CS12 CH:10

Problem Solving with Loops

Mr. Gullo

November 5, 2025

Learning Objectives

- Apply the 6-step problem-solving methodology to complex programming challenges
- Implement while and for loops to solve iterative problems
- Understand and apply optimization techniques to improve algorithm efficiency
- Master unsigned data types for handling large positive integers
- Develop algorithms for mathematical sequences and number theory problems
- Implement template-based code structure with clear TODO sections

The 6-Step Problem Solving Process

A Systematic Approach to Complex Problems

1. **Explore the Question**
   - Understand all requirements
   - Identify constraints and edge cases
   - Simplify when possible

The 6-Step Problem Solving Process

A Systematic Approach to Complex Problems

1. **Explore the Question**
   - Understand all requirements
   - Identify constraints and edge cases
   - Simplify when possible
2. **Code Something Simple First**
   - Start with a simplified version
   - Test with smaller numbers/simpler cases

The 6-Step Problem Solving Process

A Systematic Approach to Complex Problems

1. **Explore the Question**
   - Understand all requirements
   - Identify constraints and edge cases
   - Simplify when possible
2. **Code Something Simple First**
   - Start with a simplified version
   - Test with smaller numbers/simpler cases
3. **Run With Examples**
   - Verify against known solutions
   - Test edge cases

The 6-Step Problem Solving Process

A Systematic Approach to Complex Problems

1. **Explore the Question**
   - Understand all requirements
   - Identify constraints and edge cases
   - Simplify when possible
2. **Code Something Simple First**
   - Start with a simplified version
   - Test with smaller numbers/simpler cases
3. **Run With Examples**
   - Verify against known solutions
   - Test edge cases
4. **Get Something Working**
   - Focus on correctness over efficiency
   - Ensure the core algorithm functions

The 6-Step Problem Solving Process

A Systematic Approach to Complex Problems

1. **Explore the Question**
   - Understand all requirements
   - Identify constraints and edge cases
   - Simplify when possible
2. **Code Something Simple First**
   - Start with a simplified version
   - Test with smaller numbers/simpler cases
3. **Run With Examples**
   - Verify against known solutions
   - Test edge cases
4. **Get Something Working**
   - Focus on correctness over efficiency
   - Ensure the core algorithm functions
5. **Make Optimizations**
   - Improve efficiency after correctness is confirmed
   - Reduce unnecessary computations
6. **Try Another Approach**
   - Consider alternative algorithms

Problem Set Overview

Today's Challenges

1. **Basics of while and for loops**
   - Generate sequences with specific patterns
2. **Division without operators**
   - Implement division and modulus using subtraction
3. **Largest Prime Factor**
   - Factor large numbers and identify prime factors
4. **Smallest Multiple**
   - Find LCM of numbers 1-20
5. **Sum of Powers**
   - Calculate geometric series
6. **Greatest Common Factor**
   - Implement Euclidean algorithm
7. **Fibonacci Sequence**
   - Generate terms in specified ranges
8. **Palindromic Numbers**
   - Check and find palindrome products

Key Concept: Unsigned Data Types

Handling Large Positive Integers

Signed Integers

- Use one bit for sign
- Range: -2^{n-1} to $2^{n-1} - 1$
- Example: `int` (32-bit)
  - Range: $-2,147,483,648$ to $2,147,483,647$

Unsigned Integers

- All bits for magnitude
- Range: 0 to $2^n - 1$
- Example: `unsigned int` (32-bit)
  - Range: 0 to $4,294,967,295$

Common Issue:

```
int testNum = 600851475143;  // ERROR! Too large
unsigned long long testNum = 600851475143;  // Works!
```

Problem 1: Loop Basics

Template Exercise

Exercise File: `problem1_loops.cpp` (Template with TODOs)
Objective: Complete the TODOs to generate sequences.

Problem 1: Loop Basics

Template Exercise

Exercise File: `problem1_loops.cpp` (Template with TODOs)
Objective: Complete the TODOs to generate sequences.

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

int main()
{
    // TODO 1: Use a while loop to output multiples of 7 between 0
        and 77 inclusive
    // Expected output: 0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70,
        77

    // TODO 2: Use a for loop to output: 1000, 800, 600, ..., -1000
    // Expected output: 1000, 800, 600, 400, 200, 0, -200, -400,
        -600, -800, -1000

    return 0;
}
```

Problem 4: Largest Prime Factor

Template Exercise

Exercise File: `problem4_primeFactor.cpp` (Template with TODOs)
Objective: Find the largest prime factor of a number.

Problem 4: Largest Prime Factor

Template Exercise

Exercise File: `problem4_primeFactor.cpp` (Template with TODOs)
Objective: Find the largest prime factor of a number.

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

int main()
{
    unsigned long long testNumber;
    unsigned long long largestPrime = 0;
    unsigned long long currentFactor = 2;

    cout << "Enter a positive integer greater than 1: ";
    cin >> testNumber;

    // TODO 1: Implement algorithm to find largest prime factor
    // Hint: Divide out factors starting from 2
    // When testNumber % currentFactor == 0:
    //    - Update largestPrime
    //    - Divide testNumber by currentFactor
    // Otherwise: increment currentFactor
    // Continue until testNumber equals 1
```

Problem 5: Smallest Multiple

Template Exercise

Exercise File: `problem5_smallestMultiple.cpp` (Template with TODOs)
Objective: Find smallest number divisible by 1-20.

