

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：

$$expr \rightarrow expr + term \mid expr - expr \mid term$$
$$term \rightarrow term * factor \mid term / factor \mid factor$$

2. 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：

$$stmt \rightarrow \mathbf{do\ stmt\ while\ } bexpr;$$

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：

$$vardef \rightarrow id \mid id[num: num]$$
$$variable \rightarrow id \mid id[expr]$$

三、实验分析与设计

(1) 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：

$$expr \rightarrow expr + term \mid term$$
$$term \rightarrow term * factor \mid factor$$

$\text{factor} \rightarrow (\text{expr}) \mid \text{variable} \mid \text{num} \mid \text{letter} \mid \text{true} \mid \text{false}$

乘法“*”，“/”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：

$\text{expr} \rightarrow \text{expr} - \text{term}$

$\text{term} \rightarrow \text{term} / \text{factor}$ ，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）。

(2) 实现 do while 语句

do while 语句的文法如下：

$\text{stmt} \rightarrow \text{do stmt while bexpr};$

翻译模式：

```
stmt → do {  
    stmt1.code  
    bexpr_rep = addLabel(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。

(3) 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[a1];      //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
    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：

```
C:\Users\27542\Desktop\实验四\实验4....
请输入分析的文件名: expr1.txt
List intermediate code(列表中间代码)?(Y/N)y
Optimize the intermediate code(优化中间代码)?(Y/N)y
Run object code(运行目标代码)?(Y/N)y
the identifiers in block 0:
main type=      form=procedure  lev=0 address=1 start=0 end=0
-----
the identifiers in block 1:
a1 type=int  form=variable  lev=1 address=0 start=0 end=0
b type=int   form=variable  lev=1 address=2 start=0 end=0
x type=int   form=variable  lev=1 address=4 start=0 end=0
-----
intermediate code(中间代码):
[0] (JUMP, -, -, L1)
[1] (MENTRY, 6, 6, L1)
[2] (READ, -, -, a1)
[3] (READ, -, -, b)
[4] (MUL, a1, 5, T1)
[5] (DIV, b, 3, T2)
[6] (SUB, T1, T2, T3)
[7] (ASS, T3, -, x)
[8] (WRITE, -, -, x)
[9] (RET, -, -, -)
object code(目标码):
[0] JUMP 1
[1] PUSH bp
[2] MOV bp top
[3] LDA top 12 bp
[4] IN ax 0
[5] ST ax 0 bp
[6] IN ax 0
[7] ST ax 2 bp
[8] LD ax 0 bp
[9] LDC bx 5
[10] MUL ax ax bx
[11] ST ax 6 bp
[12] LD ax 2 bp
[13] LDC bx 3
[14] DIV ax ax bx
[15] ST ax 8 bp
[16] LD ax 6 bp
[17] LD bx 8 bp
[18] SUB ax ax bx
[19] ST ax 10 bp
[20] LD ax 10 bp
[21] ST ax 4 bp
[22] LD ax 4 bp
[23] OUT ax 0
[24] MOV top bp
[25] POP bp
[26] POP pc
start SIMPLE!
Input an int value: 5
Input an int value: 3
output an int value: 24
```

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

(2) 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:

```

C:\Users\27542\Desktop\实验四\实验4....
请输入分析的文件名:dowhile.txt
List intermediate code(列表中间代码)?(Y/N)y
Optimize the intermediate code(优化中间代码)?(Y/N)y
Run object code(运行目标代码)?(Y/N)y
the identifiers in block 0:
main type=      form=procedure  lev=0 address=1 start=0 end=0
-----
the identifiers in block 1:
a type=int  form=variable  lev=1 address=0 start=0 end=0
b type=int  form=variable  lev=1 address=2 start=0 end=0
x type=int  form=variable  lev=1 address=4 start=0 end=0
-----
error : sem:过程缺少return
error : syn:缺右大括号
intermediate code(中间代码):
[0]  (JUMP, -, -, L1)
[1]  (MENTRY, 0, 6, L1)
[2]  (ASS, 1, -, x)
[3]  (ASS, 100, -, b)

object code(目标码):
[0]  JUMP 1
[1]  PUSH bp
[2]  MOV bp top
[3]  LDA top 6 bp
[4]  LDC ax 1
[5]  ST ax 4 bp
[6]  LDC ax 100
[7]  ST ax 2 bp
start SIMPLE!

output an int value: 3
output an int value: 5
output an int value: 7
output an int value: 9
output an int value: 11
output an int value: 13
output an int value: 15
output an int value: 17
output an int value: 19
output an int value: 21

```

(3) 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()
```

