

Data representation: Integers, Fixed Point

Yipeng Huang

Rutgers University

October 9, 2026

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate π

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate π

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Canvas timed quiz 4 and programming assignment 2

Quiz 4

1. Spanning Friday 10/2 - Friday 10/16.
2. 60 minutes.
3. Three tries.
4. Memory bugs and bits, bytes, and integers.

Programming assignment 2

1. Due Tuesday 10/13.
2. Data structures mastery.

`inplaceSwap.c`: Swapping variables without temp variables.

How does it work?

Bitwise Operations & The XOR Swap

- ▶ **Bitwise Operators:** Manipulate the raw binary representation of integers.
 - ▶ **~ NOT:** Flips all bits.
 - ▶ **& AND:** 1 only if both bits are 1.
 - ▶ **| OR:** 1 if either bit is 1.
 - ▶ **^ XOR:** 1 if exactly one bit is 1.
- ▶ **The In-Place XOR Swap:** Swaps two variables without a temporary third variable using three XOR assignments.
- ▶ **The Fatal Flaw:** Fails when a variable is swapped with itself.
 - ▶ Any value XORed with itself is zero ($x \oplus x = 0$).
 - ▶ Swapping a variable with itself instantly zeroes it out, destroying the original data.

Don't confuse bitwise operators with logical operators

Bitwise operators

- ▶ ~
- ▶ &
- ▶ |
- ▶ ^

Logical operators

- ▶ !
- ▶ &&
- ▶ ||
- ▶ != (for bool type)

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Representing negative and signed integers

Ways to represent negative numbers

1. Sign magnitude
2. 1s' complement
3. 2's complement

Representing Negative Integers: Sign-Magnitude

- ▶ **Concept:** The Most Significant Bit (MSB) acts as a sign flag (0 = positive, 1 = negative). The remaining bits encode the absolute magnitude.
- ▶ **Range (8-bit):** -127 to $+127$.
- ▶ **Fatal Flaws:**
 - ▶ **Dual Zeros:** Creates two representations for zero (0000_0000 for $+0$, 1000_0000 for -0).
 - ▶ **Addition Fails:** Standard binary addition circuits produce incorrect results (e.g., adding $+1$ and -1 yields -2).

Representing Negative Integers: Ones' Complement

- ▶ **Concept:** Represent a negative number by flipping all the bits of its positive counterpart.
- ▶ **Addition Fix:** Binary addition mostly works, but requires a **carry-around rule** (if a carry overflows the MSB, it must be added back to the Least Significant Bit).
- ▶ **The Math:** The leading bit effectively carries a weight of -127 (for 8-bit), meaning the system complements against the value represented by all ones.
- ▶ **Lingering Issue:** Still suffers from the dual zero problem (0000_0000 and 1111_1111).

Representing Negative Integers: Two's Complement

- ▶ **The Standard:** The universally adopted encoding for modern hardware architectures.
- ▶ **The Rule:** Flip all bits and add 1.
- ▶ **The Math:** The leading bit functions as a large negative weight (exactly -128 for an 8-bit integer), while all other bits contribute standard positive powers of two.
- ▶ **Advantages:**
 - ▶ Eliminates the dual representation of zero.
 - ▶ Binary addition works natively without any special carry-around rules.

Representing negative and signed integers

2's complement

signed char	weight in decimal
00000001	1
00000010	2
00000100	4
00001000	8
00010000	16
00100000	32
01000000	64
10000000	-128

Table: Weight of each bit in a signed char type

- ▶ what is the most positive value you can represent? 127
- ▶ what is the most negative value you can represent? -128
- ▶ how to represent -1? 11111111
- ▶ how to represent -2? 11111110

Representing negative and signed integers

2's complement

signed char	weight in decimal
00000001	1
00000010	2
00000100	4
00001000	8
00010000	16
00100000	32
01000000	64
10000000	-128

Table: Weight of each bit in a signed char type

- ▶ MSB: 1 for negative
- ▶ To make a number negative: flip all bits and add 1.
- ▶ Addition in 2's complement is sound

