《编译技术》课程设计文 档

学号：\_\_\_\_\_\_\_16061184\_\_\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_王冬冬\_\_\_\_\_\_\_\_\_\_\_\_

2019年 1 月 4 日

# 一．需求说明

## 1．文法说明

获取的文法如下：

1、＜加法运算符＞ ::= +｜-

范例：c=a+b，d=a-b

分析：加法运算符前后是两个项，进行加减运算，或者在一个项前表示正负

2、＜乘法运算符＞  ::= \*｜/

范例：c=a\*b,d=a/b

分析：乘法运算符前后是两个项，进行乘除运算

3、＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==

范例：if(a!=b){}

if(c==d){}

分析： <表示小于；<=表示小于等于；>表示大于；>=表示大于等于；!=表示不等于；==表示恒等于；这些比较符号多用于条件语句中，作为执行条件

4、＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

示例：char flag::=v;

分析：字母包括小写字母a-z和大写字母A-Z和下划线

5、＜数字＞   ::= ０｜１｜．．．｜９

示例：int line::=90;

分析：数字包括0至9的数字

6、＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'

示例：’+’，’0’

分析：字符包括+，-，\*，/，字母和数字，不包括特殊符号

7、＜字符串＞   ::=  "｛十进制编码为32,33,35-126的ASCII字符｝"

示例：chat s=”\*bu6 6\_a^5”

分析：字符串和字符没有确切关系，包含广泛，长度不受限制

8、＜程序＞    ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞

示例：const int a=5,b=7;

const char sll=’v’,sdd=’b’;//常量说明

int c,line[10];//变量说明

int fun(int x,char y){fun分程序(略)}

void sort(int d[5],int e){sort分程序(略)}

int main(){主程序(略)}

分析：常量说明部分与变量说明部分可有可无但顺序不能变，有返回值函数与无返回值函数可以没有可以有多个，且顺序没有要求，主函数必须在最后，且只有一个

9、＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}

示例：const int a=5,b=7;

const char sll=’v’,sdd=’b’;

分析：常量说明部分可以有多个const，但每一个const中只包含一种类型int或char，该类型的声明至少有一个

10、＜常量定义＞   ::=   int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

范例：int line = 20, cc = 50;

char s = ’k’, flag = ‘ a’;

分析：常量定义只包含一种类型，int或char，每种类型都可以有多个声明

11、＜无符号整数＞  ::= ＜数字＞｛＜数字＞｝

范例：int a = 200398;

分析：无符号整数由0到9之间的字符串组成，没有长度顺序限制

12、＜整数＞        ::= ［＋｜－］＜无符号整数＞

范例：整数在无符号整数的前面加上加号或减号，加号或减号不一定要有

13、＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

示例：char lost2first = ‘d’;

分析：标识符的第一个字符必须是字母，其后的字符为字母或数字，顺序数量不限

14、＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞

示例：int fun(char s,int a){}

分析：声明头部用于声明函数的类型和名字，类型有int、char两种，名字为标识符

15、＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}

示例：int add, line[22];

char s, str[45];

分析：变量声明包括至少一个变量定义

16、＜变量定义＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’ )}  //＜无符号整数＞表示数组元素的个数，其值需大于0

示例：int add, line[22];

分析：每一个变量定义包含一个int或char，其后为一般变量或数组，数组方括号中必须为无符号整数，一般变量和数组一共至少有一个，声明顺序不限

17、＜类型标识符＞      ::=  int | char

示例：int a = 8;char prinf(int num,char cs){}

分析：类型标识符包括int类型和char类型，用于声明函数或变量、常量的类型

18、＜有返回值函数定义＞  ::=  ＜声明头部＞‘(’＜参数表＞‘)’ ‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’  //第一种选择为有参数的情况，第二种选择为无参数的情况

示例：int fun(int a, int b){

int add, line[22];

if(a<b){

add=3;

}

else{

add=2;

}

return add;

}

分析：有返回值函数定义首先为声明头部，用于定义返回值的类型，有参数的函数有参数表，无参数的函数无参数表，函数主体为复合语句。

19、＜无返回值函数定义＞  ::= void＜标识符＞(’＜参数表＞‘)’‘{’＜复合语句＞‘}’| void＜标识符＞{’＜复合语句＞‘}’//第一种选择为有参数的情况，第二种选择为无参数的情况

示例：void sort{

printf(“there is an error!”);

