**C0编译器实验报告**

### 一、成员分工：

**钟辉**：词法分析、语法语义分析、生成中间代码，出错处理、图形界面

**刘嘉兴**：解释器

### 二、项目特点：

1、严格按照老师给出的C0编译器规范来编写，最终的结果与文档所要求的完全一致。

2、可以清楚的分辨全局变量和局部变量，优先使用局部变量，不会因全局变量和局部变量的定义而抛出重定义的错误。

3、对于函数，可以先调用再定义。

4、对出出错处理，可以定位到出错符号。

5、改进了名字表，便于进行作用域判断，增加了一项所属列，对于全局变量定义为“all”，对于局部变量定义为其所在的函数名。

6、图形界面可以显示C0源代码，名字表，中间指令，运行结果等。

### 三、本项目主要由以下部分组成：

**Constant**：定义常量、关键字类

这其中有关键字定义，如：“int” “if” “else”等

操作符定义，如：“LIT”,”LOD”,”CAL”等

**Lex**：语法分析类

定义了返回值类型类TypeValue，存储单词的类型和值。

getsym()每次读取一个单词，返回一个TypeValue对象，里面存有单词的 值和类型。

**Program**：语法语义分析类

FuncStruct类：处理先调用后定义的函数名和指令位置

excute()函数：处理主程序

block()函数：处理分程序（参数有IsMain：区分是在Main还是普通函数）

statement()函数：处理语句序列

assignDeal()函数：处理赋值和函数调用语句

scanfDeal()函数：处理输入语句

printDeal()函数：处理输出打印语句

ifDeal()函数：处理条件语句

whileDeal()函数：处理循环语句

returnDeal()函数：处理返回语句

intDeal()函数：处理变量声明语句

expression()函数：处理表达式

term()函数：处理项

factor()函数：处理因子

enter()函数：填入名字表

**Interpreter**：解释器类

ArrayList<Instruction> code：程序地址寄存器

int[] runStack：运行栈

int top\_Addr: 栈顶指针

int base\_Addr ：基址寄存器

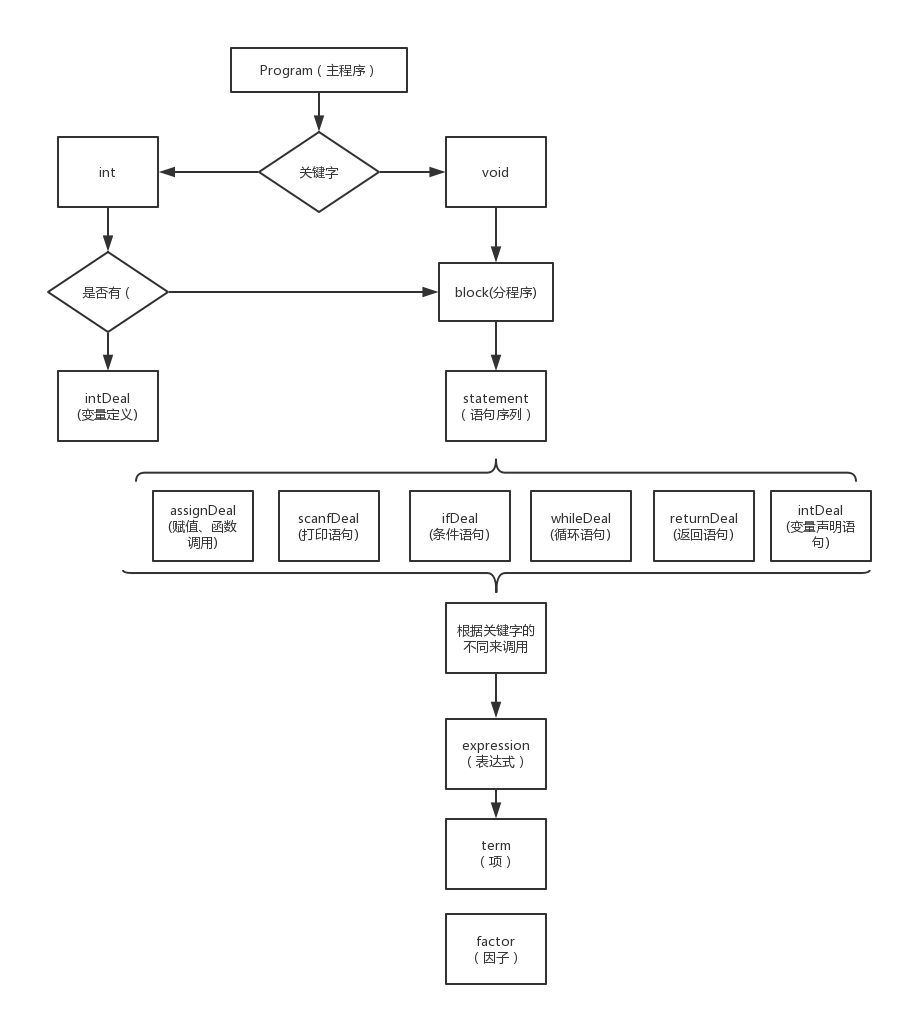
int current\_Addr ：指令寄存器当前正在解释执行的目标指令

paser()函数：解释执行函数

**Graph**：图形化界面类

设置了五个窗口，分别是源代码界面，名字表界面，目标指令界面，输出内容，输入框。这里不做过多介绍了。

**语法分析流程**：其中带中文括号说明的是函数（子程序）。



### 四、实现某些功能的代码举例

**1.实现函数调用在前，定义在后：**

解决方法：当读到函数调用时，遍历名字表发现无此函数名时，暂时先生成JMP指令，第二个操作数是当前指令表的大小，并记住该指令位置，当程序分析完，再遍历名字表，找到到之前的变量名，若还是没有就报“函数未定义错误”，若找到则回填修改JMP指令改成CAL指令。

