# Homework 5

Andrew Kowalczyk

December 03, 2012

## 3

### 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
#include <stdio.h>

#define MAX_INT (1.0 / 0)

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

int arrayMin(int *a, int size) {
  return minFinder(a, size, MAX_INT, 0);
}

int main () {
  int a[10] = {5, 6, 7, 12, 3, 1, 234, 123, 67, 9000};
  int b[4] = {-432, 3, 1, 234};
  int c[7] = {-12, -11, -4, 1344234, -87653, 0, 75};
  printf("%d, %d, %d\n", arrayMin(a, 10), arrayMin(b, 4), arrayMin(c, 7));
}
```

### 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

```
package main

import "fmt"

const MaxInt = int(^uint(0) >> 1)

func ArrayMin(a []int) int {
  return MinFinder(a, MaxInt)
}

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

func main() {
  a := []int{5, 6, 7, 12, 3, 1}
  b := []int{-432, 3, 1, 234}
  c := []int{-12, -11, -4, 1344234}
  fmt.Println(ArrayMin(a), ArrayMin(b), ArrayMin(c))
}
```

## 4

We run the risk of getting stuck in an infinite loop.

## 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

$2, [2, 3, 4]$ is printed.

### Call by reference

$2, [2, 2, 4]$ is printed.

### Call by name

$2, [2, 2, 4]$ 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

look at old quizzes

### Java

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

### Python

### Javascript

```
var nextOdd = function () {
  var x = -1;
  return function () {return x += 2;};
}();
```

**C++**