**编译器实践**

**----课程报告**

**1191002008 冯路阳**

# 总体设计

## 编译器目的

目标是实现一个WHI语言的编译器，将高级语言翻译成简单栈式抽象机可识别并执行的中间代码。拟用c语言实现。实现顺序：首先将实现编译器的词法分析部分，在进行调试后依次进行语法分析，代码生成等模块的设计。

## WHI的文法

说明：无符号整型常量（N）、变量（V）、表达式（E）、语句（S）、语句列表（L）、程序（P）

1. P ::= [ X ‘;’ ] L

说明：程序P可以是一个语句列表或者是一个变量定义成分+一个语句列表

1. X ::= ‘var’ V { ‘,’ V }

说明：定义语句X可以是定义一个变量V，也可以是定义多个变量V

1. L ::= S [ ‘;’ L ]

说明：语句列表可以接在语句后，亦可以没有语句列表，仅有语句

1. S ::= ‘skip’ | V ‘:=’ E | ‘read’ ‘(’ V ‘)’ | ‘write’ ‘(’ E ‘)’ | ‘if’ E ‘then’ L ‘else’ L ‘fi’ | ‘while’ E ‘do’ L ‘od’

说明：语句可以是跳过（skip），是变量赋值，是读入（read），是写入（write），是if条件句，是while循环语句

1. E ::= T [ TS ]

说明：表达式E形式多样，我们只讨论他的FIRST集

FIRST(E) = { FIRST(N), FIRST(V), ‘(’, ‘~’}

1. TS ::= ‘+’ T [ TS ] | ‘-’ T [ TS ]

FIRST(TS) = {‘+’, ‘-’}

1. T ::= F [ FS ]

FIRST(T) = { FIRST(N), FIRST(V), ‘(’, ‘~’}

1. FS ::= ‘\*’ F [ FS ] | ‘/’ F [ FS ]

FIRST(FS) = {‘\*’, ’/’}

1. F ::= D [ D1 ]

FIRST(F) = { FIRST(N), FIRST(V), ‘(’, ‘~’}

1. D1 ::= ‘=’ D | ‘<’ D

FIRST(D1) = {‘=’, ‘<’}

1. D ::= N | V | ‘(’ E ‘)’ | ‘~’ E

FIRST(D) = { FIRST(N), FIRST(V), ‘(’, ‘~’}

1. N ::= G { G }

说明：整数N由数字组成

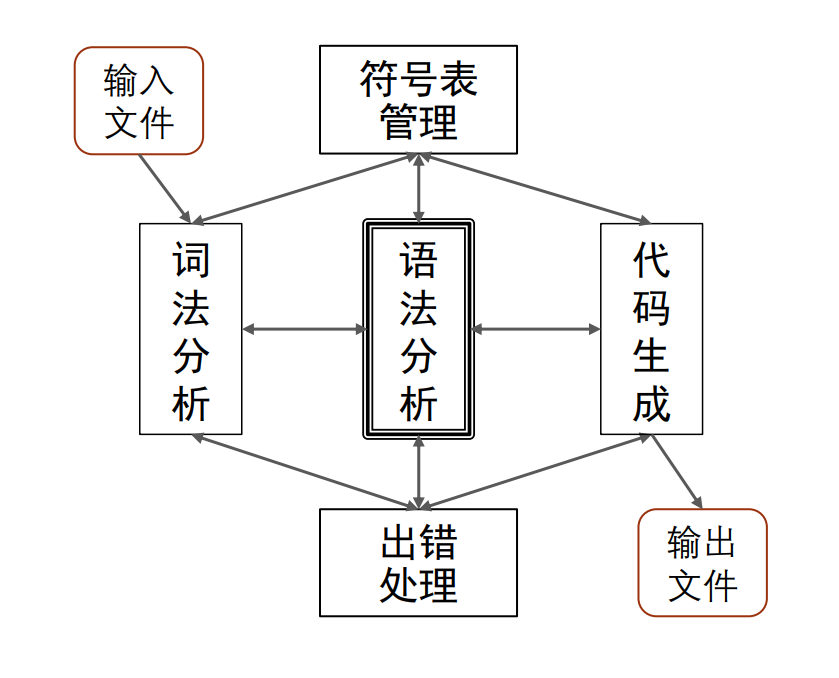
1. V ::= C { C | G }

说明：变量由a-z的字符开头，后可接多个数字或者a-z

1. G ::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’
2. C ::= ‘a’ | ‘b’ | … | ‘y’ | ‘z’

## 项目实现的工作流程

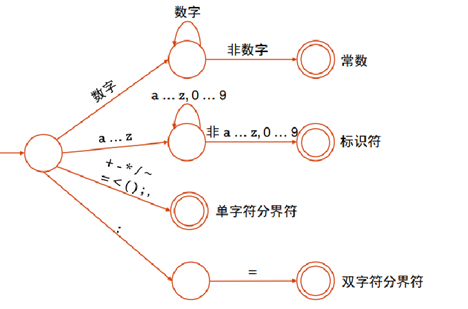
WHI语言🡪单词分析🡪语法分析🡪中间代码生成🡪交给简单栈式抽象机编译成可执行文件



图表 1 WHI语言编译器实现流程

# 词法分析

## 词法分析器设计



图表 2 词法分析器流程图

