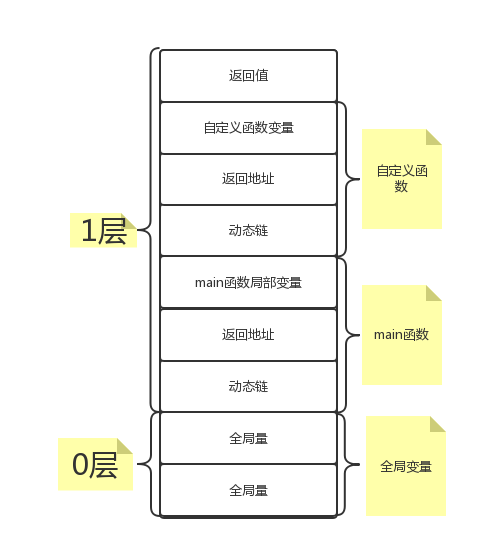
目录：

1. 编译器设计
2. JavaCC介绍
3. 测试设计
4. 异常检测
5. 附录

一、编译器设计

1.栈式结构



每个函数设立两个隐式单元，分别存储上层函数的栈基地址以及上层函数CAL指令下一条指令在指令序列表中的索引，以便退栈时使用。每个函数的返回值在栈顶存放，函数退栈时，将栈顶值暂存，退栈，然后将返回值放在栈顶，以达到返回值一直在栈顶的目的。

2.解释执行程序

数据结构：

**public class** Instruction {  
 **public enum** InstructionName {  
 ***LIT***, ***LOD***, ***STO***, ***CAL***, ***INT***, ***JMP***, ***JPC***,  
 ***ADD***, ***SUB***, ***MUL***, ***DIV***, ***RED***, ***WRT***, ***RET*** }  
 *//指令类型* **public** InstructionName **instructionName**;  
 *//变量使用与定义之间的层差* **public int t**;  
 *//变量相对地址或常数值* **public int a**;  
}

指令的数据结构，用来存储指令的信息。在解释执行程序的主类中存在一个指令的列表，用来存储所有指令并按顺序执行

辅助类：

**public class** MyError {  
 **public static enum** ErrCode{  
 ***ArithmeticException***,*//算数异常* ***InputParamErrException***,*//输入参数错误* ***AskMemoryTooBigException***,*//申请内存空间太大* ***CrossBroderException***,*//越界* ***StackOverflowException***,*//栈溢出* }  
 **public static boolean** *errFlag*=**false**;  
 **public static void** ShowErrMsg(**int** errCode,String errMsg){  
 **switch** (errCode){  
 **case** 1:{System.***err***.println(**"错误信息:"**+ErrCode.***ArithmeticException***.toString()+**" "**+errMsg);*errFlag*=**true**;**break**;}  
 **case** 2:{System.***err***.println(**"错误信息:"**+ErrCode.***InputParamErrException***.toString()+**" "**+errMsg);*errFlag*=**true**;**break**;}  
 **case** 3:{System.***err***.println(**"错误信息:"**+ErrCode.***AskMemoryTooBigException***.toString()+**" "**+errMsg);*errFlag*=**true**;**break**;}  
 **case** 4:{System.***err***.println(**"错误信息:"**+ErrCode.***CrossBroderException***.toString()+**" "**+errMsg);*errFlag*=**true**;**break**;}  
 **case** 5:{System.***err***.println(**"错误信息:"**+ErrCode.***StackOverflowException***.toString()+**" "**+errMsg);*errFlag*=**true**;**break**;}  
 **default**:{}  
 }  
 }  
}

