CIS 624

Spring 1998

Final (100 points)

Name: ....

1	Environments vs Stores I	20 pts	
2	Environments vs Stores II	10 pts	
3	Basic Continuations	10 pts	
4	Object-Oriented Concepts	15 pts	
5	Programming	10 pts	
6	Types	15 pts	
7	Exceptions using call/cc	20 pts	
8	Extra Credit	10 pts	

Environments vs Stores I (20 points)

Trace the values of the environment and the store in the following program:

```
class Test {
  public static void main (String[] args) {
    int x = 11;

    // Env = .........................................................

    // Store = .......................................................

    int z = 0;

    // Env = .........................................................

    // Store = .......................................................
    {
      int a = 1;

      // Env = .......................................................

      // Store = .....................................................

      z = a+b;

      // Env = .......................................................

      // Store = .....................................................
    }

    // Env = .........................................................

    // Store = .......................................................

  labelOne: {
    int m = 111;

    // Env = .........................................................

    // Store = .......................................................

    z = m + z;

    // Env = .........................................................
```

```
    // Store = ......................................................

    break labelOne;
  }

    // Env = ........................................................

    // Store = ......................................................

    z = z + x + y;

    // Env = ........................................................

    // Store = ......................................................

    System.out.println("Result: z = " + z);
  }
}
```

Environments vs Stores II (10 points)

Consider writing an interpreter for the subset of Java programs with less than 10 identifiers each. What is the maximum size of the store that would be needed by such programs? Justify your answer using a Java program containing less than 10 identifiers.

Basic Continuations (10 points)

Write the value of each expression:

```
(call/cc (lambda (k) (+ 1 (+ 2 3))))
```

```
(call/cc (lambda (k) (+ 1 (+ 2 (k 3)))))
```

```
(+ 5 (call/cc (lambda (k) (+ 1 (+ 2 (k 3))))))
```

```
(+ 5 (call/cc (lambda (k) (k (+ 1 (+ 2 (k 3)))))))
```

```
(call/cc (lambda (k1)
  (+ 1 (call/cc (lambda (k2)
         (+ 2 (call/cc (lambda (k3)
                (k2 8)))))))))
```

Object-Oriented Concepts (15 points)

Write the value printed next to each `println` statement:

Program I

```
class Earthling {
  private String name = "Earthling";

  public String getName () { return name; }

  public String toString () {
    return getName();
  }
}

class African extends Earthling {
  private String localname = "African";

  public String getName () { return localname; }

  public String toString () {
    return getName() + " " + super.toString();
  }
}

class Test1 {
  public static void main (String[] args) {
    African amr = new African();
    System.out.println(amr); // ........................................
  }
}
```

Program II

```
class A {
  int x = 1;
}

class B extends A {
  int x = 2;
}

class C extends B {
  int x = 3;
}

class Test2 {
  public static void main (String[] args) {
    A i;
    B j;
    C k;

    k = new C();
    System.out.println(k.x); // ........................................

    j = k;
    System.out.println(j.x); // ........................................

    i = j;
    System.out.println(i.x); // ........................................

    j = new B();
    System.out.println(j.x); // ........................................

    i = j;
    System.out.println(i.x); // ........................................
  }
}
```

Programming (10 points)

The sieve of Eratosthenes is a neat algorithm that generates all the prime numbers. The idea is to start with the following infinite sequence of natural numbers:

$$2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, \ldots$$

The first number of that sequence is the prime number 2. We now generate a new infinite sequence that excludes all the multiples of 2:

$$3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, \ldots$$

The first number of the new sequence is the prime number 3. We now generate a new infinite sequence that excludes all the multiples of 3:

$$5, 7, 11, 13, 17, 19, 23, 25, \ldots$$

The first number of the new sequence is the prime number 5. We now generate a new infinite sequence that excludes all the multiples of 5:

$$7, 11, 13, 17, 19, 23, \ldots$$

and so on. Using this algorithm, one can easily get the first n prime numbers for any n.