## 词法分析器实现

### 数据与结构设计

#### 宏定义

#define MAX\_LEN (20) //单词长度限制

#define dummy\_len (10) //保留字每个长度

#define dummy\_num (11) //保留字个数

#### 枚举类

enum tok\_kind{

    TK\_ID, //这是个变量名

TK\_NUM, //这是个常量

TK\_SEMI, //这是个边界符

TK\_ASSIN, //这是个赋值符号

    TK\_DUMMY, //这是个保留字符

TK\_EOF, //这是个文件结束标志

TK\_ERR //这是个错误字符

};

enum lex\_err{

    err\_char, //错误的单词

err\_len //错误的单词长度

};

enum dummy\_kind{

    d\_while = 1, d\_if, d\_fi, d\_do, d\_od,

    d\_write, d\_then, d\_var, d\_skip, d\_read, d\_else

};

#### 结构体

typedef struct{

    enum tok\_kind kind; //这个单词的种类

    char str[MAX\_LEN]; //这个单词的字符串

    int n\_row; //这个单词在的行号

    int n\_col; //这个单词在的列号

    enum lex\_err err; //这个单词的错误种类

}Token;

#### 全局变量

int col = 0, row = 1; //col：列 row：行

### 函数设计

#### int overflowed(int n)

功能：判断单词是否超过定长

#### void tk\_print(Token\* tk)

功能：打印处理好的单词，显示相应信息以方便调试

#### int dummy(Token\* tk)

功能：判断一个单词是否为保留关键字

#### Token\* get\_token(FILE \*fp)

功能：将读入的文件进行分词

#### Int mian()

功能：主函数

## 模块调试

void tk\_print(Token \*tk){

switch(tk->kind){

case TK\_ID: printf("标识符：%s\n", tk->str); break;

case TK\_NUM: printf("常数：%s\n", tk->str); break;

case TK\_SEMI: printf("分届符：%s\n", tk->str); break;

case TK\_ASSIN: printf("赋值符号：%s\n", tk->str); break;

case TK\_DUMMY: printf("保留字：%s\n", tk->str); break;

}

return;

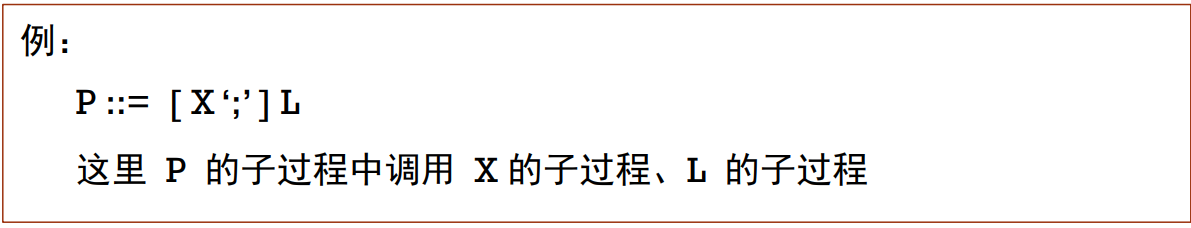
}

解释：用switch语法按需显示分词结果，其中tk\_kind = TK\_EOF是文件结束标识。tk\_kind = TK\_ERR的情况是错误情况，在此switch中不显示。

