# U. hs 编程入门

## 简单介绍

U.hs 是一门基于无类型 lambda 演算的纯函数式、懒惰求值的弱类型语言。它的灵感来源于 Unlambda.

在 U.hs 中，所有（除若干个特例外）的东西都是函数，每个函数接受一个参数（也是个函数），返回一个函数（这个返回的函数自身也是接受函数返回函数的函数）。

U.hs 使用 Lisp 风格的括号语法来表示函数的应用。同时，可以用 lambda 来定义函数。例如，我们可以定义一个单位函数 id:

(def id (\x x))

其中，\表示 lambda，即，我们定义 id 为 lambda x : x .

函数通过括号来应用到参数上。例如，(id id)就是 id. 括号内可以连续应用多个参数：

(id id id id)等价于(((id id) id) id)，也就是，id.

多个变量可以串联在同一个 lambda 中：(\a b c)等价于(\a (\b c))

有几个已经为你定义好的函数，比如，

(def K (\x y x))

(def S (\f g x ((f x) (g x))))

实际上，任何的 lambda 表达式都可以通过 K 和 S 组合而成。比如，id 就是(S K K). 使用 S 和 K 代替 lambda 可以让你的程序变得人类不可读（如果你真的想这么做的话）。

为了让程序做点有用的事，U.hs 内置了整型和浮点型的数值（数值是唯数不多的不是函数的东西）。你可以用“前缀表达式”似的方式来进行计算：

(+ (* 1 2) 3)　得到(1*2)+3 = 5

如何进行条件判断呢？U.hs 中预置了两个“布尔值”：

(def True K)

(def False (K (S K K)))

布尔值的特点是，如果 b 是一个布尔值，那么(b true_value false_value)就是：如果 b 是 True，那么 true_value；否则，false_value. 归功于懒惰求值，只有会被使用的那一分支会得到运行。于是，我们可以写出类似

(def safeinv (\x ((= x 0) 0 (/ 1 x))))

这样的函数出来。

下面的章节从不同的维度介绍 U.hs 的更多细节。

## 安装指南

随代码分发的有一份 readme.pdf. 里面详细说明了编译步骤。

最简单的运行 U.hs 程序的方法是用解释器运行它

```
$./urun helloworld.u
helloworld
```

另外，也可以把程序编译成可执行程序（需要你有一个可用的 llvm 工具链）。

```
$./ucomp helloworld.u
$./helloworld
helloworld
```

## 程序结构

每个 U.hs 程序都写在.u 文件中。一个文件中可以 import 若干别的文件，定义一些符号，同时有一个“主块”. 例如，一个实现按数值把字符串相加的.u 程序

```
(import* str)
(def strAdd (\a b (+ (atoi a) (atoi b))))
(strAdd 10 3)
```

import*表示把 str.u 中的所有符号导入进来。如果不加*而写成(import str)，那么你需要用 str.atoi 这种形式使用导入的符号。文件之间的导入关系必须是一个有向无环图。

(def name body)语句把 body 的值绑定到 name 上. body 中只能使用之前已经定义好的值。甚至，body 不能包含 name 本身！

那么如何定义递归函数呢？实际上，我们只需要一个把函数应用到自身上就行了

((\x (x x)) (\self (+ 1 (self self))))

这个定义了一个(+ 1 (+ 1 (+ 1 ...)))的无穷的表达式（死循环）。

有一个更优雅的方式是使用 Y 组合子：

(def recur (\f ((\x (f (x x))) (\x (f (x x))))))

这样我们刚才的函数就可以写成(recur (+ 1))或者(recur (\self (+ 1 self))).

主文件（就是传给 ucomp 或 urun 的文件）的主块会被求值并打印出来。非主文件（被别的文件导入的文件）的主块不会求值。这样，你可以把一个模块的测试或示例代码写在它的主块中。

可以使用 let 来在一个模块内定义只在模块内可见的符号：

(let local_method body)

同时，let 也是一个表达式：

```
(let
    (a body_a)
    (b body_b)
    c
)
```

等价于((\a ((\b c) body_b)) body_a)

## IO 操作

在纯函数式语言中进行 IO 是一个挑战。在 U.hs 中，所有的 IO 都是通过回调函数实现的。例如，可以使用 putChar 输出一个字符：

(putChar 'a' callback)

