《编译技术》
课程设计文档

学号：_____13061026 _____

姓名：_____杨东东_____

2016 年　01　月　08　日

目录

一．需求说明

1．文法说明（已更新）

文法	说明
1. <程序> :: = <分程序>.	分程序后必须有点号
2. <分程序> ::= [<常量说明部分>][<变量说明部分>]{[<过程说明部分>]|[<函数说明部分>]}<复合语句>	常量、变量顺序不能颠倒； 过程说明和函数说明顺序不定； 必须要有复合语句，其他部分可有可无。
3. <常量说明部分> ::= const<常量定义>{,<常量定义>};	常量说明以逗号隔开，分号结束
4. <常量定义> ::= <标识符>＝<常量>	这里的常量如果是数字，不能过大，否则报错； 常量若是字母，只能是一个，同时标识符区分大小写； 这里的等号是“=”，不是“：=”！
5. <常量> ::= [+| -] <无符号整数>|<字符>	+-号可有可无，负号情况需处理成负数； 字符没有正负号，且只有一个字母
6. <字符> ::= ’<字母>’ | ’<数字>’	常量在定义时只能有一个字母或者数字，由‘’包含起来。
7. <字符串> ::= ″{十进制编码为 32,33,35-126 的 ASCII 字符}″	包括空格，感叹号，#等，但是不包括括双引号，注意可以包括单引号，斜杠/，反斜杠\
8. <无符号整数>::= <数字>{<数字>}	整数可以以 0 开头的数字
9. <标识符> ::= <字母>{<字母>|<数字>}	标志符不能以数字开头，必须以字母开头，其后才能数字\字母
10. <变量说明部分> ::= var <变量说明> ; {<变量	每个变量说明以分号隔开，分号结束

说明>;}	
11. <变量说明> ::= <标识符>{, <标识符>} : <类型>	如果是同一类型，标志符可以是多个，标识符内部以逗号隔开； 最后必须要有变量的类型
12. <类型> ::= <基本类型>|array'['<无符号整数>']' of <基本类型>	array 的长度不能是变量，必须要是无符号的整数，格式符合要求； 空格的个数不限；
13. <基本类型> ::= integer | char	不是 int，而是 integer 没有别的类型
14. <过程说明部分> ::= <过程首部><分程序>{; <过程首部><分程序>};	过程说明结束时必须有分号“；”；
15. <函数说明部分> ::= <函数首部><分程序>{; <函数首部><分程序>};	函数说明结束时必须有分号“；”；
16. <过程首部> ::= procedure<标识符>[<形式参数表>];	无返回值的可以看成函数； 形式参数表可有可无； 没有左括号和右括号； 过程的标志符和普通的标志符要区分处理
17. <函数首部> ::= function <标识符>[<形式参数表>]: <基本类型>; //（说明其返回值为基本类型）	形式参数表可有可无； 没有左括号和右括号； 函数的标志符和普通的标志符要区分处理； 必须要有返回值类型；
18. <形式参数表> ::= '('<形式参数段>{; <形式参数段>}')'	形式参数段有多个，不单只有一个； 形式参数段内部以逗号隔开，外部以分号隔开，最外面必须有左右括号； 同时，形式参数表需要判断过程或者函数的个数是否一致
19. <形式参数段>::= [var]<标识符>{, <标识符>}: <基本类型>	
20. <语句> ::= <赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<for 循环语句>|<空>	<空语句>的情况使得<语句>的各个其他部分要以条件判断进入，当都不符合时为空语句； 各个语句之间可能会有相互调用； 一条语句的结束还有下一条语句，则之间必须要有分号间隔开； 函数过程主函数的显示出来最后一条语句

	可以有分号，也可以没有分号，如果有分号，则最后一条语句实际是空语句，否则该条语句即为最后一条一句
21. <赋值语句> ::= <标识符> :=<表达式>|<函数标识符> :=<表达式>	<标识符>'['<表达式>']':=<表达式>	x:='a';在赋值语句中这么写不符合文法； 赋值语句以“：=”来赋值，不同于常数定义； 函数标识符的赋值语句表示该函数的返回值，过程则没有这一条语句； 允许数组的赋值，数组的索引可以是表达式(其中也就包括了单个的数字)
22. <函数标识符 ::= <标识符>	表示函数标志符包含于标志符
23.<表达式> ::= [+|-]<项>{<加法运算符><项>}	表达式的运算，+-均属于加法运算； 项与项之间由加法运算符隔开； 表达式可以以+-号开头
24. <项> ::= <因子>{<乘法运算符><因子>}	项为乘除法运算，意思在这里也就是*/的优先级高于+-；
25. <因子>::= <标识符>|<标识符>'['<表达式>']'|<无符号整数>| '('<表达式>')' | <函数调用语句>	可以是不带左右括号的过程/函数调用，这里为函数的标志符； 也可以是带左右括号的过程函数调用； 也可以是普通的标志符； 也可以是数字
26. <函数调用语句> ::= <标识符>[<实在参数表>]	注意函数被调用时，实在参数表可能是有的，也可能是没有的，得根据函数的定义来判断
27. <实在参数表> ::='('<实在参数>{,<实在参数>}')'	调用的个数得和定义该函数的个数匹配， 以分号间隔开，实在参数可以是数组元素； 如果有实在参数表，必须要有左右括号
28. <实在参数> ::= <表达式>	实在参数可以是复杂的表达式，而不只是表达式得出的最终结果，表达式就可以很复杂了
29. <加法运算符> ::= +|-	
30. <乘法运算符 ::= *|/	
31. <条件> ::= <表达式><关系运算符><表达式>	此处为条件判断，因此可能会出现 a+b+c=f(d,e,f)其中 a、b、c 为标识符这样的神奇语句，此时，等号左边的表达式需要用一个临时变量代替 不允许有赋值操作； 这里的等号“=”与常数定义的“=”一