# 语法分析

## 语法分析器设计

利用递归下降分析器的思想来设计，形如下图模式



图表 3 非终结符P的子过程调用

利用词法分析器设计的无二义性文法，对文法中每一个非终结符（如 P、X、L、S、E等）设置子过程，以分析其所对应的源程序段在非终结符U的子过程中,调用U产生式右边各个非终结符所对应的子过程。

应该另外根据逻辑关系设计一个文件专门处理语法分析任务，不能和词法分析混在一起。递归下降分析器需要借助已有读入单词来进行按序逐一分析。在进入每个非终结符的子过程时，采取无条件进入，进入后再移动指针并判断是否语法正确。在进入可选择或可重复的子程序时，若进入某一子程序后判断为跳出此子程序，则需要跳出之前将单词指针也一并回退一格。避免指针错乱。此语法分析器检测到错误即停止，没有自修正继续执行其他语法分析的功能。检测到错误后也会一并报告出问题的单词位置和原因。

## 语法分析器实现

### 数据与结构设计

全局变量：

int len = 0, i = 0;

char\*\* ID\_bag;

int bag\_len = 0;

Token\*\* tk\_p;

**说明：**

**len**: 表示当前程序所包含单词个数

**i**: 当前单词符号表里指针所处的位置

**ID\_bag**：存储所有已定义变量

**ID\_place**: 给每个变量分配存储空间

**bag\_len**: 已定义变量个数

**tk\_p**：二维Token类型指针，用来存放所有单词的特征

### 主要 函数设计

#### void move\_i(); 功能：实现二维单词数组的指针前进动作

#### void back\_i(); 功能：功能：实现二维单词数组的指针后退动作

#### void err\_loc\_print 功能：实现错误位置打印

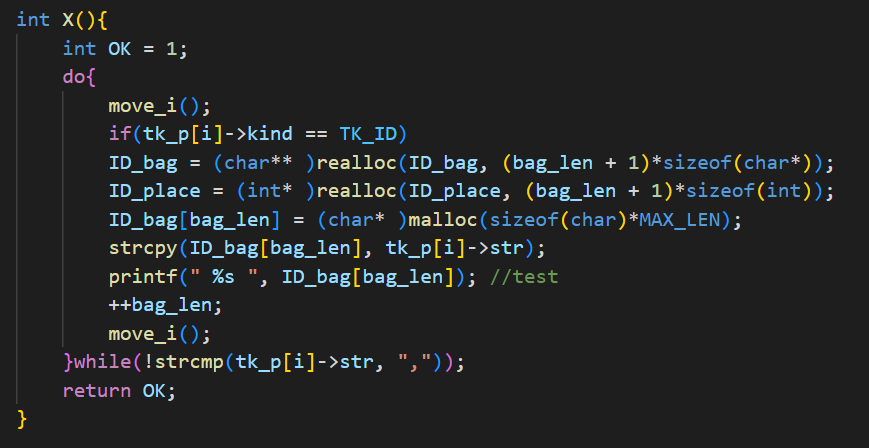