**for**(FuncStruct fs:Main.*undefiedFunc*){*//未提前声明的函数回填* **boolean** v=**false**;  
 **for**(TableStruct ts:Main.*tableStructs*){  
 **if**(fs.**typevalue**.**value**.equals(ts.**name**)&&(ts.**type**.equals(***VOID\_PROCEDUR***))){  
 v=**true**;  
 Main.*aimCode*.get(fs.**pos**).**operator**=***CAL***;  
 Main.*aimCode*.get(fs.**pos**).**second**=ts.**addr**;  
 }  
 **else if**(fs.**typevalue**.**value**.equals(ts.**name**)&&(ts.**type**.equals(***INT\_PROCEDUR***))){  
 v=**true**;  
 Main.*aimCode*.get(fs.**pos**-1).**operator**=***CAL***;  
 Main.*aimCode*.get(fs.**pos**-1).**second**=ts.**addr**;  
 Main.*aimCode*.get(fs.**pos**).**operator**=***LOD***;  
 Main.*aimCode*.get(fs.**pos**).**frist**=0;  
 Main.*aimCode*.get(fs.**pos**).**second**=999;  
 }  
 }  
 **if**(v==**false**){  
 Graph.*jt\_out\_console*.append(**"函数未定义错误："**+**"Error at "**+fs.**typevalue**.**value**);**break**;  
 }  
}

**2.实现同名全局变量与局部变量：**

解决方法：名字表设置作用域范围，优先遍历局部变量

**int** position(TypeValue tv,String range) {  
 **for** (TableStruct table : **nametable**) {  
 **if** (table.**name**.equals(tv.**value**)&&table.**belong**.equals(range)) {  
 **return nametable**.indexOf(table);*//先找当前作用域的* }  
 **if** (table.**name**.equals(tv.**value**)&&table.**belong**.equals(**"all"**)) {  
 **return nametable**.indexOf(table);*//找全局变量* }  
 }  
 **return** -1;  
}

### 五、运行结果

**1.测试所用的C0代码为**：

包含了所有语句，覆盖了本编译器的全部功能。包括了全局量，函数的先调用后定义，四则运算（包括括号），用变量给变量赋值，循环语句（条件为表达式），读语句，写语句。

**2.C0代码实现的功能：**

先输出0，输入一个大于0任意的实数n，会输出0，0+1, 0+1+2，······，0+1+2+······+n且每个数占一行

**3.C0的源码**

int aa;

int m1()

