_CODE[index].src1);

para\_num = get\_func\_para\_num(now\_func\_name);

for (i = 0; i < para\_num; i++) {

fprintf(final\_code, "lw $s7 0($sp)\n");

fprintf(final\_code, "subi $sp $sp 8\n");

fprintf(final\_code, "sw $s7 0($sp)\n");

fprintf(final\_code, "addi $sp $sp 12\n");

}

fprintf(final\_code, "move $s7 $fp\n");

fprintf(final\_code, "move $fp $sp\n");

fprintf(final\_code, "subi $fp $fp 4\n");

fprintf(final\_code, "sw $s7 0($fp)\n");

fprintf(final\_code, "sw $ra -4($fp)\n");

fprintf(final\_code, "subi $sp $sp %d\n", size \* 4);

break;

}

case 11: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

break;

}

case 12: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "li $s6 4\n");

fprintf(final\_code, "mul $s6 $s6 $t1\n");

address = new\_index\_in\_tab(CODE\_LIST.MID\_CODE[index].result);

if (new\_global\_flag) {

fprintf(final\_code, "la $s7 %s\n", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "add $s7 $s7 $s6\n");

fprintf(final\_code, "sw $t0 0($s7)\n");

}

else {

fprintf(final\_code, "subi $s7 $fp %d\n", (address + 1) \* 4);

fprintf(final\_code, "add $s7 $s7 $s6\n");

fprintf(final\_code, "sw $t0 0($s7)\n");

}

break;

}

/\*case 13:{

save\_scene();

break;

}\*/

case 14: {//分为要赋值和不要

/\*1.保存现场

2.jal

3.可能需要把v0复制给result

\*/

fprintf(final\_code, "jal %s\n", CODE\_LIST.MID\_CODE[index].src1);

if (strcmp(CODE\_EMPTY, CODE\_LIST.MID\_CODE[index].result) != 0) {

fprintf(final\_code, "move $t0 $v0\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

}

break;

}

case 15: {

fprintf(final\_code, "jal %s\n", CODE\_LIST.MID\_CODE[index].src1);

break;

}

case 16: {

generate\_label(label1);

generate\_label(label2);

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "ble $t0 $t1 %s\n", label1);

store\_data("$zero", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "j %s\n", label2);

fprintf(final\_code, "%s:\n", label1);

fprintf(final\_code, "li $t2 1\n");

store\_data("$t2", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "%s:\n", label2);

break;

}

case 17: {

generate\_label(label1);

generate\_label(label2);

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "blt $t0 $t1 %s\n", label1);

store\_data("$zero", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "j %s\n", label2);

fprintf(final\_code, "%s:\n", label1);

fprintf(final\_code, "li $t2 1\n");

store\_data("$t2", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "%s:\n", label2);

break;

}

case 18: {

generate\_label(label1);

generate\_label(label2);

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "bge $t0 $t1 %s\n", label1);

store\_data("$zero", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "j %s\n", label2);

fprintf(final\_code, "%s:\n", label1);

fprintf(final\_code, "li $t2 1\n");

store\_data("$t2", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "%s:\n", label2);

break;

}

case 19: {

generate\_label(label1);

generate\_label(label2);

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "bgt $t0 $t1 %s\n", label1);

store\_data("$zero", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "j %s\n", label2);

fprintf(final\_code, "%s:\n", label1);

fprintf(final\_code, "li $t2 1\n");

store\_data("$t2", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "%s:\n", label2);

break;

}

case 20: {

generate\_label(label1);

generate\_label(label2);

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "beq $t0 $t1 %s\n", label1);

store\_data("$zero", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "j %s\n", label2);

fprintf(final\_code, "%s:\n", label1);

fprintf(final\_code, "li $t2 1\n");

store\_data("$t2", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "%s:\n", label2);

break;

}

case 21: {

generate\_label(label1);

generate\_label(label2);

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "bne $t0 $t1 %s\n", label1);

store\_data("$zero", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "j %s\n", label2);

fprintf(final\_code, "%s:\n", label1);

fprintf(final\_code, "li $t2 1\n");

store\_data("$t2", CODE\_LIST.MID\_CODE[index].result);

fprintf(final\_code, "%s:\n", label2);

break;

}

case 22: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

fprintf(final\_code, "beq $t0 $zero %s\n", CODE\_LIST.MID\_CODE[index].src2);

break;