	样，因此等号得根据上下文赋予不同的含义
32. <关系运算符> ::= <|<=|>|>= |=|<>	不等号为<>,而不是!= 只有这几种关系，没有与或非!!!
33. <条件语句> ::= if<条件>then<语句> | if<条件>then<语句>else<语句>	在条件语句中的判断条件中 = 为 EQL 为“==”
34. <当循环语句>::= do<语句> while<条件>	没有 while 开头的当循环； 注意语句的结束没有分号；

35.
<for 循环语句> ::= for <标识符> := <表达式> （downto | to） <表达式> do <语句> //步长为 1
	是 downto/to，不是 down/to；
	步长仅为 1；
	downto/to 后面表达式的判断。因为比较纠结举个例子。
	i:=1;
	for i:=0 to 10 do …
	//进入 for，最终结果为循环 11 次，i 的最后跳出循环后的结果为 10，不是 11
	
	i:=100;
	for i:=0 to 10 do …
	//不进入循环，优先判断 100 与 10 的关系

36. <过程调用语句> ::= <标识符>[<实在参数表>]	过程调用的实在参数表的内容可能是数字，标志符，数组元素或者待运算的表达式等等这几种情况！
37. <复合语句> ::= begin<语句>{; <语句>}end	
38. <读语句> ::= read’(’<标识符>{,<标识符>}’)’	read 没有 read(a[i]);这种形式的！而 write 则有 write(a[i]);
39. <写语句> ::= write’(’<字符串>,<表达式>’)’|write’(’<字符串>’)’|write’(’<表达式>’)’	write("%d", a) write(a)----------此时若是 char，它不可能是一个字符串，只会是一个字符，按一个字符来处理 write("Hello, compilier!\n")
40. <字母> ::= a|b|c|d…x|y|z |A|B…|Z	没有别的
41.	函数内是可以重新新的变量的，但是属于

<数字> ::= 0|1|2|3…8|9	函数主部分 begin...end 则不行，同时，主函数内部不能申明变量

2. 目标代码说明

系统调用	syscall	
取立即数	li $s1, 100	$s1 = 100
加	add s1,s2,$s3	s3=s1+$s2
立即数加	addi s1,s2,100	s3=s1+100
立即数减	subi s1,s2,100	s3=s1-100
减	sub s1,s2,$s3	s3=s1-$s2
乘	mult s2,s3	Hi，Lo=s2?s3
除	div s2,s3	Lo=s2/s3Hi=s2 mods 3
取字	lw s1,100(s2)	s1=memory[s2+100]
存字	sw s1,100(s2)	memory[s2+100]=s1
beq	beq s1,s2,100	if（s1==s2） goto PC+4+400
bne	beq s1,s2,100	if（$s1! =$s2） goto PC+4+400
slt	slt s1,s2, $s3	if（s2<s3） $s1=1; else $s1=0;
j	j 2000	goto 8000
jr	ja $ra	goto $ra
jal	jal 2500	$ra=PC+4;goto 10000;

3. 优化方案*

【说明需要完成的优化方案及其要求】

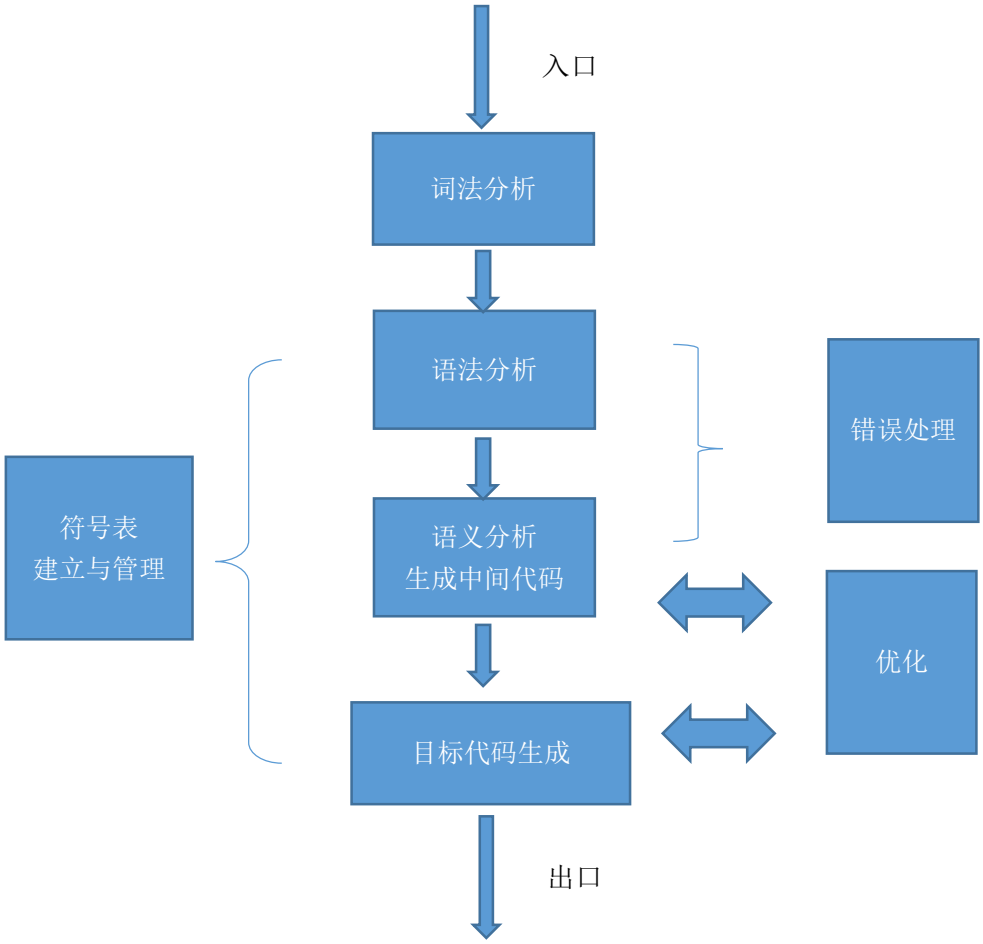
基本块内部的公共子表达式删除（DAG 图）；
全局寄存器分配（引用计数或着色算法）；数据流分析（通过活跃变量分析，或利用定义-使用链建网等方法建立冲突图）；
其它优化自选；
代码生成时合理利用临时寄存器（临时寄存器池），并能生成较高质量的目标代码；

