Bài tập 1

Cho biết kết quả của biểu thức sau viết bằng Scala:

List(3,5,9,10).foldLeft(0)((x,y)=>x+y)

- (•) 27
- ( ) List(27)
- ( ) List(3,5,9)
- ( ) 3

Chính xác

Bài tập 2

Cho filter là hàm bậc cao nhận vào một vị từ (predicate - hàm có kết quả kiểu luận lý) và trả về một danh sách con gồm các phần tử thoả vị từ đó và cho phép toán % là phép toán tính modulo. Hãy viết biểu thức để trả về một danh sách con chỉ gồm các giá trị chẵn của danh sách vào? List(2,3,4,5,6)._____ => List(2,4,6)

- ( ) List(2,3,4,5,6).filter(x => x % 2)
- ( ) List(2,3,4,5,6).filter(x % 2 == 0)
- (•) List(2,3,4,5,6).filter(x => x % 2 == 0)
- ( ) List(2,3,4,5,6).filter((x,y)=> x % 2 == y)

Chính xác

Bài tập 3

Cho một danh sách List(4,2,6,9), hãy cho biết dùng hàm bậc cao nào để tạo được một danh sách là các giá trị bình phương của các phần tử trong danh sách được cho, tức List(16,4,36,81)

- [ ] foldLeft
- [x] map
- [ ] filter
- [ ] forall

Chính xác, khi danh sách kết quả có cùng số phần tử với danh sách vào và mỗi phần tử kết quả là từ mỗi phần tử của danh sách vào thì map là hàm bậc cao thích hợp nhất

Bài tập 4

Cho reverse là hàm đảo ngược thứ tự các phần tử của một danh sách (ví dụ List(1,4,2).reverse => List(2,4,1)) và :: dùng để nối 1 phần tử vào đầu một danh sách (a :: List(b,c,d) => List(a,b,c,d)). Với lst đang chứa một danh sách các giá trị nguyên, cho biết biểu thức nào dưới đây có kết quả tương tự reverse?

- [ ] lst.map(x => x :: List())
- [ ] lst.foldLeft(List())((x,y)=>x::y)
- [ ] lst.forAll(x => x::List())
- [x] lst.foldRight(List())((x,y) => x::y)

Chính xác, hãy kiểm tra lại trên Scala.

Bài tập 5

Hãy viết một hàm (forAllExist) nhận vào 3 thông số gồm 1 danh sách các số nguyên và 2 predicate, chỉ trả về kết quả true nếu danh sách có tất cả phần tử thoả predicate thứ nhứt và có ít nhất 1 phần tử thoả predicate thứ hai.

- (•) def forAllExist(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.forall(f1) && lst.exists(f2)
- ( ) def forAllExist(lst:List[Int],f1:Boolean,f2:Boolean) = lst.forall(f1) && lst.exists(f2)
- ( ) def forAllExist(lst:List[Int],f1:Int,f2:Int) = lst.forall(f1) && lst.exists(f2)
- ( ) def forAllExist(lst:List[Int],f1:Boolean=>Boolean,f2:Boolean=>Boolean) = lst.forall(f1) && lst.exists(f2)

Chính xác

Chưa chính xác, predicate là một hàm nên nó phải có kiểu vào và kiểu ra và được viết dưới hình thức <kiểu vào> => <kiểu ra>. Ngoài ra, predicate phải có kiểu ra là Boolean nên nó phải có dạng <kiểu vào> => Boolean

Chưa chính xác, predicate là một hàm nên nó phải có kiểu vào và kiểu ra và được viết dưới hình thức <kiểu vào> => <kiểu ra>. Ngoài ra, predicate phải có kiểu ra là Boolean nên nó phải có dạng <kiểu vào> => Boolean

Chưa chính xác, theo yêu cầu của câu hỏi, các phần tử của danh sách số nguyên thoả predicate thứ nhứt, do đó, hàm predicate phải có kiểu vào là kiểu phần tử của danh sách (trong trường hợp này là kiểu nguyên).

Bài tập 6

Hãy viết một hàm (doubleCheck) nhận vào 3 thông số gồm 1 danh sách các số nguyên và 2 predicate, chỉ trả về kết quả true nếu danh sách có ít nhất một phần tử thoả predicate thứ nhứt và có ít nhất một phần tử thoả predicate thứ hai.