}

分析：无返回值函数定义开始必须为void<标识符>，有参数的函数有参数表，无参数的函数无参数表，函数主体为复合语句。

20、＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞

示例：const int a=5,b=7;

int add, line[22];

if(a<b){

add=3;

}

分析：复合语句中常量说明和变量说明可有可无，但顺序不可变，其后为语句列

21、＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}

示例：int fun(int a, char b){}

分析：参数表用于有参数的函数，参数由函数外部输入，进入函数参与运算，函数内部对参数的改变会影响外部

22、＜主函数＞    ::= void main‘(’‘)’‘{’＜复合语句＞‘}’

示例：void main(){

Printf(“Hello Word!”);

}

分析：主函数不返回值，开头为void，由于为最外层的函数，括号内没有参数，函数主体为复合语句。

23、＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}  //[+|-]只作用于第一个<项>

示例：-30+ fun(int a,int b)

分析：表达式至少包含一个项，相邻两项间必须有加法运算符+或-，第一个项前面可以有+或-表示正负，也可以没有默认无符号

24、＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}

示例：line \* sd

分析：项由至少一个因子组成，相邻因子间必须有乘法运算符\*或/

25、＜因子＞    ::= ＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’|‘(’＜表达式＞‘)’｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞

示例：line[2]

fun(int a,int b)

分析：因子可以为标识符，标识符必须有值，可以为数组元素，可以为(表达式)，可以为整数，可以为字符，计算时转为assic码，可以为有返回值调用语句，因子必须能够计算出值，能够用于计算

26、＜语句＞    ::= ＜条件语句＞｜＜循环语句＞| ‘{’＜语句列＞‘}’| ＜有返回值函数调用语句＞;|＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;|＜返回语句＞;

示例：while(a){

printf(“ad”);

a=a-1;

}

分析：语句包含条件语句、循环语句、{语句列}、有返回值函数调用语句、无返回值函数调用语句、赋值语句、读语句、写语句、空语句、返回语句，只含有一个上述成分

27、＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞

示例：first=a+b;

line[2]=c+d;

分析：赋值语句等号前面为标识符或数组元素，等号后的表达式的值赋给该成分

28、＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞[else＜语句＞]

示例：if(a==b){

printf(“true”);

}

else{

printf(“false”);

}

分析：条件语句必须有if和紧跟其后的语句成分，else及其后的语句成分可有可无

29、＜条件＞    ::=  ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真

示例：a+b<=c

分析：条件为一个表达式或两个表达式和其之间的关系运算符，表达式可以计算出值来进行比较，关系运算符为<｜<=｜>｜>=｜!=｜==，用于比较得出true或flase，进行条件判断

30、＜循环语句＞   ::=  while ‘(’＜条件＞‘)’＜语句＞ |for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞')'＜语句＞  
 示例：for(a=3; a<10;a=a+1){

printf(a);

}

分析：循环语句包括while循环和for循环，while循环的括号内为条件，主体为语句，for循环括号内有三部分，第一部分为＜标识符＞＝＜表达式＞，用于给标识符赋初值，第二部分为条件，判断是否结束循环，第三部分为＜标识符＞＝＜标识符＞(+|-)＜步长＞，改变表示符的值，进行下一次循环和判断

31、＜步长＞::= ＜无符号整数＞

示例：6

分析：步长用于每次循环结束后对标识符进行改变

32、＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

示例：add(a+b, c);

分析：有返回值函数调用语句用于调用有返回值函数，得到返回值，有参数时括号内为传入函数并进行运算的参数

33、＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

示例：fun(char c)；

分析；无返回值函数调用语句用于调用无返回值函数，进行相关操作，有参数时括号内为传入函数并进行运算的参数

34、＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}

示例：a+b,c

分析：值参数表至少包含一个表达式，相邻表达式间有逗号隔开，每个表达式都是一个参数，传入函数进行相关运算。

35、＜语句列＞   ::= ｛＜语句＞｝

示例：add(a+b, c);

fun(char c)；

for(a=3; a<10;a=a+1){

printf(a);

}

分析：语句列包含任意多个语句，可以为空

36、＜读语句＞    ::=  scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

示例：scanf(a,b,c);

分析：读语句读取外部输入的值并传递给变量

37、＜写语句＞    ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞‘)’

示例：printf(“hello world”,a+b);

分析：写语句用于输出字符换或表达式的值，每个写语句最多包含一个字符串和一个表达式，且至少包含其中之一，字符串和表达式同时存在时，顺序不能变

38、＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]

示例：return a;

