SEAL 语法手册

一、 简介

Seal（Simple Education Accompanying Language）是一款用于编译原理教学的编程语言。是一种静态类型语言。

二、 基本语法

2.1基本数据类型

Seal 的基本数据类型包括了 **Int**、**Float**、**String**、**Bool** 和 **Void**。其中，**Int** 默认为 64 位整数（同 C++里的 long long），**Float** 为 64 位浮点数（同 double），**String** 为字符串，**Bool** 为布尔变量；**Void** 为空类型，用于函数标识无返回值，以及无类型表达式的值，例如空表达式。其中，**Int** 类型可以写作十进制形式，15，也可以写为十六进制格式，如 0xf、0xF、0XF 等。**Bool** 有两个值，即 **false** 和 **true**。浮点数 **Float** 即按照正常的十进制表示法，即如 1.214，不允许 0.和.5 的形式，所有的浮点数必须以 x.x 的形式，例如 8 的浮点形式为 8.0，如果写作 8 则视为整型。字符串值参考 2.8 节。

除了五个基本数据类型的名称是以大写字母开头，其余所有变量、函数命名一定要以小写英文字母做开头，名字的其他部分可以用到大小写字母、数字以及下划线。

2.2变量声明

Seal 可以在函数体外、函数体内声明，基本格式如

```
var x Float;
var y Float;
```

赋值

```
x = 1.0;
```

其中，**var** 是关键词，用于指示变量声明，**Float** 为类型，放在末尾。要求，每一行只能声明一个变量。但是请注意，在函数声明的范围之外仅仅可以声明变量，不可以执行赋值等任何语句。

2.3函数调用和定义

Seal 中的函数声明如下格式

```
func max(x Float, y Float) Float{
    if x > y {
        return x;
    }
    return y;
}
```

可以看到函数的形参的声明部分，也是变量名 x 在前，类型 **Float** 在后。关键词 **func** 用于指示函数声明，形参 x，y 括号的后面，是返回值的类型，返回值最多只能有一个。返

回关键词仍然为 **return**，后面跟要返回的值，当返回值为空时，该处填为 **void**。在调用时，可以直接如 max(1.1,1.2)来调用。必须有一个 main() Void 函数，即该函数参数为空，且返回值为空，作为整个程序的入口。

此外，为了在目标代码生成阶段的方便，限制每个函数最多有 6 个参数。

2.4条件语句

基本条件语法同 C++，如

```
if x>10 {
    x = x + 1;
}
```

或

```
if x==0 {
    x = 3;
}else {
x = 2;
}
```

注意，在 Seal 中，**if** 的条件判断部分不用带小括号，所以这里直接就是 x>10

2.5循环语句

Seal 的循环语法有 **while** 和 **for** 两种，**While** 形如

while <condition> Body

condition 为 while 的条件部分，不可缺失，必须由 bool 类型的表达式构成。Body 是一个表达式块，即用大括号括起来的一系列语句。具体执行时，按照<condition> Body <condition> Body 的顺序。形式上的例子如

```
while x < 10 {
    x = x+1;
    y = 3 + 2;
}
```

for 形如

for <initexpr>;<condition>;<loopact> {Stmts;}

这里的 initexpr 为循环执行前执行的表达式，condition 为执行循环体之前的条件判断语句，loopact 为每轮循环体结束的执行动作。例如

```
for x = 0; x < 10; x = x + 1{
    s = s + x;
    p = x + 3;
}
```

最开始先会执行 x=0，而后判断 x<10 是否成立，成立则进入循环体，循环体执行完毕之后，执行 loopact 语句 x=x+1，而后再判断 x<10 是否成立，以此类推。

同样的，**while** 和 **for** 的条件部分都是不用加括号的。**for** 语句中，由分号隔开的 initexpr、condition、loopact 三个部分，可以缺失其中的任何或者全部，例如

```
for ;x<10; {
```

x = x+1;

}

对于两个循环语句类型，存在关键字 **continue** 和 **break**，**break** 能够直接中断循环体，执行后续的部分；**continue** 则跳出当前循环体，**while** 语句直接到<condition>部分，**for** 语句直接到 loopact 部分。

有一个特殊的函数，printf，与 C++中的类似，其参数第一个为 String 类型，后面可以跟若干个其他类型的参数，用于输出一个字符串。而对应参数在第一个参数字符串中必须有占位符，对应的占位符如下表

Int	%lld
Float	%f
Bool	%lld
String	%s

例如可以调用写作 printf("a=%f, b=%d", 1.1, b);

这个函数不可以重新声明，在语义分析部分，要检查是否重定义、第一个参数类型是否为 String；目标代码生成部分，相关的部分已经给出。

2.6表达式

Seal 的运算符，将其大致上分为算术运算符和逻辑运算符。在实际操作中，我们将每一个表达式会标识为某个类型，例如 Int 类型的 a 和 b，式子 a+b 的结果类型为 Int，而(a+1)>b 的类型为 Bool。对于循环和条件语句的条件部分，必须由 Bool 类型的表达式构成。

