CS 1160, Computing Fundamentals-Lab 2020/2021 *Week of December. 13- 17, 2020*

FUNCTIONS IN C -2

In this assignment we are going to demonstrate:

- Call by Reference in C language - Passing Arrays to Functions.
- The use of local and global variables in C language.
- Recursion functions in C language.

Part 1: Calling Functions in C language (Call by Value and Call by Reference).

Call by value:

- Copy of argument passed to function.
- Changes in function do not effect original.
- Use when function does not need to modify argument.
- Avoids accidental changes.

For example, below program prints:

x is 7 before the call to fun
x is 9 in fun
x is 7 after the call to fun

```
#include<stdio.h>
// function prototype, also called function declaration
void fun(int x);

// main function, program starts from here
int main() {
   int x = 7;
   printf("x is %d before the call to fun\n", x);
 // function call
   fun(x);
   printf("x is %d after the call to fun\n", x);
   return 0;
}

// function definition
void fun(int x) {
   x = x + 2;
   printf("x is %d in fun\n", x);
}
```

Call by reference:

- Passes original argument.
- Changes in function effect original.
- Only used with trusted functions.

In this assignment we are going to demonstrate "Pass Arrays to Functions" as an example of Call by reference.

```
#include<stdio.h>
// function prototype, also called function declaration
void Modify (float [], int);

// main function, program starts from here
int main() {
   float array[] = {1, 6, 8, 5, 9};
   int i;
   printf("Array elements before the call to Modify-fun:\n");
   for (i = 0; i < 5; i++)
      printf("%.2f\t", array[i]);
   // function call
   Modify(array, 5);
   printf("\nArray elements After the call to Modify-fun:\n");
   for (i = 0; i < 5; i++)
      printf("%.2f\t", array[i]);
}

// function definition
void Modify(float array[], int size)
{
   int i;
   for (i = 0; i < size; i++) {
      array[i] = array[i] + 2;
   }
   printf("\nArray elements in the Modify-fun:\n");
   for (i = 0; i < size; i++)
      printf("%.2f\t", array[i]);
}
```

Above program prints:

```
Array elements before the call to Modify-fun:
1.00    6.00    8.00    5.00    9.00
Array elements in the Modify-fun:
3.00    8.00    10.00   7.00    11.00
Array elements After the call to Modify-fun:
3.00    8.00    10.00   7.00    11.00
```

Part 2: The use of local and global variables in C language.

local variables: are declared within a function and can be only accessed from within the function.
global variables: are declared outside of a function and thus have a program scope; i.e., they can be accessed anywhere in a program.
For example, below program prints:

```
Local: 1
Global: 1
Local: 1
Global: 2
```

```c
#include <stdio.h>
int Local();
int Global();
int C = 0; // global variable

int main() {
   printf("Local: %i\n", Local());
   printf("Global: %i\n", Global());
   printf("\n");
   printf("\n");
   printf("Local: %i\n", Local());
   printf("Global: %i\n", Global());
   printf("\n");
   printf("\n");
}

int Local() {
   int A = 0; // local variable
   A++;
   return (A);
}
int Global() {
   C++;
   return (C);
}
```

Static variables are initialized only once. The compiler persists with the variable till the end of the program. Static variables can be defined inside or outside the function. They are local to the block. The default value of static variables is zero. The static variables are alive till the execution of the program. For example, below program prints: *1, 2*

```
#include<stdio.h>
int fun()
{
  static int count = 0; // static variable
  count++;
  return count;
}
int main()
{
  printf("%d , ", fun());
  printf("%d ", fun());
  return 0;
}
```

Part 3: Recursion functions in C language**.**

- Functions that call themselves.
- Can only solve a base case.
- Eventually base case gets solved
  - Gets plugged in, works its way up and solves whole problem.

Example Using Recursion: The Fibonacci Series:

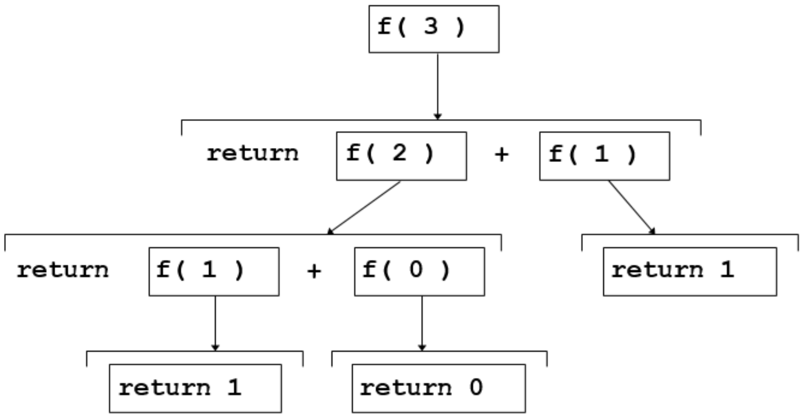
Fibonacci series: 0, 1, 1, 2, 3, 5, 8...

- Each number is the sum of the previous two.
- Can be solved recursively: **Fibonacci (n) = Fibonacci (n - 1) + Fibonacci (n – 2)**

Code for the **Fibonacci** function:

```
long Fibonacci (long n)

{

        if (n == 0 || n == 1) // base case

  return n;

        else

  return Fibonacci (n - 1) + Fibonacci (n – 2);

}
```

- Set of recursive calls to function **Fibonacci:**



TASKS:

1. Write a program to find the maximum value in an array of floats of size 15. The processing of finding the maximum must be in a separate function.

2. Write a C program that displays transpose of a matrix. It is obtained by interchanging rows and columns of a matrix. For example, if a matrix is:

   1 2 3
   4 5 6

   Then transpose of above matrix will be:

   1 4
   2 5
   3 6

   You have to create the following functions, and then call them in the main:

   a) Function Transpose that interchanges rows and columns of a matrix (of any size).
   b) Function **FillMatrix** that allows the user to insert elements into the matrix.
   C) Function **DisplayMatrix** which displays the matrix elements on the screen.

3. Write a C program that calculate the factorial of a number entered by the user. The process of calculating the factorial must be in separate recursive function.