Recursion – Basic Concepts

Arnab Ganguly, Assistant Professor
Department of Computer Science, University of Wisconsin – Whitewater
Data Structures (CS 223)

1 Recursion

Recursion is the ability to define something in terms of itself. We are only going to concentrate on recursive functions from the context of programming.

The idea behind recursion is simple: to compute the value of any function, we break the function down into that of computing a simpler function (using the so-called *recursive rule*), compute the simpler function, and then use the computed value to compute the value of the original function. Often this breaking down is carried on until we reach a function which is so simple (the so-called *base-case*) that it can be directly computed.

The following are the two main components of any recursive code:

- **Base-case:** Smallest function, which is simple enough so that the value of the function is obvious or easy to compute.
- **Recursive Rule:** A set of rules by which we can compute the answer to a function, given the answer to a suitable smaller function.

1.1 Example: Factorial

$factorial(n)$ = product of all positive integers from 1 to $n = 1 * 2 * \cdots * n$. Then,

Base-case: $factorial(1) = 1$
Recursive Rule: $factorial(n) = 1 * 2 * \cdots * (n-1) * n = n * factorial(n-1)$

Computing the factorial of a number

```
int factorial(int n) {
      if (n == 1)
            return 1;
      else
            return n*factorial(n-1);
}
```

1.2 Example: Sum of Digits of a Number

Consider a number 5671. Sum of its digits is $5 + 6 + 7 + 1 = 19$. Let $sumDigits(num)$ be the sum of the digits of the number. Then,

Base-case: $sumDigits(n) = n$ when $n < 10$
Recursive Rule: $sumDigits(n) = sumDigits(num/10) + num\%10$ when $n \geq 10$

Computing the sum of the digits of a number

```
int sumDigits(int n) {
    if (n < 10)
        return n;
    else
        return sumDigits(num/10) + num%10;
}
```

1.3 Example: Fibonacci Sequence (Two Base-cases)

The fibonacci sequence is: $1, 1, 2, 3, 5, 8, \ldots$. The nth term in the fibonacci sequence is the sum of $(n-1)$th and the $(n-2)$th terms of the series.
Define $fibonacci(n)$ = the nth term of the fibonacci sequence. Then,

Base-case: $fibonacci(1) = 1$ and $fibonacci(2) = 2$
Recursive Rule: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$

Finding the n^{th} Fibonacci number

```
int fibonacci(int n) {
    if (n <= 2)
        return 1;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Note: This is an awful use of recursion. The complexity is $O(\phi^n)$, where $\phi = \frac{\sqrt{5}+1}{2}$ is the Golden Ratio. To get an idea as to why this happens, draw a recursive tree for say $fibonacci(10)$ and you shall see a lot of repeated computation.

1.4 Example: Sum of Numbers in an Array

Let $sumArray(k)$ = sum of the first k numbers in an array $A[0, n-1]$. Then,

Base-case: $sumArray(1) = A[0]$
Recursive Rule: $sumArray(k) = A[k-1] + sumArray(k-1)$

Computing the sum of numbers in an array

```
int sumArray(int k) {
    if (k == 1)
        return A[0];
    else
        return A[k-1] + sumArray(k-1);
}
```

1.5 Example: Minimum Value in an Array

Let $minArray(k)$ = minimum value among the first k numbers in an array $A[0, n-1]$. Then,

Base-case: $minArray(1) = A[0]$
Recursive Rule: $minArray(k)$ = minimum of $A[k-1]$ and $minArray(k-1)$

Computing the minimum value in an array

```
int minArray(int k) {
    if (k == 1)
        return A[0];
    else {
        int min = minArray(k-1);
        if (min < A[k-1])
            return min;
        else
            return A[k-1];
    }
}
```

1.6 Example: Sum of all positive integers upto n that are divisible by 3

If $n = 9$, the desired sum is $(3 + 6 + 9) = 18$. If $n = 10$, the desired sum is $(3 + 6 + 9) = 18$. If $n = 14$, the desired sum is $(3 + 6 + 9 + 12) = 30$.

Define $sumPosDiv3(n)$ = the sum of all positive integers upto n that are divisible by 3. Then,

Base-case: if $n < 3$, $sumPosDiv3(n) = 0$.
Recursive Rule: If $n\%3 == 0$, $sumPosDiv3(n) = n + sumPosDiv3(n-3)$ else, $sumPosDiv3(n) = sumPosDiv3(n - n\%3)$.

Computing the sum of all positive integers upto n that are divisible by 3

```
int sumPosDiv3(int n) {
    if (n < 3)
        return 0;
    else {
        if (0 == n %3)
            return n + sumPosDiv3(n-3);
        else
            return sumPosDiv3(n-n%3);
    }
}
```