	算数运算符	逻辑运算符
Int	+ - * / %（余）-(单目，取负) &（按位与） |（按位或） ^（按位异或） ~（单目、按位取反）=	> >= == != <= <
Bool	=	&& || ! != == ^（异或）
Float	+ - * / -(单目，取负) =	> >= == != <= <

原则上，只有相同类型的数据类型能够进行算术运算，得到的结果是相同类型的表达式。但是，对于数值**比较**，可以在 Int 与 Float 之间进行，+ - * /也可以在 Int 与 Float 之间执行，而得到的表达式类型为 Float。

特别要注意的是，String 类型并没有支持任何的运算符，意即我们不会对 String 类型的值做任何修改操作。

我们考虑运算符的优先级，规定：

	算数运算符	逻辑运算符
最先算	~ & | ^（左）	！（无结合律）
	-（单目取负，无结合律）	< <= >= >（无结合律）
	* / %（左）	== != （无结合律）
	+ -（减法）（左）	&&（右）
		||（右）
最后算	=（无结合律）	

当然，小括号内的是最优先的。

2.7语句块

特别需要强调一下语句块，语句块即后文语法形式定义中的 $StmtBlock$，也即利用大括号括起来的语句们。典型的如 **while** 语句的 Body 部分即是一个语句块。语句块内如果声明局部变量，则其所有的局部变量声明语句必须位于语句块中的开头部分，然后是正常的语句。具体的语句块如何构成，可以参考语法形式部分。

2.8字符串

Seal 的字符串有两种表达形式，双引号"括起来和反撇号`括起来。以下列输出的字符串为例（注意到里面有个\t，这里不是水平制表符，而是两个字符即反斜杠\和小写字母 t）：

```
This is a test string.
It's a new line\t.
```

双引号模式的形式为 s = "This is a test string.\nIt's a new line\\t."，注意到为了输出反斜杠，需要转义即\\，换行符为\n 的形式；如果不想手动输入换行符，要形成一个多行字符利用双引号也可以以

s = "This is a test string.\
It's a new line\\t."

也即在行末加上转义字符，让编译器了解到字符串并未结束。但是反斜杠仍然需要转义。

反撇号括起来的字符串就更为方便，反撇号是所见即所得，即上述定义为

s = `This is a test string.
It's a new line\t.`

即可输出原结果。

2.9注释

Seal 语言的注释分为单行注释和多行注释。单行注释以//起始，一直持续到行末换行符结束。而多行注释以/*开始，以*/结束。如果一个*/符号没有与之配对的/*，则视为违规。

三、 语法形式定义

Seal 所包含的语法形式定义如下

$$Program := [Decl]^{+}$$

$$Decl := VariableDecl \mid CallDecl$$

$$VariableDecl := \textbf{var}\ Variable\ ;$$

$$Variable := ObjectID\ TYPEID$$

$$TYPEID := \textbf{Int} \mid \textbf{Float} \mid \textbf{Bool} \mid \textbf{String} \mid \textbf{Void}$$

$$CallDecl := \textbf{func}\ object([Variable[\![,Variable]\!]^{*}])\ TYPEID\ StmtBlock$$

$$StmtBlock := \{VariableDecl^{*}\ Stmt^{*}\}$$

$$Stmt := ; \mid Expr; \mid IfStmt \mid WhileStmt \mid ForStmt \mid BreakStmt \mid ContinueStmt \mid ReturnStmt \mid StmtBlock$$

$IfStmt$:=**if** $Expr\ StmtBlock$ [**else** $StmtBlock$]

$WhileStmt$:=**while** $Expr\ StmtBlock$

$ForStmt$:=**for** [$Expr$] ;[$Expr$];[$Expr$] $StmtBlock$

$ReturnStmt$:=**return** [$Expr$];

$ContinueStmt$:=**continue**;

$BreakStmt$:=**break**;

$Expr$:=$ObjectID\ = Expr$ |$Constant$| $Call$ |($Expr$)|$ObjectID$ |
$Expr + Expr$ | $Expr - Expr$ | $Expr * Expr$ | $Expr/Expr$ | $Expr\%Expr$ |
$- Expr$ | $Expr < Expr$ | $Expr <= Expr$ | $Expr == Expr$ | $Expr! = Expr$ |
$Expr >= Expr$ | $Expr > Expr$ | $Expr\&\&Expr$ | $Expr||Expr$ | $Expr\hat{}Expr$ | $!Expr$ |
$\sim Expr$ | $Expr\&Expr$ | $Expr|Expr$

$Call$:=$ObjectID(Actuals)$

$Actuals$:=[$Expr$[,$Expr$]*]

$Constant$:=$intConstant$ | $floatConstant$ | $boolConstant$ | $stringConstant$

其中，竖线|表示或者，[]内的内容为可选择，双方括号同样为可选择，后面跟*号表示可以重复 0 个或者多个，需要特别强调说明的是，基本的数据类型都是大写开头的。