错误信息类，在解释执行过程中，可能会发生一些异常和错误，我们在错误信息类中列举了常见的错误信息，当发生错误时可以通知用户。

主类模块划分：

1.程序入口，提供两个重载，一个为手动输入目标文件地址（用户手动执行调用），一个传入目标文件地址（供给编译程序调用）。

**public void** start()：

**public void** start(String SourcePath)：

2.将指令读入指令列表

**public boolean** readInstructionsFile(String fileName);

3.执行函数

**public void** execute();

4.分析指令函数

需要分析指令在指令列表的索引

**public void** execute\_word(**int** index)

5.指令函数

**private void** INT(**int** t,**int** a)

**private void** LIT(**int** t,**int** a)

**private void** LOD(**int** t,**int** a)

**private void** STO(**int** t,**int** a)

**private void** CAL(**int** t,**int** a)

**private void** JMP(**int** t, **int** a)

**private void** JPC(**int** t, **int** a)

**private void** ADD(**int** t, **int** a)

**private void** SUB(**int** t, **int** a)

**private void** MUL(**int** t, **int** a)

**private void** DIV(**int** t, **int** a)

**private void** RED(**int** t, **int** a)

**private void** WRT(**int** t, **int** a)

**private void** RET(**int** t, **int** a)

3.编译程序

数据结构：

指令的数据结构,同解释执行程序中的指令结构,用于生成和存储中间指令，并提供输出方法。

**public class SymbolItem {  
 public SymbolItem(SymbolType type) {  
 this.type = type;  
 this.name = "";  
 this.val = 0;  
 this.level = 0;  
 this.adr = 0;  
 this.size = 0;  
 this.returnType = 0;  
 }  
 public String toString(){  
 return type.toString()+"\t"+name+"\t"+adr+"\t"+returnType;  
 }  
 public enum SymbolType{  
 intSym,  
 functionSym  
 }  
 private String name;  
 private SymbolType type;  
 private int val;  
 private int level;  
 private int adr;  
 private int size;  
 private int returnType;  
 public int getReturnType() {  
 return returnType;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public SymbolType getType() {  
 return type;  
 }  
 public void setType(SymbolType type) {  
 this.type = type;  
 }  
 public void setLevel(int level) {  
 this.level = level;  
 }  
 public int getAdr() {  
 return adr;  
 }  
 public void setAdr(int adr) {  
 this.adr = adr;  
 }  
}**

变量信息类，用来存储变量或函数的一些信息，包括层差，地址，值，名字，函数还有变量个数等信息。

辅助类：

**public class** MyError {  
 **public static boolean** *errFlag*=**false**;  
 **public static void** ShowErrMsg(**int** errCode,String errMsg){  
 *errFlag*=**true**;  
 **switch** (errCode){  
 **case** 1:{*showMessage*(ErrorCode.***ArithmeticException***,errMsg);**break**;}  
 **case** 2:{*showMessage*(ErrorCode.***InputParamErrException***,errMsg);**break**;}  
 **case** 3:{*showMessage*(ErrorCode.***VariableException***,errMsg);**break**;}  
 **case** 4:{*showMessage*(ErrorCode.***FunctionException***,errMsg);**break**;}  
 **default**:{*showMessage*(ErrorCode.***VariableException***,errMsg);**break**;}  
 }  
 }  
 **private static void** showMessage(ErrorCode errCode, String errMsg){  
 System.***err***.println(**"Error-"**+errCode.toString()+**": "**+errMsg);  
 *errFlag*=**true**;  
 }  
}

错误信息类，在编译过程中，除了C0检测到的一些语法错误，可能会发生一些其他的错误，例如重复定义变量等，我们在错误信息类中列举了常见的错误信息，当发生错误时可以通知用户。

主类（未编译的JavaCC文件，编译后会自动生成一些词法和语法的辅助类）：

Java代码部分：

主函数：

public static void main(String args[]);

新建变量函数：

Token：单词信息，Type：空间类型（0-全局空间，1-局部空间）

**public static void** createSymbol(Token t,**int** type);

新建函数函数：

Token：单词信息，Type：返回值类型（0-void，1-int）

**public static void** createFunction(Token t,**int** type);

开辟变量空间函数：

Type：空间类型（0-全局空间，1-局部空间）

**public static void** createSymbolSpace(**int** type);

生成存取变量指令函数：

Token：单词信息，Type：操作类型（0-取操作LOD，1-存操作STO）

**public static void** getVariableByName(Token t, **int** type);

生成调用函数指令函数：

**public static void** getFunctionByName(Token t);

调用主函数函数：

**public static void** callMainFunction();

输出目标代码函数：

**public static boolean** outPut**MiddleCodeList**();

输出全局变量和函数函数：

**public static void** outPutPartList();

写入中间指令至文本文件:

**public static void writeMiddleCodeList()**

解释执行函数：

**public static void** execute();

JavaCC代码部分：

编译程序入口：

void start()

变量定义：

void VariableDefinition(int Type);

自定义函数定义：

void FunctionDefinition();

主函数定义：

void MainFunction();

分程序定义：

void PartProgram();

语句序列定义：

void WordList();

语句定义：

void Word();

条件语句定义：

void If\_Word();

循环语句定义：

void For\_Word();

赋值语句定义:

void Equals\_Word();

返回语句定义：

void Return\_Word();

读语句定义：

void Scanf\_Word();

写语句定义：

void Printf\_Word();

表达式定义：

void Expression();

项定义：

void Term();

因子定义：

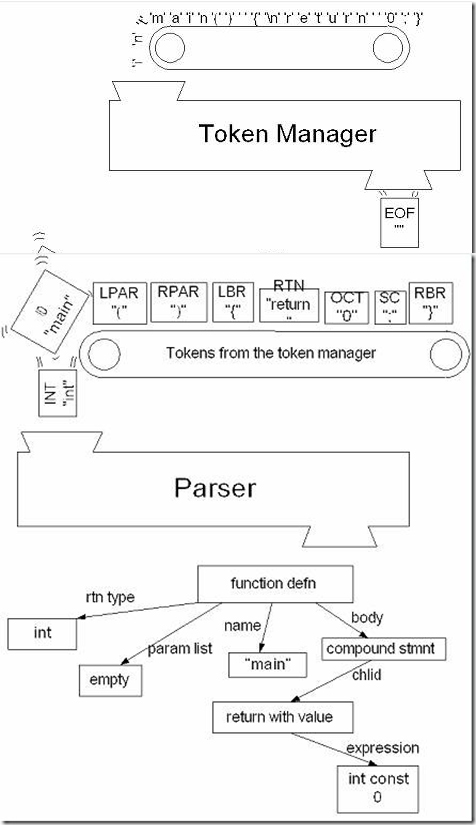
void Factor();

自定义函数定义：

void DefinitionFunction();

JavaCC的主要用法为将语法对应的正规式用JavaCC语法表达，并在合适的时机进行中间指令的生成，所有的语法错误将会以异常的形式抛出，所以其实该模块的每一个函数对应语法定义中的一条规则。

二、JavaCC介绍



其中，TokenManager是词法分析器，他将输入的源程序的字符根据用户定义的词法规则，转化为一个个Token，并投入Parser（语法分析器），在Parser中根据构造的自动机来进行状态的转换。通过JavaCC与BNF结合的形式，可以在合适的时机生成对应的指令，并在出现异常时成功捕获异常并反馈。

编译器定义的词法规则如下：（语法规则详见编译器设计\编译程序\JavaCC代码）

/\*\* 空字符过滤 \*/  
SKIP:  
{ " " | "\t" | "\n" | "\r" }  
/\*\* TOKEN定义 \*/  
TOKEN:  
{ <IF\_TOKEN : "if">  
 |  
 <INT\_TOKEN : "int">  
 |  
 <VOID\_TOKEN : "void">  
 |  
 <ELSE\_TOKEN : "else">  
 |  
 <MAIN\_TOKEN : "main">  
 |  
 <WHILE\_TOKEN : "while">  
 |  
 <SCANF\_TOKEN : "scanf">  
 |  
 <PRINTF\_TOKEN : "printf">  
 |  
 <RETURN\_TOKEN : "return">  
 |  
 <ID\_TOKEN : ["a"-"z","A"-"Z","\_"](["a"-"z","A"-"Z","\_","0"-"9"])\*>  
 |  
 <NUM\_TOKEN : (["0"-"9"])+>  
 |  
 <COMMA\_TOKEN: ",">  
 |  
 <SEMICOLON\_TOKEN: ";">  
 |  
 <PLUS\_TOKEN: "+">  
 |  
 <MINUS\_TOKEN: "-">  
 |  
 <TIMES\_TOKEN: "\*">  
 |  
 <DIVIDE\_TOKEN: "/">  
 |  
 <EQUAL\_TOKEN: "=">  
 |  
 <LBRACE\_TOKEN: "{">  
 |  
 <RBRACE\_TOKEN: "}">  
 |  
 <LPAREN\_TOKEN: "(">  
 |  
 <RPAREN\_TOKEN: ")"> }