分析：返回语句用于返回函数值或跳出循环，没有表达式时，不返回，跳出循环或函数

## 2．目标代码说明

将四元式转换为MIPS指令，MIPS指令包括如下：

1. sw

示例：sw $1,10($2)

含义：memory[$2+10]=$1

1. lw

示例：lw $1,10($2)

含义：$1=memory[$2+10]

3、mul

示例：mult rd,rs,rt

含义：$rd=$rs\*$rt

4、addu

示例：addu $1,$2,$3

含义：$1=$2+$3

5、sub

示例：sub $1,$2,$3

含义：$1=$2-$3

6、subi

示例：sub $1,$2,,-1

含义：$1=$2-(-1)

7、beq

示例：beq $1,$2,10

含义：if($1==$2)    
  goto PC+4+40

8、bgez

示例：bgez rs,10000

含义：if $rs>=0

then goto 10000

9、bltz

示例：bltz rs,10000

含义：if $rs<0

then goto 10000

1. bne

示例：bne $1,$2,10

含义：if($1!=$2)   
  goto PC+4+40

11、blez

示例：blez rs,10000

含义：if $rs<=0

then goto 10000

12、div

示例：div rs,rt

含义：(HT,LO)<-$rs/$rt

13、j

示例：j 10000

含义：goto 10000

1. jr

示例：jr $31

含义：goto $31

15、jal

示例：jal 10000

含义：$31<-PC+4;  
  goto 10000

16、move

示例：move $1,$2

含义：$1<-$2

.data区的内容包括数组，全局变量，字符串

## 3. 优化方案

1、基本块优化：构建基本块的DAG图可以完成消除局部公共子表达式等优化，采用从DAG导出中间代码的启发式算法还可以重新导出进一步优化的中间代码，方便以后的其他优化。

2、窥孔优化：由每一条中间代码的输入得到目标代码后，会存在大量的冗余指令或低效率指令，窥孔优化通过关注目标指令的一个较短序列，能够发现多余的指令或者化简指令。好的窥孔优化方案需要对目标体系结构、指令集都有充分的了解。

3、常数合并和传播：这种优化能在将编译时计算出值得表达式用其相应的值代替。

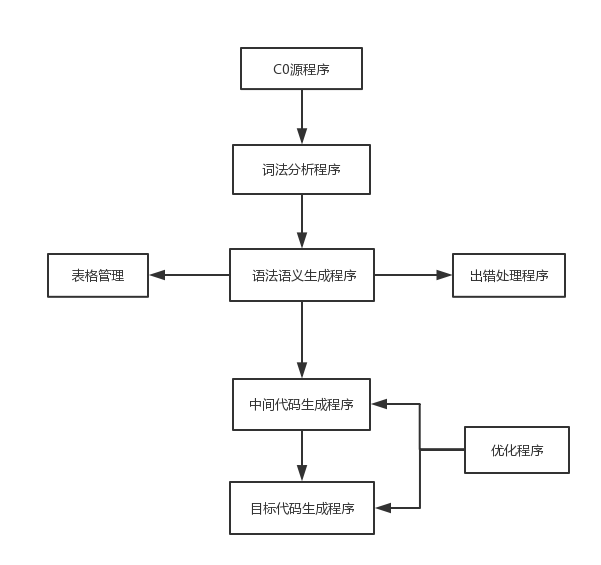
4、全局优化：通过数据流分析，计算每个程序执行点的可用表达式来判断公共子表达式；对于形如x=y的赋值语句，采用复制传播优化；对于不再被引用的定义变量，删除其定义点。

5、循环优化：循环体内部优化包含多重循环内外层交换以提高访问速度；循环展开以空间换取时间；代码外提，消除无意义的语句循环；代码强度削弱，代码长度开销换取运算速度。

6、全局寄存器分配：引用计数，需要统计每个函数内的变量的使用次数，引用次数多的变量有更大的机会被分配寄存器

# 二．详细设计

## 1．程序结构



## 2．类/方法/函数功能

void label\_creat()：生成label名

void t\_creat()：生成临时变量名

void str\_creat()：生成data数据区的字符串的名字

int getword(FILE \*fp, char \*w)：词法分析程序

void err\_mes(int n)：出错处理程序

int tra\_num(int a, char b[])：根据整数和正负符号输出整数

int var\_type(char func\_name[], char var\_name[])：查找符号表，返回变量的类型

void enter\_sym\_tab(char sym\_name[], int sym\_type, int sym\_num, int sym\_addr, int level)：建立符号表