其中，callback 是一个函数。putChar 'a'的结果（0 表示成功，其他表示失败）会传给 callback 并继续运行。下面是把输入的一个字符送到输出并退出的程序：

(getChar (\a (putChar a (_ (exit 0)))))

exit 把程序的运行结束，所以不需要回调函数。为了让这种级联的回调更好写，U.hs 中有一个 run 结构：

```
(run
    (a getChar)
    (_ (putChar a))
    (exit 0)
)
```

和上面的程序是等价的。

另外一种写法是使用 do 结构。(do ...)就是(\return (run ...))。例如：

```
(def readTwo (do
    (a getChar)
    (b getChar)
    (return (pair a b))
))
```

pair 的定义会在后面的章节讲到。

除了 getChar、putChar，U.hs 中还有如下内置的 IO 函数：

open filename mode callback

filename 是文件名。mode 是 ReadMode、WriteMode、AppendMode、ReadWriteMode 之一. 这个操作到一个文件句柄并传给 callback.

getCharF handle callback

用句柄 handle 指向的文件读入一个字符，并把结果传给 callback. 如果遇到 EOF，那么得到-1. getChar 就是(getChar stdin)

putCharF handle char callback

输出一个字符。返回 0 或者错误码。

peekCharF handle callback

非获取性的读入（读入但在缓存中保持这个字符）一个字符。-1 表示 EOF.

getArg callback

读入命令行参数中的下一个字符。如果命令行参数已经没有更多字符，返回-1.

close handle callback

关闭文件。返回 0 或者错误码（如果 handle 不合法的话）。

system cmdline callback

新建一个 shell，运行 cmdline. 退出值返回给 callback.

原则上，以上 IO 函数使得 U.hs 成为一个有可能自举的语言。当然，进行自举需要相当大的工作量。

## 基本数据类型

每个文件都隐式的导入了 prelude.u. prelude 中包含了各种常用的数据结构的定义。

例如，之前举过的布尔型的例子：

```
(def if id)
(def True (\a b a))
(def False (\a b b))
```

于是，可以像其他语言一样写 if then else 表达式：

```
(if condition
    true_block
    false_block
)
```

布尔之外，最常用的就是 list 了. 一个 list 定义为：

```
(def empty (\a b a))
(def cons (\head tail (\a b (b head tail))))
```

注意，empty 和 True 是一模一样的。甚至，你可以写成(def empty True)或者(def empty K). 类型系统的缺乏（万物皆函数）使得 U.hs 事实上成为了一门弱类型的语言。如果你混用了数据类型，除非涉及到内置的数值或函数（例如把数值作用到另一个数值上会产生一个异常），通常并不会导致立即报错。这也使得 U.hs 的程序调试起来很需要想象力。

对于一个 list，你可以用类似 pattern matching 方式来访问它：

(l null_value (\head tail non_null_value))

当列表 l 是空的时候，得到 null_value. 否则，针对 head 和 tail 对得到 non_null_value

cons 把一个元素和一段列表连接起来（放到头部）. 有一个 list 结构来简化列表的构造：

(list a b c)等价于(cons a (cons b (cons c empty))

注意，list 是个语法构造（就像 run 一样）而不是函数。这是因为它在每个表达式中的参数个数是需要在编译时分别确定的。

字符串被实现为 int 的列表。”abc”等价于(list 'a' 'b' 'c')或(list 97 98 99)

传给 open 和 system 的列表里如果有非整型的参数（用 toInt 和 toFloat 可以在整数与浮点数之间相互转化），会产生错误。

另外一些有用的数据类型包括：

pair：((pair a b) f) = (f a b)

访问元素：(fst (pair a b)) = a，(snd (pair a b)) = b

Maybe:  (nothing a b) = a  ((just x) a b) = (b x)

此外，还有一些用于列表操作的函数（例如 map, foldl, foldr）定义在 prelude.u 中。

## 标准库、示例程序

随源码附带了几个例子与模块。包括：

str.u : 字符串处理（atoi，itoa 等）

io.u : 基本的输入、输出（readLine，readInt 等）

algorithm.u ： sort 等

有几个测试用的例子：

sorttest.u 把 stdin 读入的行按字典序排序之后输出