08 - Recursion

Post-Lab Exercises

Exercise 1

The following expansion of e^x is based on Taylor series.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

It can be implemented iteratively using a while-loop based code that detects convergence as presented here.
Convert this iterative function into a recursive function. Your function prototype should be same as the one for the given function. You must be mindful of proper usage of static type variables for this exercise. Another thing to consider is the factorial function required for this function. You may use it from our previous codes.

```
double exp(double x)
{
    int term_no=0;
    double prev_term=1, new_term=0, temp=0;
    while(fabs(prev_term-new_term)>0.0001)
    {
        prev_term=temp;
        new_term=prev_term+(pow(x,term_no)/factorial(term_no));
        temp=new_term;
        ++term_no;
    }
    return new_term;
```

Code

```
#include <iostream>
#include <cmath>
using namespace std;

double my_exp(double x);
int factorial(int y);

int main ()
{
    double x;
    cout<<"Enter the Power of Exponential you want to calculate = ";
    cin>>x;
    cout<<"\nexponent("<<x<<") = "<<my_exp(x)<<endl<<endl;
    return 0;
}
```

```
double my_exp(double x)
{
    static int a = 0;
    static double prev_term = 1, new_term = 0, temp = 0;
    if(fabs(prev_term - new_term) > 0.0001)
    {
        prev_term = temp;
        new_term = prev_term + (pow(x,a)/factorial(a));
        temp = new_term;
        a++;
        return my_exp (x);
    }
    else
        return new_term;
}

int factorial(int y)
{
    int i;
    int F = 1;
    for (i = 1; i <= y; i++)
        F *= i;
    return F;
}
```

Output

```
Enter the Power of Exponential you want to calculate = 2

exponent(2) = 7.38905

Program ended with exit code: 0
```

Explanation

To convert the given iterative exponential function into a recursive function we will replace the while loop with an if and else statement in the body of the function. The if statement will have the same condition as the while loop and the recursive function call will be made over here making it the inductive step. On the other hand, the else statement will return our final calculated value and will act as a base case. The variables in the body of the exponential function will be made static to prevent them from reinitializing on every recursive function call.

Exercise 2

Write function prototype and definition for palindrome detection of a string using both recursive and iterative implementations.

Iterative Implementation: Code

```
# include<iostream>
#include<string>
using namespace std;

bool isPal(const string my_word);

int main ()
{
    string my_word;
    cout<<"Enter word to check if it is a palindrome : ";
    getline(cin, my_word);
    if(isPal(my_word))
        cout<<"\nThe word you entered is a Palindrome\n\n”;
    else
        cout<<"\nThe word you entered is not a Palindrome\n\n";

    return 0;
}

bool isPal(const string my_word)
{
    int start=0, end=my_word.length()-1;
    while (start < end)
    {
        if (my_word[start++] != my_word[end--])
            return false;
    }
    return true;
}
```

Iterative Implementation: Output

```
Enter word to check if it is a palindrome : civic

The word you entered is a Palindrome

Program ended with exit code: 0
```

Iterative Implementation: Explanation

In the **iterative** model, the palindrome function takes a string named `my_word` as argument and treats it as an array of characters. It compares the letters progressively by first comparing the last letter with the first and then working towards the middle of the string. It does so by increasing the index count at the start and decreasing the index count at the end. All of this work is done in the body of the function.

Recursive Implementation: Code

```
#include<iostream>
#include<string>
using namespace std;

bool isPal(const string str, int start, int end);

int main ()
{
    string my_word,str;
    cout<<"Enter word to be checked: ";
    getline(cin, my_word);

    if (isPal(my_word, 0, my_word.length()-1))
        cout<<"\nThe word you entered is a Palindrome\n\n";
    else
        cout<<"\nThe word you entered is not a Palindrome\n\n";

    return 0;
}

bool isPal(const string str, int start, int end)
{
    if (start >= end)
        return true;
    if (str[start] != str[end])
        return false;
    return isPal(str, ++start, --end);
}
```

Recursive Implementation: Output

```
Enter word to be checked: civic

The word you entered is a Palindrome

Program ended with exit code: 0
```

Recursive Implementation: Explanation

In the **recursive** model the palindrome function does the same thing by comparing the letters at the end and the start. The only difference is that it manipulates the index count in the function arguments.