Focus on Software Engineering: Recursion vs. Iteration

**CONCEPT:** Recursive algorithms can also be coded with iterative control structures. There are advantages and disadvantages to each approach.

Any algorithm that can be coded with recursion can also be coded with an iterative control structure, such as a while loop. Both approaches achieve repetition, but which is best to use?

There are several reasons not to use recursion. Recursive algorithms are certainly less efficient than iterative algorithms. Each time a function is called, the system incurs overhead that is not necessary with a loop. Also, in many cases an iterative solution may be more evident than a recursive one. In fact, the majority of repetitive programming tasks are best done with loops.

Some problems, however, are more easily solved with recursion than with iteration. For example, the mathematical definition of the GCD formula is well-suited for a recursive approach. The QuickSort algorithm is also an example of a function that is easier to code with recursion than iteration.

The speed and amount of memory available to modern computers diminishes the performance impact of recursion so much that inefficiency is no longer a strong argument against it. Today, the choice of recursion or iteration is primarily a design decision. If a problem is more easily solved with a loop, that should be the approach you take. If recursion results in a better design, that is the choice you should make.

**Review Questions and Exercises**

**Short Answer**

1. What is the base case of each of the recursive functions listed in questions 12, 13, and 14?
2. What type of recursive function do you think would be more difficult to debug, one that uses direct recursion, or one that uses indirect recursion? Why?
3. Which repetition approach is less efficient, a loop or a recursive function? Why?
4. When should you choose a recursive algorithm over an iterative algorithm?
5. Explain what is likely to happen when a recursive function that has no way of stopping executes.

**Fill-in-the-Blank**

6. The _________ of recursion is the number of times a function calls itself.
7. A recursive function’s solvable problem is known as its _________. This causes the recursion to stop.
8. ________ recursion is when a function explicitly calls itself.
9. ________ recursion is when function A calls function B, which in turns calls function A.

**Algorithm Workbench**

10. Write a recursive function to return the number of times a specified number occurs in an array.
11. Write a recursive function to return the largest value in an array.

**Predict the Output**

What is the output of the following programs?

12. ```
#include <iostream>

using namespace std;

int function(int);

int main()
{
    int x = 10;
    cout << function(x) << endl;
    return 0;
}

int function(int num)
{
    if (num <= 0)
        return 0;
    else
        return function(num - 1) + num;
}
```  

13. ```
#include <iostream>

using namespace std;

void function(int);

int main()
{
    int x = 10;
    function(x);
    return 0;
}

void function(int num)
{
    if (num > 0)
    {
        for (int x = 0; x < num; x++)
            cout << '*';
        cout << endl;
        cout << endl;
        function(num - 1);
    }
}  ```
14. #include <iostream>
#include <string>
using namespace std;

void function(string, int, int);

int main()
{
    string mystr = "Hello";
    cout << mystr << endl;
    function(mystr, 0, mystr.size());
    return 0;
}
void function(string str, int pos, int size)
{
    if (pos < size)
    {
        function(str, pos + 1, size);
        cout << str[pos];
    }
}

Programming Challenges

1. Iterative Factorial
Write an iterative version (using a loop instead of recursion) of the factorial function shown in this chapter. Test it with a driver program.

2. Recursive Conversion
Convert the following function to one that uses recursion.
void sign(int n)
{
    while (n > 0)
    {
        cout << "No Parking\n";
        n--;
    }
}
Demonstrate the function with a driver program.

3. QuickSort Template
Create a template version of the QuickSort algorithm that will work with any data type. Demonstrate the template with a driver function.

4. Recursive Array Sum
Write a function that accepts an array of integers and a number indicating the number of elements as arguments. The function should recursively calculate the sum of all the numbers in the array. Demonstrate the function in a driver program.

5. Recursive Multiplication
Write a recursive function that accepts two arguments into the parameters \( x \) and \( y \). The function should return the value of \( x \) times \( y \). Remember, multiplication can be performed as repeated addition:
\[
7 \times 4 = 4 + 4 + 4 + 4 + 4 + 4 + 4
\]