二. 详细设计

【应包括但不限于以下内容】

1. 程序结构

【从总体上描述程序的结构，文字或图示均可】



2．类/方法/函数功能（已更新）

【描述各类/方法或函数的功能，以及关键算法】

函数名	作用和功能
词法分析	
void getch();	获取一个字符
int getsym();	获取一个词语
语法分析：	
void wprogram();	程序主程序
void program();	程序递归子程序

void varstate();	变量声明递归子程序
void vardef();	变量定义递归子程序
void constdef(int tclass);	常量定义递归子程序
void conststate();	常量声明递归子程序
void sentencelist();	语句列递归子程序
void complexsentence();	复杂语句递归子程序
void sentence();	语句递归子程序
void condition();	条件递归子程序
void loopsentence();	循环语句递归子程序
void valueofpara();	值参数表递归子程序
void readsentence();	读语句递归子程序
void writesentence();	写语句递归子程序
void parametertable();	参数表递归子程序
void returnsentence();	返回语句递归子程序
void expression();	表达式递归子程序
void item();	项递归子程序
void factor();	因子递归子程序
中间代码：	
void insmidcode(char* op, char* t1, char* t2, char* t3);	插入中间代码
void insmidcode(char* op, int t1, int t2, char* t3);	

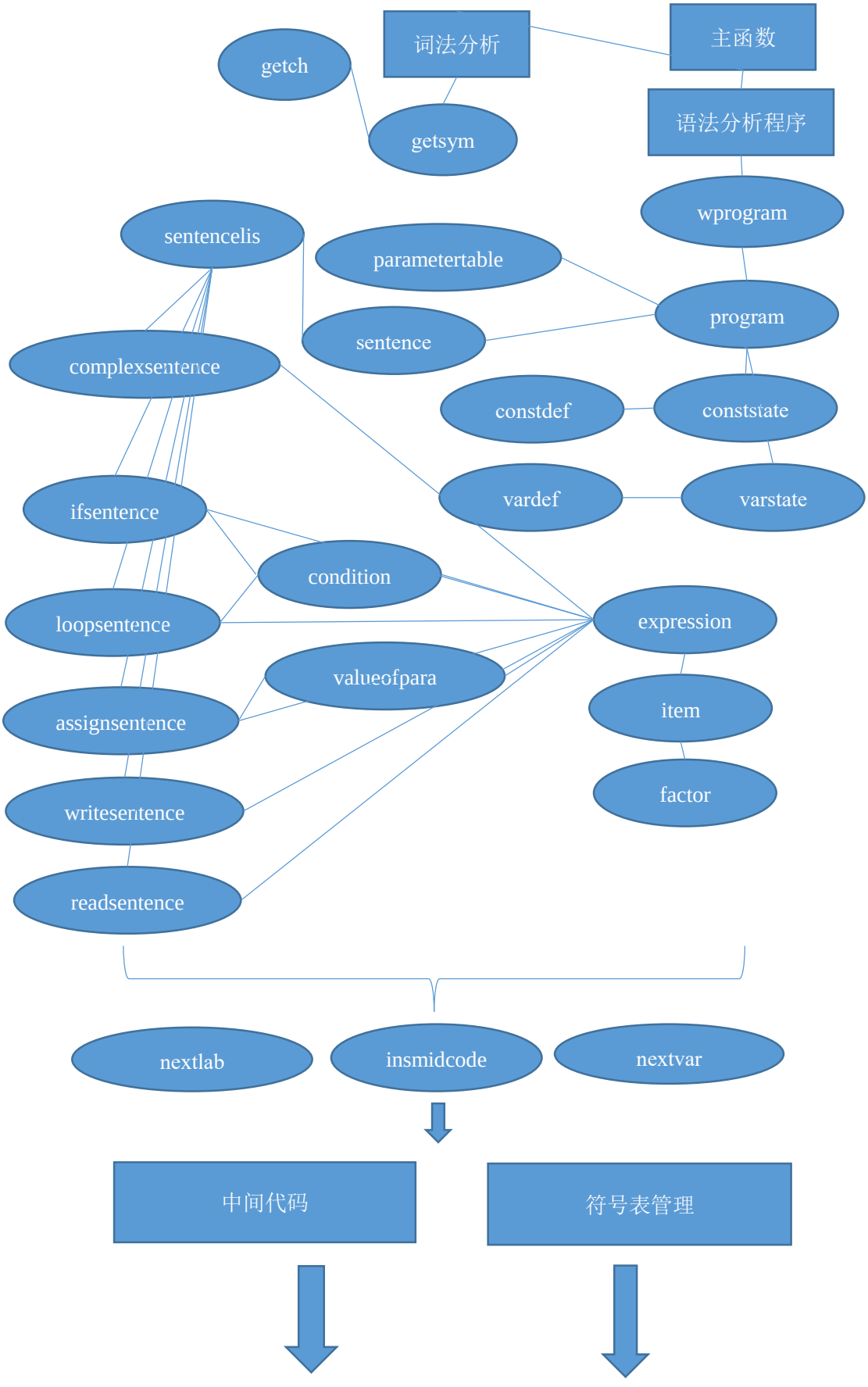
void insmidcode(char* op, char t1, int t2, char* t3);	
char* nextlab();	返回下一个 label 名
char* nextvar();	返回下一个临时变量名
# 汇编相关：	
void midcode2asm();	中间代码转汇编
int findvartable(char *name);	在变量表中查找
void midcode2asm();	中间代码转汇编
void insertaddress(int kind, int addr = -1, int nmi = -1);	向变量表中插入一个变量和地址
void pushstack(char* item = "0", int lenth = 1);	压栈 lenth*4 个字节
void inittempvar();	初始化局部变量
void beginasm();	函数过程开始
void funcasm();	开始函数定义
int varaddr(char *name);	求变量的相对 fp 的地址
void jmpasm();	跳转
void printint();	写一个整数语句
void callasm();	调用函数
void setlabasm();	放置标签
void addasm();	四则运算
void subasm();	
void mulasm();	