Here is most of a Java implementation of the algorithm: it is missing the implementation of the class `Primes`. Finish it!

```
abstract class InfiniteSequence {
  abstract int getNext();
}

class FromTwoSeq extends InfiniteSequence {
  private int current;

  FromTwoSeq () { current=1; }

  int getNext() { return ++current; }
}

class FilteredSeq extends InfiniteSequence {
  private InfiniteSequence source;
  private Function filter;

  FilteredSeq (InfiniteSequence s, Function f) { source=s; filter=f; }

  int getNext() {
    int i = source.getNext();
    while (! filter.apply(i)) {
```

```
      i = source.getNext();
    }
    return i;
  }
}

abstract class Function {
  abstract boolean apply (int n);
}

class NotMultipleOf extends Function {
  private int i;

  NotMultipleOf (int i) { this.i = i; }

  boolean apply (int j) { return !(j % i == 0); }
}



class Primes {




  Primes () {






  }



  // The first call should return 2; the next call 3; the next 5; and so on.
  int nextPrime() {






  }
}
```

Types (15 points)

Write the type of each occurrence of the statement `throw new E();` below:

```
class E extends Exception {}

class A {
  int m1 (boolean b) throws E {
    if (b) return 1;
    else throw new E(); // ..................................
  }

  boolean m2 (boolean b) throws E {
    if (b) return false;
    else throw new E(); // ..................................
  }

  void m3 (boolean b) throws E {
    if (b) return;
    else throw new E(); // ..................................
  }
}
```

What can you conclude about the *general* type of the statement `throw new E();`?

Exceptions using call/cc (20 points)

One of the reasons Scheme does not have handleable exceptions is that it is rather simple to define them in the language. Assuming, for simplicity, that we only have one exception type, here are Scheme definitions for `throw` and `try_catch`:

```
(define global_handler
  (lambda ()
    (error 'global_handler "Uncaught exception")))

(define throw
  (lambda ()
    (global_handler)))

(define try_catch
  (lambda (body local_handler)
    (call/cc (lambda (k)
      (let ((old_handler global_handler))
        (begin
          (set! global_handler (lambda ()
                                 (begin
                                   (set! global_handler old_handler)
                                   (k (local_handler)))))
          (let ((result (body)))
            (begin
              (set! global_handler old_handler)
              result))))))))
```

Like Java, throwing an exception transfers control to the (dynamically) closest handler and resumes the computation at that point. In the following example, the `library_function` performs some presumably complicated computation that involves division. The `user_code` attempts to call the `library_function` and handles the exception by printing an error message.

```
(define library_function
  (lambda (a b c d)
    (let ((divide (lambda (x y)
                    (if (zero? y)
                        (throw)
                        (/ x y)))))
      (divide (divide (divide a b) c) d))))

(define user_code
  (lambda ()
    (try_catch
     (lambda () (library_function 8 2 0 2))
     (lambda () (error 'user_code "Division by zero in library")))))
```

This model of exceptions is somewhat limited as it is impossible for `user_code` to resume `library_function` at the point that caused the exception.

Assume that for the particular purposes of `user_code`, it is acceptable to treat `(/ 1 0) = 100`. Wouldn't it be nice if `user_code` could somehow resume the computation in `library_function` passing the value 100 as the result of the offending division by zero?

Of course, this is possible in Scheme. Here is the basic idea: when invoking an exception handler, pass a continuation to the handler that can be used to resume the computation that threw the exception. Using this idea, modify the definitions of `global_handler`, `throw`, and `try_catch` to allow for resumable exceptions. A call to `(user_code)` below should evaluate to 50:

```
(define user_code
  (lambda ()
    (try_catch
     (lambda () (library_function 8 2 0 2))
     (lambda (resume) (resume 100)))))
```

Extra Credit (10 points)

Here is an example of using inner classes in Java:

```
abstract class Function {
  abstract int apply (int i);
}

class A {
  int[] elems = {1, 2, 3, 4, 5};

  void map (Function f) {
    for (int i=0; i<elems.length; i++) {
      elems[i] = f.apply(elems[i]);
    }
  }

  public String toString () {
    String res = "[ ";
    for (int i=0; i<elems.length; i++) {
      res += elems[i] + " ";
    }
    return res + "]";
  }
}

class Inner {
  public static void main (String[] args) {
    A x = new A();
    System.out.println(x); // prints [ 1 2 3 4 5 ]
    x.map(new Function () {
      int apply (int i) { return i+1; }
    });
    System.out.println(x); // prints [ 2 3 4 5 6 ]
    x.map(new Function () {
      int apply (int i) { return i*i; }
    });
    System.out.println(x); // prints [ 4 9 16 25 36 ]

    int global = 3;
    x.map(new Function () {
      int apply (int i) { return global + i; } // does not compile
    });
  }
}
```

The last attempt to use an inner class is rejected with the following error message:

```
Inner.java:38: Attempt to use a non-final variable global from a
different method. From enclosing blocks, only final local variables
are available.
      int apply (int i) { return global + i; } // does not compile
```

Can you explain why the code is rejected and the difficulties in implementing it if it were accepted?