题目：C0编译器的设计与实现（10周）

C0语言的语法结构定义如下：

<程序>->[<变量定义部分>] {<自定义函数定义部分>} <主函数>

<变量定义部分>-> int id {, id};

<自定义函数定义部分>-> ( int id | void id) '(' ')' <分程序>

<主函数>->void main'(' ')' <分程序>

<分程序>->'{' [<变量定义部分>] <语句序列> '}'

<语句序列>-><语句> {<语句>}

<语句>-> <条件语句>｜<循环语句> | '{'<语句序列>'}' | <自定义函数调用语句>

| <赋值语句> | <返回语句> | <读语句> | <写语句> | ;

<条件语句>->if '('<表达式>')' <语句> [else <语句>]

<循环语句>->while '(' <表达式>')' <语句>

<自定义函数调用语句>-><自定义函数调用>;

<赋值语句>->id = <表达式>;

<返回语句>->return ['(' <表达式> ')'] ;

<读语句>->scanf '(' id ')';

<写语句>->printf '(' [ <表达式>] ')';

<表达式>-> [+｜-] <项> { (+｜-) <项>}

<项> -> <因子>｛(\*｜/) <因子>｝

<因子> -> id｜'(' <表达式>')' | num | <自定义函数调用>

<自定义函数调用>->id '(' ')'

其中，id代表标识符，num代表整数，其含义及构成方式与C语言相一致；C0源程序中的变量需先定义后使用，其作用域与生存期与C语言相一致；自定义函数可超前使用（调用在前，定义在后）。

根据上面给定的C0文法及其说明和下列定义的假想栈式指令系统，按递归下降分析法设计并实现该C0语言的编译器，生成栈式目标代码；编写栈式指令系统的解释执行程序，输出目标代码的解释执行结果。

要求所实现程序可带运行选项，以便对输出内容进行选择。所实现程序能分阶段依次输出下列内容：C0源程序及其编译结果（生成第一个函数目标代码前的符号表内容—通过选项选择、目标代码，出错信息）；解释执行目标代码，动态显示运行栈的内容—通过选项选择，输出执行结果。

假想的栈式指令系统表

|  |  |
| --- | --- |
| LIT 0 a | 将常数值取到栈顶，a为常数值 |
| LOD t a | 将变量值取到栈顶，a为相对地址，t为层数 |
| STO t a | 将栈顶内容送入某变量单元中，a为相对地址，t为层数 |
| CAL 0 a | 调用函数，a为函数地址 |
| INT 0 a | 在运行栈中为被调用的过程开辟a个单元的数据区 |
| JMP 0 a | 无条件跳转至a地址 |
| JPC 0 a | 条件跳转，当栈顶值为0，则跳转至a地址，否则顺序执行 |
| ADD 0 0 | 次栈顶与栈顶相加，退两个栈元素，结果值进栈 |
| SUB 0 0 | 次栈顶减去栈顶，退两个栈元素，结果值进栈 |
| MUL 0 0 | 次栈顶乘以栈顶，退两个栈元素，结果值进栈 |
| DIV 0 0 | 次栈顶除以栈顶，退两个栈元素，结果值进栈 |
| RED 0 0 | 从命令行读入一个输入置于栈顶 |
| WRT 0 0 | 栈顶值输出至屏幕并换行 |
| RET 0 0 | 函数调用结束后,返回调用点并退栈 |

程序=Pg

变量定义部分=Var

自定义函数定义部分=Fun

主函数=Main

分程序=Other

语句序列=SenList

语句=Sen

条件语句=If

循环语句=While

自定义函数调用语句= Personal

赋值语句=Value

返回语句=Return

读语句=Read

写语句=Write

表达式= Exp

自定义函数调用=PerFun

项= Term

因子= Factor

Pg -> Var Fun Main

Var -> int id {, id} ;

Fun -> ( int id | void id) ( ) Other

Main ->Void main ( ) Other

Other ->'{' [Var] SenList '}'

SenList -> Sen { Sen }

Sen -> If｜While | '{' SenList '}' | Personal

| Value | Return | Read | Write | ;

If ->if '(' Exp ')' Sen [else Sen]

While->while '(' Exp ')' Sen

Personal -> PerFun;

Value ->id = Exp;

Return ->return ['(' Exp ')'] ;

Read ->scanf '(' id ')';

Write ->printf '(' [ Exp] ')';

Exp -> [+｜-] Term { (+｜-) Term }

Term -> <因子>｛(\*｜/) Factor｝

Factor -> id｜'(' Exp ')' | num | PerFun

PerFun ->id '(' ')'