C Programming: Using recursion, pointers, arrays, bugs, debugging

Yipeng Huang

Rutgers University

February 8, 2022

Table of contents

Next quiz and programming assignments

Next quiz

- Spans Wednesday through Friday.
- Same format, two 45-minute tries.

Programming assignment 1

- Due Thursday evening.

Table of contents

Challenges in CS programming assignments, strategies to get unstuck, resources

In CS 111, 112, 211, what are reasons programming assignments are challenging?

- Not sure where to start.
- It isn't working.
- The CS 211 teachers say that knowing Java helps programming in C, but C is nothing like Java.

What are strategies to get unstuck?

What are resources to learn something that was not explained well?

Lessons and ways in which programming in class is not like the real world.

- Coding deliberately is important. Have a plan. Understand the existing code. Test assumptions. Don't code by trial and error.
- Less code is better, and more likely to be correct.
- Reading code is as important and takes more time than writing code.
- In the real world, people work in teams. Here, assignments are individual work. If the class can collectively crack a difficult assignment via Piazza, that is a more realistic model of real-world engineering.

Table of contents

`dynamicProgramming.c`: Minimum number of multiplies needed for matrix chain multiplication

Learning objectives

- Review and master recursion.
- Array subsetting using pointer arithmetic.
- Using pass-by-reference to return computed results.
- A new algorithm that most classmates have not seen before.

Cost of multiplying matrices: the number of multiplies

- $A_{l \times m} \times B_{m \times n}$
- Needs $l \times m \times n$ number of multiplies
- (Well-kept secret: fewer multiplications possible, see Strassen's algorithm)

`dynamicProgramming.c`: Minimum number of multiplies needed for matrix chain multiplication

$$A \times B \times C = \begin{bmatrix} a_{0,0} & a_{0,1} \end{bmatrix}_{1\times 2} \times \begin{bmatrix} b_{0,0} \\ b_{1,0} \end{bmatrix}_{2\times 1} \times \begin{bmatrix} c_{0,0} & c_{0,1} \end{bmatrix}_{1\times 2}$$

Parenthesization 1: 4+4 = 8 multiplies

$$\begin{aligned} A \times (B \times C) &= \begin{bmatrix} a_{0,0} & a_{0,1} \end{bmatrix}_{1\times 2} \times \begin{bmatrix} b_{0,0}c_{0,0} & b_{0,0}c_{0,1} \\ b_{1,0}c_{0,0} & b_{1,0}c_{0,1} \end{bmatrix}_{2\times 2} \\ &= \begin{bmatrix} \left(a_{0,0}b_{0,0}c_{0,0} + a_{0,1}b_{1,0}c_{0,0}\right) & \left(a_{0,0}b_{0,0}c_{0,1} + a_{0,1}b_{1,0}c_{0,1}\right) \end{bmatrix}_{1\times 2} \end{aligned}$$

Parenthesization 2: 2+2 = 4 multiplies

$$\begin{aligned} (A \times B) \times C &= \left(a_{0,0}b_{0,0} + a_{0,1}b_{1,0}\right) \times \begin{bmatrix} c_{0,0} & c_{0,1} \end{bmatrix}_{1\times 2} \\ &= \begin{bmatrix} \left(a_{0,0}b_{0,0}c_{0,0} + a_{0,1}b_{1,0}c_{0,0}\right) & \left(a_{0,0}b_{0,0}c_{0,1} + a_{0,1}b_{1,0}c_{0,1}\right) \end{bmatrix}_{1\times 2} \end{aligned}$$

`dynamicProgramming.c`: Minimum number of multiplies needed for matrix chain multiplication

$A \times B \times C \times D$

First partitioning

- $A(BCD)$; but what is cost of finding (BCD)? Needs decomposition.
- $(AB)(CD)$
- $(ABC)D$; but what is cost of finding (ABC)? Needs decomposition.

Second partitioning

- $A(B(CD))$
- $A((BC)D)$
- $(AB)(CD)$
- $(A(BC))D$
- $((AB)C)D$

On the blackboard: work through `dynamicProgramming/tests/test3.txt` together.

Then, compare our algorithm against the source code.

Table of contents

`matMul.c`: Function for matrix-matrix multiplication

What to pay attention to

- How `matMulProduct` result is given back to caller of function.
- How and where memory is allocated and freed.

Table of contents

Approaches to Software Reliability

- Social
  - Code reviews
  - Extreme/Pair programming
- Methodological
  - Design patterns
  - Test-driven development
  - Version control
  - Bug tracking
- Technological
  - "lint" tools, static analysis
  - Fuzzers, random testing
- Mathematical
  - Sound type systems
  - Formal verification

Less "formal": Lightweight, inexpensive techniques (that may miss problems)

This isn't an either/or tradeoff… a spectrum of methods is needed!

Even the most "formal" argument can still have holes:

- Did you prove the right thing?
- Do your assumptions match reality?
- Knuth: *"Beware of bugs in the above code; I have only proved it correct, not tried it."*

More "formal": eliminate *with certainty* as many problems as possible.

From: https://www.seas.upenn.edu/~cis500/current/lectures/lec01.pdf

Strategies for debugging

Reduce to minimum example

- Check your assumptions.
- Use minimum example as basis for searching for help.

Debugging techniques

- Use assertions.
- Use debugging tools: Valgrind, Address Sanitizer, GDB.
- Use debugging statements.