#### got \_if(); 功能：遇到保留字if需要调用此子程序

#### int got\_while(); 功能：遇到保留字while需要调用此子程序

#### int P();; 功能：执行非终结符P的子程序

### 关键函数实现细节

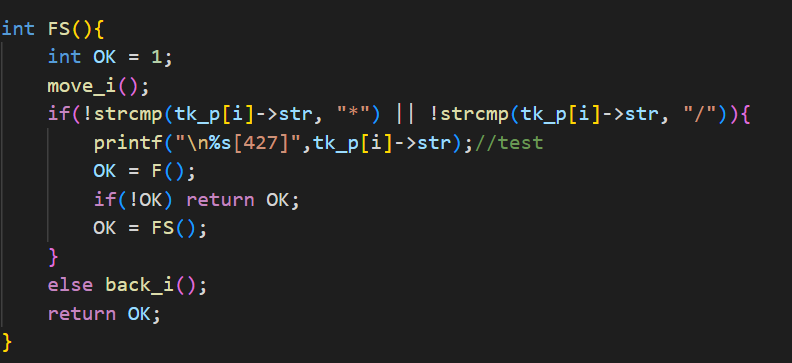
#### 单词存储与符号表设计



图表 4 符号表实现代码

说明：利用ID\_bag作为二维指向字符串的数组指针存储所有变量名，并实采取动态分配内存的方式进行最节能的存储，ID\_place作为存储每个已经定义过的变量的实值与ID\_bag的下标相同，亦方便进行变量访存。

#### 子程序设计



图表 5 其中一个非终结符的递归下降方法实现

说明：根据WHI的文法。详见[总体设计—WHI的文法](#_WHI的文法)，可以知道为解决二义性进行了非终结符的扩展，并用BNF范式表达文法。可以看到有很多可选项或者可重复项，针对这些，我采取了每个非终结符右部的可选项或可重复项无条件进入的方法，让每个子程序内部移动单词指针并判断是否需要执行可选项或可重复项。例如FS->F[FS]，若发现下个单词既不是‘\*‘也不是’/‘则back\_i();即退回单词指针。所有的非终结符子程序均采用判断是否满足当前非终结符的FIRST集中值的方式来进行语法分析。

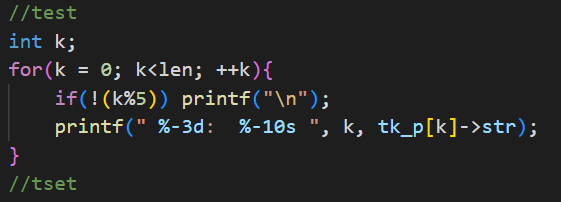
#### OK的利用

说明：只要OK一次为0程序就会结束，可以很快停止在出错的位置，不会过多执行剩余代码，最高效报错。

### 模块调试

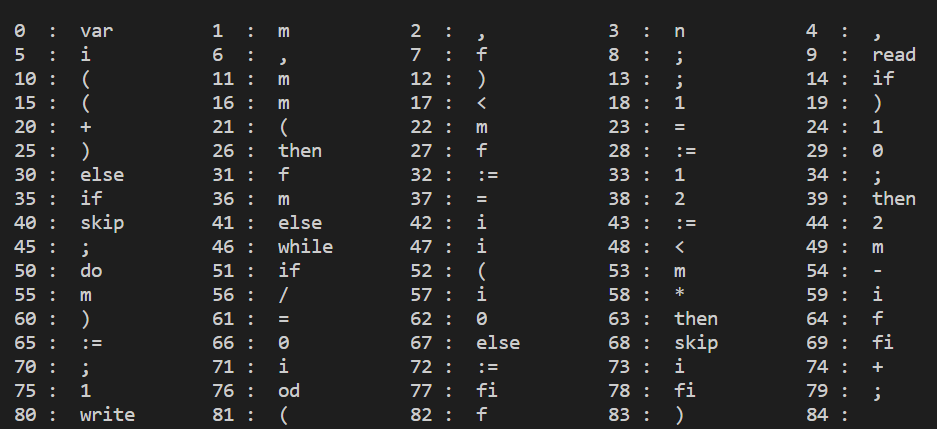
可以说，语法分析阶段属于编程容易，做严谨难，在起初没有思考好实现方法后，编写的程序单词指针乱飞，一会就不知道分析到哪一个单词，又是那一个单词出了问题。所以其实这个模块的编写中调试是最耗费我心思的。我采用了如下的调试策略。

