CMPT 120

Topic: Recursion – Part 2
Last Lectures

• Recursion
  • Examples of recursion occurring in the real world
  • Examples of recursion occurring in the mathematical world
  • What is recursion
  • Iterative code versus Recursive code
  • Demo of recursion
Learning outcomes

At the end of this course, a student is expected to:

• Describe the concept of *recursion*, of recursive definitions and recursive functions.

• Use *recursion* to solve problems:
  • Use box tracing to predict the result of simple recursive code (drawing a call stack)
  • Design recursive code
  • Design recursive functions that recurse, for example, on lists and strings
Today’s Menu

1. Design and implement a recursive function
2. Box trace (hand trace with boxes) a recursive function to figure out what it produces
How to design/implement a recursive function?

• First, we need to think recursively

http://www.notonthehighstreet.com/britishandbespoke/product/paint-your-own-russian-dolls-for-children
Think recursively … hum! Easy for you to say!

- Let’s give it a try!

- **Problem Statement**
  - Write a Python function that returns the factorial of a number passed as a parameter
How? Remember the 2 properties

• Remember the two properties we saw in the definition of recursion:
  1. A simple base case (or cases)—a terminating scenario that does not use recursion to produce an answer
  2. A set of rules that reduce all other cases toward the base case
Using these 2 properties as a procedure

1. **Base Case**
   - Its purpose is to **stop** the function from calling itself indefinitely
     - So it does not use recursion to produce an answer

2. **Recursive Case**
   - Use recursion to eventually produce an answer
     - So, this is where the function calls itself with a modified (often smaller) form of the problem
     - How do we get this “modified form of the problem”
       - -> **A set of rules**

3. **Function interface**
   - Function name
   - Parameters
   - Returned value
General form of recursive function

def recursiveFunction( ... ):  
   if (base case):  <- may be > 1  
      result = base case  
   else:  
      result = expression including recursive call(s)  

return result
Let’s give it a go: factorial function

• Recursive definition of factorial:

  • Base cases:

  • Rule:
Let's give it a go: factorial function
Let’s test our recursive function

- How?
- By box tracing our recursive function
  - Box tracing -> hand tracing using stack frames (boxes)
    - Basically, we mimic what the visualizer does but on paper
Box tracing a recursive function

- Knowing how to hand/box trace code is very important
- Even more important when dealing with recursive functions
- **Why?**
- Because following the execution of recursive functions can become very complicated very quickly
  - Box tracing allows us to figure out **what a recursive function does** (keeping track of its parameters and local variables), **what it produces** (keeping track of its returned value) and **where the execution flow returns to** in a systematic way
Box trace of factorial function
That was fun, no?

• Let’s try again!

• **Problem Statement**
  • Write a Python function that multiplies two numbers without using the multiplication operator
Let's give it a go: multiplication function

- Brainstorming:
Let’s give it a go: multiplication function

- Recursive definition of a multiplication:
  - Base cases:
  - Rule:
Let's give it a go: multiplication function
Another example: find a character in a string

- **Problem Statement**
  - Write a Python function that finds a character in a string
Another example: find a character in a string

• Base case:

• Rule:
Another example: find a character in a string
Summary

1. Design and implement a recursive function
2. Box trace (hand trace with boxes) a recursive function to figure out what it produces
Next Lecture

• Revisiting List