COMPSCI 1JC3
Introduction to Computational Thinking
Fall 2017

05 Recursion 1

William M. Farmer

Department of Computing and Software
McMaster University

October 3, 2017

Admin

- Student Accessibility Services (SAS) is looking for a note taker for COMPSCI 1JC3; see

  `https://sas.mcmaster.ca/notetaking/#notetaking-for-others`

  for details.
- Assignment 2 is posted and due Friday, October 20.
- Office hours: To see me please send me a note with times.
- Are there any questions?

M&Ms (iClicker)

What is your relationship with M&Ms?

A. They are great; I submit them every week.
B. I submit them only to get the marks.
C. They are valuable, but I am too lazy to always submit them.
D. They are of no value to me; I don't submit them.

Advice

1. Use the midterm recess to both catch up and to relax!
2. Read your McMaster e-mail regularly!

Review

1. Propositional formulas.
2. Defining any boolean function using not, and, and or.
3. Conditional expressions.
4. Bitwise operations.
5. While loops.

What is Recursion?

- Recursion is a method of defining something (usually a function) in terms of itself.
  - One of the most fundamental ideas of computing.
  - An alternative to iteration (loops).
  - Can make some programs easier to write, understand, and prove correct.
  - Learning to use recursion is one of the very best ways to develop computational thinking!
- In almost all programming languages, functions can be defined by recursion.
- The use of recursion requires care and understanding.
  - Recursive definitions can be nonsensical (i.e., nonterminating).
  - Sloppy use of recursion can lead to total confusion.
  - Correctness is proved by (mathematical) induction.

Recursion Question (iClicker)

Given a string `s` as input, what does the function `mystery` return?

```
mystery :: String -> String

mystery s
  | s == ""     = s
  | otherwise   = (mystery (tail s)) ++
                  [head s]
```

A. The reverse of the string `s`.
B. The string `s`.
C. Every other character from the string `s`.
D. The empty string.
E. None of the above.

How does Recursion Work?

A problem is solved by recursion as follows:

1. The simplest instances of the problem are solved directly.
2. Each other instance of the problem is solved by reducing the instance to simpler instances of the problem.
3. As a result of 1 and 2, each instance can be solved by reducing the instance to simpler instances and then reducing these instances to simpler instances and continuing in this fashion until a simplest instance is reached, which has already been solved.

Notice that recursion employs a divide and conquer strategy.

How does Recursion Work with Functions?

- In the typical recursive definition of a function:
  - An instance of the function is a set of inputs for the function.
  - Each instance I is assigned a natural number $n(I)$.
  - An instance I is a "simplest instance" if $n(I) = 0$.
  - An instance I' is "simpler than an instance I if $n(I') < n(I)$.
- A recursive definition of a function is nonsensical if some instance I is reduced to an instance I' such that I' is not simpler than I, i.e., $n(I') \geq n(I)$.

Example: Reverse

```
reverse1 :: Eq a => [a] -> [a]

reverse1 x
  | x == []      = x
  | otherwise    = (reverse1 (tail x)) ++
                   [head x]


reverse2 :: [a] -> [a]

reverse2 []      = []
reverse2 (x:xs)  = (reverse2 xs) ++ [x]
```


Example: Factorial

```
factorial :: Integer -> Integer

factorial n
  | n == 0   = 1
  | n > 0    = n * factorial (n - 1)
```


Example: Bogus Recursive Functions

```
bogus1 :: Integer -> Integer

bogus1 n = bogus1 n


bogus2 :: Integer -> Integer

bogus2 n = n * bogus2 (n - 1)


bogus3 :: Integer -> Integer

bogus3 n
  | n == 0   = 1
  | n > 0    = n * bogus3 (n + 1)
```