- [ ] def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.exists(x => f1(x) && f2(x))
- [x] def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.exists(f1) && lst.exists(f2))
- [ ] def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.forall(x => f1(x) && f2(x))
- [ ] def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.forall(f1) && lst.forall(f2))

Chưa chính xác, biểu thức này trả về true chỉ khi danh sách có 1 phần tử thoả cùng lúc 2 điều kiện

Chính xác

Chưa chính xác, biểu thức này chỉ trả về true khi tất cả các phần tử của danh sách thoả cả hai predicate

Chưa chính xác, biểu thức này chỉ trả về true khi tất cả các phần tử của danh sách thoả cả hai predicate

Bài tập 7

CHo một hàm được định nghĩa như sau:

def increaseClosures(n:Int)(x:Float) = x + n

Một lênh khai báo biến inc3 được viết như sau:

val inc3 = increaseClosures(3) _

sẽ làm cho inc3 được cất giữ giá trị gì?

- ○ Không cất giữ gì cả vì lệnh gọi hàm increaseClosures bị sai, không đủ thông số
- ○ 3
- ○ x + 3
- ● Một hàm có kiểu là Float => Float

Chưa chính xác, khi khai báo các thông số ở trong các dấu () độc lập thì hàm có thể được gọi mà không cần đủ số thông số

Chưa chính xác

Chưa chính xác, biến inc3 không cất giữ biểu thức x + 3

Chính xác, hàm này là (x:Float) => x + 3

Bài tập 8

Cho định nghĩa hàm như sau:

def foo(x:Int)(y:Float) = x * y

Khi định nghĩa x như sau: val x = foo(3) _ thì x sẽ có kiểu là gì

- Int
- Float
- Không có kiểu gì cả vì việc gọi hàm foo bị sai do thiếu thông số
- Kiểu hàm: Float => Float

Chưa đúng

Chưa đúng

Chưa đúng

Chính xác, vì hàm foo có kiểu Int => Float => Float nên khi gọi hàm foo với chỉ 1 thông số foo thì sẽ nhận kiểu trả về là Float => Float

Bài tập 9

Cho hàm foo được định nghĩa như sau:

def foo(x:Boolean,y:Float)(z:Float)= if (x) y else z

và m được dịnh nghĩa như sau: val m = foo(true) _ _

Cho biết kiểu của m?

- Float
- Không có kiểu gì cả vì lệnh gọi foo chưa đúng
- Kiểu hàm Float => Float
- Kiểu hàm Float => Float => Float

Chưa đúng

Chính xác

Chưa chính xác

Chưa chính xác

Cho listA là một danh sách có 3 phần tử {a, b, c} và listB là một danh sách có 3 phần tử là {d,e,f}, nếu tác vụ listA.append(listB) sẽ làm cho listA trở nên có 6 phần tử gồm 3 phần tử ban đầu của listA và 3 phần tử của listB thì tác vụ append là?

- ○ Immutable
- ◉ Mutable

Chính xác. tác vụ append làm thay đổi listA nên nó là một tác vụ mutable.

Bài tập 2

Cho listA là một danh sách có 3 phần tử {a, b, c} và listB là một danh sách có 3 phần tử là {d,e,f}, nếu tác vụ listA.append(listB) trả về một danh sách mới có 6 phần tử gồm 3 phần tử của listA và 3 phần tử của listB, trong khi listA và listB không đổi, thì tác vụ append là?

- ◉ Immutable
- ○ Mutable

Chính xác, tác vụ append không làm thay đổi các thông số của nó nên append là immutable.

Bài tập 3

Cho x chứa chuỗi "abc", nếu x.toUpper sẽ làm cho x trở thành chứa chuỗi "ABC" thì toUpper là?

- ○ Immutable
- ◉ Mutable

Chính xác.

Bài tập 4

Giả sử Scala là một ngôn ngữ lập trình hàm tinh khiết (pure functional programming language) và giả sử Scala có phát biểu while như sau:

while (a < 1) { ... }

Thân vòng lặp là một hộp đen, không rõ về nội dung. Bạn có thể suy luận gì về lệnh lặp trên?