Importance of paying attention to limits of encoding

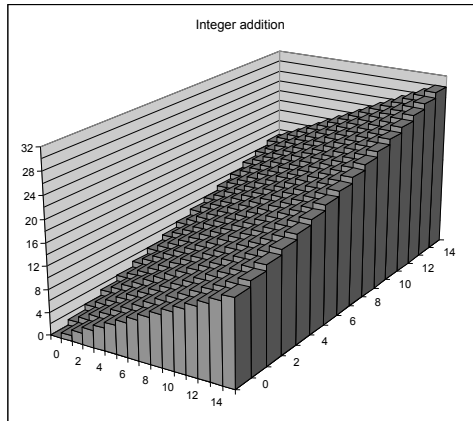


Figure: Image credit: CS:APP

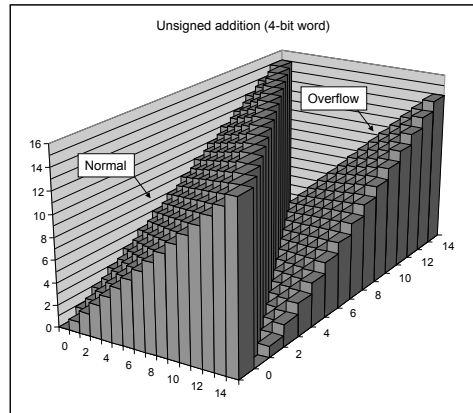


Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

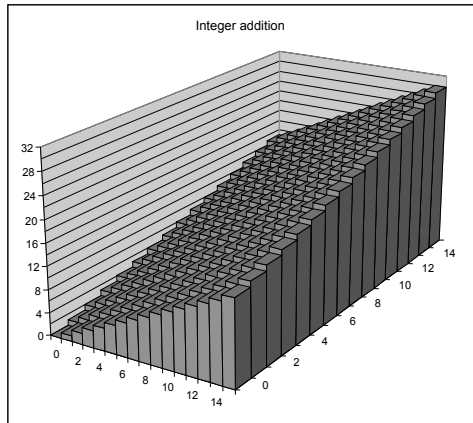


Figure: Image credit: CS:APP

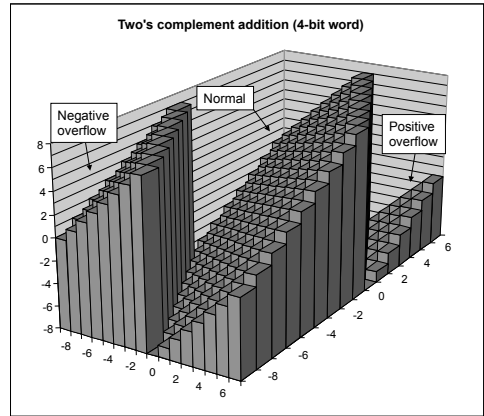


Figure: Image credit: CS:APP

<https://www.theatlantic.com/technology/archive/2014/12/how-gangnam-style-broke-youtube/383389/>

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Unsigned fixed-point binary for fractions

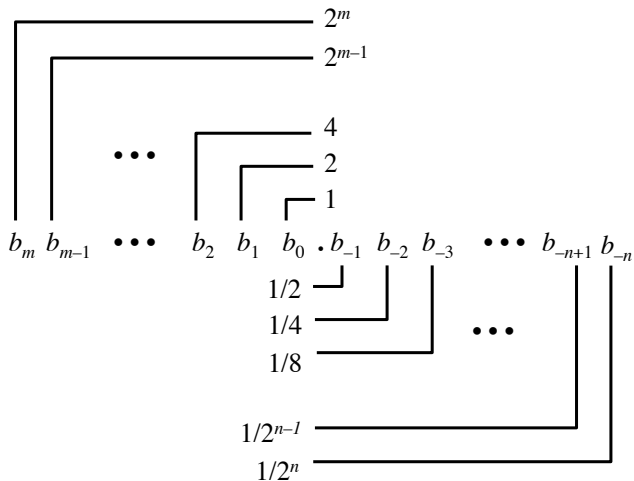


Figure: Fractional binary. Image credit CS:APP

Unsigned fixed-point binary for fractions

unsigned fixed-point char example	weight in decimal
1000.0000	8
0100.0000	4
0010.0000	2
0001.0000	1
0000.1000	0.5
0000.0100	0.25
0000.0010	0.125
0000.0001	0.0625