#### 打印所有单词



图表 6 实现代码（在后期制作过程中已删除）

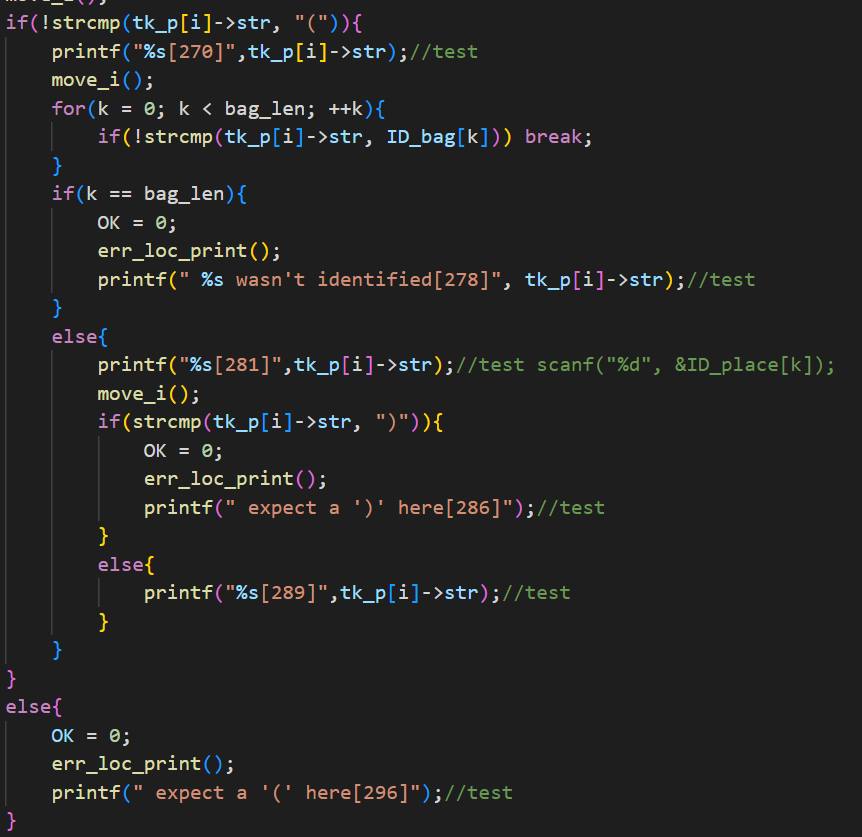
通过上述程序得到如下输出，此处采用鸡兔同笼问题的程序，得到了84个单词，包括一个EOF文件终结符。通过在结尾增加当前文件指针的输出可以快速知晓当前单词的错误位置。