- ○ Không thực thi được lần nào
- ○ Sẽ thực thi bình thường như một lệnh lặp while trên các ngôn ngữ khác (Java)
- ○ Luôn luôn lặp vô hạn
- ◉ Hoặc không thực thi hoặc lặp vô hạn

Đúng, trên ngôn ngữ lập trình hàm thuần khiết, các biến đại diện cho một giá trị không đổi, do đó, biểu thức a < 1 hoặc là luôn luôn sai hoặc là luôn luôn đúng nên lệnh while sẽ hoặc là không được thực thi (khi a < 1 sai) hoặc là lặp vô hạn (khi a < 1 đúng). Vì vậy trên các ngôn ngữ lập trình hàm thuần khiết, không có lệnh lặp dựa vào biểu thức điều kiện như while, do while.

Bài tập 1

Trên Scala, phép toán :: dùng để nối 1 phần tử vào 1 danh sách (3::List(2,1,5) sẽ tạo thành danh sách List(3,2,1,5)). Giả sử biến sl đang chứa giá trị List(3,5,6,2), hãy cho biết phép match sau có thành công không và nếu có thì giá trị của h và t là bao nhiêu?

```
sl match {
    case h :: tail => ....
```

- ○ Thành công, h là 3 và t là List(5,6,2)
- ○ Không thành công, phải ghi case List(3,5,6,2) thì mới thành công
- ◉ Không thành công, phải ghi case List(t,h) mới thành công
- ○ Nếu h đang là 3 và t đang là List(5,6,2) thì mới thành công

Chính xác

Không đúng, mặc dù ghi List(3,5,6,2) cũng thành công

Không đúng, nếu ghi List(t,h) sẽ không trùng vì sl là danh sách có 4 phần tử trong khi List(t,h) chỉ có 2 phần tử

Chưa đúng, h và t là biến tự do (chưa nhận giá trị nào) thì có thể trùng với bất kỳ giá trị nào

Bài tập 2

Trên Scala, kiểu tuple là kiểu kết hợp nhiều giá trị với nhau, ví dụ val m = (3,4.3, True) là một giá trị kiểu tuple kết hợp 3 kiểu Int, Double và Boolean. Cho biết với m định nghĩa trong ví dụ trên, trong phép match sau thì thành công ở case nào?

```
m match {
  case (3,1.0,True) => ...
  case (3,4.3,False) => ...
  case (_,_,_) =>
}
```

- ○ Không thành công ở case nào cả
- ○ Thành công ở case đầu tiên vì có giá trị 3 trùng nhau
- ○ Thành công ở case thứ hai
- ○ Thành công ở case cuối

Chưa chính xác

Chưa chính xác, phải cả 3 giá trị trùng nhau thì mới thành công

Chưa chính xác

Chính xác, vì _ có thể trùng với bất kỳ giá trị nào

Bài tập 1

Cho một khai báo hàm như sau:

def foo(x:Boolean,y:Int,z:Int) = if (x) y else z

Hỏi khai báo trên sẽ gây ra lỗi gì?

- ( ) Không gây ra lỗi gì
- ( ) Lỗi biên dịch vì không khai báo kiểu trả về
- (•) Có thể gây ra lỗi khi thực thi tuỳ theo cách gọi hàm
- ( ) Luôn gây ra lỗi runtime

Đúng nhưng chưa đủ. Khai báo hàm như trên không gây ra lỗi từ vựng, cú pháp hay ngữ nghĩa. Tuy nhiên, khi gọi hàm như là: foo(a==0,1,b/a) thì các biểu thức a==0, 1, b/a sẽ được tính để có giá trị truyền cho các thông số hình thức a,b,c nên sẽ gây ra lỗi chia cho 0 khi a bằng 0.

Chưa đúng, vì Scala có thể suy diễn được kiểu trả về là Int do cả 2 vế của if đều trả về Int

Đúng, tuỳ theo cách gọi hàm, có thể xảy ra lỗi khi thực thi. Ví dụ foo(a==0,1,b/a) sẽ gây ra lỗi chia cho 0 khi a bằng 0. Hoặc foo(x == null,null,x.m()) sẽ gây ra lỗi Null pointer reference.

