sw-compiler

软件设计说明书

作者:XXX

版本:XXX

日期:XX/XX/XX

1．介绍

本编译器语法的实现基于 swift 语言 sw 子集，并针对 swift 源语法进行了一系列的语法扩展，同时设计了编译器的图形界面，便于使用

程序整体架构：

编译器程序：有两个模式，通过 compiler.h 中的__DEBUG__开关调整

非__DEBUG__模式：接收需要编译的 sw 代码文件名称的命令行参数，运行获得编译错误信息及中间代码的 json 文件，不进行解释交互工作

__DEBUG__模式：直接在命令行中输入需要编译的 sw 代码文件名，编译后调用解释器工具直接在命令行解释交互

GUI 暨解释器程序：提供具有关键字高亮功能的 sw 代码编辑器，调用编译器程序生成中间文件，然后通过中间文件输出错误信息并解释交互，有两个运行模式

直接运行模式：直接运行中间代码到程序结束

单步运行模式：可以单步运行、从头再次运行，切换到直接运行，该模式下提供当前运行中间代码位置追踪及显示当前运行栈功能

2．编译器系统结构

2.1 编译器

2.1.1 **sw 语法综述**

sw 语法结构为单层定义声明方式，不允许函数声明内部的嵌套声明，仅使用整形数据进行运算，支持常量使用，函数可以使用传值类型参数以及返回值使用，在语句的定义上支持 while/for range(闭区间及半开区间)/repeat while 循环、if 语句、调用函数语句、赋值语句、输入输入语句、返回语句，可以使用形如/* comment */的块注释和形如// comment 的行注释

符号表方面支持较为完整的运算符使用，包括基本四则运算（加、减、乘、除）、求余、逻辑运算(与、或、非)、位运算（位与、位或、位取反、异或、移位）、自增、自减、比较运算符（等于、不等于、大于、大于等于、小于、小于等于）以及这些运算符对应的自运算赋值符（如+=），相应运算符优先级如下：

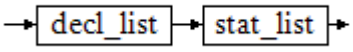
- 单目运算优于双目运算。如正负号。
- 先乘除（模），后加减。
- 先算术运算，后移位运算，最后位运算。请特别注意：1 << 3 + 2 & 7 等价于 (1 << (3 + 2))&7

- 逻辑运算最后计算

运算符类型	运算符	结合方向
表达式运算	()	从左到右
一元运算符	! ~ ++ -- + -	从右到左
二元运算符	* / % + - >> << < > <= >= == !=	从左到右
位运算符	& ^ | && ||	从左到右
赋值运算符	= += -= *= /= %= >>= <<= &= ^= |=	从右到左

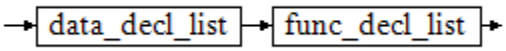
2.1.2 sw 语言语法图（红色标注为扩增或修改部分）

Problem 程序的入口：包括声明部分以及执行语句部分



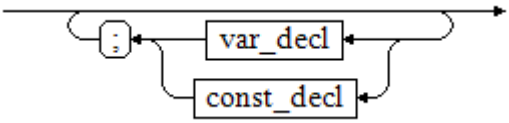
（Ⅰ）声明部分：

decl_list：先进行数据声明再进行函数声明

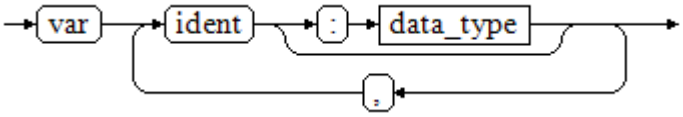


数据声明部分 data_decl_list：包括变量声明和常量声明

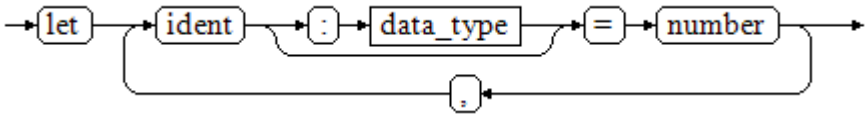
扩展点：增加常量声明



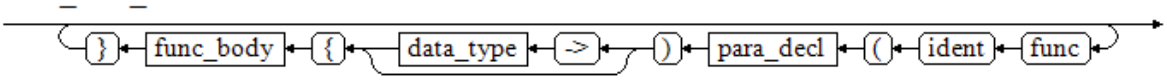
var_decl：变量声明，支持单行声明多变量，并声明变量类型（如 var a:int, b, c:bool;）



const_decl：常量声明，声明同时确定常量的值与类型（如 let a:int=1, b=2;）

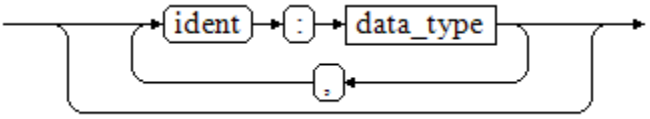


函数声明部分 func_decl_list：在函数名声明后的括号中，可以进行参数声明，参数声明包括对参数数据类型的声明（如 a:int, b:bool），单个函数最多支持 15 个参数，之后可以声明函数是否有返回值及返回类型，如 func a() -> int 表示返回值类型为 int，删去-> int 则表示无返回值，此后进入函数体声明