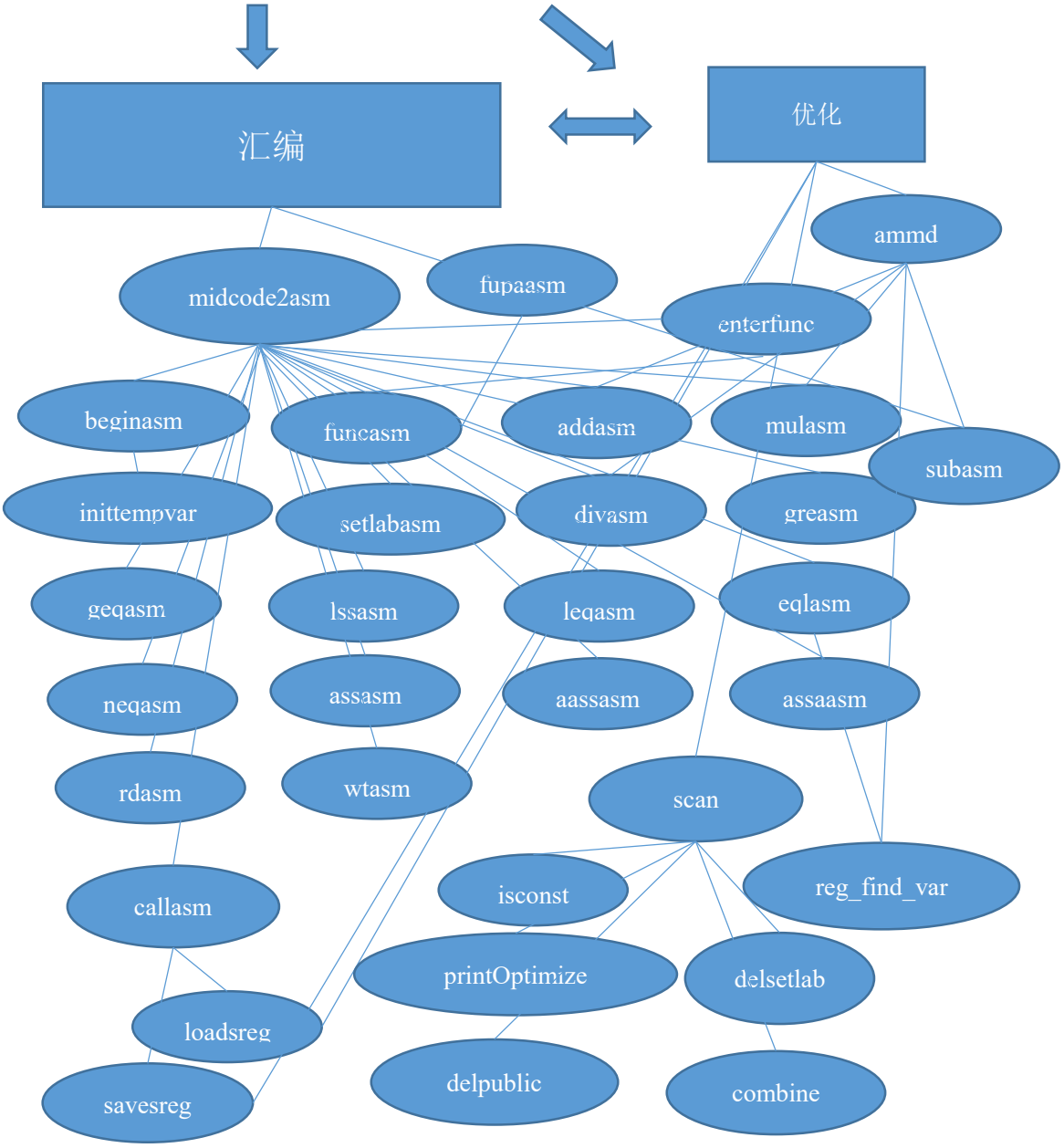
void divasm();	
void greasm();	比较运算
void geqasm();	
void lssasm();	
void leqasm();	
void eqlasm();	
void neqasm();	
void assasm();	变量赋值语句
void aassasm();	包含数组元素的操作
void assaasm();	
void rdasm();	读语句
void wtasm();	写语句
void fupaasm();	填入参数语句
void retasm();	函数返回
void paraasm();	参数声明处理
void jneasm();	否定则跳转
void constvarasm();	变量常量定义语句
# 优化	
int isconst(char name[]);	是否为常数
void savesreg();	保存寄存器的值
void ammd(char* oper);	加减乘除优化模板

