姓名：黄珀芝 学号：20201050331 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**

在SIMPLE基础上，扩展完成如下几个功能：

* 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do** **while语句（40%）**

**do while循环的语法可以通过*stmt*增加如下候选式：**

***stmt*→do *stmt* while *bexpr*;**

其中，***bexpr*为假退出循环。**

**3、增加一维数组（20%）**

将***vardef*和*variable*的定义扩展为**如下的规则：

***vardef*→ id**⏐**id**[**num**: **num**]

***variable*→id**⏐**id[*expr*]**

1. **实验分析与设计**
2. **实现减法(-)、除法(/)**

考虑将一开始的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

乘除法“\*”，“/”和加减法“+”，“-”满足左结合，且乘除优先于加减法，由此可以改写。

可以将expr增加两项产生式：

expr→expr – term

term→term / factor，方便实现乘法和除法。

* 为了完成上述功能，以下是具体设计与分析步骤：
* 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，增加判断当前token是否为减号。如果是，则使用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（即减号）。
* 在分析term的程序中，增加判断当前token是否为除号。如果是，则使用gettoken()获取下一个token，随后进行factor的分析，一直循环，直到当前token的类型不再是divtoken（即除号）。

1. **实现do while语句**

do while语句的文法如下：

***stmt*→do *stmt* while *bexpr*;**

翻译模式：

stmt → do {

stmt1.code

bexpr\_rep = addrLabel(NewLabel());

stmt\_next = addLabel(NewLabel());

stmt.code = (LAB, stmt\_next, NULL, NULL)

|| bexpr.code

|| (JPC, bexpr.addr, NULL, bexpr\_rep)

|| stmt1.code

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

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

} while bexpr

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

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

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

为了完成上述功能，以下是具体设计与分析步骤：

* 在函数中，首先需要记录下当前语句的起始位置，随后进入 stmt 的语法分析程序，并返回一个具有中间代码分量的结构体指针。
* 接着，使用 gettoken()获取一个新的 token，判断是否为 whiletoken。
* 如果不是，则说明语法错误，进行报错处理。如果是，则进入 bexpr 的语法分析程序，并返回一个具有中间代码分量的结构体指针。由于 do...while 循环至少会执行一次，因此无需对返回的结构体的值进行判断。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。

1. **增加一维数组**

为了支持数组类型的变量，需要在符号表中增加一个标识符类型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。

在符号表中，需要增加起始下标与结束下标：start\_index, end\_index来存储数组的范围信息。

|  |
| --- |
| 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。

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，即将其压入栈中以保护现场。

**四、实现与结果分析**

1. 实现减法、除法例子

用编译器对下面的程序进行分析与运行：

main()

{ int a1,b;

int x;

read(a1,b);

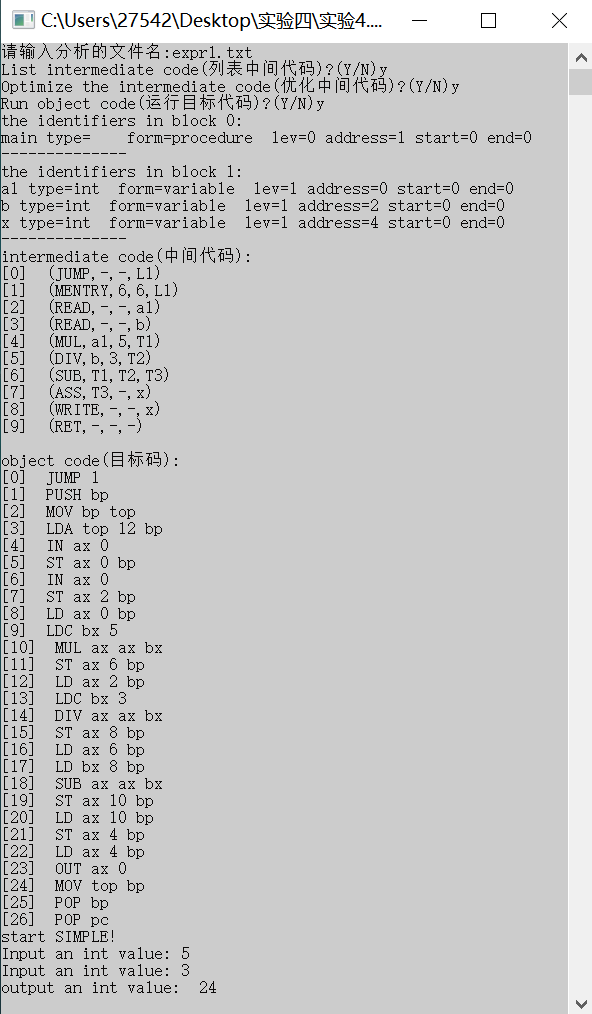
x=a1\*5-b/3;

write(x);

return;

}

结果：



x=5\*5-3/3=24

1. 实现do while语句例子

用编译器对下面的程序进行分析与运行：

main()

{ int a,b,x;

x=1;

b=100;

do {

x=x+2;