图表 7 打印效果（最终程序不会出现）

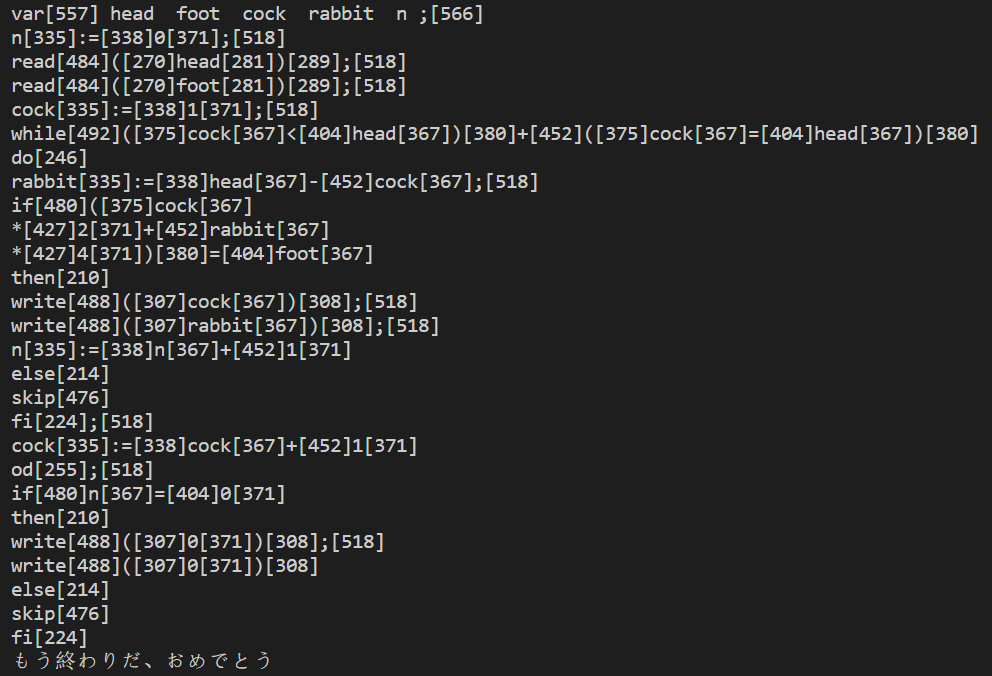
#### 显示每条打印语句所在行号

起初的程序写完，我完全不知道哪行打印出的单词是出自哪个语句，这也是自己程序的高耦合性导致的不良结果。但是我为每个printf加了如下数值辅助我知晓每一个单词的语法分析情况。



图表 8 语法分析阶段带调试的部分代码展示（后期取消了行号）

比如左括号右括号，在read，write，非终结符D的子程序中军可能出现判断，在输出行同时加上[xxx]即可知道哪一行告诉我缺少左括号或右括号。打印的鸡兔同笼问题如下如所示，虽然可读性降低，但对于已经熟知程序内容的我来说，语法分析出了问题我能很快定位出问题的程序调用模块。



图表 9 语法分析阶段打印效果（与最终不同）

#### Vscode断点调试功能

由于人工模拟一遍程序很费时间，可以考虑通过上文初吻题的行进行断点设置，然后通过单步执行随电脑一起模拟文法分析过程，从而可以让我很快理解出问题的病源所在。

# 中间代码生成器

## 生成器设计

由于目标是生成输入给简单栈式抽象机的代码，所以需要熟知WHI语言的指令代码，如下图



图表 10 栈式抽象机包含的指令

根据指令规律，在递归下降分析器中逐一分析生成代码的位置。

### 表达式代码生成

由于add操作是对栈顶和次栈顶元素进行加和计算，所以add的指令生成在TS ::= ‘+’T@add[TS]的“@add”处。同理，sub也在此对应位置，mul生成在FS::=’\*’F@mul[FS]的“@mul”处。同理，div也在此对应位置。equ, lod, lth, lit分别出现在如下位置：D1::=’=’D@equ | ‘<’D@lth, D ::=N@lit | V@lookup@lod | ‘(’E’)’ | ‘~’E@lit@equ。其中@lookup是指需要创建一个查询变量V是否已定义的函数。其中，lit和lod需要一个参数，参数的值即为变量或常量对应的整型值。

### 变量说的代码生成

由于构造符号表的工作我在语法分析时就已经完成，故没有特别的符号表建立过程。

### 赋值语句的代码生成

赋值语句涉及到sto指令生成，具体为：S🡪V: E@lookup@sto 中“@sto”的位置。其中sto涉及到参数，参数值为通过lookup找到变量的存储位置。

### 输入输出的代码生成

输入输出涉及内容简单，在S🡪read(V)@lookup@red中“@red”处生成指令，其中参数为lookup找到的变量地址。While相对不同点在于不需要@lookup。

### 分支语句的代码生成。

当来到分支语句后就涉及到一个关键的问题——跳转，跳转分为无条件跳转和有条件跳转，即jmp和jpc。且两条指令后面涉及的参数都是随着语句继续执行才能知道的，会需要“回填”的操作。这里我才用链表技术来完成每个跳转指令参数的回填。具体操作如下：

