UMER FAYYAZ – C++ LABORATORY

What is Infinite Recursion?

Definition: Infinite recursion occurs when a recursive function does not have a base case, or the base case is never reached. As a result, the function keeps calling itself indefinitely, leading to what is known as infinite recursion.

Key Concepts in Infinite Recursion

1. **Recursive Call**:
   - A recursive function is one that calls itself in its definition.
2. **Base Case**:
   - The base case is a condition in a recursive function that stops the recursion by not making another recursive call. It's essential to ensure the function doesn't run indefinitely.
3. **Infinite Recursion**:
   - If every recursive call results in another recursive call without ever reaching a base case, the function falls into infinite recursion.
   - This means the function continues to execute forever theoretically, without terminating.
4. **Memory Allocation**:
   - Each time a recursive function is called, the system allocates memory for the function's local variables and formal parameters (the arguments passed to the function).
   - The system also keeps track of the return address, so it knows where to go back once the function completes.
5. **System Memory**:
   - Computer memory is finite, meaning it has a limit.
   - As infinite recursion keeps calling the function, it keeps consuming more and more memory.
   - Eventually, the system runs out of memory, leading to a stack overflow, and the program crashes or terminates abnormally.

Why is Infinite Recursion Dangerous?

- **Stack Overflow**:
  - In programming, each function call is placed on the call stack (a special region of memory). When recursion doesn't terminate, the call stack keeps growing until the system runs out of memory, causing a stack overflow.
- **Abnormal Termination**:

    - When the system can no longer allocate memory for the new calls, it will force the program to stop, often leading to an unexpected crash.

How to Prevent Infinite Recursion?

To avoid infinite recursion, you must carefully design your recursive functions. Here's how:

1. **Understand the Problem Requirements**:
    - Make sure you understand what the problem is asking for. Knowing what you want to achieve helps in structuring the recursion properly.
2. **Determine the Limiting Conditions**:
    - Identify the limits of the problem. For example, if you are working with a list, the limit might be the length of the list.
3. **Identify the Base Cases**:
    - Clearly define the base case(s). A base case is a condition under which the function does not call itself and instead returns a value.
    - The base case ensures that the recursion will eventually stop.
4. **Identify the General Cases**:
    - The general case is where the function calls itself with a modified version of the input. Each recursive call should bring the problem closer to the base case.

Example of Infinite Recursion

Let's take a look at an example of infinite recursion:

```
void infiniteRecursion() {
    infiniteRecursion();  // No base case
}
```

Dry Run:

- **Step 1**: infiniteRecursion() is called.
    - The function immediately calls infiniteRecursion() again.
- **Step 2**: The second call to infiniteRecursion() happens.
    - It again calls infiniteRecursion().
- **Step 3**: This process continues indefinitely.
    - Since there is no base case, the function never stops calling itself.

Outcome: The system will keep allocating memory for each call until it runs out of memory, causing a stack overflow error.

Proper Recursive Design:

Now, let's see how to design a recursive function properly to avoid infinite recursion.

Example: Factorial Function with a Base Case:

```
int factorial(int n) {
    if (n == 0) {
        return 1;  // Base case: stop recursion when n is 0
    } else {
        return n * factorial(n - 1);  // Recursive case
    }
}
```

Dry Run for factorial(3):

- **Step 1**: factorial(3) is called.
    - Since n is not 0, it calls factorial(2).
- **Step 2**: factorial(2) is called.
    - Again, n is not 0, so it calls factorial(1).
- **Step 3**: factorial(1) is called.
    - n is still not 0, so it calls factorial(0).
- **Step 4**: factorial(0) is called.
    - Now, n is 0, so the base case is reached, and 1 is returned.
- **Step 5**: The return value 1 is passed back to factorial(1), which then returns 1 * 1 = 1.
- **Step 6**: The return value 1 is passed back to factorial(2), which returns 2 * 1 = 2.
- **Step 7**: The return value 2 is passed back to factorial(3), which returns 3 * 2 = 6.

Outcome: The recursion stops when the base case is met, and the result is 6.

Conclusion

Infinite Recursion is a critical concept to understand in recursion. It highlights the importance of having a well-defined base case in recursive functions. Without a base case, the recursion will continue indefinitely, leading to a stack overflow and abnormal termination of the program.

Key Takeaways:

- Always ensure your recursive functions have a clear and reachable base case.
- Understand the problem and design your recursion carefully to avoid infinite loops.
- If a function seems to be running forever or causing a crash, check if it's due to infinite recursion by analyzing the presence and logic of the base case.