Lab 07 Implementation of Recursion

Objectives:

- To understand the basics of recursive functions.
- To convert an iterative solution to a problem to its recursive counterpart.
- To understand the advantages and disadvantages of recursion

Pre-Lab

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

What is base condition in recursion?

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

How a particular problem is solved using recursion?

The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

Why Stack Overflow error occurs in recursion?

If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

```c
int fact(int n)
{
    // wrong base case (it may cause
    // stack overflow).
    if (n == 100)
        return 1;

    else
        return n*fact(n-1);
}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

How memory is allocated to different function calls in recursion?

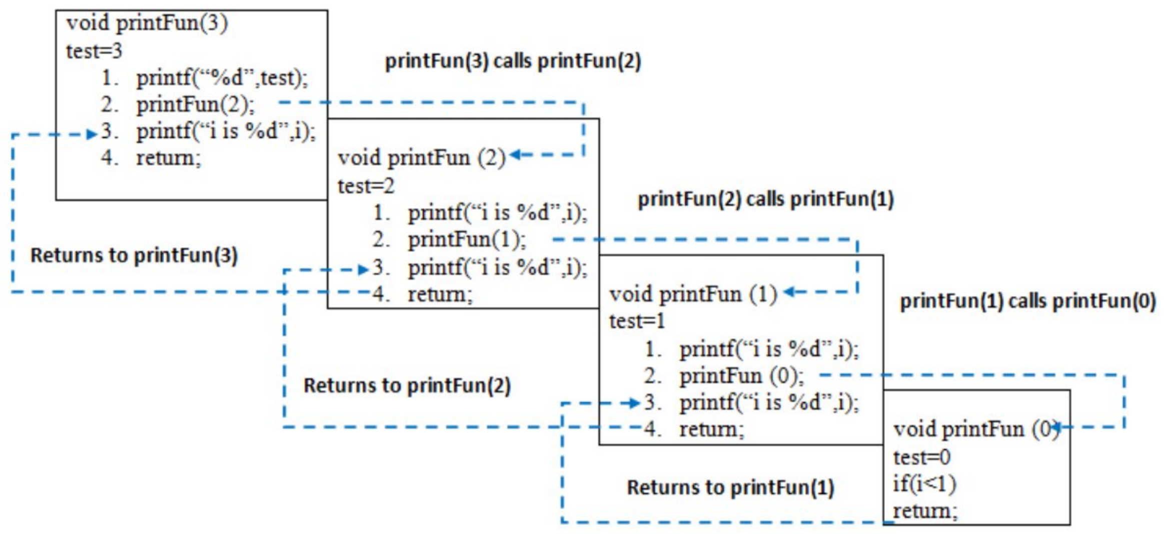
When any function is called from main(), the memory is allocated to it on the stack. A recursive function calls itself, the memory for a called function is allocated on top of memory allocated to calling function and different copy of local variables is created for each function call. When the

```c
// A C program to demonstrate working of recursion
#include<stdio.h>
#include<stdlib.h>

void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        printf("%d ", test);
        printFun(test-1);    // statement 2
        printf("%d ", test);
        return;
    }
}

int main()
{
    int test = 3;
    printFun(test);
}
```

base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.

Let us take the example how recursion works by taking a simple function.

When printFun(3) is called from main(), memory is allocated to printFun(3) and a local variable test is initialized to 3 and statement 1 to 4 are pushed on the stack as shown in below diagram. It first prints ‘3’. In statement 2, printFun(2) is called and memory is allocated to printFun(2) and a local variable test is initialized to 2 and statement 1 to 4 are pushed in the stack. Similarly, printFun(2) calls printFun(1) and printFun(1) calls printFun(0). printFun(0) goes to if statement and it return to printFun(1). Remaining statements of printFun(1) are executed and it returns to printFun(2) and so on. In the output, value from 3 to 1 are printed and then 1 to 3 are printed. The memory stack has been shown in below diagram.



Read more about recursion in the book ***“Data Structure using C”*** by Reema Thareja.

In-Lab Task 01 : Convert the following iterative function to a recursive one.

```
bool test_palindrome_itr(char * test_word)
{
    /// Get the length of the string.
    /// 'strlen()' returns total length including the NULL character
    int length = strlen(test_word)-1;

    int i;

    for(i = 0; i<=(length/2); i++)
    {
        /// Compare the 1st and the Last letters of the word
        if((*(test_word+i)) != (*(test_word+length-i-1)))
            break;
    }

    i--;     /// 'i' would have been one greater because of the 'i++' in the loop

    return(i==(length/2));
}
```

CodeListing 1: Function to test if a given word is a Palindrome (Iterative method)

Palindromes

A palindrome is a word that is the same when reversed, e.g. ‘madam’. The string can be reversed using an iterative method (given in **CodeListing 1**), but there is also a recursive method. A word is a palindrome if it has fewer than two letters, or if the first and last letter are the same and the middle part (i.e. without the first and last letters) is a palindrome.

Complete the function ‘bool **test_palindrome_itr**(char * test_word)’ in the provided skeleton code.

<u>Task 02</u> Convert the following recursive function to iterative one.

The greatest common divisor of two numbers (integers) is the largest integer that divides both the numbers. We can find the GCD of two numbers recursively by using the Euclid’s algorithm that states:

$$\text{GCD (a, b)} = \begin{cases} \text{b, if b divides a} \\ \text{GCD (b, a mod b), otherwise} \end{cases}$$

GCD can be implemented as a recursive function because if **b** does not divide **a** , then we call the same function (GCD) with another set of parameters that are smaller than the original ones. Here we assume that **a** > **b** . However if **a** < **b**, then interchange **a** and **b** in the formula given above. In ***CodeListing 2*** you are given a recursive implementation.

```
int GCD_rec(int x, int y)
{

    int rem;

    rem = x%y;

    if(rem==0)
            return y;
      else

    return (GCD_rec(y, rem));
}
```

CodeListing 2: Recursive implementation of finding GCD

Your task is to write an iterative function for computing GCD.

<u>Post Lab</u>

Write a program to reverse a string using recursion.

******** End of document ********