void enter\_mid\_tab(int mid\_id, char mid\_argu1[], char mid\_argu2[], char mid\_result[], int mid\_num)：建立中间代码表

void enter\_func\_tab(char func\_name[])：建立函数信息表

void enter\_var\_tab(char var\_name[], int type, int addr)：建立每个函数的变量信息表

void enter\_tab(int mid\_num)：遍历中间代码表，建立函数信息表和相应的变量信息表

int look\_tab(char sym\_name[])：建立符号表时，遇到新的常量、变量声明，查表是否已声明

int look\_func(char sym\_name[])：遇到函数声明，查符号表看是否已声明

int over\_con\_type(char var\_name[])：查符号表，返回全局变量的类型

int local\_var\_type(char func\_name[], char var\_name[])：查符号表，返回局部函数的类型

int var\_pos1(char func\_name[], char var\_name[])：返回变量在该函数中的位置(建表时查)

int var\_pos2(char func\_name[], char var\_name[])：返回变量在该函数中的位置(建完表时查)

int func\_pos(char func\_name[])：返回函数在函数信息表中的位置

int addr\_var(char func\_name[], char var\_name[])：查找标量信息表中变量的地址

void int\_ass()：语法分析过程中，处理int类型的变量定义

void char\_ass()：语法分析过程中，处理char类型的变量定义

void array\_def()：语法分析过程中，处理数组的定义

void con\_def()：语法分析过程中，处理常量的定义

void var\_def()：语法分析过程中，处理变量的定义

void comp\_state()：语法分析过程中，处理复合语句

void statement()：语法分析过程中，处理语句

void while\_state()：语法分析过程中，处理while循环语句

void for\_state()：语法分析过程中，处理for循环语句

void if\_statement()：语法分析过程中，处理if条件语句

void scanf\_state()：语法分析过程中，处理读语句

void printf\_state()：语法分析过程中，处理读语句

void return\_state()：语法分析过程中，处理返回语句

void condition()：语法分析过程中，处理条件

void expression()：语法分析过程中，处理表达式

void item()：语法分析过程中，处理项

void factor()：语法分析过程中，处理因子

void value\_para\_tab()：语法分析过程中，处理形参表

void para\_tab()：语法分析过程中，处理参数表

void return\_value()：语法分析过程中，处理有返回值函数

void no\_return\_value()：语法分析过程中，处理无返回值函数

void main\_fun()：语法分析过程中，处理主函数

void mid\_code(int mid\_num)：中间代码生成程序

void numtostr(int num)：将数字变为相应字符串

void mid\_update(int mid\_num)：常量合并

void goal\_code(int mid\_num,int sym\_num)：目标代码生成程序

void creat\_data(int mid\_num,int sym\_num,FILE \*goal)：生成目标代码程序的data区

void creat\_text(int mid\_num, FILE \*goal)：生成目标代码程序的text区

int tolow\_com(char w[])：将关键字统一变为小写

void low\_sym(char w[])：将标识符统一变为小写

void optimize()：代码优化程序

void update(int i)：优化过程中更新中间代码

void quote\_count()：引用计数程序

void sort\_var\_info()：引用计数后重新排列变量信息表

## 3．调用依赖关系

语法分析程序依赖词法分析程序的输出结果

函数间的大量调用主要存在于语法分析程序中，语法分析程序完全按照文法规则中的关系，由主函数逐步调用各个分程序及语句函数。

目标代码生成程序需要调用中间代码程序

语法分析程序、中间代码生成程序和目标代码生成程序调用建表、查表和其他函数

### 4．符号表管理方案

符号表主要包括标识符和函数的名字、类型、值、作用域的嵌套层次，对于数组值为下标。

在语法分析过程中，遇到新的变量、常量、函数声明则先查表，若已存在相同的name，则报错处理，否则填入符号表，并记录相关属性信息；遇到标识符时则查表，看是否已定义，若未定义，则报错处理

## 5．存储分配方案

由于存在递归函数的可能，所以C0程序存储采用动态存储分配。

动态存储分配用运行栈来实现。遇到函数调用时，则在运行栈分配空间，自上至下分别为，传入调用函数的参数，返回地址，函数返回值，函数用到的所有变量，其中对于数组，和其他变量分配相同大小的空间，改变其值时则在data区，对于函数的前8个变量存入s寄存器，当需要使用或者改变变量的值时，若变量为前8个，则直接读写s寄存器，否则，读写运行栈内存。函数调用前需要将s寄存器值写回运行栈，调用结束后重新读回。