}

case 23: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

fprintf(final\_code, "bne $t0 $zero %s\n", CODE\_LIST.MID\_CODE[index].src2);

break;

}

case 24: {

fprintf(final\_code, "%s:\n", CODE\_LIST.MID\_CODE[index].src1);

break;

}

//?

case 25: {

fprintf(final\_code, "lw $ra -4($fp)\n");

fprintf(final\_code, "addi $sp $fp 4\n");

fprintf(final\_code, "lw $fp 0($fp)\n");

fprintf(final\_code, "jr $ra\n");

break;

}

case 26: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

fprintf(final\_code, "move $v0 $t0\n");

fprintf(final\_code, "lw $ra -4($fp)\n");

fprintf(final\_code, "addi $sp $fp 4\n");

fprintf(final\_code, "lw $fp 0($fp)\n");

fprintf(final\_code, "jr $ra\n");

break;

}

case 27: {

if (strcmp(CODE\_INT, CODE\_LIST.MID\_CODE[index].src1) == 0) {

fprintf(final\_code, "li $v0 5\n");

fprintf(final\_code, "syscall\n");

fprintf(final\_code, "move $t0 $v0\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

}

else {

fprintf(final\_code, "li $v0 12\n");

fprintf(final\_code, "syscall\n");

fprintf(final\_code, "move $t0 $v0\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

}

break;

}

case 28: {

if (strcmp(CODE\_LIST.MID\_CODE[index].src1, CODE\_EMPTY) != 0) {

fprintf(final\_code, "la $a0 S%d\n", index);

fprintf(final\_code, "li $v0 4\n");

fprintf(final\_code, "syscall\n");

}

if (strcmp(CODE\_LIST.MID\_CODE[index].src2, CODE\_EMPTY) != 0) {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src2);

if (strcmp(CODE\_LIST.MID\_CODE[index].result, CODE\_CHAR) == 0) {

fprintf(final\_code, "move $a0 $t0\n");

fprintf(final\_code, "li $v0 11\n");

fprintf(final\_code, "syscall\n");

}

else {

fprintf(final\_code, "move $a0 $t0\n");

fprintf(final\_code, "li $v0 1\n");

fprintf(final\_code, "syscall\n");

}

}

break;

}

case 29: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "add $t0 $t0 $t1\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

break;

}

case 30: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "sub $t0 $t0 $t1\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

break;

}

case 31: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "mul $t0 $t0 $t1\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

break;

}

case 32: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

load\_data("$t1", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "div $t0 $t1\n");

fprintf(final\_code, "mflo $t0\n");

store\_data("$t0", CODE\_LIST.MID\_CODE[index].result);

break;

}

case 33: {

fprintf(final\_code, "j %s\n", CODE\_LIST.MID\_CODE[index].src1);

break;

}

case 34: {

fprintf(final\_code, "li $v0 10\n");

fprintf(final\_code, "syscall\n");

break;

}

case 35: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src1);

fprintf(final\_code, "subi $sp $sp 4\n");

fprintf(final\_code, "sw $t0 0($sp)\n");

break;

}

case 36: {

load\_data("$t0", CODE\_LIST.MID\_CODE[index].src2);

fprintf(final\_code, "li $s6 4\n");

fprintf(final\_code, "mul $s6 $s6 $t0\n");

address = new\_index\_in\_tab(CODE\_LIST.MID\_CODE[index].src1);

if (new\_global\_flag) {

fprintf(final\_code, "la $s7 %s\n", CODE\_LIST.MID\_CODE[index].src1);

fprintf(final\_code, "add $s7 $s7 $s6\n");

fprintf(final\_code, "lw $t1 0($s7)\n");

store\_data("$t1", CODE\_LIST.MID\_CODE[index].result);

}

else {

fprintf(final\_code, "subi $s7 $fp %d\n", (1 + address) \* 4);

fprintf(final\_code, "add $s7 $s7 $s6\n");

fprintf(final\_code, "lw $t1 0($s7)\n");

store\_data("$t1", CODE\_LIST.MID\_CODE[index].result);

}

break;

}

}

}

}

int new\_index\_in\_tab(char name[]) {

int index, i;

new\_const\_flag = 0;

new\_var\_flag = 0;

new\_para\_flag = 0;

new\_global\_flag = 0;

if (strcmp(now\_func\_name, CODE\_EMPTY) == 0) { //全局量，在全局里面找

for (index = 0; new\_Table.List[index].kind != 2; index++) {

if (strcmp(new\_Table.List[index].name, name) == 0)

break;