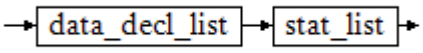


扩展点：参数声明、返回值类型声明

para_decl：参数声明，形如 a:int, b:bool

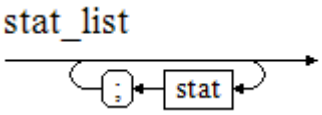


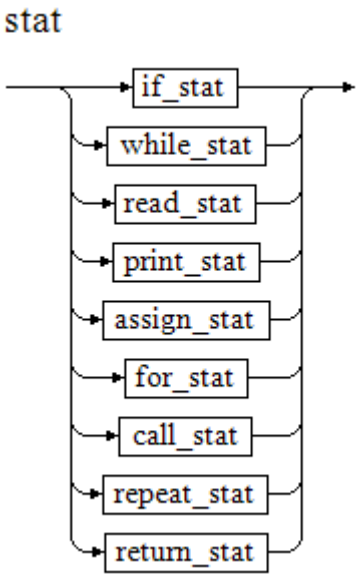
func_body：函数体部分，包括数据声明以及执行语句部分



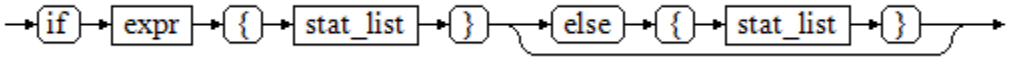
（Ⅱ）执行语句部分：stat_list 为单条语句的循环，每条语句包括语句体 stat 和一个分隔符“；”，stat 内为支持的语句类型，包括赋值语句、输入语句、输出语句、函数调用 call 语句、if 语句、while 循环语句、for range 循环语句、repeat while 循环语句、return 语句

扩展点：赋值语句内扩增自增自减式写法以及多种运算符的自赋值运算、for range 支持..<表示的右半开区间、repeat while 循环、return 功能

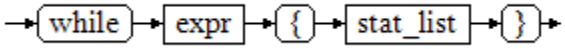




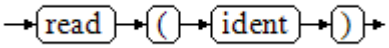
if_stat：if 条件语句，可以有或没有 else 部分



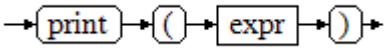
while_stat：while 循环语句，符合表达式 expr 非负条件则继续循环



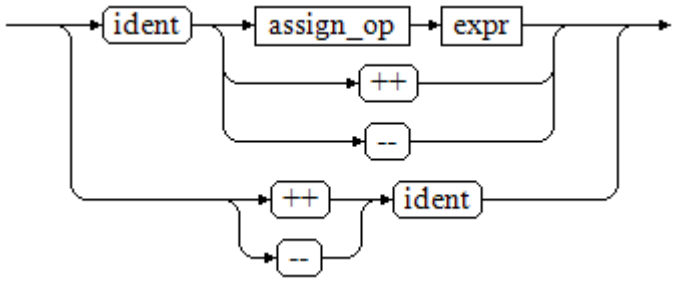
read_stat：输入语句，一次为一个变量输入值



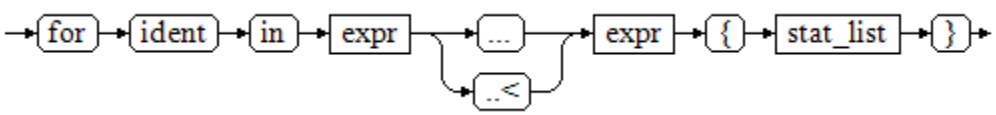
print_stat：输出语句，一次输出一个表达式 expr 的结果



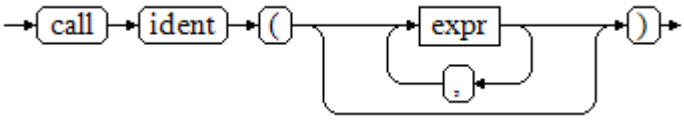
assign_stat：赋值语句，除基本的赋值号赋值外，支持各种运算符对应的的自运算赋值，以及以语句形式表示的自增自减操作



for_stat：for 循环语句，支持...表示的闭区间 range 循环以及..<表示的右半开区间循环

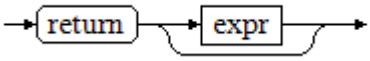


call_stat：调用函数 call 语句，直接调用函数，有参数声明的函数要对对应的参数以一个 expr 传值，不获取返回值



repeat_stat：repeat while 循环语句，先执行循环再判断，当 while 中的条件符合时继续执行循环体

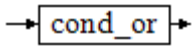
return_stat：return 返回语句，可以以一个 expr 返回对应值或者空值返回，取决于相关函数声明，函数声明返回为空则可以无 return 语句且使用 return 语句时无需返回值，否则函数体中至少有一个 return 语句且都需要返回值