int reg_find_var(char* name);	查找分配好的寄存器
void scan();	扫描四元式同时分割块
void delsetlab();	删除无效的标签
void delpublic();	删除公共表达式，创建 DAG 图，打印新的四元式算法
void combine();	
void loadsreg();	加载寄存器的值
void cnt(char * name);	引用计数法
void cntopt();	
void printOptimize();	打印优化后的四元式

3．调用依赖关系（已更新）

【说明各类之间的关系，方法/函数之间的调用关系】


词法分析
主函数
getch
getsym
语法分析程序
wprogram
sentencelis
parametertable
program
sentence
complexsentence
constdef
conststate
vardef
varstate
ifsentence
condition
loopsentence
expression
valueofpara
item
assignsentence
factor
writesentence
readsentence
nextlab
insmidcode
nextvar
中间代码
符号表管理


汇编
优化
ammd
midcode2asm
fupaasm
enterfunc
beginasm
funcasm
addasm
mulasm
subasm
inittempvar
setlabasm
divasm
greasm
geqasm
lssasm
leqasm
eqlasm
neqasm
assasm
aassasm
assaasm
rdasm
wtasm
scan
callasm
isconst
reg_find_var
printOptimize
delsetlab
loadsreg
delpublic
combine
savesreg

4．符号表管理方案（已更新）

【说明符号表的数据结构、管理算法】

//需添加一个标签来区别　函数标识符与普通标识符

```
//      kind 的域
#define CONST       0   //常亮
#define VARIABLE    1   //变量
#define FUNCTION    2   //函数
#define PROCEDURE         22       //过程
#define PARA        3   //参数
#define CINT              4        //常量 int 类型
#define CCHAR             5        //常量 char 类型
#define AINT              6        //数组 int 类型
#define ACHAR             7        //数组 char 类型
#define VINT              8        //变量 int 类型
#define VCHAR             9        //变量 char 类型

typedef struct {
        char name[MAXIDENLEN];  //identifier name
        int kind;
        int value;              //2 对函数来说返回 char，1 对于函数来说返回为
Int，0 返回值为 void（对应 procedure）
        int address;            //记录相对位置
        int paranum;            //参数个数
        bool is_return;
}symb;

typedef struct {
        symb element[MAXSYMTABLENUM];
        int index;
        //总的索引
        int ftotal;
        //分程序总数
        int findextable[MAXSYMTABLENUM]; //从 1 开始，分程序索引数组，function
index table,用于将函数程序之间区别开来,记录的是 function 在总的 index 中的序号，也
就是使用“栈”的思想
}symtable;
extern symtable maintable;
```

5．存储分配方案（已更新）

【说明运行时的存储组织及管理方案，运行栈结构】

运行栈	（说明）
（高地址）	
主函数的 fp	主函数区
全局变量	
参数区	一层函数的运行栈，其他函数的栈同理往下压
上一层函数的 fp 值	
调用该函数 fp 值	
返回值 ra	
当前函数的 fp	
局部变量区	
…	下一层函数的运行栈
…	
…	
.	
.	
.	
$s0-$s7 的变量区	
（低地址）	

6. 解释执行程序*

【说明解释执行程序的数据结构，关键算法，及解释执行过程】

本编译器不采用解释执行方式。

7. 四元式设计*（已更新）

【对采用的四元式进行详细说明】

```
= , 2 , , temp                    temp = 2;
[]= , a , i , t                   a[i] = t;    数组赋值语句
数组变量声明
inta, 0, num, name
chara,0, num, name
int , , , a                      int a;
//这个由原来的 int, , ,a 更新为 int , , ,a, level(所在层数)
//char 同理,const 也一样
const,int,5 , a                   const a = 5;

char, , 30, a                     var a:array[30] of char;
fupa, , , a                       a is a function parameter
```

```
call, f , , a                    a = f()
call, f , ,                      f()
<=, a , b ,                      a <= b
<>,
jne , , , lable                  if not satisfy(==false) then jump
jmp , , , label                  jump to label
lab:, , , labx                   set label
geta, a , n , b                  b = a[n]
ret , , , (e)                    f:=(expression);
wt , a , b , symbol              write("a", b) symbol 存的是 b 的类型
rd , , , a                       read(a)
func,int, , f                    start of function f:integer
para,int, , a                    f(a:integer; ...)
end , , , ,                      end 结束当前函数
lab:, , , label$NO               设定一个 label
```