S↑lst🡪if E @brf↑la then

L1↑lst1 @br↑lb

else @patch↓la

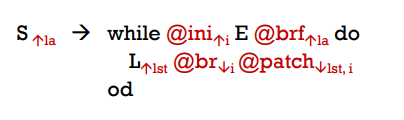
L2↑lst2

if @endif↓lb,lst,lst2↑lst

其中带↑↓分别对应需要生成新的回填链表，还是回填指定回填链表。用jmp还是jpc根据逻辑关系判断。整个子程序最后也会返回一个回填链表。

### 循环语句的代码生成

与分支语句大同小异，故不多做赘述。



### 语句列表的代码生成

L↑lst🡪S↑lst

L↑lst🡪S↑lst0；@patch↓lst0 L1↑lst

整个非终结符L的子程序结束后也会返回一个回填链表。

### 程序的代码生成



在P的子过程中，需要在@patch后面新生成一个nop指令，以防止前面的跳转指令越界。

### 特别说明

除了分支和循环指令中涉及大量回填操作，其他的子程序也涉及了或多或少的的返回链表操作，但均是返回空链表，在本编译器中设计的是链表元素数值为-1时即为空。

## 生成代码器实现

### 全局变量

Stc\*\* stc\_p;

int stc\_len = 0;

说明：

stc\_p：是指向指令的二维指针。

stc\_len: 记录指令当前的长短

### 设计数据结构

**指令结构体**

typedef struct{

char kind[4];

int val;

}Stc;

**回填链表**

typedef struct Link{

int val;

struct Link\* next;

}Link;

注：当val为-1时代表为空链表

### 主要函数

由于在实现阶段，感觉和语法分析阶段可以和中间代码生成一起做，故非终结符的子程序并没有结构上的改变，但是加入了指令生成和回填链表操作。

#### void alloc(char\* kind, int val)

说明：生成指令的函数，kind表示指令类型，val表示参数，val为-1即表示该指令不需要参数。

#### Link\* link\_init()

说明：给新定义的回填链表开辟存储空间和做初始化。

### 特色算法

由于在lod指令时，需要知道相应常数的值，而我在保存常量时采取的是保存字符串，所以需要如下字符串转整型的算法：

char \*s;

int j, len = strlen(tk\_p[i]->str);

s = (char\*)malloc(len\*sizeof(char));

strcpy(s, tk\_p[i]->str);

for(j = 0; j<len; ++j){

a = a + (s[j]-'0')\*pow(10, (len-j-1));

}

按位依次计算数值。

## 代码生成器调试

在这一步的调试，我主要使用了vscode自带断点调试工具。后期结合多个WHI语言程序案例来验证代码生成器。

t1.txt：鸡兔同笼问题

t2.txt：二元一次方程

t3.txt：猜数

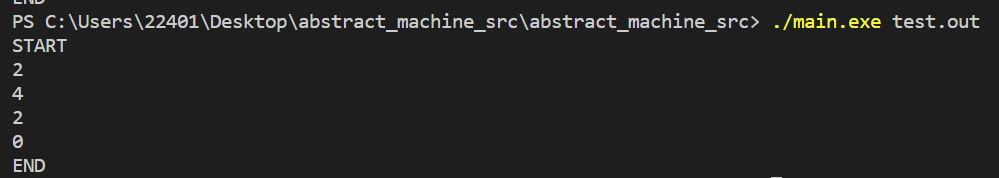
t4.txt：判断素数

t5.txt：最大公约数和最小公倍数

# 编译器实现结果展示

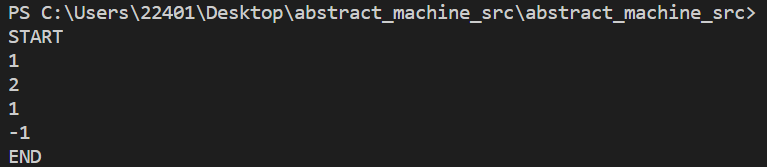
由于编译器生成的打印结果过长，故此处不做展示，指利用生成指令放到简单栈式抽象机中进行检验和展示。