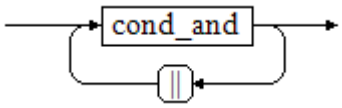
（Ⅲ）表达式部分：expr 是表达式部分的入口，由此按照运算符优先级逐层生成各个运算符语法树，相较于原本的 sw 语法，新增了逻辑运算、位运算、移位运算、自增自减运算以及求余运算

扩展点：逻辑运算、位运算、移位运算、自增自减运算、求余

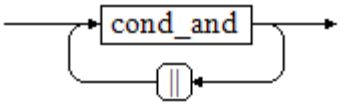
expr



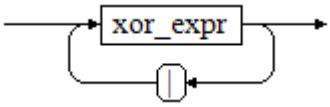
cond_or：逻辑或运算



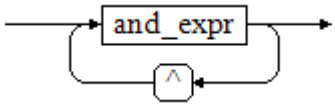
cond_and：逻辑与运算



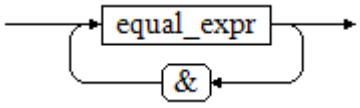
or_expr：位或运算



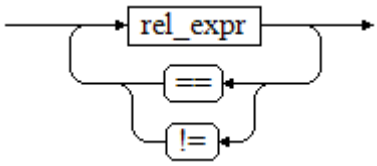
xor_expr：异或运算



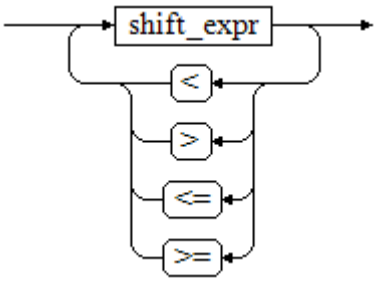
and_expr：位与运算



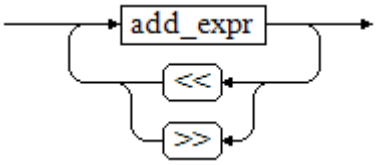
equal_expr：等式判断



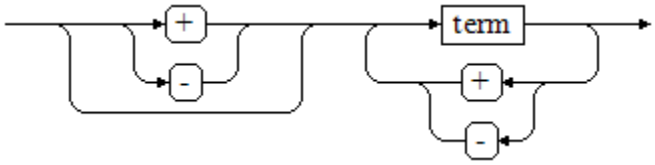
rel_expr：不等式判断



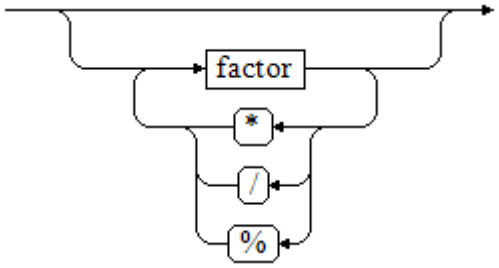
shift_expr：移位运算



add_expr：加减运算



term：乘除，以及求余运算



factor：运算因子部分

运算因子包括：

（a）变量

（b）常量

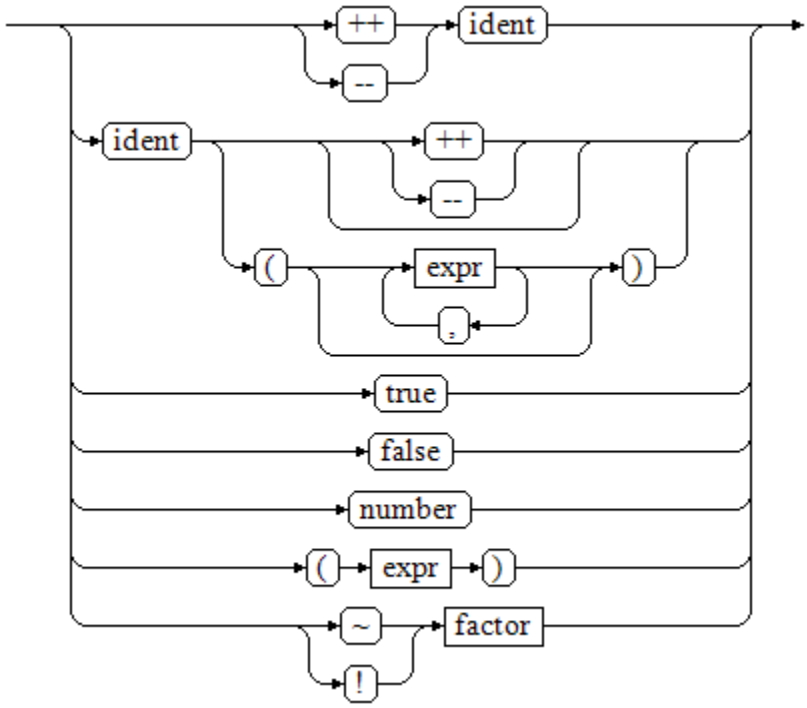
（c）符合长度要求的数字

（d）可以传参的函数

（e）“true”“false”关键字，即代表 1、0

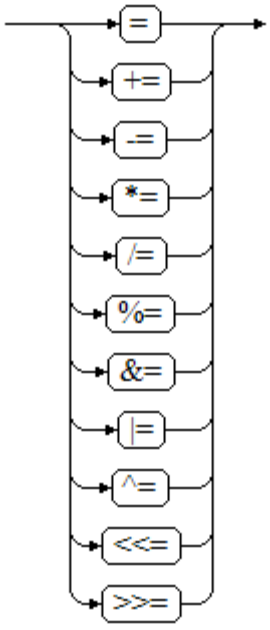
（f）用括号分隔的表达式

同时在这层可以对变量进行自增自减操作(不允许递归使用)，或进行取反和取否的单目运算

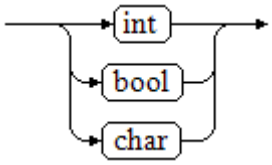


（IV）枚举符号部分：将一些同时使用的同类型符号进行了归并，在此列出

assign_op：赋值号以及各个运算符对应的赋值号



data_type：使用的数据类型



2.1.2 判断是否符合两条限制规则（全部符合）

2.1.3 符号表结构说明

Name：标识符名称

Kind：标识符类型

Datatype：标识符数据类型，对于函数该字段为返回值类型

Val：常量标识符的数值、函数声明的参数数量

Level：标识符声明层次，常量类型不使用

Adr：地址，常量类型不使用

Size：需要分配的数据空间，函数类型使用

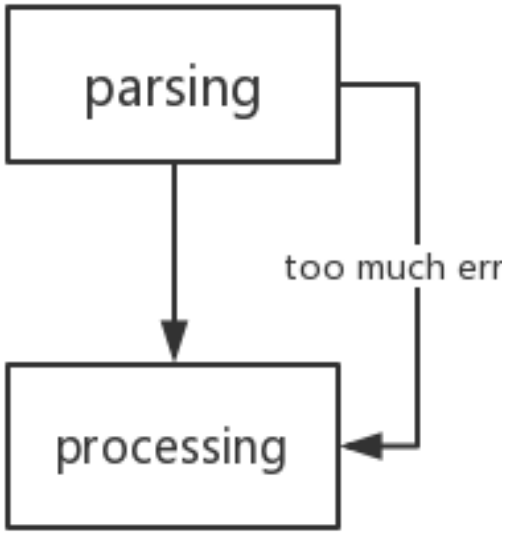
2.1.4 编译程序总体结构

编译程序总体分为如下两个部分

Parsing：即解析部分，这一部分完成词法、语法及语义分析工作

Processing：即处理结果部分，这一部分将 parsing 分析的结果生成相关的中间文件并进行收尾工作

如下图所示，从 parsing 到 processing 有两个入口，一个是在 parsing 运行完毕后直接进入 processing 的正常入口，另一个是当已发现的错误数量达到了可同时缓存错误数量上限时从 error 处理程序直接进入 processing 并强制跳出 parsing 的入口



2.1.5 语法出错表定义

出错编号	错误信息
4	declaration lacks identity
5	lack primary token ';'
6	a wrong statement start token or function declaration
