Fibonacci Fun - Programming Exercise

Difficulty: Beginner → Intermediate
Topic: Loops, Arrays, Logic
Time Limit: 1 second per test case
Memory Limit: 256 MB

Problem Description

The Fibonacci sequence is a famous series of numbers where:

- Fib(1) = 1
- Fib(2) = 1
- Fib(n) = Fib(n-1) + Fib(n-2) for n > 2

So the sequence starts: `1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...`

Task

1. Read an integer `m` — the number of test cases
2. For each test case, read an integer `n` and print the **n-th Fibonacci number**

Input Format

- First line: integer `m` ($1 \leq m \leq 100$) — number of test cases
- Next `m` lines: each contains integer `n` ($1 \leq n \leq 92$)
- Note: `n ≤ 92` ensures result fits in 64-bit signed integer

Output Format

- For each test case, output the n-th Fibonacci number on a new line

Sample Input

```
3
1
5
10
```

Sample Output

```
1
5
55
```

Explanation

- Fib(1) = 1
- Fib(5) = 5 (sequence: 1, 1, 2, 3, **5**)
- Fib(10) = 55 (sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, **55**)

🚨 Important Notes

- **Do NOT use recursion** for large n - it's exponentially slow!
- Use iterative approach for efficiency
- Consider space optimization using only two variables
- Use appropriate data types for large numbers

Solution Approaches

Approach 1: Basic Iterative (Recommended for Beginners)

C Solution (Clean and Efficient):

```c
```

```c
#include <stdio.h>

int main() {
    int m;
    scanf("%d", &m);

    while (m--) {
        long long n;
        scanf("%lld", &n);

        if (n == 1 || n == 2) {
            printf("1\n");
            continue;
        }

        long long prev = 1, curr = 1;
        for (long long i = 3; i <= n; i++) {
            long long next = prev + curr;
            prev = curr;
            curr = next;
        }
        printf("%lld\n", curr);
    }
    return 0;
}
```

C++ Solution:

cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int m;
    cin >> m;

    while (m--) {
        int n;
        cin >> n;

        if (n <= 2) {
            cout << 1 << endl;
            continue;
        }

        long long fib1 = 1, fib2 = 1, current;

        for (int i = 3; i <= n; i++) {
            current = fib1 + fib2;
            fib1 = fib2;
            fib2 = current;
        }

        cout << current << endl;
    }

    return 0;
}
```

Python Solution:

```python
```

```
def fibonacci(n):
    if n <= 2:
        return 1

    fib1, fib2 = 1, 1

    for i in range(3, n + 1):
        current = fib1 + fib2
        fib1 = fib2
        fib2 = current

    return fib2

# Main program
m = int(input())
for _ in range(m):
    n = int(input())
    print(fibonacci(n))
```

Java Solution:

java

```
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int m = scanner.nextInt();

        while (m-- > 0) {
            int n = scanner.nextInt();
            System.out.println(fibonacci(n));
        }

        scanner.close();
    }

    public static long fibonacci(int n) {
        if (n <= 2) {
            return 1;
        }

        long fib1 = 1, fib2 = 1, current = 0;

        for (int i = 3; i <= n; i++) {
            current = fib1 + fib2;
            fib1 = fib2;
            fib2 = current;
        }

        return current;
    }
}
```

Approach 2: Space-Optimized (Advanced)

C Alternative (More Traditional Style):

```
c
```

```c
#include <stdio.h>

long long fibonacci(long long n) {
    if (n <= 2) return 1;

    long long prev = 1, curr = 1;

    for (long long i = 3; i <= n; i++) {
        long long temp = prev + curr;
        prev = curr;
        curr = temp;
    }

    return curr;
}

int main() {
    int m;
    scanf("%d", &m);

    while (m--) {
        long long n;
        scanf("%lld", &n);
        printf("%lld\n", fibonacci(n));
    }

    return 0;
}
```

C++ Optimized:

cpp

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

long long fibonacci(int n) {
    if (n <= 2) return 1;

    long long prev = 1, curr = 1;

    for (int i = 3; i <= n; i++) {
        long long temp = prev + curr;
        prev = curr;
        curr = temp;
    }

    return curr;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int m;
    cin >> m;

    while (m--) {
        int n;
        cin >> n;
        cout << fibonacci(n) << "\n";
    }

    return 0;
}
```

Approach 3: Pre-computation (For Multiple Queries)

C++ with Pre-computation:

```cpp
```

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    // Pre-compute all Fibonacci numbers up to 92
    vector<long long> fib(93);
    fib[1] = fib[2] = 1;

    for (int i = 3; i <= 92; i++) {
        fib[i] = fib[i-1] + fib[i-2];
    }

    int m;
    cin >> m;

    while (m--) {
        int n;
        cin >> n;
        cout << fib[n] << endl;
    }

    return 0;
}
```

🎯 Practice Exercises

Exercise 1: Basic Implementation

Implement the basic iterative solution in your preferred language.

Exercise 2: Optimization Challenge

- Measure the time difference between recursive and iterative approaches
- Try with n = 40, 50 (be careful with recursion!)

Exercise 3: Extension Problems

1. **Sum of First N Fibonacci Numbers**: Calculate sum of Fib(1) + Fib(2) + ... + Fib(n)
2. **Even Fibonacci Numbers**: Find sum of all even Fibonacci numbers ≤ n
3. **Fibonacci Modulo**: Calculate Fib(n) mod 1000000007

💡 Key Learning Points

1. **Time Complexity:**
   - Recursive: O(2^n) - exponential, very slow!
   - Iterative: O(n) - linear, efficient
   - Pre-computed: O(1) per query
2. **Space Complexity:**
   - Basic iterative: O(1)
   - Pre-computation: O(n)
3. **Common Pitfalls:**
   - Using recursion for large n
   - Integer overflow (use long long in C++)
   - Off-by-one errors in indexing
4. **Optimization Techniques:**
   - Space optimization using two variables
   - Pre-computation for multiple queries
   - Fast I/O for competitive programming

🧪 Test Your Solution

Test Case 1

```
Input: 5
    1 2 3 4 5
Output: 1 1 2 3 5
```

Test Case 2

```
Input: 3
    10 15 20
Output: 55 610 6765
```

Edge Cases

- n = 1, n = 2 (base cases)
- n = 92 (maximum value)

- Multiple test cases with varying n

Happy coding! 🚀