Homework 5

Andrew Kowalczyk

December 03, 2012

3

For C and Go, go to the links provided to see the function in action. Codepad for C and play.golang.org for Go.

Python

```python
def list_min(list):
    def min_finder(l, m):
        if not l:
            print m
        else:
            min_finder(l[1:], m if m < l[0] else l[0])
    min_finder(list, float("inf"))
```

C

```c
// Function in action at http://codepad.org/D4reNoZ5

int arrayMin(int *a, int size, int min, int index) {
    if (index == size - 1) {
        return min;
    } else {
        min = min < a[index] ? min : a[index];
        index++;
    }
    return arrayMin(a, size, min, index);
}
```

Javascript

```javascript
var arrayMin = function (array) {
    var minFinder = function (a, m) {
        return a.length === 0 ? m : minFinder(a.slice(1), m < a[0] ? m : a[0]);
    }
    return minFinder(array, Infinity);
}
```

Go

```
// Function in action at http://play.golang.org/p/KU4Jn1Qkcx

func ArrayMin(a [] int, min int) int {
  if (len(a) == 0) {
    return min
  } else {
    if (a[0] < min) {
      min = a[0]
    }
  }
  return ArrayMin(a[1:], min)
}
```

4

There is a possibility of getting stuck in an infinite loop. The user would potentially need to stop the program on their own if this were to happen.

5

Javascript

```
var a = function () {alert("first");};
var b = function () {alert("second");};
var both = function (a, b) {};

both(a(), b());
```

We can see after running this code in a shell or jsFiddle that JavaScript evaluate subroutines in the order that they are passed into a function (left-to-right).

6

If the program outputs $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ this is how the program runs: It first pushes the return address of i on the stack (which is 0). Print i out. The increment it by 1. When foo is called a second time, it looks up the variable i again and simply increments it. If the program ran in this way, it is likely that the stack was cleared when the program was run.

There is also a possibily that one can see all zeroes. This is because when the program allocated a stack frame for the this, it allocated it in such a place where it just so happened to be 0. Basiclly, the i overlayed the j in memory. This could mean that the stack was not initialized on that particular system.

8

The old version of Fortran printed 3 because it passed by reference. The modern version of Fortran prints 2 because it passes pointers to copies of rvalues. This is due to the fact that the compiler placed the value of the literal 2 somewhere in memory when foo was first called. Whenever there is 2 in the program, the compiler told it to look in that memory address. The value was changed when foo was called thus explaining why it printed 3.

10

Call by value: $1, [2, 3, 4]$ **is printed.**

Call by value-result: $2, [2, 2, 4]$ **is printed.**

Call by reference: $2, [2, 3, 4]$ **is printed.**

Call by name: $2, [2, 3, 4]$ **is printed.**

11

12 - XC

This is a bad idea because an object should not (cannot) have more than one class. As opposed to inheritance (*i.e. IS-A*), this society of classes should be built by aggregation (*i.e. HAS-A*). Aggregation should not be confused with composition. In other words, each *Person* HAS-A *Job*. The class *Job* can live it on its own without a *Person* having that particular *Job*. Each *Job* will have its own class to store properties specific to that *Job*. The person class should be the only class denoting a person. This person class can have its set of jobs or roles as a property.

13

Java

```java
public class OddGenerator {

  private int x = -1;
  public int nextOdd() {
    return x += 2;
  }

  public static void main (String[] args) {
    OddGenerator odds = new OddGenerator();
    System.out.println(odds.nextOdd());
```

```java
11      System.out.println(odds.nextOdd());
        System.out.println(odds.nextOdd());
13      System.out.println(odds.nextOdd());
    }
15 }
```

Python

```python
class odd_generator:
2    def __init__(self):
        self.__x = -1
4
    def next_odd(self):
6        self.__x = self.__x + 2
        return self.__x
8
z = odd_generator()
10 print z.next_odd()
   print z.next_odd()
12 print z.next_odd()
   print z.next_odd()
```

Javascript

```javascript
1 var nextOdd = function () {
     var x = -1;
3    return function () {return x += 2;};
   }();
5
   nextOdd();
7 nextOdd();
   nextOdd();
9 nextOdd();
```

C++

```cpp
1 #include <iostream>
   using namespace std;
3
   class OddGenerator {
5    private:
        int x = -1;
7    public:
        int nextOdd();
9 };
```

```cpp
int OddGenerator::nextOdd() {
    return x += 2;
}

int main () {
    OddGenerator odds;
    cout << odds.nextOdd() << endl;
    cout << odds.nextOdd() << endl;
    cout << odds.nextOdd() << endl;
    cout << odds.nextOdd() << endl;
}
```