8. 目标代码生成方案*(已更新）

【说明代码生成有关的数据结构、关键算法】
符号表在目标代码生成阶段，仍旧有用到，但是使用的情况和语法语义部分不太一样，因此重新建立符号表，针对各个不同的功能而定。

```
typedef struct {
        char name[100];
        int level;
}constvarlevel;//变量所在层数
extern constvarlevel cvlevel[1000];
extern int cvl;

typedef struct {
        char name[100];
        int kind;
        int paranum;
        bool isParaVar[20];
        int level;
}funcclass;
extern vector<funcclass> fc;
//函数表

typedef struct {
        char name[100];
        int address;
        int kind;
        int cnt;
```

```
        //int level;
}tempvaraddress;//变量表
extern tempvaraddress addrtable[1000];
//临时变量在栈中的地址表
typedef struct {
        int symbnum;
        int cnt;
}cntstruct;
extern cntstruct cnttable[200];
extern int ap;
extern int mi;
extern int sp;
```

9. 优化方案*（已更新）

【说明代码优化有关的数据结构、关键算法】

0. 划分基本快：

按照数据流的分析划分基本块。

1. 窥孔优化：

通过窥孔优化合并常数，不断读取四元式，一旦发现有常数运算的立即变为赋值操作。

2. 消除公共子表达式：

1. 通过扫描基本块，由上向下找出有相同的 op（对应 DAG 图中的父节点），var1（对应 DAG 图中的左子节点），var2（对应 DAG 图中的右子节点）且 var3 为临时变量的四元式对（A, B）；
2. 从 B 向下继续扫描四元式，寻找所有与 B 具有相同 var3 的四元式 C1，C2，…；
3. 将 Ci 的 var3 置为 A.var3；
4. 删除 B，将 B 之后的四元式前移。

3. 删除冗余代码标签：

很多情况下，程序出现 if 语句中即不存在 else。

此外，还有很多情况会导致在转化成四元式的时候两个甚至多个 label 连在一起，这样就导致了代码冗余。

算法：

扫描整个四元式，一旦遇到 lab:,删除其后的所有相邻 lab 代码，并更改所有 var3 为被删除的 lab 的四元式中跳转指令的 var3 为最先发现的 lab，反复执行直到读完整个四元式。

4. 通过引用计数寄存器分配方案：

本程序使用 s0 到 s7 存储局部变量。实际运行时的情况举例：

其中 symbnum 指的是变量在变量表中的编号，cnt 为计数的积分，越高使用频率就越大。

通过引用计数算法，扫描整个函数，查找每个变量使用的频度值，从大到小排序，分配寄存器。

10. 出错处理（已更新）

【说明出错处理方案、错误信息及含义】

接口：

```
void error(int _errsig, int signal)
```

出错类型和相关信息：

_errsig	编码	说明
#define NOSUCHFILE	404	//文件不存在
#define FILEINCOMPLETE	0	//文件不完整
#define DOUBLEQUOTESLACK	1	//双引号缺失
#define UNACCEPTATLECHAR	2	//不可以接受的 char
#define SINGLEQUOTESLACK	3	//单引号缺失
#define OUTOFTABLE	4	//符号表
#define SYMBOLCONFLICT	5	//字符串冲突
#define CSTDEFINEFAIL	6	//常量声明失败
#define VARNOTINIT	7	//变量未初始化
#define UNKNOWRIGHTSYM	8	//等号右边字符非法
#define SEMICOLONLACK	9	//丢失“;”
#define KEYWORDERROR	10	//关键字错误
#define IDENTIFIERLACK	11	//丢失标志符
#define RIGHTBRACKLACK	12	//丢失“]”
#define FUNCTIONNOTFOUND	13	//调用函数未定义
#define FORMALPARANUMUNMATCH	14	//形参个数不匹配
#define VARNOTDEFINE	15	//未定义变量
#define LEFTPARENTLACK	16	//丢失“(”
#define RIGHTPARENTLACK	17	//丢失“)”
#define IMMLACK	18	//丢失立即数
#define LBRACKLACK	19	//丢失“[”

#define FUNCTIONRETURNNULL	20	//函数无返回值却被用于参数
#define EXPRESSIONERROR	21	//表达式缺失或错误
#define UNACCEPTABLESENTENCE	22	//句子不合法
#define ASSIGNCONST	23	//给常数赋值
#define OFLACK	24	//缺失 of
#define NONERETURN	25	//缺少返回值
#define PLUSMINULACK	26	//缺少‘+’或‘-’
#define MAINLACK	26	//缺少 main 函数
#define MOREWORD	28	//main 函数后面有多余字符
#define CONSTNOTINIT	29	//常量没有初始化
#define PERIODLACK	30	//缺失主程序后的点号"."
#define COLONLACK	31	//缺失：
#define ARRAYLENGTHLACK	32	//数组缺失长度
#define DOLACK	33	//缺失 do
#define BEGINLACK	34	//缺失 begin
#define ENDLACK	35	//缺失 end
#define DIVIDEZERO	36	//除数为 0
#define DOWNTOLACK	37	//downto/to 缺失
#define COMMALACK	38	//逗号缺失
#define ARRAYLACK	39	//数组元素不存在
#define UNKNOWNOPRRATOR	40	//未定义关系运算符
#define RETURNTYPELACK	41	//函数缺少返回值类型
#define PARANUMZERO	42	//函数（） 内参数个位为 0
#define UNKNOWN	50	//未知错误

三．操作说明

1．运行环境

【说明搭建运行环境的步骤】

visual studio professional 2015（注意是 2015 版）!