}

if (new\_Table.List[index].kind == 2)

return -1;

else {

new\_global\_flag = 1;

if (new\_Table.List[index].kind == 0) {

new\_const\_flag = 1;

return new\_Table.List[index].value;

}

else if (new\_Table.List[index].kind == 1) {

new\_var\_flag = 1;

return new\_Table.List[index].in\_address;

}

return -1;

}

}

else {

for (index = 0; index < new\_Table.num\_of\_func; index++) {

i = new\_Table.index\_of\_func[index];

if (strcmp(new\_Table.List[i].name, now\_func\_name) == 0)

break;

}

if (index < new\_Table.num\_of\_func) {

for (index = i + 1; (index <= new\_Table.list\_index) && (new\_Table.List[index].kind != 2); index++)

if (strcmp(new\_Table.List[index].name, name) == 0)

break;

if ((index <= new\_Table.list\_index) && (new\_Table.List[index].kind != 2)) {

if (new\_Table.List[index].kind == 0) {

new\_const\_flag = 1;

return new\_Table.List[index].value;

}

else if (new\_Table.List[index].kind == 1) {

new\_var\_flag = 1;

return new\_Table.List[index].in\_address;

}

else if (new\_Table.List[index].kind == 3) {

new\_para\_flag = 1;

return new\_Table.List[index].in\_address;

}

return -1;

}

else {//去全局量找

for (index = 0; new\_Table.List[index].kind != 2; index++) {

if (strcmp(new\_Table.List[index].name, name) == 0)

break;

}

if (new\_Table.List[index].kind == 2)

return -1;

else {

new\_global\_flag = 1;

if (new\_Table.List[index].kind == 0) {

new\_const\_flag = 1;

return new\_Table.List[index].value;

}

else if (new\_Table.List[index].kind == 1) {

new\_var\_flag = 1;

return new\_Table.List[index].in\_address;

}

return -1;

}

}

}

else {//error

return -1;

}

}

return -1;

}

int get\_func\_para\_num(char name[]) {

int index;

for (index = 0; index < new\_Table.num\_of\_func; index++) {

if (strcmp(name, new\_Table.List[new\_Table.index\_of\_func[index]].name) == 0)

break;

}

if (index < new\_Table.num\_of\_func) {

return(new\_Table.List[new\_Table.index\_of\_func[index]].size);

}

else {

return -1;

}

}

int get\_func\_total\_size(char name[]) {

int i, index;

for (i = 0; i < new\_Table.num\_of\_func; i++) {

index = new\_Table.index\_of\_func[i];

if (strcmp(name, new\_Table.List[index].name) == 0)

break;

}

if (i == new\_Table.num\_of\_func) {

error(FUNC\_NO\_DEF\_ERROR, line);

return -1;

}

else { //找到了同名函数

if (i < new\_Table.num\_of\_func - 1) {

return(new\_Table.List[new\_Table.index\_of\_func[i + 1] - 1].in\_address);

}

else {

return(new\_Table.List[new\_Table.list\_index].in\_address);

}

}

}

//生成最终代码的总处理

void generate\_all\_final\_code() {

get\_new\_Table();

strcmp(now\_func\_name, CODE\_EMPTY);

final\_code = fopen("final.asm", "w");

fprintf(final\_code, ".data\n");

data\_area\_generate();

fprintf(final\_code, ".text\n");

fprintf(final\_code, "move $fp $sp\n");

fprintf(final\_code, "j main\n");

generate\_text();

fclose(final\_code);

}

int main()

{

char data\_path[100];

scanf("%s", data\_path);

if ((compile\_file = fopen(data\_path, "r")) == NULL) {

printf("Can not find the file.\n");

return 0;

}

output\_file = fopen("result.txt", "w");

initial\_table();

init\_code\_list();

init\_func\_list();

CHAR = get\_ch();

symbol = insymbol(); //There is only one symbol here, but after reading it, the pointer already points to the next position

program();

fprintf(output\_file, "finish compiling\n");

print\_mid\_code();

generate\_all\_final\_code();

fclose(compile\_file);

fclose(output\_file);

return 0;

}