姓名：白文强 学号：20191060064 专业：计算机科学与技术

**一、实验目的**

1. 理解编译程序的基本逻辑过程；
2. 理解作用域管理和符号表的组织与实现；
3. 理解活动记录的基本设计，过程的调用和返回的实现；

**二、实验内容**

1. 实现减法(-)、除法(/)（40%）

将expr的定义扩展为：

expr→expr + term  expr - exprterm

term→term \* factor  term / factor  factor

1. 实现while语句（40%）

while循环的语法可以通过stmt增加如下候选式：

stmt→while bexpr do stmt

其中，bexpr为真执行do后面的语句，为假退出循环。

3、实现数组（20%）

将vardecl和variable的定义扩展为如下的规则：

vardef→ id | id[num: num]

variable→id | id[expr]

例如，int a[2:5]定义一个下标从2到5的4个元素的整型数组。数组元素可以在表达式中被引用。

1. **实验分析与设计**

（一）实现减法与除法

乘除法“\*”，“/”和加减法“+”，“-”满足左结合，且乘除优先于加减法。将一开始的expr定义改写为如下的产生式：

expr→expr + term | term

term→term \* factor | factor

factor→(expr) | variable | num | letter | true | false

为了实现减法与除法，需要将expr增加两项产生式，即：

expr→expr – term

term→term / factor

为了完成这两项功能，需要在单字符的符号表skind中添加减号(-)与除号(/)，在分析expr的程序中，需要增加判断当前token是否为减号即subtoken，若是，则要使用gettoken()获取下一个token，分析term语法，一直循环，直到通过gettoken()的token的类型不再是subtoken。

在分析term时，需要增加判断当前token是否为除号即divtoken，若是，则要使用gettoken()获取下一个token，随后进行factor的分析，一直循环，直到当前token的两类型不再是divtoken。

（二）实现while语句

while语句的文法如下：

stmt→while bexpr do stmt。

翻译模式：

stmt→while bexpr do stmt1

{

bexpr\_rep=addrLabel(NewLabel());

stmt\_next=addLabel(NewLabel());

stmt.code=(LAB, bexpr\_rep, NULL,NULL)

|| bexpr.code

|| (JPC, bexpr.addr, NULL, stmt\_next)

|| stmt1.code

|| (JUMP, NULL, NULL, bexpr\_rep)

|| (LAB, stmt\_next, NULL, NULL)

}

为了实现对while语句的分析，需要在reservedword中增加while与do两个保留关键字，将符号分别设置为whiletoken与dotoken。

在分析stmt文法时，需要增加判断当前token是否为whiletoken，如果是，则进入WHILEstmt()函数对while语句进行分析，函数声明如下。

int WHILEstmt(int lev, int \*tx, int \*off)

在函数中，使用gettoken()获取一个新的token，随后进入bexpr的语法分析程序，并返回一个具有中间代码分量的结构的结构体指针。由于while的条件语句的值必须为布尔类型，即值为true或false。对返回的结构体的值进行判断，如果其中的type字段不是布尔类型，那么则表示while后的条件语句是不可判断真假的类型，无法进行后面的运算，说明该语句出现问题，此时要进行报错处理。

分析过bexpr后，接下来需要判断当前token的类型，while语句中必须包含do关键字，如果当前token不是do，那么则说明，while语句缺少do语句，进行报错处理。

如果当前token是do，则需要继续对stmt进行分析。

随后，需要设置两个标号，分别表示while语句入口以及while语句结束后跳转地址，接着进行中间代码生成工作。

（三）实现数组

为了实现数组，需要添加一个标识符类型array来表示该符号是数组类型，同时，为了支持数组的定义，还需要增加两个符号‘[’和‘]’，分别用lbracket和rbracket表示。

由于数组变量需要存储起始下标，因此需要对符号表进行修改，增加起始下标与结束下标：start\_index, end\_index

struct tablestruct

{

char name[al]; //符号的名字

enum idform form; //标识符的类型 ,可以通过符号的数据类型进行区分

enum datatype type; //符号的数据类型

int level; //符号所在的层

int address; //符号的地址

int start\_index;

int end\_index;

};

同时增加插入array变量的函数，enter\_array()，将数组变量插入符号表时，调用该函数完成，而不是原来的enter()，在enter\_array()中，大部分与enter()函数相同，只是增加了根据数组元素的个数计算存储地址偏移的功能。

if (tk == inttype)

