Lecture 25

Topics:

1. **Recursion – Introduction**
2. **Data Validation – Brief**

---

1. Recursion – Introduction

Consider a mathematical equation of,

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

2.1 Examples

Let us look at the examples below where a function displays a simple message.

Example 1

```c
/**
 * Program Name: cis27L2511.c
 * Discussion:   Recursion
 */

// Header/include File
#include <stdio.h>

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

// Application Driver
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: cis27L2512.c
 * Discussion:   Recursion
 */

// Header/include File
#include <stdio.h>

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

// Application Driver
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 follows,

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

How would the above expression be turned into actual code?

2.2 Direct and Indirect Recursive Functions

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

For example, function **a()** calls function **b()**, function **b()** calls function **c()**, and function **c()** calls function **a()**. Because of the recursive nature, these functions must observe the two basic rules of above.

2.3 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 function calls. Iterative structure would not inherit this function overhead.

However, in the current and future generations of computers, the consideration of speed and memory may not be that gravely important for some systems and/or applications. 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)                                  (Expr. 3)
     = 1              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)                          (Expr. 4)
          = n                if n is 0 or 1
```

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

2. Data Validation – Brief

Recall that writing a computer program is to provide a solution to some given problem. A general computer program execution may be depicted in **Figure 4** below.

The **Data Validation** block will validate the input data before being sent to the **Processing** block. If designed properly in applications, incorrect data will be rejected and new set of input data may be re-entered or selected and then validated again.

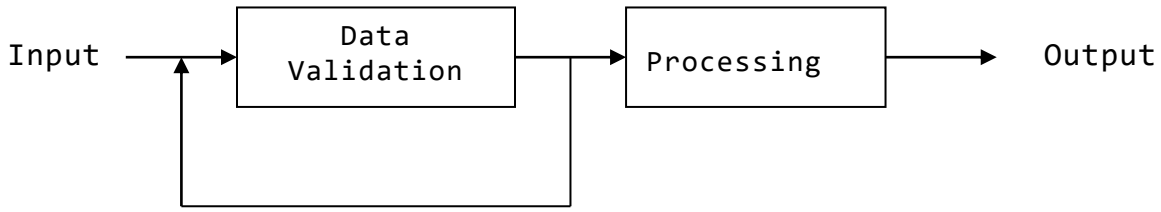

Figure 4. Input/output and data processing

Using the input data, the processing block has all the algorithms and steps to produce the desired result (i.e., output). The processing block(s) will be designed and implemented specifically for individual applications.

The most general form of input data would be **typed/entered** through keyboard or given as character strings (or just strings). These strings should be validated and converted to numerical values. What kinds of numerical values are they convertible to?