Crash Course Session 1—Recursion, Iteration, Lists

CS 1102S—Data Structures and Algorithms

Martin Henz

14 January, 2010
<table>
<thead>
<tr>
<th>Languages and Language Processors</th>
<th>Is Scheme Compiled or Interpreted?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursion and Iteration</td>
<td>T-Diagrams</td>
</tr>
<tr>
<td>Lists</td>
<td>Interpreters</td>
</tr>
<tr>
<td></td>
<td>Translators</td>
</tr>
<tr>
<td></td>
<td>Combinations</td>
</tr>
</tbody>
</table>
Languages vs Implementation

Programming language
Programming languages are the languages in which a programmer writes the instructions that the computer will ultimately execute. *Encyclopedia Britannica*

Programming system
Set of tools that help achieving this execution.

Same language, different tools
For the same language, different tools are available for different purposes.
T-Diagrams

x86 Processor

Program “GTA” (x86 code)

“GTA” running on x86
Interpreter

- Interpreter is program that executes another program
- The interpreter’s *source language* is the language in which the interpreter is written
- The interpreter’s *target language* is the language in which the programs are written which the interpreter can execute
Interpreters

Interpreter for Scheme (x86 machine code)

Scheme

x86
Interpreting a Program

Scheme program “GTA” running on x86 using interpretation
Hardware Emulation

“GTA” x86 executable running on a PowerPC using hardware emulation
Translators

- Translator translates from one language—the *from-language*—to another language—the *to-language*
- Compiler translates from “high-level” language to “low-level” language
- De-compiler translates from “low-level” language to “high-level” language
T-Diagram of Translator

Scheme → C

C

x86

Scheme-to-C compiler written in x86 machine code
Compilation

Compiling “GTA” from Scheme to C
Two-stage Compilation

Compiling “GTA” from Scheme to C to x86 machine code
Compiling a Scheme-to-x86 compiler from C to x86 machine code
Compiling “HelloWorld” from Java to JVM code, and running the JVM code on a JVM running on an x86
Factorial Function

```scheme
(define (factorial i)
  (if (<= i 0)
      1
      (* i (factorial (- i 1))))
)
Factorial In Java

```java
public static int factorial(int i) {
    if (i <= 1) {
        return 1;
    } else {
        return i * factorial(i - 1);
    }
}
```
Iteration vs Recursion in Scheme

Iteration
A (recursive) Scheme function is **iterative**, if the recursive call is always the last thing to do in its body.

Is Factorial Iterative?
No! \((\ast \ i \ (\text{factorial} \ (- \ i \ 1)))\)

In Java: \(i \ \ast \ \text{factorial}(i \ - \ 1);\)
Iterative Factorial Function In Scheme

(define (iterfactorial i acc)
  (if (<= i 1)
      acc
      (iterfactorial (- i 1) (* acc i))))

(define (iterativefactorial i)
  (iterfactorial i 1))
Iterative Factorial Function In Java?

```java
private static int iterFactorialTry(int i, int acc)
{
    if (i <= 1) {
        return acc;
    } else {
        return iterFactorialTry(i - 1, acc * i);
    }
}

public static int iterativeFactorialTry(int i) {
    return iterFactorialTry(i, 1);
}
```
The Sad Truth about Java

Java has no iterative recursion!
Every function call requires space on a Java runtime stack.

Recursion is always recursive!
A recursive function in Java will never use constant space.
Loops to the rescue!

Loop constructs
Java contains loop constructs such as while and for.

Iteration in Java
Iteration can only be achieved using loops in Java.
Iterative Factorial Function In Java

```java
public static int iterativeFactorial(int i) {
    int acc = 1;
    while (i > 1) {
        acc = acc * i;
        i = i - 1;
    }
    return acc;
}
```
Lists in Scheme and Java

Lists in Scheme
Built-in, using cons, car, cdr, ’() , null ?.

Lists in Java
There is a List interface in Java, see http://java.sun.com/j2se/1.5.0/docs/api/java/util/List.html

Start with List of Integers
Here, we study a restricted form of lists first: IntList .
Lists in Scheme

(cons 1 2) ;; a pair
'(()) ;; an empty list
(cons 7 '(()) ;; a list with one integer
(cons 4 (cons 9 '(()))) ;; a list with two integers
Built-in Operations on Lists in Scheme

\[
\text{'}() \quad \text{;; an empty list}
\]

\[
\text{(cons 1 2)} \quad \text{;; a pair}
\]

\[
\text{(car alist)} \quad \text{;; first component (head)}
\]

\[
\text{(cdr alist)} \quad \text{;; second component (tail)}
\]

\[
\text{(null? alist)} \quad \text{;; whether list is empty}
\]
Operations on Lists of Integers in Java