(\*off) = (\*off) + 2 \* (end - start + 1);

else if (tk == booltype || tk == chartype)

(\*off) = (\*off) + 1 \* (end - start + 1);

由于数组的名字也是一个id，为了判断该id是不是数组类型的变量，需要在vardef()函数中进行修改。需要多读一个token，判断其是否是lbracket即‘[’，如果是，则说明该id是数组，否则是普通变量。

采用多读一个token来判断该id是不是数组变量的方式具有一个缺陷是难以将代码中缺少左中括号即‘[’的情况查找出来，我的解决方式是如果多读的token不是‘[’，在else语句中判断该token是否为num类型，如果是，则说明数组定义或访问缺少‘[’。

如果是数组，将form设置为array，接下来对‘num:num]’进行分析，如果出现错误，则进行报错。否则，使用enter\_array()函数将该符号插入到符号表中。

对数组元素进行访问时，由于数组的下标不是从0开始，而是由定义时指定，那么对应array指针中的地址的计算方式为：array\_index = (i – start\_index) \* type\_width。

在对数组元素进行赋值时，通过addrvar()函数获取的内容是元素的地址、类型等信息，在代码运行时，需要通过地址找到该元素在存储单元中的位置，因此，需要修改sourceOperandGen()与thirdOperandGen()函数，增加对数组类型的判断，在其中需要对寄存器bp进行保护，即将其压入栈中。

1. **实现与结果分析**

（一）减除的实现

用编译器对下面的程序进行分析与运行：

main()

{ int a,b;

read(a,b);

b=2+3;

{

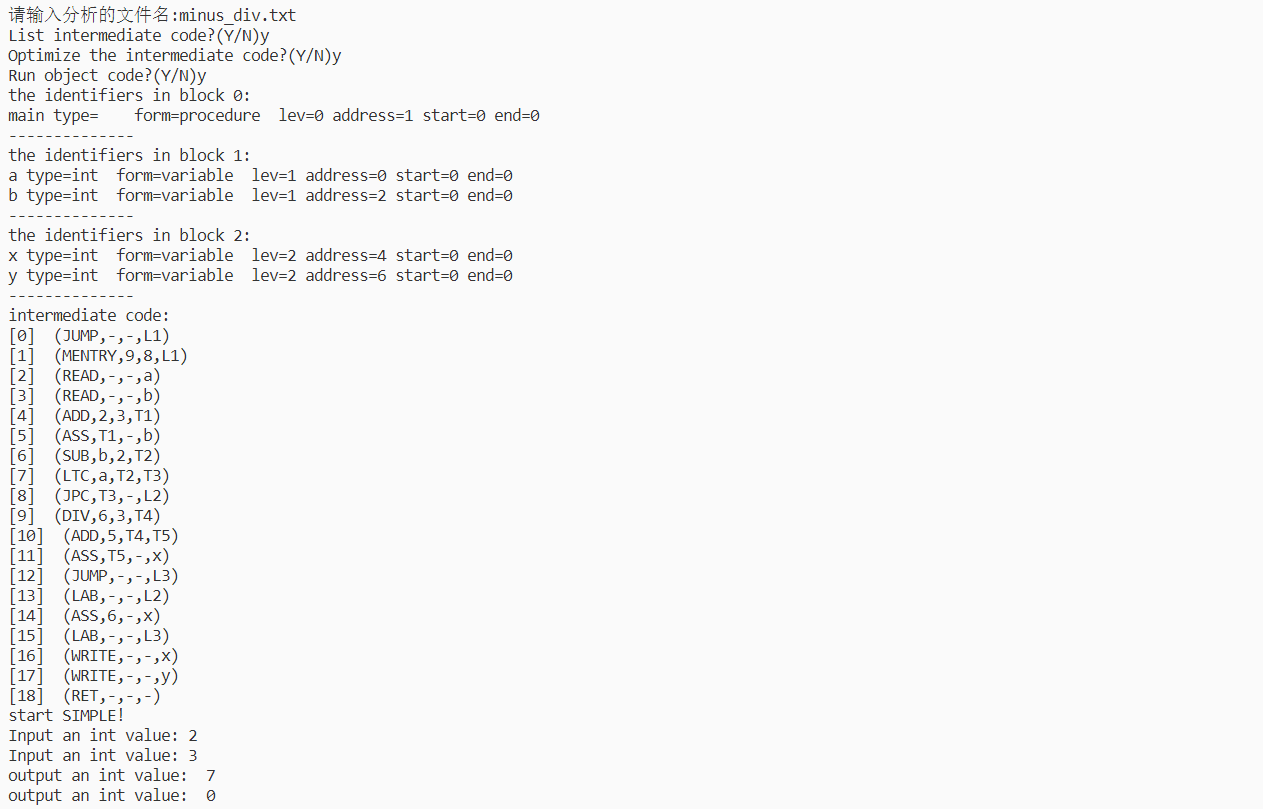
int x,y;

if a<b-2 then x=5+6/3; else x=6;

write(x,y);