7	a wrong follow token after declaration
8	a wrong follow token after statement
9	parsing incomplete
10	a function can receive no more than 15 parameters
11	a no-declaration identity
12	a variable type need for lhs
13	lack of become token for assignment statement
14	invalid call statement without function identity
15	a declaration of constant type must initialize with a value
16	a non-variable type instance cannot be auto-increment or auto-decrement

Project Name/Model No:XXXXX

17	a datatype token is needed after ':' for a data declaration
18	a datatype token is needed after '->' for a function declaration to return
19	statement ending with a wrong follow symbol
20	a relation operator is lost
21	invalid conversation for a function identity
22	parameter declaration need ':' for datatype declaration
23	number of passing parameters cannot match declaration
24	a wrong start token for factor
25	non-variable identity cannot be read by input
26	a parameter without name
30	too long for a number
31	number is out-of-range
33	invalid statement for lacking a token ')'
34	invalid statement for lacking a token '('
36	function needs '()' token
37	statement needs start after '{' token
38	statement needs ending with '}' token
39	a for statement needs token 'in'
40	a for statement needs a left range
41	a for statement needs a right range
42	a for statement needs a range symbol '...'
43	no variable to use for auto-increment statement
44	no variable to use for auto-decrement statement
45	a while symbol is need for repeat statement
46	a wrong follow symbol for logic or judgement
47	a wrong follow symbol for logic and judgement
48	the function needs a return statement
49	a value must return for the function