{

int m;

scanf(m);

return m;

}

void main()

{

int m,g,h,i,x,y;

m=1\*1\*(1+5\*3-75/5)-1;

g=m;

i=0;

h=0;

aa=0;

printf(aa);

x=m1();

while(x-i+1)

{

m=m+i;

i=i+1;

printf(m);

}

y=g1();

while(y-h+1)

{

g=g+h;

h=h+1;

printf(g);

}

}

int g1()

{

int m;

scanf(m);

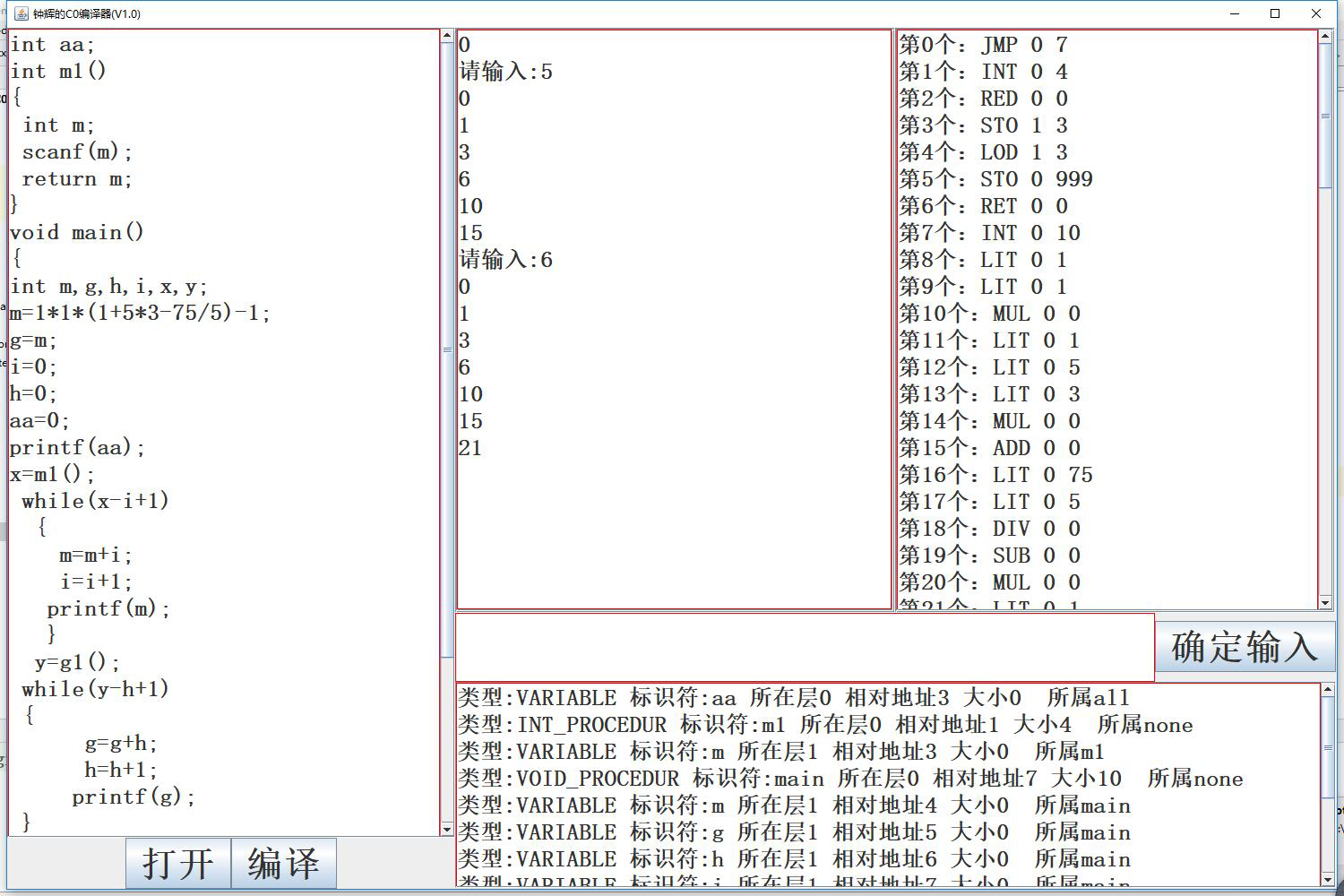
return m;