若为 vs 2013 或者其他，则环境需要调整，步骤如下：

（1） 点击项目右键，属性

（2） 配置属性->常规->平台工具集: 修改为 visual studio 2013 v

（3） 配置属性->c/c++->预处理器: 在预处理器定义末尾添加如下

; _CRT_SECURE_NO_DEPRECATE

2．操作步骤

【详细说明操作步骤】

（1） 点击: 生成->生成解决方案；
（2） 点击: 调试->开始执行（不调试）；
（3） 拖动: 将测试文档拖到控制台中
（4） 回车
（5） 将在工程目录 compiler/compiler 下产生的 rst.asm 文件在 Mars4.5 中运行。

四．测试报告

1．测试程序及测试结果

【给出提供的测试程序以及每个程序的测试结果，至少 5 个正确程序，5 个错误程序，无需截屏】

综述：

测试名称	测试说明
正确测试 1	实现阶乘运算
正确测试 2	实现交换运算，包括传地址与传值
正确测试 3	实现求最大公约数
正确测试 4	实现冒泡排序
正确测试 5	实现快速排序
错误测试 1	测试函数参数不匹配
错误测试 2	测试数组的定义 测试赋值语句:=
错误测试 3	测试函数 function 是否有返回值
错误测试 4	测试常量定义和变量定义颠倒位置 测试未定义变量
错误测试 5	测试重复定义变量 测试 if…then 固定句型

详述：

（1）正确测试一

程序：

```
const X=0;
var x:integer;

function test(a:integer):integer;
        begin
                if a > 1 then
                        test := a * test(a - 1)
```

```
                else
                        test := a
        end
;
begin
        write(" ",X);
        read(x);
        write(" x! = ",test(x));
end
.
```

测试结果：

```
输入：5
输出：  0 120
```

（2）正确测试二

程序：

```
const
        SIZE = 5, MAX = 100;

var
        x1,x2,x3 : integer;
        c1,c2 : char;
        list : array[10] of integer;

procedure swap(var a,b : integer);
var t : integer;
begin
        t := a;
        a := b;
        b := t
end;

procedure swap2(a,b : integer);
var t : integer;
begin
        t := a;
        a := b;
        b := t
end;

begin

x1:=10;
```

```
x2:=5;
swap2(x1,x2);
write(" x1:",x1);
write(" x2:",x2);
swap(x1,x2);
write(" x1:",x1);
write(" x2:",x2);
end
.
```

测试结果：

```
  x1:10 x2:5 x1:5 x2:10
```

（3）正确测试三

程序：

```
const
        SIZE = 5, MAX = 100;

var
        x1,x2,x3 : integer;
        c1,c2 : char;
        list : array[10] of integer;


function gcd(a,b : integer) : integer;
        function mod(a,b : integer) : integer;
                begin
                        mod := a - a/b*b
                end;

        begin
                if a*b = 0 then
                        gcd := a+b
                else
                        gcd := gcd(mod(a,b),mod(b,a))
        end;

begin
        write("gcd test begin!");
        read(x1,x2);
        write(gcd(x1,x2));
end
.
```

测试结果：

```
输入：
```

```
12
18
输出:
6
```

（4）正确测试四

```
const
        SIZE = 5;

var
        list : array[10] of integer;

procedure swap(var a,b : integer);
        var t : integer;
        begin
                t := a;
                a := b;
                b := t
        end;

procedure sort;
        var
                i,j : integer;

        begin
                for i := SIZE-1 downto 1 do
                        for j := 0 to i - 1 do
                                if list[j] > list[j+1] then
                                begin
                                        swap(list[j],list[j+1]);
                                end
        end;

procedure readList;
        var
                i,t : integer;

        begin
                for i := 0 to SIZE-1 do
                begin
                        read(t);
                        list[i] := t
                end
        end;
```

```
procedure writeList;
        var
                i : integer;
        begin
                for i := 0 to SIZE-1 do
                begin
                        write(" ",list[i])
                end
        end;

begin
        readList;
        sort;
        writeList;
end
.
```

测试结果：

输入：

```
-10
5
1
200
3
```

输出：

```
 -10 1 3 5 200
```

（5）正确测试五

程序：

```
var
        arr:    array[7] of integer;
        ft,i:   integer;

procedure QuickSort(from,til:integer);
        var ft,re:integer;
                key:integer;
                flag:integer;

begin
        ft := from;
        re := til;
        key := arr[ft];
```

```
        if ft < re then
        begin
                do
                begin
                        do
                                if ft < re then
                                        if arr[re] >= key then
                                        begin
                                                re := re - 1;
                                                flag := 1
                                        end
                                        else
                                                flag := 0
                                else
                                        flag := 0
                        while flag = 1
                        ;
                        arr[ft]:=arr[re];

                        do
                                if ft < re then
                                        if arr[ft] <= key then
                                        begin
                                                ft := ft + 1;
                                                flag := 1
                                        end
                                        else
                                                flag:=0
                                else
                                        flag:=0
                        while flag = 1
                        ;
                        arr[re] := arr[ft];
                end
                while ft < re
                ;

                arr[ft] := key;

                QuickSort(from,ft-1);
                QuickSort(ft+1,til);

        end
        ;
```

```
end;

begin
        ft:=5;
        arr[0]:=4;
        arr[1]:=8;
        arr[2]:=1;
        arr[3]:=3;
        arr[4]:=5;
        QuickSort(0, 4);
        for i:=0 to 4 do
                write(" ",arr[i]);
end.
```