三、测试设计

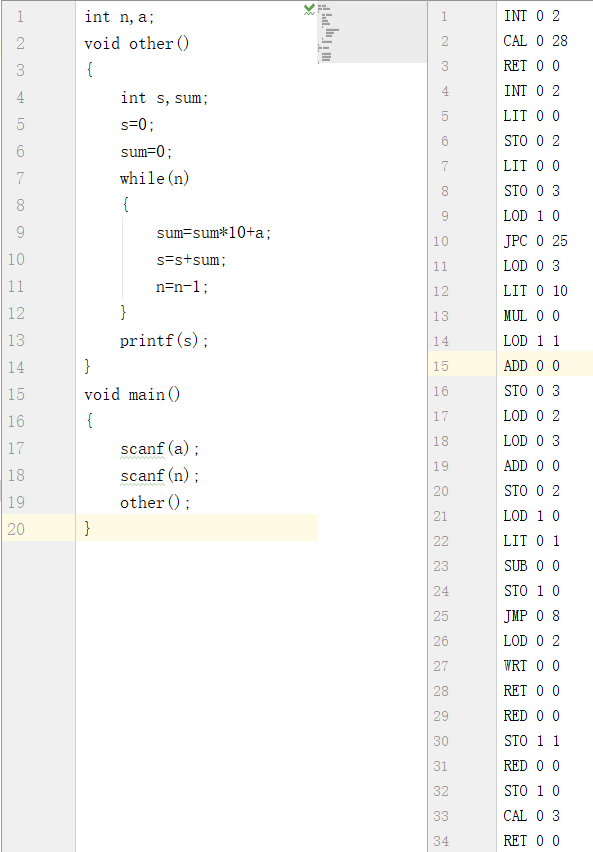
再设计测试测试程序时，我们从不同的角度选取了四个具有代表性的程序，分别从不同的角度对编译程序加以测试。