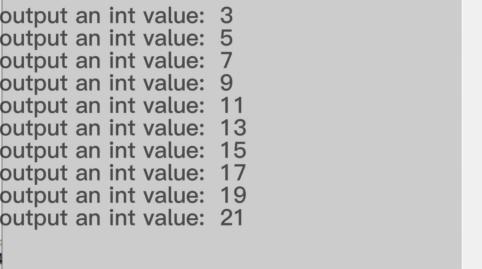
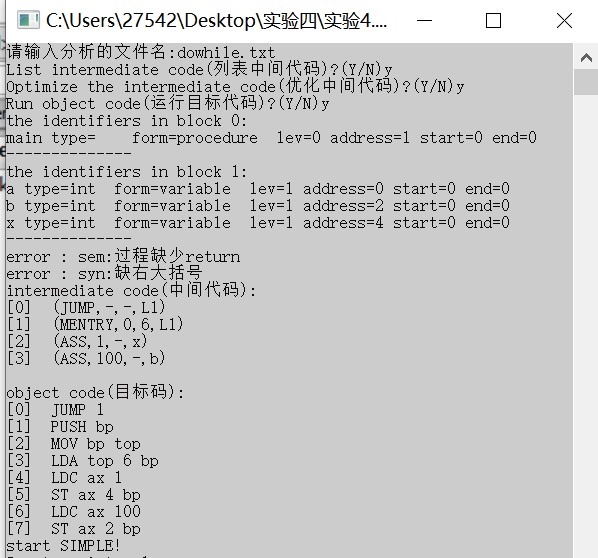
write(x);}

while x<20;

return;

}

结果：

****

1. 增加一维数组例子

用编译器对下面的程序进行分析与运行：

main()

{

int a[2:3],b;

a[2]=4;

a[3]=5;

write(a[2]);

write(a[3]);

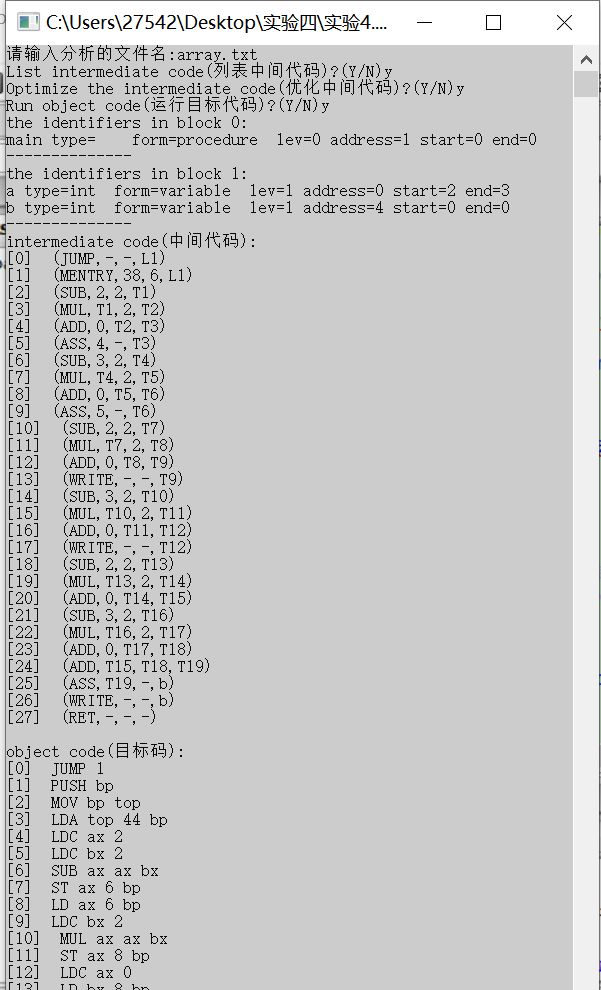
b=a[2]+a[3];

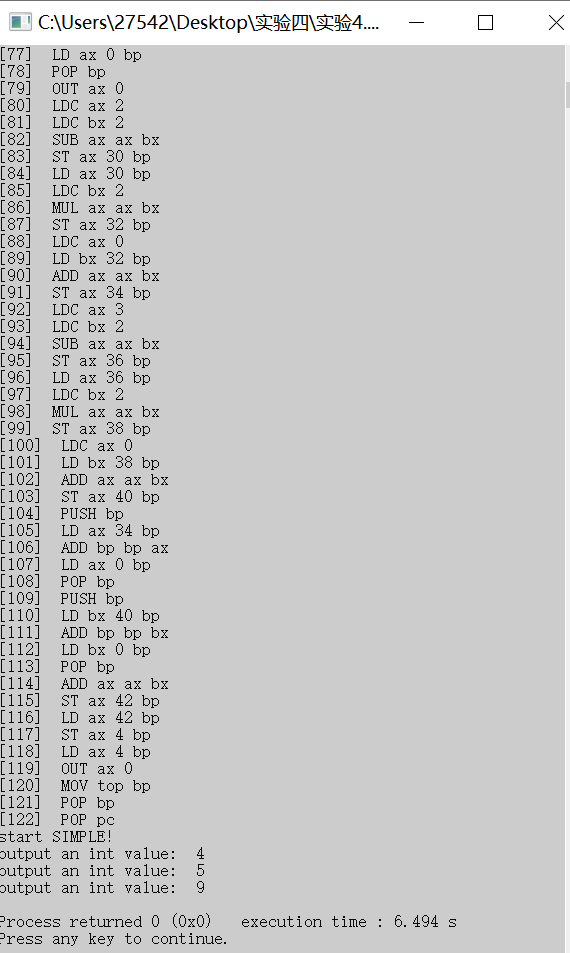
write(b);

return;

}

结果：结果过长，没有截取中间过程





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