测试结果：

```
输出：
 1 3 4 5 8
```

（6）错误测试一

将正确测试一的代码第 15 行：

```
        write(" x! = ",test(x));
```

改为：

```
        write(" x! = ",test(x,x));
```

报错信息：

```
error 1: line 15: near "x" : formal para num unmatch
error 2: line 15: near ")" : right parent lack
```

（7）错误测试二

将正确测试二的代码第 7 行：

```
        list : array[10] of integer;
```

改为：

```
        list : array[SIZE] of integer;
```

将正确测试二的代码第 27 行：

```
x1:=10;
```

改为：

```
x1=10;
```

报错信息：

```
error 1: line 7: near "[" : array length lack
error 2: line 27: near "x1" : function not found
error 3: line 28: near ";" : semicolon lack
```

（8）错误测试三

程序：

```
var b,c:char;
function swap:integer;
        var tmp:char;
        begin
                tmp:=b;
                b:=c;
                c:=tmp;
        end
;

begin
 read(b);
 read(c);
 swap;
 write(" ",b);
 write(" ",c);
end.
```

报错信息：

```
error 1: line 9: near ";" : none return
```

（9）错误测试四

将正确测试四的代码第 25 行：

```
        swap(list[j],list[j+1]);
```

改为：

```
        swap(list[j],list[j+h]);
```

在正确测试四的代码第 32 行添加：

```
        const j='a';
```

报错信息：

```
error 1: line 25: near "h" : var not define
Error 2: Line 32, 常量定义位置错误!
```

（10）错误测试五

将正确测试五的代码第 3 行：

```
                ft,i:   integer;
```

改为：

```
                ft,i,i: integer;
```

将正确测试五的代码第 36 行：

```
                if arr[ft] <= key then
```

改为：

```
                if arr[ft] <= key
```

将正确测试五的代码第 45 行：

```
                while flag := 1
```

改为：

```
                while flag = 1
```

报错信息；

```
error 1: line 3: near ":" : symbol conflict
error 2: line 30: near "flag" : unknow relation operator
error 3: line 30: near "flag" : end lack
error 4: line 31: near "1" : unknow relation operator
error 5: line 37: near "key" : keyword error
error 6: line 49: near "end" : end lack
error 7: line 49: near "while" : semicolon lack
error 8: line 49: near "while" : semicolon lack
error 9: line 49: near "while" : predix lack
error 10: line 49: near "while" : main lack
error 11: line 52: near ";" : more word
```

报错说明：
error 1 说明了第一个错，重复定义；
error 5 说明了第二个错，then 删除，这里

2．测试结果分析

【说明上述测试程序对语法成分的覆盖情况】

说明见表下方。

测试程序号	覆盖的语法成分
正确测试一	1,2-11,15,17,18,22-24,26-33,37,38,40,41 2.<分程序> ::= <常量说明部分><变量说明部分><函数说明部分><复合语句> 12.<类型> ::= <基本类型> 13.<基本类型> ::= integer 19.<形式参数段>::= <标识符>: <基本类型> 20.<语句> ::= <赋值语句>|<条件语句>| <读语句>|<写语句> |<空> |<过程调用语句> 25.<因子>::= <标识符>|<无符号整数>|| <函数调用语句> 39.<写语句> ::= write'('<字符串>,<表达式>')'
正确测试二	1-14,16,18,19,22,26-28,36,37,39-41 20.<语句> ::= <赋值语句>|<过程调用语句>|<复合语句>|<写语句>|<空> 21.<赋值语句> ::= <标识符> := <表达式> 23.<表达式> ::= <项> 24.<项> ::= <因子> 25.<因子>::= <标识符>

正确测试三	1-13,15,17,18,22-24,26-31,33,36-41 19.<形式参数段>::= <标识符>{, <标识符>}: <基本类型> 20.<语句> ::= <赋值语句>|<条件语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<空> 21.<赋值语句> ::= <标识符> := <表达式>| <函数标识符> := <表达式> 25.<因子>::= <标识符>|<无符号整数>| <函数调用语句> 32.<关系运算符> ::= =
正确测试四	1-14,16,18-19,23,26-29,31,35-41 20.<语句> ::= <赋值语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<for 循环语句>|<空> 21.<赋值语句> ::= <标识符> := <表达式>| <标识符>'['<表达式>']':= <表达式> 24.<项> ::= <因子> 25.<因子>::= <标识符>|<标识符>'['<表达式>']'|<无符号整数>| <函数调用语句>
正确测试五	1,7-12,14,16,18,22-23,26-29,31,34-37,39-41 2.<分程序> ::= <变量说明部分><过程说明部分><复合语句> 13.<基本类型> ::= integer 20.<语句> ::= <赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>|<复合语句>|<写语句>|<空> 21.<赋值语句> ::= <标识符> := <表达式>| <标识符>'['<表达式>']':= <表达式> 24.<项> ::= <因子> 25.<因子>::= <标识符>|<标识符>'['<表达式>']'|<无符号整数>| <函数调用语句> 32.<关系运算符> ::= <|<=|>= |= 33.<条件语句> ::= if<条件>then<语句>else<语句>

(说明：覆盖的语法成分的编号与 1.1 文法说明相同。
若完全覆盖则直接用编号，否则若只有该成分的部分则单独说明)
经过上面分析，可知测试程序覆盖所有的文法。
同时，测试程序中考虑了 judge 课设要求中显式写出来的要求：
大小写、函数调用、不同作用域的同名变量（参见测试三）、递归调用

五．总结感想

【说明在完成课程设计中的收获、认识和感想】

不要再见吧，编译！被你虐的，真心无爱了!!!