一、A+AA+AAA

该题目是一道经典的ACM试题，主要测试循环。

用户输入a(0-9)和n(最后一项A的个数)m程序给出A+AA+AAA的结果。

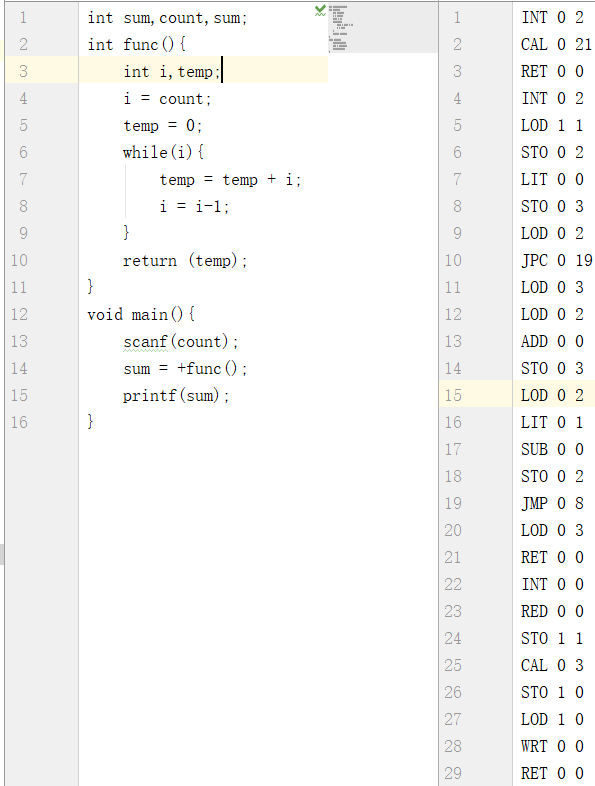
源代码及中间指令：



二、连加

用户输入A，程序自动计算从1到A之间数字的和并且给出结果

源代码及中间指令：

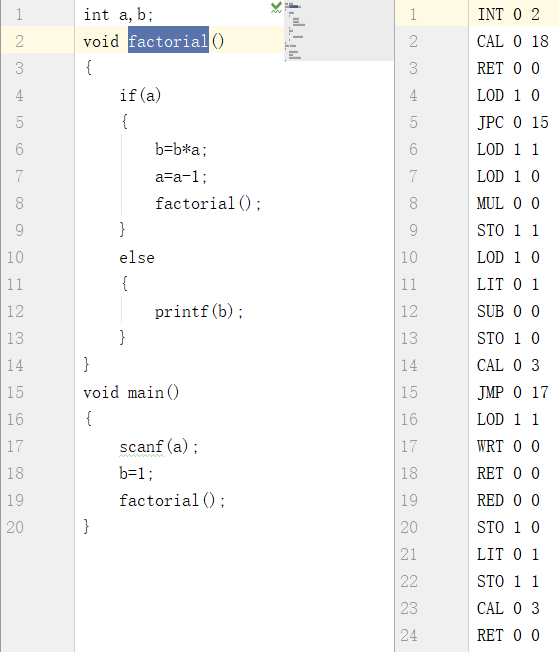


三、阶乘

利用递归计算数字的阶乘，主要检验函数的调用部分。

用户输入一个数字A，程序自动计算该数字的阶乘

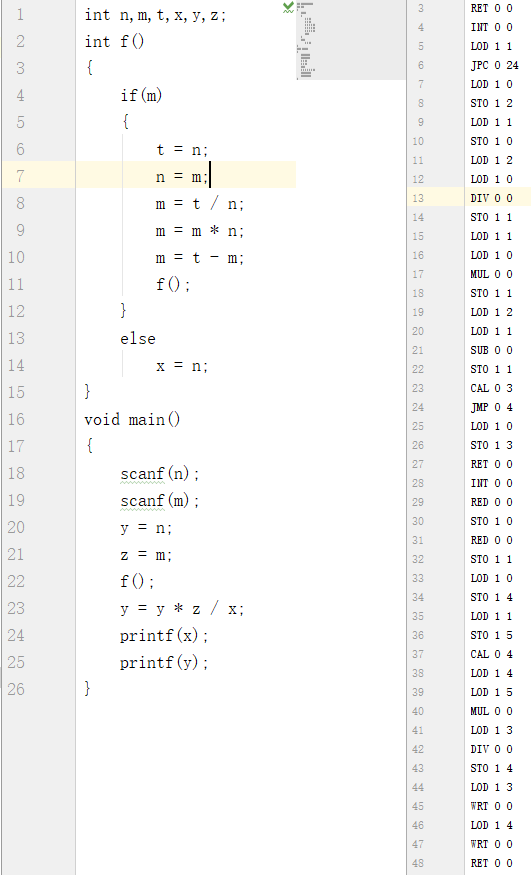
程序及中间指令：



四、求两个数的最大公因数和最小公倍数

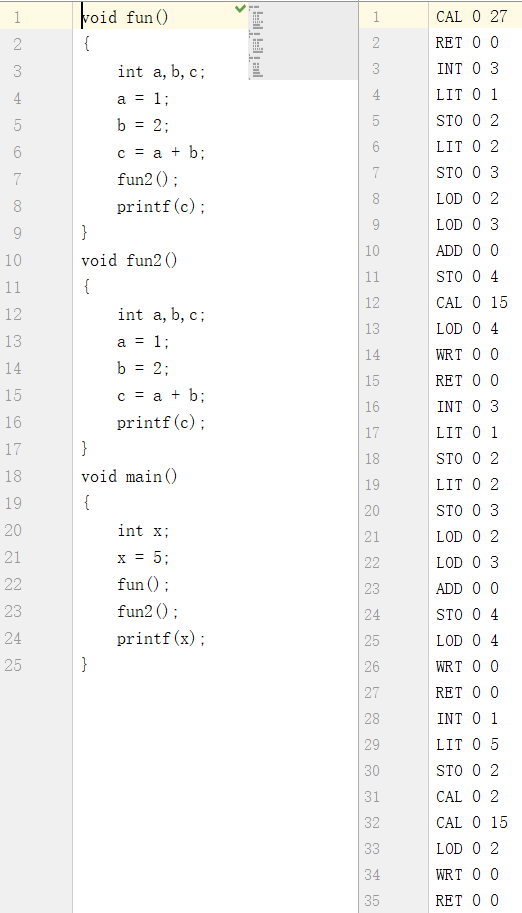
对程序各个模块的综合考察

用户输入两个数字，程序计算并给出两个数的最大公因数和最小公倍数



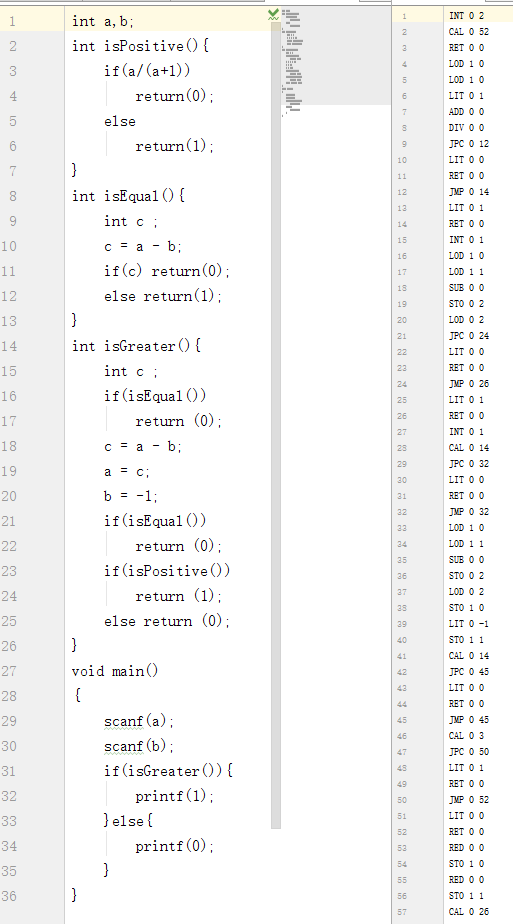
五、测试超前调用

该测试程序没有输出，仅仅用来测试函数的超前调用。



六、大于

由于C0语言没有逻辑运算，此程序代替基本逻辑运算（小于、等于、大于等于、小于等于）用来检测逻辑运算功能。



四、异常检测

编译器可以实现编译过程中的异常及错误信息。

1.变量重复定义

输入：int a,a;

错误提示：Error-VariableException: 变量[a]已经定义！

2.变量未定义

输入：int n,a;

……

m=0;

错误提示：Error-VariableException: 变量[m]不存在

3.超前调用未定义函数

输入：int n,a;  
 void main()  
 {  
 scanf(a);  
 scanf(n);  
 other();  
 }

错误提示：Error-FunctionException: 函数other未定义！

4.返回值为int类型的函数没有return语句

输入：int factorial()  
 {

}

错误提示：Error-FunctionException: 返回值类型为int的函数必须有返回值!

5. 返回值为int类型的函数返回空

输入：int factorial()  
 {

return ；

}

错误提示：Error-FunctionException: 返回值类型为int的函数必须有返回值!

6. 返回值为void类型的函数返回int

输入：void factorial()  
 {

return 1；

}

错误提示：Error-FunctionException: 返回值为void类型的函数只能返回null

注意：当返回值类型为void的函数没有return语句时，将不会报错。

7.缺少分号

输入：int a,b

void factorial()

错误提示：Compiler.ParseException: Encountered " "void" "void "" at line

2, column 1.

Was expecting one of:

"," ...

";" ...

8.函数定义缺少返回值类型

输入：int a,b;

factorial()

错误提示：Compiler.ParseException: Encountered " <ID\_TOKEN> "factorial ""

at line 2, column 1.