60	program is too long --end
61	Displacement address is too big --end

2.2 虚拟机

2.2.1 虚拟机指令结构

每条虚拟机指令包括三个字段：

f：虚拟机指令名称

l：引用层与声明层的层次差或某些相近功能指令的区分码

a：某些指令可传入的参数或运算指令的区分码

2.2.2 虚拟机解释器结构说明

基本结构：

p：指令指针

c：当前指令指针

b：指令基址

t：栈顶指针

i：当前运行指令

s：运行栈

reg：临时寄存器（从编号 1 开始共 15 个可用）

Python 版本 GUI 解释器 utils.py **说明**

封装类 Instruction：中间代码类，每个实例代表一条中间代码

封装类 Interpret：解释器类，除了基本结构，还将中间代码集合保存在内部属性 code 中，另外新增输出缓冲区属性 buf，功能方法说明如下：

judge：每一步运行结束用于判断解释器是否运行结束，结束返回 False，否则返回 True

send：运行到输入指令时使用，将用户输入作为参数 data 传入并置于运行栈顶部

recv：运行到输出指令时使用，将输出值传到 buf 里再通过该方法获取

showStack：返回当前运行栈的情况

sg_step：单步运行函数

2.2.3 虚拟机指令码表

注：空白位置可为任意值

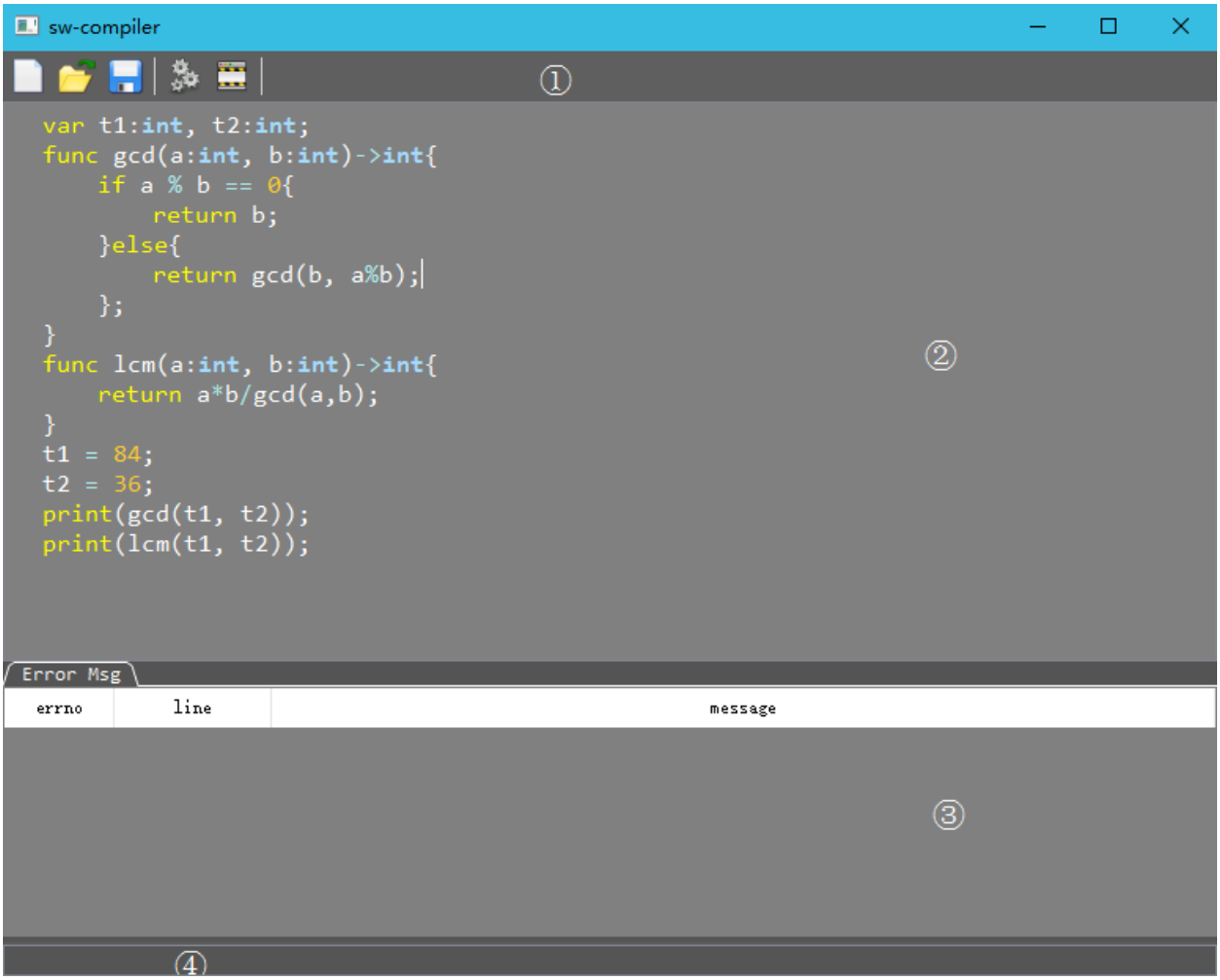
指令名称	层次差或区分码	细分码或参数	指令作用
lit		Num	将常数 num 移入运行栈
opr		0	函数执行完毕后返回
		1	栈顶值取反
		2	次栈顶加上栈顶值
		3	次栈顶减去栈顶值
		4	次栈顶乘以栈顶值
	0	5	次栈顶除以栈顶值
	1		次栈顶求余栈顶值
		6	栈顶值位与 1（判断奇偶）
	0	7	判断栈顶次栈顶是否相等
	1		判断栈顶次栈顶是否不等
	0	8	判断次栈顶是否小于次栈顶
	1		判断次栈顶是否小于等于次栈顶
	0	9	判断次栈顶是否大于次栈顶
	1		判断次栈顶是否小于等于次栈顶
		10	次栈顶位与栈顶值
		11	次栈顶位或栈顶值
		12	栈顶值位取反
		13	次栈顶异或栈顶值
		14	次栈顶逻辑与栈顶值
		15	次栈顶逻辑或栈顶值
	0	16	次栈顶左移位栈顶值位数
	1		次栈顶右移位栈顶值位数
lod	l	a	将层差为 l，偏移量为 a 的变量数据取至栈顶处
sto	l	a	将栈顶数据存至层差为 l，偏移量为 a 的变量处
cal	l	a	调用层差为 l，起始代码地址为 a 的函数
ini		a	在栈顶初始化大小为 a 的数据区
jmp		a	无条件跳转至编号为 a 的中间代码
jeq		a	若栈顶值为真跳转至编号为 a 的中间代码
jne		a	若栈顶值为假跳转至编号为 a 的中间代码