## 鸡兔同笼问题



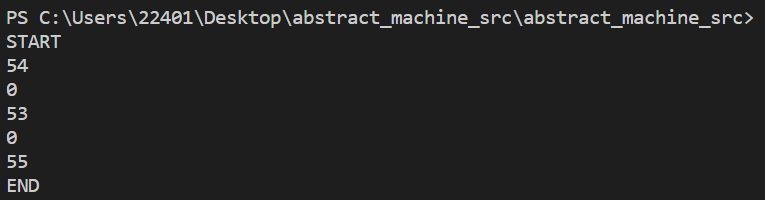
2个头，四只脚，说明有俩鸡，没有兔子。

## 二元一次方程



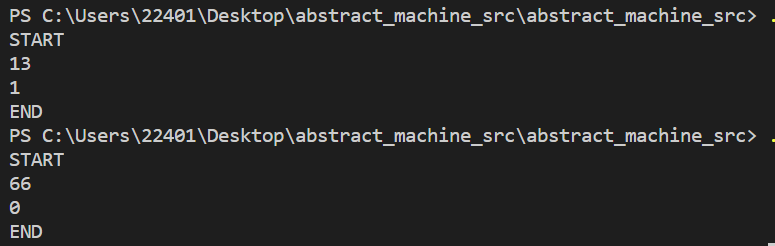
分别输入二元一次方程的三个系数a,b,c，最后结果是ax2+bx+c=0的解，只考虑实数集，可能零解，可能是两个相同的解（默认输出其中一个），可能是两个不相同的解。

## 猜数



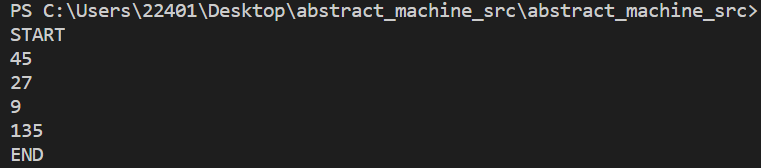
目标数字为55，当猜错了不会跳出循环，一直显示0表示猜错了。显示对就退出。

## 判断素数



若为素数，显示1，若非素数，就显示0。

## 求最大公约数和小公倍数



计算45和27的最大公约数和最小公倍数，很明显45，27的最小公倍数是9，最大公约数是135。

# 编译器反思

在进行从一开始的单词的存储到后来指令的存储，我都采用的是动态分配内存空间的方法，初衷是节省资源，但经过老师知道和自我思考，每次动态分配存储空间可能导致在大型项目中的时间资源上的浪费，所以在指令生成部分，未来可以考虑一次生成多个指令，同一时间分配多个存储空间。例如int x后面接的lit，sto指令，可以根据x个数，直接分配x\*2\*sizeof(stc)的指令存储空间。

另外，一个好的项目一定是高内聚低耦合的。我的这个项目在文法分析和中间代码生成阶段，是一起的，如果一个模块发生了问题，很可能导致两个功能的受损。可以考虑更好地分开两个功能。在语法分析部分，我的语法错误是有一个报一个，不会说有一个统一的变量去记录错误，最后一起报。只能同一时间处理一个语法错误。可以说不够智能，未来可以考虑更好的规划错误输出以及适当的错误自修正能力。

# 致谢

首先非常感谢李老师地悉心教导，从一开始请老师给我的词法分析做检查开始，就非常细心地解决了我的诸多疑惑，在代码生成阶段也给我提了不少有用的建议，让我能更好的完善自己的小项目。分文件写就是老师建议地，本来也想懒省事写在一个文件里，但真的努力分好后软件清爽了许多。

另外，还要感谢我的室友白同学，起初选编译器的课程就是他推荐的，我俩一起学之后，从第一次认真使vscode开始就提供了给我有干货的参考文档，后期还在讨论设计方法时探寻到更优雅的思路。最后的编译器检验也给我分享了自己设计的WHI语言程序做参考。

能顺利地完成此次编译器设计少不了老师和同学的帮忙。学到很多！感觉对理论课的帮助也很大。