Was expecting one of:

"," ...

";" ...

编译错误提示还有很多，这里不一一举例，只举一些典型的实例。

解释执行程序在运行时同样可以给出一些错误或异常的信息。

1.除数为0

输入：s=a/0;

错误提示：错误信息:ArithmeticException(算数异常) 错误指令所在行数为:第41行

除此以外还有一些其他的异常的捕捉，这里便不一一介绍。

五、附录

附测试用例源代码及中间指令：

一、A+AA+AAA

源代码：

int n,a;

void other()

{

int s,sum;

s=0;

sum=0;

while(n)

{

sum=sum\*10+a;

s=s+sum;

n=n-1;

}

printf(s);

}

void main()

{

scanf(a);

scanf(n);

other();

}

中间指令：

INT 0 2

CAL 0 28

RET 0 0

INT 0 2

LIT 0 0

STO 0 2

LIT 0 0

STO 0 3

LOD 1 0

JPC 0 25

LOD 0 3

LIT 0 10

MUL 0 0

LOD 1 1

ADD 0 0

STO 0 3

LOD 0 2

LOD 0 3

ADD 0 0

STO 0 2

LOD 1 0

LIT 0 1

SUB 0 0

STO 1 0

JMP 0 8

LOD 0 2

WRT 0 0

RET 0 0

RED 0 0

STO 1 1

RED 0 0

STO 1 0

CAL 0 3

RET 0 0

二、连加

源代码：

int sum,count,sum;  
int func(){  
 int i,temp;  
 i = count;  
 temp = 0;  
 while(i){  
 temp = temp + i;  
 i = i-1;  
 }  
 return (temp);  
}  
void main(){  
 scanf(count);  
 sum = +func();  
 printf(sum);  
}

中间代码：

INT 0 2  
CAL 0 21  
RET 0 0  
INT 0 2  
LOD 1 1  
STO 0 2  
LIT 0 0  
STO 0 3  
LOD 0 2  
JPC 0 19  
LOD 0 3  
LOD 0 2  
ADD 0 0  
STO 0 3  
LOD 0 2  
LIT 0 1  
SUB 0 0  
STO 0 2  
JMP 0 8  
LOD 0 3  
RET 0 0  
INT 0 0  
RED 0 0  
STO 1 1  
CAL 0 3  
STO 1 0  
LOD 1 0  
WRT 0 0  
RET 0 0

三、阶乘：

源代码：

int a,b;  
void factorial()  
{  
 if(a)  
 {  
 b=b\*a;  
 a=a-1;  
 factorial();  
 }  
 else  
 {  
 printf(b);  
 }  
}  
void main()  
{  
 scanf(a);  
 b=1;  
 factorial();  
}

中间指令：

INT 0 2  
CAL 0 18  
RET 0 0  
LOD 1 0  
JPC 0 15  
LOD 1 1  
LOD 1 0  
MUL 0 0  
STO 1 1  
LOD 1 0  
LIT 0 1  
SUB 0 0  
STO 1 0  
CAL 0 3  
JMP 0 17  
LOD 1 1  
WRT 0 0  
RET 0 0  
RED 0 0  
STO 1 0  
LIT 0 1  
STO 1 1  
CAL 0 3  
RET 0 0

四、最大公因数和最小最小公倍数

源代码：

int n,m,t,x,y,z;  
int f()  
{  
 if(m)  
 {  
 t = n;  
 n = m;  
 m = t / n;  
 m = m \* n;  
 m = t - m;  
 f();  
 }  
 else  
 x = n;  
}  
void main()  
{  
 scanf(n);  
 scanf(m);  
 y = n;  
 z = m;  
 f();  
 y = y \* z / x;  
 printf(x);  
 printf(y);  
}