}

**4.运行结果图：**

结果正确执行！

其中源代码保存在了tmp.txt，名字表保存在了nametable.txt中，中间代码保存在了code.txt中，已打包在附上的文件中。



**产生的名字表如下：**

类型:INT\_PROCEDUR 标识符:m1 所在层0 相对地址1 大小4 所属none

类型:VARIABLE 标识符:m 所在层1 相对地址3 大小0 所属m1

类型:VOID\_PROCEDUR 标识符:main 所在层0 相对地址7 大小9 所属none

类型:VARIABLE 标识符:m 所在层1 相对地址3 大小0 所属main

类型:VARIABLE 标识符:g 所在层1 相对地址4 大小0 所属main

类型:VARIABLE 标识符:h 所在层1 相对地址5 大小0 所属main

类型:VARIABLE 标识符:i 所在层1 相对地址6 大小0 所属main

类型:VARIABLE 标识符:x 所在层1 相对地址7 大小0 所属main

类型:VARIABLE 标识符:y 所在层1 相对地址8 大小0 所属main

类型:INT\_PROCEDUR 标识符:g1 所在层0 相对地址71 大小4 所属none

类型:VARIABLE 标识符:m 所在层1 相对地址3 大小0 所属g1

**产生的中间代码如下：**

第0个 JMP 0 7

第1个 INT 0 4

第2个 RED 0 0

第3个 STO 1 3

第4个 LOD 1 3

第5个 STO 0 999

第6个 RET 0 0

第7个 INT 0 9

第8个 LIT 0 1

第9个 LIT 0 1

第10个 MUL 0 0

第11个 LIT 0 1

第12个 LIT 0 5

第13个 LIT 0 3

第14个 MUL 0 0

第15个 ADD 0 0

第16个 LIT 0 75

第17个 LIT 0 5

第18个 DIV 0 0

第19个 SUB 0 0

第20个 MUL 0 0

第21个 LIT 0 1

第22个 SUB 0 0

第23个 STO 1 3

第24个 LOD 1 3

第25个 STO 1 4

第26个 LIT 0 0

第27个 STO 1 6

第28个 LIT 0 0

第29个 STO 1 5

第30个 CAL 0 1

第31个 LOD 0 999

第32个 STO 1 7

第33个 LOD 1 7

第34个 LOD 1 6

第35个 SUB 0 0

第36个 LIT 0 1

第37个 ADD 0 0

第38个 JPC 0 50

第39个 LOD 1 3

第40个 LOD 1 6

第41个 ADD 0 0

第42个 STO 1 3

第43个 LOD 1 6

第44个 LIT 0 1

第45个 ADD 0 0

第46个 STO 1 6

第47个 LOD 1 3

第48个 WRT 0 0

第49个 JMP 0 33

第50个 CAL 0 71

第51个 LOD 0 999

第52个 STO 1 8

第53个 LOD 1 8

第54个 LOD 1 5

第55个 SUB 0 0

第56个 LIT 0 1

第57个 ADD 0 0

第58个 JPC 0 70

第59个 LOD 1 4

第60个 LOD 1 5

第61个 ADD 0 0

第62个 STO 1 4

第63个 LOD 1 5

第64个 LIT 0 1

第65个 ADD 0 0

第66个 STO 1 5

第67个 LOD 1 4

第68个 WRT 0 0

第69个 JMP 0 53

第70个 RET 0 0

第71个 INT 0 4

第72个 RED 0 0

第73个 STO 1 3

第74个 LOD 1 3

第75个 STO 0 999

第76个 RET 0 0

**5.对于出错处理**：

使用了未定义函数“func”，结果如下，可以指出错误位置

