Slide 1

Computing 1A (COMP1011)

List Processing and Debugging #3

Manuel M. T. Chakravarty

Slide 2

Assignment 1

- Read the Message Board. It has important additional information.
- There are two bugs in the examples of `getCell` and `setCell`

Slide 3

A Brief Look at Sorting

- Sorting of a list of elements is a frequently used operation
- One of the simplest sorting algorithms is insertion sort
- How does insertion sort work?
- Insertion sort's central component is a function

```
insertSorted :: Ord a => a -> [a] -> [a]
```

which given a sorted list, inserts a single new element such that the ordering is respected

- For example,

```
insertSorted 7 [2, 4, 6, 9]  ⇒  [2, 4, 6, 7, 9]
```

◇

- How can we use `insertSorted` to sort an unsorted list?
  ① Given an unsorted list
  ② Start with an empty result list
  ③ Successively, `insertSorted` each element of the unsorted list into the result list

◇

Slide 4

```
insertSorted            :: Ord a => a -> [a] -> [a]
insertSorted e []        = [e]
insertSorted e (x:xs)
  | e < x                = e : x : xs
  | otherwise            = x : insertSorted e xs

isort          :: Ord a => [a] -> [a]
isort []        = []
isort (x:xs)    = insertSorted x (isort xs)
```

Slide 5

Testing and Debugging

What is Testing?

- Any attempt to determine whether a program meets its specification by executing the program

What is important when testing?

- Coverage:
  - How much of the code is exercised during testing?
  - How many of the possible control paths are exercised?
  - How many of the possible inputs are covered?
  - Are all corner cases covered?
- Test small and simple components first, then complex ones
- Testing is always imperfect. As Edsger Dijkstra put it, "Program testing can be used to show the presence of bugs, but never their absence."

What is Debugging?

- Finding and eliminating a bug discovered by testing

Slide 6

Debugging

Different methods of debugging:

- Code inspection:
  - Given the failing test case, inspect the program code
  - Stepwise evaluation on paper
- Tracing:
  - Adding trace output that prints intermediate values
  - Tool collecting and replaying program traces
- Code animation:
  - Source code debugger
  - Program can be stopped and current values be inspected

Slide 7

Adding Trace Output in Haskell:

- Import the module `Debug.Trace`
- It provides the function

```
trace :: String -> a -> a
```

- It prints a message before evaluating its second argument; e.g.,

```
trace "Trace Message" ⟨some expression⟩
```

✧(enter msg)

Converting non-string values to strings:

- The Prelude provides the function

```
show :: Show a => a -> String
```

✧(print args)

Slide 8

Tracing Two Bugs in Insertion Sort

Bug #1: Forgotten base case:

- Let's assume we forget to add the base case for `isort`
- Trace the invocations of `isort` ✧

Bug #2: Forgot to add head of list to result:

- Let's assume we drop `x` in second case of `insertSorted`
- Trace arguments and results of calls to `insertSorted` from `isort`: wrapper function ✧

Slide 9

Evaluation Order and Laziness

Haskell is a lazy language:

- ➜ It computes only what is necessary, when it becomes necessary
  - `head [3, 2 ‘div‘ 0]`
  - Unused argument ✧
- ➜ For example, enables computations on infinite structures
- ➜ But can be confusing during tracing

Enforcing the usage of a value:

- ➜ We can enforce the use of a value with

```
seq :: a -> b -> b
```

- ➜ We use it in the form

⟨force this⟩ `‘seq‘` ⟨actual computation⟩

✧