中间指令：

INT 0 6  
CAL 0 27  
RET 0 0  
INT 0 0  
LOD 1 1  
JPC 0 24  
LOD 1 0  
STO 1 2  
LOD 1 1  
STO 1 0  
LOD 1 2  
LOD 1 0  
DIV 0 0  
STO 1 1  
LOD 1 1  
LOD 1 0  
MUL 0 0  
STO 1 1  
LOD 1 2  
LOD 1 1  
SUB 0 0  
STO 1 1  
CAL 0 3  
JMP 0 4  
LOD 1 0  
STO 1 3  
RET 0 0  
INT 0 0  
RED 0 0  
STO 1 0  
RED 0 0  
STO 1 1  
LOD 1 0  
STO 1 4  
LOD 1 1  
STO 1 5  
CAL 0 4  
LOD 1 4  
LOD 1 5  
MUL 0 0  
LOD 1 3  
DIV 0 0  
STO 1 4  
LOD 1 3  
WRT 0 0  
LOD 1 4  
WRT 0 0  
RET 0 0

五、超前调用

源代码：

void fun()  
{  
 int a,b,c;  
 a = 1;  
 b = 2;  
 c = a + b;  
 fun2();  
 printf(c);  
}  
void fun2()  
{  
 int a,b,c;  
 a = 1;  
 b = 2;  
 c = a + b;  
 printf(c);  
}  
void main()  
{  
 int x;  
 x = 5;  
 fun();  
 fun2();  
 printf(x);  
}

中间指令：

CAL 0 27  
RET 0 0  
INT 0 3  
LIT 0 1  
STO 0 2  
LIT 0 2  
STO 0 3  
LOD 0 2  
LOD 0 3  
ADD 0 0  
STO 0 4  
CAL 0 15  
LOD 0 4  
WRT 0 0  
RET 0 0  
INT 0 3  
LIT 0 1  
STO 0 2  
LIT 0 2  
STO 0 3  
LOD 0 2  
LOD 0 3  
ADD 0 0  
STO 0 4  
LOD 0 4  
WRT 0 0  
RET 0 0  
INT 0 1  
LIT 0 5  
STO 0 2  
CAL 0 2  
CAL 0 15  
LOD 0 2  
WRT 0 0  
RET 0 0

六、大于

源代码：

int a,b;  
int isPositive(){  
 if(a/(a+1))  
 return(0);  
 else  
 return(1);  
}  
int isEqual(){  
 int c ;  
 c = a - b;  
 if(c) return(0);  
 else return(1);  
}  
int isGreater(){  
 int c ;  
 if(isEqual())  
 return (0);  
 c = a - b;  
 a = c;  
 b = -1;  
 if(isEqual())  
 return (0);  
 if(isPositive())  
 return (1);  
 else return (0);  
}  
void main()  
{  
 scanf(a);  
 scanf(b);  
 if(isGreater()){  
 printf(1);  
 }else{  
 printf(0);  
 }  
}

中间指令：

INT 0 2  
CAL 0 52  
RET 0 0  
LOD 1 0  
LOD 1 0  
LIT 0 1  
ADD 0 0  
DIV 0 0  
JPC 0 12  
LIT 0 0  
RET 0 0  
JMP 0 14  
LIT 0 1  
RET 0 0  
INT 0 1  
LOD 1 0  
LOD 1 1  
SUB 0 0  
STO 0 2  
LOD 0 2  
JPC 0 24  
LIT 0 0  
RET 0 0  
JMP 0 26  
LIT 0 1  
RET 0 0  
INT 0 1  
CAL 0 14  
JPC 0 32  
LIT 0 0  
RET 0 0  
JMP 0 32  
LOD 1 0  
LOD 1 1  
SUB 0 0  
STO 0 2  
LOD 0 2  
STO 1 0  
LIT 0 -1  
STO 1 1  
CAL 0 14  
JPC 0 45  
LIT 0 0  
RET 0 0  
JMP 0 45  
CAL 0 3  
JPC 0 50  
LIT 0 1  
RET 0 0  
JMP 0 52  
LIT 0 0  
RET 0 0  
RED 0 0  
STO 1 0  
RED 0 0  
STO 1 1  
CAL 0 26  
JPC 0 61  
LIT 0 1  
WRT 0 0  
JMP 0 63  
LIT 0 0  
WRT 0 0  
RET 0 0