## 6. 四元式设计

中间代码表包含5个部分：中间代码类型，操作数1，操作数2，结果，相关值。相关值为数组的下标，常量值等。

按照中缀表达式格式输出中间代码，四元式的种类包括：

1. 函数声明

源码形如：

int foo(int a ,char b)

中间代码：

int foo()

para int a

para char b

1. 函数调用

源码形如：

i=foo(a,b)

中间代码：

push a

push b

call foo

i=RET

1. 函数返回

源码形如：

return x

中间代码：

ret x

1. 常量声明

源码形如：

const int a = 10

中间代码：

const int a = 10

1. 变量声明

源码形如：

int a

中间代码：

Var int a

1. 表达式

源码形如：

x=a\*(b+c)

中间代码：

@t1=b+c

@t2=a\*@t1

x=@t2

1. 条件判断

源码形如：

x == y

中间代码：

x == y

1. 条件或无条件跳转

中间代码：

GOTO @label1

BNZ @label2

1. 带标号语句

源码形如：

@label1：x=a+b

中间代码：

@label1:

x=a+b

10、数组赋值或取值

源码形如：

a[i]=b\*c[j]

中间代码：

@t1=c[j]

@t2=b\*@t1

a[i]=@t2

11、读语句

源码形如：

scanf(a)

中间代码：

scanf a

12、写语句

源码形如：

printf(a)

中间代码：

printf a

## 7. 目标代码生成方案

生成目标代码的过程，需要用到中间代码表，每条中间代码翻译为执行效率最高的目标代码，同时还需要维护运行栈和寄存器，如进入调用函数需要申请运行栈空间，写值，调用结束后需要清空申请运行栈，读值。

对于全局变量、数组和字符串要申请空间存储在.data区内。

## 8. 优化方案

1、常数合并与传播

将整个程序中全局常量替换为常数声明的值，对于局部常量，只在当前函数内替换。对于能够计算出结果的常量表达式，直接在中间代码阶段计算出值。这样可以节省目标代码中关于常数的存取和计算

2、引用计数

遍历函数，统计变量的使用次数并重新排列变量信息表，排在前面的变量更有几率被分配寄存器。这样可以使读写次数更多的变量使用寄存器读写，节省读写开销。

3、复制传播

对于不影响程序的赋值语句，可以复制给其他语句，并传播这个过程。这样避免了刚写回内存，就读出来使用的无用开销。

4、窥孔优化

对于无用、累赘的中间代码，如

Goto @label1

@label1

的代码，对程序没用影响，且会增加开销

## 9. 出错处理

出错处理方案：跳转到下一个分号后面的token

部分出错信息及含义如下：

1. 缺失标识符
2. 缺失分号
3. 缺失类型标识符
4. 缺失赋值符号
5. 缺失无符号整数
6. 缺失单引号
7. 非法字符
8. 多字符
9. 数组缺失元素个数

10、数组元素个数为0

11、缺少右方括号

12、缺失右小括号

13、缺失左大括号

14、缺失左小括号

15、缺失等号

16、缺失加法运算符

17、空字符串

18、双引号不匹配

19、缺失右大括号

20、缺失主函数

21、空字符

22、字符串非法

23、类型不匹配

24、参数数目不匹配

25、标识符未定义

26、重定义标识符

27、函数未定义

28、因子为无返回值函数

29、赋值语句左侧为不是变量

30、数组不存在

31、有返回值函数缺少返回值

32、无返回值函数有返回值

33、数组越界

34、数组下标不为整型

35、标识符为关键字

36、条件类型错误

# 三、操作说明

## 1、运行环境

Visual Studio 2017，MARS

## 2、操作步骤

运行工程文件，输入测试文件路径，若进行优化，输入Y，否则，输入N，生成的mid\_code.txt为中间代码文件，goal\_code.asm为目标代码运行文件，进入MARS，打开oal\_code.asm并运行即可。