in		0	将用户输入读至栈顶
		a(1-15)	将编号 a 的寄存器中值读入栈顶
out		0	将栈顶值输出给用户
		a(1-15)	将栈顶值送入编号 a 的寄存器

3．GUI 说明及功能介绍

GUI 共有代码编辑和程序运行两个界面，其中程序运行是在代码编辑界面执行编译命令编译成功后才出现，且运行界面未关闭时编译功能将不可用，因此每次只能打开一个运行界面

代码编辑界面：



功能区说明：

①工具栏，从左至右依次是：

Ⅰ新建代码文件(ctrl+N)

Ⅱ打开代码文件(ctrl+O)

Ⅲ保存代码文件(ctrl+S)：打开已有文件后保存会覆盖原文件，新建文件保存会弹出对话框选择保存位置

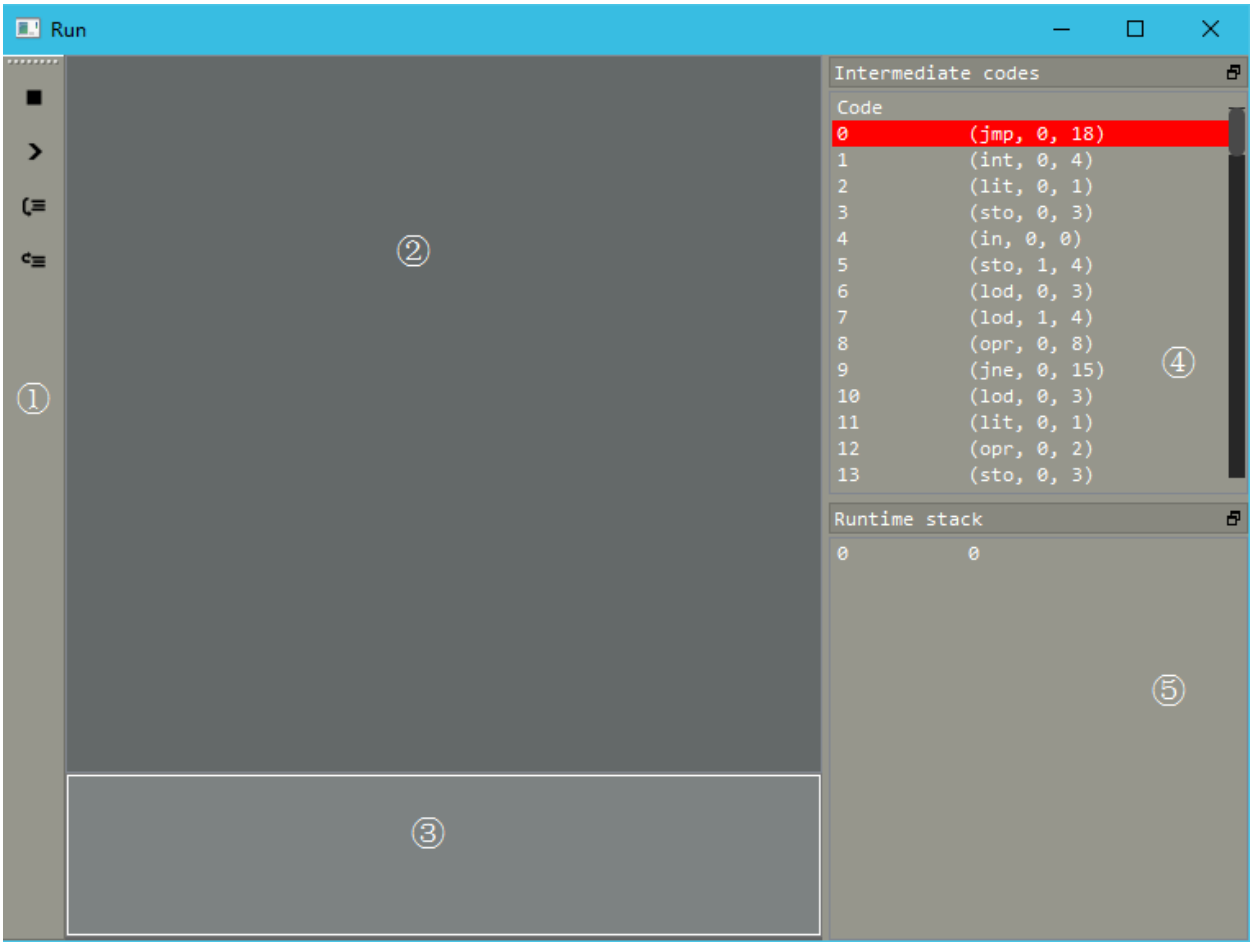
Ⅳ直接运行模式(ctrl+B)