}

return;

}

（二）对while语句的实现

用编译器对下面的程序进行分析与运行：

main()

{

int a,b;

read(a,b);

{

while a<b do a=a+1;

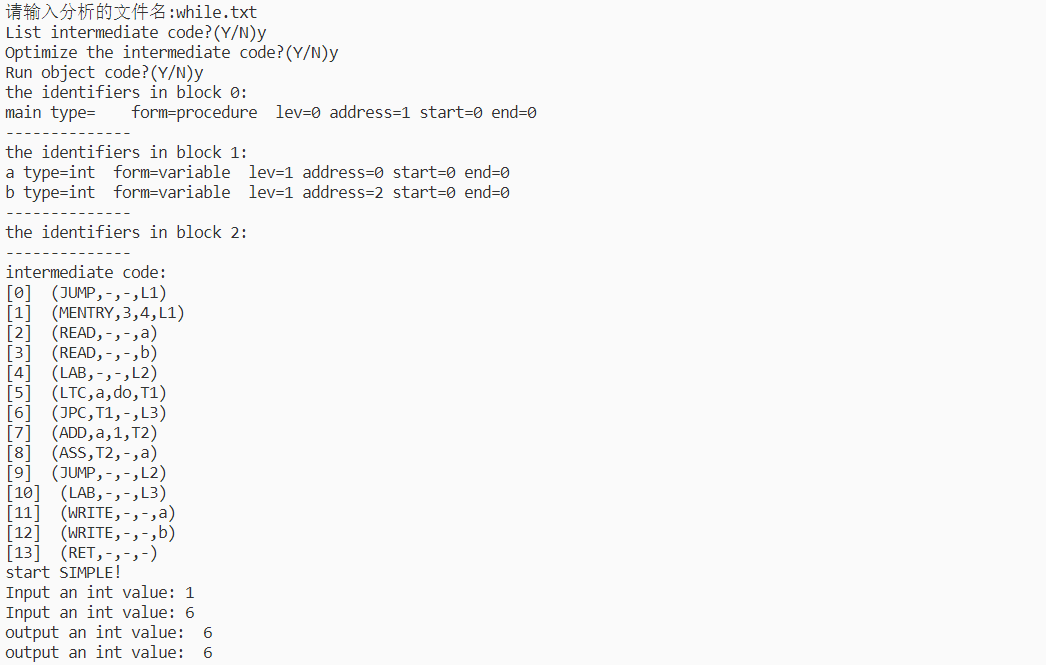
write(a,b);

}

return;

}

该程序实现的功能是：如果a是小于b的，那么将a每次加1，直到a与b相等。



（三）对数组的实现

使用编译器对下面的程序进行分析:

main ()

{ int x[1:3];

read(x[1]);

read(x[2]);