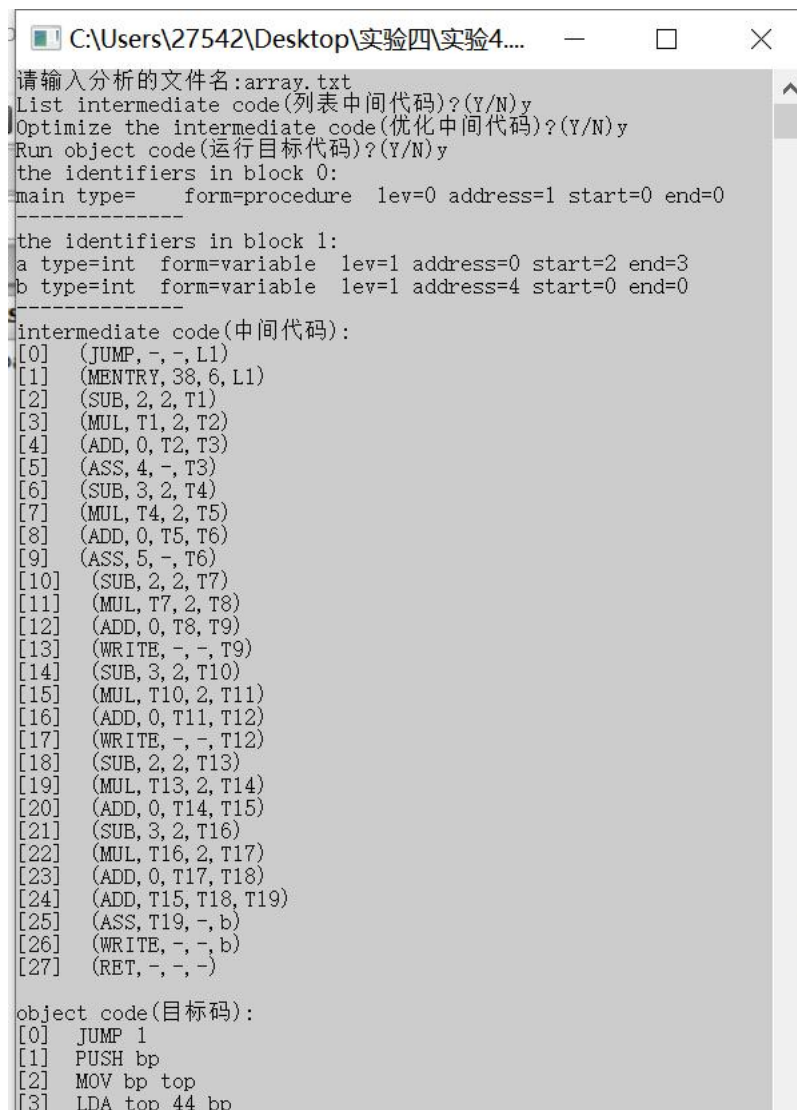
```

{
    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



```

C:\Users\27542\Desktop\实验四\实验4....
请输入分析的文件名:array.txt
List intermediate code(列表中间代码)?(Y/N)y
Optimize the intermediate code(优化中间代码)?(Y/N)y
Run object code(运行目标代码)?(Y/N)y
the identifiers in block 0:
main type=      form=procedure  lev=0 address=1 start=0 end=0
-----
the identifiers in block 1:
a type=int  form=variable  lev=1 address=0 start=2 end=3
b type=int  form=variable  lev=1 address=4 start=0 end=0
-----
intermediate code(中间代码):
[0] (JUMP, -, -, L1)
[1] (MENTRY, 38, 6, L1)
[2] (SUB, 2, 2, T1)
[3] (MUL, T1, 2, T2)
[4] (ADD, 0, T2, T3)
[5] (ASS, 4, -, T3)
[6] (SUB, 3, 2, T4)
[7] (MUL, T4, 2, T5)
[8] (ADD, 0, T5, T6)
[9] (ASS, 5, -, T6)
[10] (SUB, 2, 2, T7)
[11] (MUL, T7, 2, T8)
[12] (ADD, 0, T8, T9)
[13] (WRITE, -, -, T9)
[14] (SUB, 3, 2, T10)
[15] (MUL, T10, 2, T11)
[16] (ADD, 0, T11, T12)
[17] (WRITE, -, -, T12)
[18] (SUB, 2, 2, T13)
[19] (MUL, T13, 2, T14)
[20] (ADD, 0, T14, T15)
[21] (SUB, 3, 2, T16)
[22] (MUL, T16, 2, T17)
[23] (ADD, 0, T17, T18)
[24] (ADD, T15, T18, T19)
[25] (ASS, T19, -, b)
[26] (WRITE, -, -, b)
[27] (RET, -, -, -)

object code(目标码):
[0] JUMP 1
[1] PUSH bp
[2] MOV bp top
[3] LDA top 44 bp

```

```
C:\Users\27542\Desktop\实验四\实验4....  —  □  ×

[77] LD ax 0 bp
[78] POP bp
[79] OUT ax 0
[80] LDC ax 2
[81] LDC bx 2
[82] SUB ax ax bx
[83] ST ax 30 bp
[84] LD ax 30 bp
[85] LDC bx 2
[86] MUL ax ax bx
[87] ST ax 32 bp
[88] LDC ax 0
[89] LD bx 32 bp
[90] ADD ax ax bx
[91] ST ax 34 bp
[92] LDC ax 3
[93] LDC bx 2
[94] SUB ax ax bx
[95] ST ax 36 bp
[96] LD ax 36 bp
[97] LDC bx 2
[98] MUL ax ax bx
[99] ST ax 38 bp
[100] LDC ax 0
[101] LD bx 38 bp
[102] ADD ax ax bx
[103] ST ax 40 bp
[104] PUSH bp
[105] LD ax 34 bp
[106] ADD bp bp ax
[107] LD ax 0 bp
[108] POP bp
[109] PUSH bp
[110] LD bx 40 bp
[111] ADD bp bp bx
[112] LD bx 0 bp
[113] POP bp
[114] ADD ax ax bx
[115] ST ax 42 bp
[116] LD ax 42 bp
[117] ST ax 4 bp
[118] LD ax 4 bp
[119] OUT ax 0
[120] MOV top bp
[121] POP bp
[122] POP pc
start SIMPLE!
output an int value: 4
output an int value: 5
output an int value: 9

Process returned 0 (0x0)   execution time : 6.494 s
Press any key to continue.
```

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。