Chưa đúng, ví dụ foo(a==0,1,2) sẽ không gây ra lỗi.

Bài tập 2

Để khắc phục lỗi về ngữ dụng trong bài tập 1, cần phải sửa khai báo hàm như thế nào?

- def foo(x:Boolean)(y:Int)(z:Int) = if (x) y else z
- def foo(x:Boolean,y:Int) = (z:Int) => if (x) y else z
- def foo(x:Boolean, y: => Int, z: => Int) = if (x) y else z
- def foo(x:Boolean,y: Int, z:Int) = {

```
    lazy val m = y
    lazy val n = z
    if (x) m else n
}
```

Chưa đúng, khai báo hàm này sẽ cho phép gọi hàm mà không cần đủ các thông số (currying functions) nhưng vẫn gây ra lỗi thực thi như đã nêu ở bài tập 1

Chưa đúng, cách khai báo này là một hình thức khác của khai báo dạng curry, cho phép gọi hàm với 2 thông số nhưng vẫn gây ra lỗi như đã nêu ở bài tập 1

Đúng, khi viết => trước kiểu của thông số, thì thông số sẽ được truyền theo tên (pass-by-name). Khi đó các biểu thức thông số chỉ được tính khi thông số hình thức được sử dụng trong thân hàm.

Chưa đúng, mặc dù dung các biến m,n là lazy nhưng do các thông số y, z là eager nên khi gọi hàm foo(a==0,1,b/a) thì các thông số a==0, 1, b/a được tính trước khi truyền nên vẫn gây ra lỗi chia cho 0

Bài tập 1

Cho biết kiểu trả về của hàm sau:

def foo(a:Boolean) = if (a) 1 else List()

- ( ) Int
- ( ) List
- ( ) AnyVal
- (•) Any

Chưa đúng, kiểu Int được trả về trong trường hợp a == true, nhưng khi a != true thì kiểu Int không được trả về. Kiểu trả về của foo phải là 1 kiểu chung của cả hai vế then và else

Chưa đúng, kiểu List được trả về trong trường hợp a != true, nhưng khi a == true thì kiểu List không được trả về. Kiểu trả về của foo phải là 1 kiểu chung của cả hai vế then và else

Chưa đúng, AnyVal là lớp cha của các giá trị đơn như Int, Float, Long, Double. AnyVal không phải là lớp cha của List

Đúng, trên cây phân lớp của Scala, Any là lớp gốc nên nó là lớp tổ tiên của cả lớp Int và List

Bài tập 2

Chọn khai báo hàm đúng và ngắn nhất trong các khai báo sau:

- (•) def foo(x:Int):Int = if (x == 0) 1 else x*foo(x-1)
- ( ) def foo(x):Int = x + 3
- ( ) def foo(x:Int):Int = x + 4
- ( ) List(1,2,3).map((x:Int)=> x + 1)

Đúng, hàm đệ qui nên cần phải khai báo kiểu trả về

Chưa đúng, thông số vào cần phải được khai báo kiểu

Chưa đúng, khai báo trên đúng nhưng dư thừa kiểu trả về. Không cần khai báo kiểu trả về khi hàm không đệ qui

Chưa đúng, biểu thức trên dúng nhưng dư thừa khai báo kiểu của x vì x đại diện cho một phân tử của List(1,2,3) nên nó sẽ có kiểu Int.

Bài tập 3

Chọn biểu thức đúng và ngắn nhất trong các biểu thức sau:

- [ ] List(4,5,6).forall((x:Int) => x > 3)
- [ ] List(4,5,6).forall(x => x > 3)
- [x] List(4,5,6).forall(_ > 3)
- [ ] List(4,5,6).forall(> 3)

Đúng nhưng dư thừa khai báo kiểu của x. Kiểu của x là kiểu của member của List(4,5,6)

Đúng nhưng dư thừa khai báo x vì x chỉ xuất hiện 1 lần trong thân hàm.

Đúng

Chưa đúng vì không thể hiện phần tử của List sẽ xuất hiện ở chỗ nào trong biểu thức thân hàm