Notes on the First Assignment: Comparing Iterative and Recursive Factorial Calculations

Assignment Rules:

- **Tip**: Before diving into your assignment, take the time to thoroughly read and understand the rules and instructions. This ensures that your submission is comprehensive and accurate.

Structured Reporting: The significance of well-organized reports.

- **Tip**: Organize your reports. Utilize tables, charts, and clear discussions to present your findings. A well-structured report enhances the overall presentation and readability of your work.

Importance of Choosing Appropriate Data Types:

- **Tip**: When working with variables in programming, be sure of selecting the right data types. Choosing inadequate types may result in unexpected errors or inaccuracies in results. Pay attention to the data types that best suit the requirements of your task.

Choosing Appropriate Data Types

- C++ offers the data type '***unsigned long long***'.
- This data type uses 64 bits.
- ***Rule***: The maximum value that can be represented by ***unsigned long long*** is **$2^{64} – 1$**
- $2^{64} - 1 \approx \mathbf{18.4 \times 10^{18}}$
- Choosing data types for factorials:
- When calculating factorials, choose a data type that can store the result without overflow.
- **ensure that n! does not exceed $\mathbf{18.4 \times 10^{18}}$.**

```
cout << "Number of bits for unsigned long long: " << sizeof(unsigned long long) * 8 << " bits" << endl;
```

Cont. Recursion

Exponentiation (x^y) with Recursive Method

Exponentiation, represented as $\mathbf{x^y}$, is a fundamental mathematical operation where x is the base and y is the exponent. When dealing with exponentiation through a recursive method, we break down the problem into smaller subproblems. Assume x can be of type double and y is of type integer. **The recursive approach involves two main cases**:

1) **Base case**: If the exponent (y) is 0, the result is always 1. [x^0 is always 1]
2) **Recursive Case**: If y is greater than 0, we recursively calculate x^(y-1) and multiply the result by x.

$$x^y = x^y * \boxed{x^{y-1} * x^{y-2} * \ldots\ldots * x^0}$$

$$x^y = x^y * x^{y-1}$$

The general Recursive Formula:

So we can use recursion to define the power function:

$$\text{power}(x, y) = \begin{cases} 1 & \text{if } y = 0 \\ x * \text{power}(x, y-1) & \text{if } y > 0 \end{cases}$$

Base case

Recursive call

Pseudocode

```
function recursive_power(x, y)
    // Base case
    if y==0
        return 1
    // Recursive case: x^y = x * x^(y-1)
    return x * recursive_power(x, y - 1)
```

Example

- Given the following function:

```
int fun ( int x )
{
   if ( x )
     return 2 + fun (x – 1);
   else
      return 1;
}
```

- What is the output for the following code (**show your work**):

```
int x = 3 , result;

result = fun ( x );
cout << result;
```

Example

- **Trace of function multiply(6 , 3)   [show your work step by step]**

```
int multiply ( int m , int n )
{
   if ( n == 1  )
     return m;
   else
      return m + multiply (m , n – 1);
}
```

m * n = m + m * (n-1)   base case is m * 1 = m

Example

- Consider the following function:

```
int mystery ( int x , int y )
{
   if ( x < y  )
     return x;
   else
      return m + mystery (x - y , y);
}
```

- For each call below, indicate what value is returned: [show your work]

mystery (6 , 13)
mystery (14 , 10)

Example

- What is the result of the following function when **a=12** and **b=9**? Show your work.

```
int doit ( int a,  int b )
 {
   if (b == 0)
     return  a;
   else
    return  doit (b, a % b);
 }
```

Exercises

Exercise: Fibonacci Numbers

Problem Statement:

The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones, usually starting with 0 and 1. The sequence goes:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ….. and so on.

1. Setup the recurrence relation F(n) that define the sequence of Fibonacci numbers.
2. Write a pseudocode for the the recursive function ***fib(n)*** to calculate the nth Fibonacci number.
3. Implement the recursive function ***fib(n)*** ***ion c++ .*** *Ob*serve the function's behavior for larger values of n and note any potential increase in computation time.

[illegible]

Exercises

Write both iterative and recursive functions to calculate the value of the following expression:

$$f(n) = (1) + (2*3) + (4*5*6) + \ldots + n$$

Exercise

Based on your knowledge of Euclid’s Algorithm, Rewrite the algorithm using a recursive function named ***Rec_Euclid***.

- Recall: gcd (*a* , *b*), the greatest common divisor of two nonnegative, not both zero integers *a* and *b*

Exercise

Write pseudocode for a recursive function to find the sum of digits of a given number.

Example: n = 135 sum of digits = 9

Exercise

In the lecture, we discussed a method to raise a double to an integer power. In this question, write a recursive function that allows raising to a negative integer power as well.

Exercise

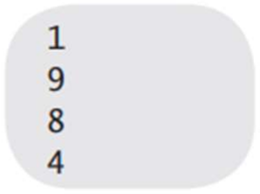
Write a recursive function to reverse a given string.

Example:
Input String: "algorithm"
Output (Reversed): "mhtirogla"

Exercise

Design a recursive function named ***displayVertically(n)***, that displays the digits of the given number nonnegative integer ***n*** vertically on the screen.

For example, if n = 1984, the vertical display would be:



```
1
9
8
4
```

1. What is the base case?
2. Design the recurrence case.
3. Write pseudocode.
4. Implement the recursive function.

Exercise

Design a recursive function with one parameter, a positive integer. When called, the function should write its argument backward to the screen. For instance, if the argument is 1234, the function should output the following to the screen: 4321

Exercise

Design a recursive function with one parameter, a positive integer. When called, the function should write its argument backward to the screen. For instance, if the argument is 1234, the function should output the following to the screen: 4321

In the upcoming chapters of this course, we will explore other interesting problems and challenges that can be elegantly solved using recursive approaches.

Advanced topics:

1.Recursive Data Structures:
- Explore how recursion plays a pivotal role in understanding and manipulating recursive data structures such as trees and graphs.

2.Divide and Conquer Strategies:
- Uncover the power of "divide and conquer" strategies, where recursion becomes a key player in solving complex problems efficiently.

3.Dynamic Programming :
- A technique that often involves recursive thinking to optimize solutions for a variety of problems.

Key Points

- Any task that can be accomplished using recursion can also be done in some other way without using recursion (nonrecursive version).
- The recursive version may use more storage and run somewhat slower, because the computer must do a good deal of work manipulating the stack in order to keep track of the recursion.
- However, since the system does all this for you automatically, using recursion can sometimes make your job as a programmer easier and can sometimes produce code that is easier to understand

Interactive tools and simulations

some websites that provide interactive tools and simulations to help students understand recursion and the use of the call stack:

1. **Pythontutor.com** : allows you to visualize the execution of Python code, including recursion, step by step. It provides a visual representation of the call stack.

- Website: http://pythontutor.com

2. **Visualgo.net:** offers visualizations for various data structures and algorithms, including recursion. It allows you to see how the call stack works for different programming languages.

- Website: https://visualgo.net/en/recursion