Ⅵ单步运行模式(ctrl+alt+B)

②代码编辑器：支持语法高亮功能

③编译结果信息输出：编译成功会显示解析成功的信息，否则会按行输出错误信息

程序运行界面：



功能区说明：

①工具栏，从上至下依次是：

Ⅰ停止运行(shift+F8)

Ⅱ单步运行(F7)

Ⅲ切换至直接运行到底模式(F4)

IV跳回至开头重新运行(ctrl+F7)

其中ⅡⅢIV仅在单步运行模式下可以使用

②结果输出显示区：程序运行输出的的结果会在这里显示，同时需要输入时会显示一行>>>输入提示标识

③输入区：运行到输入指令时会激活这一区域，输入并按下回车会完成输入

④中间代码显示：列出全部中间代码，并对当前正在运行的中间代码红色高亮标注(仅单步运行模式可用)

⑤运行栈显示：会显示每一步运行时的运行栈内部情况（仅单步运行模式可用）

4．全局数据结构、常量和变量

宏定义工具：

__DEBUG__：测试模式开关，宏定义该名称后可以开启命令行测试模式，即输入代码文件名后编译并在命令行模式交互，undef 该名称可关闭该模式改为以参数输入编译文件名

枚举类型：

SYMBOL：关键字符号表

OBJECT：标识符类型表

DATATYPE：数据类型表

FCT：虚拟机指令表

宏定义常量：

N_SYM：文法关键字符号数目

N_RW：文法保留字数目

N_FCT：虚拟机指令类型数目

LEN_NUM：数字容许最大长度

LEN_ID：标识符容许最大长度

LEN_L：行读取字符缓冲区容量

MAX_ERR：编译程序可缓存最大错误数，超出此数终止编译

MAX_CX：虚拟机最大数目

SIZE_TB：符号表容量

SIZE_STACK：运行栈容量

SIZE_REG：临时寄存器数目

BOUND_ADR：可用地址上界

字符类型常量：

ERR_TP：错误类型表

word：保留字字典

mnemonic：虚拟机指令集字典

布尔型常量：

declbegsys：声明部分起始符号表

statbegsys：执行语句起始符号表

facbegsys：因子部分起始符号表

全局变量：

table：符号表

code：虚拟机暨中间代码缓存

ch：getch 工具当前读取出的字符

sym：getsym 词法分析工具当前符号

id：当前读到的标识符名称

num：当前读到的数字

cc：getch 处理一行的当前位置

ll：getch 处理一行的末尾位置

line：getch 使用的行缓冲区

a：当前读到的符号名称缓冲区

line_num：当前读到的行号

err_num：已发现的编译语法错误数目

fend_tag：读取到代码文件末尾标记

rtn_num：当前处理的函数体使用到的 return 语句数目

rtn_type：当前处理的函数声明的返回类型

5．函数原型

函数原型	problem(int lev, int tx, bool* fsys)
参数描述	*Lev:程序主体层次（0） tx：符号表起始位置 fsys：follow 符号集合*

函数描述	程序主体语法分析程序
返回值	Void

函数原型	data_decl(int lev, int* ptx, int* pdx)
参数描述	*Lev：当前程序层次 ptx：当前处理符号表尾指针 pdx：变量分配地址指针*
函数描述	数据声明段语法分析程序
返回值	Void

函数原型	var_decl(int lev, int* ptx, int* pdx)
参数描述	*Lev：当前程序层次 ptx：当前处理符号表尾指针 pdx：变量分配地址指针*
函数描述	变量声明段语法分析程序
返回值	Void

函数原型	Let_decl(int lev, int* ptx, int* pdx)
参数描述	*Lev：当前程序层次 ptx：当前处理符号表尾指针 pdx：变量分配地址指针*
函数描述	常量声明段语法分析程序
返回值	Void

函数原型	data_decl(int lev, int* ptx, int* pdx)
参数描述	*Lev：当前程序层次 ptx：当前处理符号表尾指针 pdx：变量分配地址指针*
函数描述	数据声明段语法分析程序
返回值	Void