# 四、测试报告

## 1、测试程序即测试结果、

e.g.1 正确执行 输入3，输出6

1. **int** fact(**int** t)
2. {
3. **if**(t < 3)
4. **return**(t);
5. **else**
6. **return**(fact(t-1)\*t);
7. }
8. **void** main(){
9. **int** a,b;
10. **scanf**(b);
11. a=fact(b);
12. **printf**(a);
13. }

e.g.2 正确执行 输入3,4 输出a>0,3

1. **void** main(){
2. **int** a,r,x;
3. scanf(a, x);
4. **if** (a > 0){
5. printf("a>0");
6. **if** (x <= 0) {
7. r = a / x \* a;
8. printf("x<=0 ");
9. }
10. r = r + a;
11. }
12. printf(r);
13. }

e.g.3 正确执行 输入3,4 输出104

1. **char** ss{
2. **const** **int** b=5;
3. **int** i,j;
4. j=b;
5. **while**(j){
6. i=i+3;
7. j=j-1;
8. }
9. **if**(i>10){
10. **return**('a');
11. }
12. **else**{
13. **return**('b');
14. }
15. }
16. **void** main(){
17. **int** a,r,x;
18. scanf(a, x);
19. printf(a+x+ss);
20. }

e.g.4 正确执行 输出10

1. **int** x;
2. **void** ss{
3. **int** i;
4. **for**(i=0;i<5;i=i+1){
5. x=x+i;
6. }
7. }
8. **void** main(){
9. ss;
10. printf(x);
11. }

e.g.5 正确执行 输入3 输出12

1. **void** main(){
2. **const** **int** a1=1;
3. **int** a,b,c,x,d[5];
4. scanf(x);
5. d[0]=a1;
6. d[1]=2;
7. d[2]=3;
8. d[3]=4;
9. a=d[2];
10. b=d[3];
11. c=d[1];
12. printf(a+b+c+x);
13. }

e.g.6 错误执行 输出Error--------Missing semicolon!

1. **int** fact(**int** t)  {
2. **if**(t < 3)
3. **return**(t);
4. **else**
5. **return**(fact(t-1)\*t)
6. }
7. **void** main(){
8. **int** a,b;
9. scanf(b);
10. a=fact(b);
11. printf(a);
12. }

e.g.7 错误执行 输出 Error--------Missing ')'!

1. **void** main(){
2. **int** a,r,x;
3. scanf(a, x);
4. **if** (a > 0){
5. printf("a>0";
6. **if** (x <= 0) {
7. r = a / x \* a;
8. printf("x<=0 ");
9. }
10. r = r + a;
11. }
12. printf(r);
13. }

e.g.8 错误执行 输出Line 16:Error--------Missing identifier!

1. **char** ss{
2. **const** **int** b=5;
3. **int** i,j;
4. j=b;
5. **while**(j){
6. i=i+3;
7. j=j-1;
8. }
9. **if**(i>10){
10. **return**('a');
11. }
12. **else**{
13. **return**('b');
14. }
15. }
16. **void** (){
17. **int** a,r,x;
18. scanf(a, x);
19. printf(a+x+ss);
20. }

e.g.9 错误执行 输出Error--------Undefined identifier!

1. **int** x;
2. **void** ss{
3. **int** i;
4. **for**(i=0;i<5;i=i+1){
5. x=x+i;
6. }
7. b=2;
8. }
9. **void** main(){
10. ss;
11. printf(x);
12. }

e.g.10 错误执行 输出Error--------Array transboundary!

1. **void** main(){
2. **const** **int** a1=1;
3. **int** a,b,c,x,d[5];
4. scanf(x);
5. d[0]=a1;
6. d[1]=2;
7. d[2]=3;
8. d[3]=4;
9. a=d[2];
10. b=d[3];
11. c=d[1];
12. d[6]=5;
13. printf(a+b+c+x);
14. }

## 2、测试结果分析

e.g.1覆盖int返回值函数，读语句，写语句，主函数，赋值语句，有返回值函数函数调用语句，复合语句，语句

e.g.2覆盖条件语句，条件

e.g.3覆盖while循环语句，char返回值函数，返回语句

e.g.4覆盖for循环语句，无返回值函数语句，无返回值函数调用语句

e.g.5覆盖数组赋值和赋值数组

e.g.6覆盖缺失分号的错误

e.g.7覆盖缺失右小括号的错误

e.g.8覆盖缺失标识符的错误

e.g.9覆盖标识符未定义的错误

e.g.10覆盖数组越界的错误

# 五、总结感想

编译器作为计算机的最重要的课程之一，整个课程有体系的逐步引导我们做好一个编译器，使我们对编译器的整体结构和运行都有了深刻的了解，整个过程下来，明显感觉到编程理解能力有了很大的提升。

编译课设和书本结合也很好，尤其是优化，能够很好的锻炼一个程序员细致入微的处理程序的能力，可以说，整个课程对于整体提升学生编程素质有很大的作用。

希望未来老师们对于编译课设的体系能够做到更加完善，我们积累的教训也会在学弟学妹身上得到提醒，使整个课程更加趋于完美。