Topics: - work
- privacy
- property

CSC104 winter 2013
Why and how of computing
week 11

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/104/F12/
416-978-5899

Text: Picturing Programs
Outline

work

Notes
who’s got the better deal?

My parents → 40 hour work.

14 hour days 1800s (Industrial Revolution)

life with, or without, computers — which works better?

What went wrong?

How many hours per week do you expect to work? What about your parents/grandparents? Explain labour-saving devices
previous experience

does technological change automatically improve lives?

More things produced/hour
in some ways better, at least “uniform”

land cleared of people
provides wool and hands
for emerging factories

overall

Some economists report that production actually dropped for the first few decades of the Industrial Revolution. The working day certainly lengthened — to 12 or even 14 hours!
automation/computerization

what has the effect been?

Ford assembly, then and now where’d everybody go?

In 1940s, a car “cost” 35 hours. Now it’s 19 hours.
Hardware effects: early 60s

Storing information gets smaller, cheaper, faster by the decade...

What’s the effect on working lives?
do long hours matter?

... if you have an ergonomic chair and a fuzzball table?

Check out why crunch mode doesn’t work. Chart productivity/hour over a long day.
don’t operate heavy machinery...
after working (too much)

prolonged sleeplessness affects
motor skills and
judgement

> 24 hours awake
low-level skills
judgement.
utopia, dystopia?

- work 0.5 h/day
- drive

new jobs, flying cars, or no jobs, or retirement?

no more work?
not just how long, but where

telecommuting \rightarrow + flexibility
downside \rightarrow always available

trade traffic for flexibility and time?
; flatten : list -> list
(define (flatten L)
  (cond
   [(cons? L) (apply append (map flatten L))]
   [else (list L)]))

; predict what (flatten 3) does

; predict what (flatten (list 3)) does

; predict what (flatten (list 1 2 (list 3))) does
depth

; depth : list -> number
(define (depth L)
  (cond
   [(cons? L) (+ 1 (apply max (map depth L)))]
   [else 0]))

; predict what (depth 3) does

; predict what (depth (list 3 4)) does

; predict what (depth (list 3 4 (list 5 6))) does