```java
public static IntList nil = new IntList()
public static IntList cons(int i, IntList list)
public static int car(IntList list)
public static IntList cdr(IntList list)
public static boolean isNil(IntList list)
public static void print(IntList list)
```

These functions are available in the library (class) IntList.
Some Cheating (for convenience)

```java
public static IntList intList(int[] elements)
```
Length in Scheme

\[
\text{(define (length xs)}
\begin{align*}
\text{(if (null? xs) } & \text{ 0} \\
& (+ 1 (\text{length (cdr xs)})))
\end{align*}
\]
Length in Java

```java
public static int length(IntList aList) {
    if (IntList.isNil(aList)) {
        return 0;
    } else {
        return 1 + length(IntList.cdr(aList));
    }
}
```
Iterative Length in Scheme

\[
(\textbf{define} (\textit{iterlength} \textit{alist} \textit{acc})
  (\textbf{if} (\textit{null?} \textit{alist})
    \textit{acc}
    (\textit{iterlength} (\textit{cdr} \textit{alist}) (+ \textit{acc} 1))))
\]

\[
(\textbf{define} (\textit{iterativelength} \textit{alist})
  (\textit{iterlength} \textit{alist} 0))
\]
Iterative Length in Java?

```java
public static int iterativeLengthTry(IntList aList) {
    return iterativeLengthTry(aList, 0);
}
```

```java
public static int iterativeLengthTry(IntList aList, int acc) {
    if (IntList.isNil(aList)) {
        return acc;
    } else {
        return iterativeLengthTry(aList.cdr(aList), acc + 1);
    }
}
```
Iterative Length in Java!

```java
public static int iterativeLength(IntList aList) {
    int acc = 0;
    while (!IntList.isNil(aList)) {
        aList = IntList.cdr(aList);
        acc++;
    }
    return acc;
}
```
Append in Scheme

```
(define (append alist anotherlist)
  (if (null? alist)
      anotherlist
      (cons (car alist)
            (append (cdr alist) anotherlist))))
```
Append in Java

```java
public static IntList append(IntList aList, IntList anotherList) {
    if (IntList.isNil(aList)) {
        return anotherList;
    } else {
        return IntList.cons(IntList.car(aList),
                            append(IntList.cdr(aList),
                                   anotherList));
    }
}
```
Naive Reverse in Scheme

(define (naivereverse alist)
  (if (null? alist)
      ()
      (append (naivereverse (cdr alist))
              (cons (car alist) '()))))
Naive Reverse in Java

```java
public static IntList naiveReverse(IntList aList) {
    if (IntList.isNil(aList)) {
        return IntList.nil;
    } else {
        return append(naiveReverse(IntList.cdr(aList)),
                      IntList.cons(IntList.car(aList),
                                    IntList.nil));
    }
}
```
Square All in Scheme

(\texttt{define \hspace{1em}} (\texttt{squareall \hspace{1em}} \texttt{alist})
 (\texttt{\hspace{1em}} (\texttt{if \hspace{1em}} (\texttt{null? \hspace{1em}} \texttt{alist})
    '() 
    (cons \hspace{1em} (* \hspace{1em} (\texttt{car \hspace{1em}} \texttt{alist}) \hspace{1em} (\texttt{car \hspace{1em}} \texttt{alist}))
     (\texttt{squareall \hspace{1em}} (\texttt{cdr \hspace{1em}} \texttt{alist}))))))

\textbf{Examples}

Square All in Scheme
Square All in Java

```java
public static IntList squareAll(IntList aList) {
    if (IntList.isNil(aList)) {
        return IntList.nil;
    } else {
        return IntList.cons(IntList.car(aList) * IntList.car(aList),
                             squareAll(IntList.cdr(aList)));
    }
}
```
Sum in Scheme

(define (sum alist)
  (if (null? alist)
      0
      (+ (car alist) (sum (cdr alist)))))
Sum in Java

```java
public static int sum(IntList aList) {
    if (IntList.isNil(aList)) {
        return 0;
    } else {
        return IntList.car(aList) + sum(IntList.cdr(aList));
    }
}
```
Next Session

- More built-in types
- Loops
- Arrays