# Lecture 14.3

## Topics:

1. **Recursion – Review**

---

## 1. Recursion – Review

Consider a mathematical equation of,

```
f(n) = 2 * f(n – 1) + n                                    (Eqn. 1)
```

This is called a recurrence equation where the solution may be obtained from repeatedly substituting values for `n` with some initial condition such as `f( 0 ) = 0`. In programming, one can use a recursive technique to implement the above equation (and thus, this is called recursion).

In general, a recursive method is the one that calls itself. Any recursive implementation must observe two rules:

i. **Base Case**

   There must be at least one base case without recursion.

ii. **Updating Current Value**

   To continue with recursion, the recursive computation must always have calls to itself with different arguments that trend toward the base case.

### 1.1 Examples

Let's look at the examples below where a method displays a simple message.

**Example 1**

```
/**
 *Program Name:  cis27L1431.c
 *Discussion:    Recursion
 */
#include <stdio.h>

void printClassInfo(void);
void display(int count);

int main() {
  printClassInfo();

  printf("\nRecursive printing: \n");
  display(3);

  return 0;
}

/**
 *Function Name: printClassInfo()
 *Description  : Printing the class information
 *Pre          : Nothing (nothing is sent to this function)
 *Post         : Displaying class info on screen
 */
void printClassInfo() {
  printf("\n\tCIS 27 : Data Structures");
  printf("\n\tLaney College.\n");
```

```
  return;
}

/**
 *Function Name: display()
 *Discussion:    Recursive function
 *Pre:           None
 *Post:          Displaying results recursively
 */
void display(int count) {
  if (count > 0) {
    printf("\tGreeting! -- %d\n", count);
    display(count - 1);
  }

  return;
}
```

**OUTPUT**

```
        CIS 27 : Data Structures
        Laney College.


Recursive printing:
        Greeting! -- 3
        Greeting! -- 2
        Greeting! -- 1
```

**Example 2**

```
/**
 *Program Name:  cis27L1432.c
 *Discussion:    Recursion
 */
#include <stdio.h>

void printClassInfo(void);
void display2(int count);

int main() {
  printClassInfo();

  printf("\nRecursive printing: \n");
  display2(3);

  return 0;
}

/**
 *Function Name: printClassInfo()
 *Description  : Printing the class information
 *Pre          : Nothing (nothing is sent to this function)
 *Post         : Displaying class info on screen
 */
void printClassInfo() {
```

```
  printf("\n\tCIS 27 : Data Structures");
  printf("\n\tLaney College.\n");

  return;
}

/**
 *Function Name: display2()
 *Discussion:    Recursive function
 *Pre:           None
 *Post:          Displaying results recursively
 */
void display2(int iCount) {
  printf("Calling display2() -- iCount : %d\n", iCount);
  if (iCount > 0) {
    printf("\tGreeting! -- %d\n", iCount);
    display2(iCount - 1);
  }
  printf("Returning display2() -- iCount : %d\n", iCount);

  return;
}
```

**OUTPUT**

```
        CIS 27 : Data Structures
        Laney College.


Recursive printing:
Calling display2() -- iCount : 3
        Greeting! -- 3
Calling display2() -- iCount : 2
        Greeting! -- 2
Calling display2() -- iCount : 1
        Greeting! -- 1
Calling display2() -- iCount : 0
Returning display2() -- iCount : 0
Returning display2() -- iCount : 1
Returning display2() -- iCount : 2
Returning display2() -- iCount : 3
```

**Example 3**

Consider the **n** factorial (**n!**)where **n** is an integer. A recursive expression may be given as followed,

```
factorial n) = n * factorial(n – 1)    where n > 0          (Eqn 2)
             = 1                       where n = 0
```

How would the above expression be turned into actual code?

### 1.2 Direct and Indirect Recursive Methods

The above examples are termed direct recursion where a method calls itself repeatedly. There are cases where a sequence of several methods would also provide recursive behavior. This is called indirect recursive.

For example, method A calls method B, method B calls method C, and method C calls method A. Because of the recursive nature, these methods must observe the two basic rules of above.

### 1.3 Recursion versus Iteration

Any algorithm that can be implemented with recursive code can also be implemented using iterative structure. Thus, there may be questions when recursive structures were in used. Why?

In many cases, recursive implementation would be less efficient than iterative due to the overhead required in the method calls. Iterative structure would not inherit this function overhead.

In the current and future generations of computers, the consideration of speed and memory may not be that gravely important for some systems. Thus, the recursive implementation would still be an attractive option. In many of these problems, recursive approach would present an elegant and obvious solution such as **QuickSort** algorithm.

### 1.4 Recursion -- Examples

**Example 4**

**Factorial:**

$$f\ n) = n * f(n - 1) \quad \text{(Eqn. 3)}$$
$$= 1 \text{ if } n = 0$$

```
int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return (n * factorial(n - 1));
  }
}
```

**Example 5**

**Fibonacci:**

$$f(n) = f(n - 1) + f(n - 2) \quad \text{(Eqn. 4)}$$
$$= n \text{ if } n \text{ is } 0 \text{ or } 1$$

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