函数原型	func_decl_body(int lev, int tx, bool* fsys)
参数描述	*Lev：当前程序层次 tx：当前处理符号表起始位置 fsys：follow 符号集合*
函数描述	函数声明主体段语法分析程序
返回值	Void

函数原型	parameter(bool* fsys, int* ptx, int lev, int p)

参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次 p：当前传参函数的生命参数数目*
函数描述	参数调用段语法分析程序
返回值	Void

函数原型	statement(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	语句执行段语法分析程序
返回值	Void

函数原型	expression(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	表达式段语法分析程序，实际为宏定义，等价于 cond_or
返回值	Void

函数原型	cond_or(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	条件或表达式语法分析程序
返回值	Void

函数原型	cond_and(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	条件与表达式语法分析程序
返回值	Void

函数原型	or_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	位或表达式语法分析程序
返回值	Void

函数原型	xor_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	异或表达式语法分析程序
返回值	Void

函数原型	and_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	位与表达式语法分析程序
返回值	Void

函数原型	equal_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	等于不等于表达式语法分析程序
返回值	Void

函数原型	rel_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	不等式表达式语法分析程序
返回值	Void

函数原型	shift_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	移位表达式语法分析程序
返回值	Void

函数原型	add_expr(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*

函数描述	加减法表达式语法分析程序
返回值	Void

函数原型	term(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	乘除求余表达式语法分析程序
返回值	Void

函数原型	factor(bool* fsys, int* ptx, int lev)
参数描述	*fsys：follow 符号集合 ptx：当前处理符号表尾指针 Lev：当前程序层次*
函数描述	表达式因子语法分析程序
返回值	Void

函数原型	declaration(OBJECT tp, int* ptx, int lev, int* pdx)
参数描述	*tp：声明标识符类型 ptx：当前处理符号表尾指针 Lev：当前程序层次 pdx：可为标识符分配空间尾指针*
函数描述	标识符声明语法分析程序
返回值	Void

函数原型	enter(OBJECT k, int* ptx, int lev, int* pdx)
参数描述	*tp：声明标识符类型 ptx：当前处理符号表尾指针 Lev：当前程序层次 pdx：可为标识符分配空间尾指针*
函数描述	标识符登入符号表程序
返回值	Void

函数原型	position(char* id, int tx)
参数描述	*id：要查找的名字 tx：当前处理符号表尾*
函数描述	使用给定名字倒序在符号表查找，找到返回位置，否则返回 0
返回值	(int)符号表中位置或 0

函数原型	base(int l, int* s, int b)
参数描述	*L：当前过程与查找未知的层差 s：运行栈 b：动态链*

函数描述	通过过程基址求上 l 层过程的基址
返回值	(int)回溯 l 层过程的基址

函数原型	error(int n)
参数描述	*n：错误类型编号*
函数描述	根据给定错误类型号输出相关错误信息并更新错误数目，超过上限则直接终止程序编译
返回值	Void

函数原型	gen(FCT x, int y, int z)
参数描述	*x：中间代码指令名称 y：中间代码 l 字段 z：中间代码 a 字段*
函数描述	在虚拟机中生成给定中间代码
返回值	Void

函数原型	test(bool* s1, bool* s2, int n)
参数描述	*s1：需要的单词集合 s2：可恢复语法分析继续正常工作的补充单词符号集合 n：错误类型*
函数描述	在语法分析程序的入口和出口处调用测试函数 test，检查当前单词进入和退出该语法单位的合法性
返回值	Void

函数原型	inset(int e, bool* s)
参数描述	*e：给定符号 s：给定符号集合*
函数描述	判断给定符号 e 是否在给定符号集合 s 中
返回值	(int)e 在 s 中返回 1，否则返回 0

函数原型	addset(bool* sr, bool* s1, bool* s2, int n)
参数描述	*sr：新生成的符号集 s1：符号集 1 s2：符号集 2 n：符号集可包括的符号总数*
函数描述	取 s1 和 s2 的并集作为 sr
返回值	(int)0

函数原型	subset(bool* sr, bool* s1, bool* s2, int n)
参数描述	*sr：新生成的符号集 s1：符号集 1 s2：符号集 2 n：符号集*

	可包括的符号总数
函数描述	取 s1 和 s2 的差集作为 sr
返回值	(int)0

函数原型	mulset(bool* sr, bool* s1, bool* s2, int n)
参数描述	*sr：新生成的符号集 s1：符号集 1 s2：符号集 2 n：符号集可包括的符号总数*
函数描述	取 s1 和 s2 的交集作为 sr
返回值	(int)0

函数原型	getch()
参数描述	
函数描述	在 getsym 函数中使用。通过建立行缓冲区 line 用于一行行读取字符，line 被 getsym 取空后再读一行。对于每个字符，先过滤空格，再读取一个字符到 ch 中
返回值	Void

函数原型	getsym()
参数描述	
函数描述	词法分析，读取一个符号到 sym 中
返回值	Void