Problem 5: Smallest Multiple

Template Exercise

Exercise File: `problem5_smallestMultiple.cpp` (Template with TODOs)

Objective: Find smallest number divisible by 1-20.

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

int main()
{
    int smallestNumber = 0;
    bool areWeDone = false;

    // TODO 1: Optimization - identify what numbers we need to
        check
    // Hint: What's special about prime numbers
        2,3,5,7,11,13,17,19?

    while(!areWeDone) {
        areWeDone = true;
        smallestNumber += /* TODO 2: Fill in optimized increment
            value */;
```

Problem 8: Fibonacci Sequence

Understanding the Sequence

Definition:

$$f_n = \begin{cases} f_0 = f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases}$$

First 10 terms: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

Key Implementation Strategy:

- Keep track of three consecutive terms
- Use variables for: f_n, f_{n-1}, and f_{n-2}
- Shift values forward in each iteration

[Diagram showing Fibonacci sequence calculation]

Problem 8: Fibonacci Implementation

Template Exercise

Exercise File: `problem8_fibonacci.cpp` (Template with TODOs)
Objective: Display Fibonacci numbers from f_a to f_b.

Problem 8: Fibonacci Implementation

Template Exercise

Exercise File: `problem8_fibonacci.cpp` (Template with TODOs)
Objective: Display Fibonacci numbers from f_a to f_b.

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

int main()
{
    int fibN = 1;     // Current term
    int fibN_1 = 0;   // Previous term
    int fibN_2 = 0;   // Two terms back

    int a, b;
    cout << "Enter integers a and b (where 0 <= a <= b <= 50): ";
    cin >> a >> b;

    // TODO 1: Find the ath term (don't print yet)
    // Use a loop to calculate up to position a

    // TODO 2: Print from ath to bth term
    // Handle both cases: a <= b and a > b
    // Remember to shift the three terms appropriately
```

Problem 9: Palindromic Numbers

Definition and Properties

Palindrome: A number that reads the same forward and backward
Examples: 9, 232, 7007, 12321
Algorithm to Check Palindrome:

1. Make a copy of the original number
2. Reverse the digits using modulo and division
3. Compare original with reversed number

Largest Palindrome Product Problem:

- Find largest palindrome from product of two 3-digit numbers
- Brute force approach: Check all products from 999×999 down to 100×100
- Optimization: Inner loop can start at current outer loop value

[Diagram showing palindrome check algorithm]

Problem 9: Palindrome Implementation

Template Exercise

Exercise File: `problem9_palindrome.cpp` (Template with TODOs)
Objective: Find largest palindrome from two 3-digit numbers.

Problem 9: Palindrome Implementation

Template Exercise

Exercise File: `problem9_palindrome.cpp` (Template with TODOs)
Objective: Find largest palindrome from two 3-digit numbers.

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

bool isPalindrome(int number) {
    // TODO 1: Implement palindrome check function
    // Return true if number is palindrome, false otherwise
    int original = number;
    int reversed = 0;

    // Reverse the digits without destroying original
    // Compare original with reversed

    return false; // Replace with actual condition
}

int main() {
    int largestPalindrome = 0;

    // TODO 2: Implement nested loops to find largest palindrome
    // Outer loop: from 999 down to 100
```

Optimization Techniques

Making Your Code More Efficient

Problem 5 (Smallest Multiple) Optimization:

- Check only multiples of primes < 20
- Product: $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
- Reduces checks by $\sim$99%

Problem 9 (Palindrome) Optimization:

- Inner loop starts at outer loop value
- Avoids duplicate products (e.g., 123×456 and 456×123)
- Reduces iterations by approximately 50%

General Optimization Principles:

1. Get it working first, then optimize
2. Profile to identify bottlenecks
3. Consider mathematical properties of the problem
4. Reduce redundant calculations

Summary

Key Takeaways

Problem-Solving Methodology:

- Break complex problems into smaller, manageable parts
- Start with simple cases and verify your approach
- Implement a working solution before optimizing

Loop Implementation:

- While loops: When iteration count is unknown
- For loops: When iteration count is known
- Proper initialization, condition, and increment are essential

Algorithm Design:

- Consider data type limitations (unsigned for large positives)
- Look for mathematical properties to optimize
- Test with edge cases and known solutions

Next Steps:

- Complete all template exercises
- Experiment with different optimization approaches
- Apply problem-solving methodology to new challenges