

# FOPL-Homework4

昂伟 *PB11011058*

## Problem 1

见 *prob1.hs*

## Problem 2

(a)

*return* 与  $\leftarrow$  作用相反，它将一个 **pure value** 封装为一个 *IO* 对象。

(b)

```
m >>= \_ -> n
```

(c)

```
getLine :: IO [Char]
getLine = getChar >>= \c ->
    if c == '\n'
    then return []
    else
        getLine >>= \cs -> return (c:cs)
```

## Problem 3

(a)

见 *untilIO :: IO Bool -> IO ()*

```
untilIO :: IO Bool -> IO ()
untilIO x = do
    result <- x
    if result == True
    then return ()
    else
        untilIO(x)
```

(b)

见 *doGuess :: Int -> IO Bool*

```

doGuess :: Int -> IO Bool
doGuess secret = do
    guess <- getLine
    let x = read guess
    case compare x secret of
        LT -> putStrLn "Too low!" >> return False
        EQ -> putStrLn "Congratulations!" >> return True
        GT -> putStrLn "Too high!" >> return False

```

**Problem 5**

(a)

<i>Activation Records</i>		
(1)	access link	(0)
	y	0
(2)	access link	(1)
	x	0
(3)	access link	(2)
	g	.
	exn handler	
(4)	access link	(3)
	b	.
	exn handler	
(5)	access link	(4)
	exn handler	19
(6) b(...)	access link	(4)
	h	.
	x	2
	exn handler	14
	f	.
(7) g(...)	access link	(3)
	h	.
	x	4
	exn handler	7
(8) f(...)	access link	(6)
	x	6

*Closures*

&lt;(3), .&gt;

&lt;(4), .&gt;

&lt;(6), .&gt;

*Compiled**Codes*

| code for g |

| code for b |

| cod for f |

(b)

活动记录 (7) (8) 被弹出栈。因为抛出异常时，由动态作用域知，异常处理由 14 行的代码处理，所以活动记录 (6) 之上的活动记录都被弹出。

(c)

$$y = 2.$$

## Problem 6

(a)

McCarthy 定义的垃圾在我们定义中不一定是垃圾。McCarthy 的定义中不能被基址寄存器引用的内存即为垃圾，而不能被基址寄存器引用却有可能被内存中的指针引用，所以该内存可能程序的后续指令引用，所以 McCarthy 定义的垃圾不一定是我们定义的垃圾。

(b)

我们定义的垃圾一定是 McCarthy 定义的垃圾。我们的定义中不能被程序后续指令引用的内存是垃圾，既然不能被引用当然不能被基址寄存器引用，所以我们定义的垃圾一定是 McCarthy 定义的垃圾。

(c)

不可能写出一个垃圾收集器来收集完我们定义的垃圾。因为当程序执行到某点需要进行垃圾收集时，我们只能确定该点之前申请的内存空间中哪些内存一定不会被用到，但是我们无法确定程序后续指令是否一定不会用到的内存空间。所以我们只能采取相对保守的策略：只释放一定不会被用到的内存。