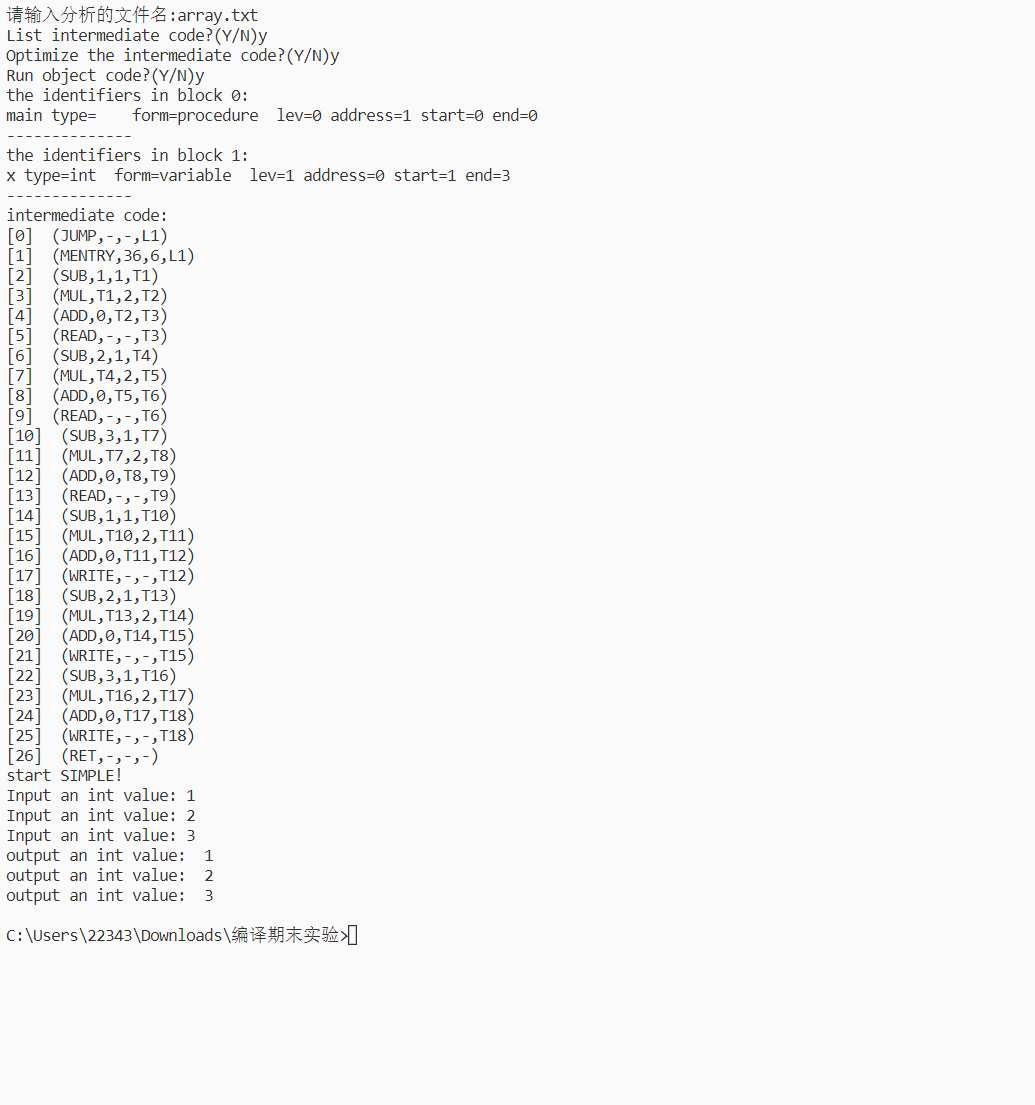
read(x[3]);

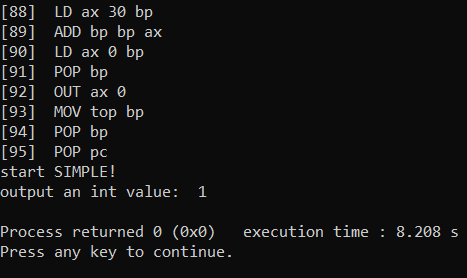
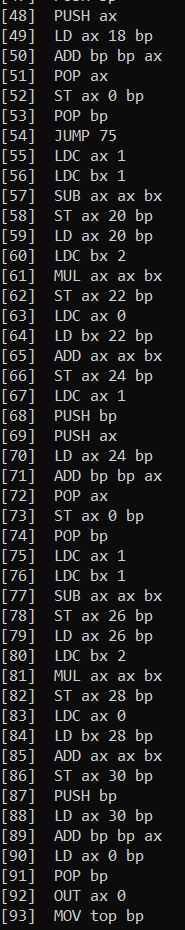
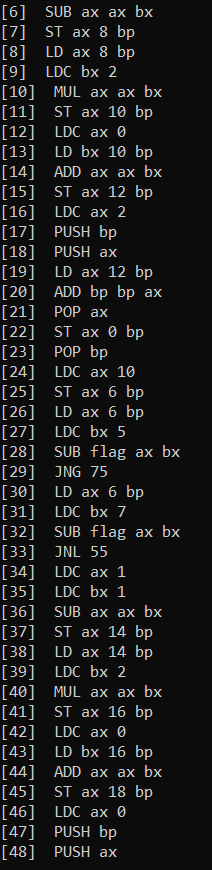
write(x[1],x[2],x[3]);

return;

}

该程序实现的功能是：输入一个数组，并将其输出。





由此可见，编译器对减法、除法、while语句、数组都实现了正确的编译和运行。