Table: Weight of each bit in an example fixed-point binary number

- ▶ $.625 = .5 + .125 = 0000.1010_2$
- ▶ $1001.1000_2 = 9 + .5 = 9.5$

Signed fixed-point binary for fractions

signed fixed-point char example	weight in decimal
1000.0000	-8
0100.0000	4
0010.0000	2
0001.0000	1
0000.1000	0.5
0000.0100	0.25
0000.0010	0.125
0000.0001	0.0625

Table: Weight of each bit in an example fixed-point binary number

- ▶ $-.625 = -8 + 4 + 2 + 1 + 0 + .25 + .125 = 1111.0110_2$
- ▶ $1001.1000_2 = -8 + 1 + .5 = -6.5$

Limitations of fixed-point

- ▶ Can only represent numbers of the form $x/2^k$
- ▶ Cannot represent numbers with very large magnitude (great range) or very small magnitude (great precision)

Bit shifting

$\ll N$ Left shift by N bits

- ▶ multiplies by 2^N
- ▶ $2 \ll 3 = 0000_0010_2 \ll 3 = 0001_0000_2 = 16 = 2 * 2^3$
- ▶ $-2 \ll 3 = 1111_1110_2 \ll 3 = 1111_0000_2 = -16 = -2 * 2^3$

$\gg N$ Right shift by N bits

- ▶ divides by 2^N
- ▶ $16 \gg 3 = 0001_0000_2 \gg 3 = 0000_0010_2 = 2 = 16/2^3$
- ▶ $-16 \gg 3 = 1111_0000_2 \gg 3 = 1111_1110_2 = -2 = -16/2^3$

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

monteCarloPi.c Using floating point and random numbers to estimate PI

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Programming assignment 2: Graphs, trees, queues, hashes

Programming Assignment 2 parts

1. hashTable: implementing a separate-chaining hash table.
2. edgelist: loading and printing a graph
3. mst: a greedy algorithm
4. solveMaze: needs either DFS (stack) or BFS (queue)
5. findCycle: needs either DFS (stack) or BFS (queue)
6. matChainMul: a dynamic programming approach to multiplying matrices with minimal operations using recursion

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate π

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Using `graphutils.h`

- ▶ The adjacency list representation
- ▶ The edgelist representation
- ▶ The query

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate π

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

Binary search tree

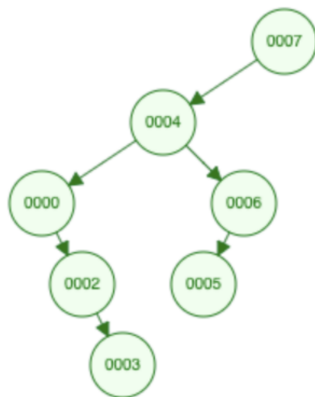


Figure: BST with input sequence 7, 4, 7, 0, 6, 5, 2, 3. Duplicates ignored.

Binary search tree level order traversal

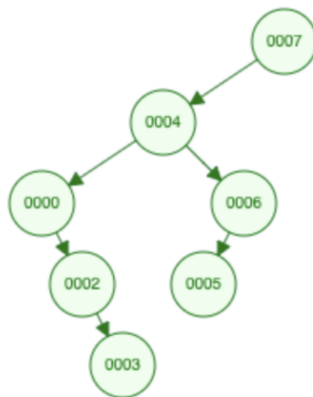


Figure: Level order, left-to-right traversal would return 7, 4, 0, 6, 2, 5, 3.

Binary search tree traversal orders

Breadth-first

- ▶ For example: level-order.
- ▶ Needs a queue (first in first out).
- ▶ Today in class we will build a BST and a Queue.

Depth-first

- ▶ For example: in-order traversal, reverse-order traversal.
- ▶ Needs a stack (first in last out).
- ▶ But in the example code implementation, *where is the stack data structure?*

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate π

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

typedef

Why types are important

- ▶ Natural language has nouns, verbs, adjectives, adverbs.
- ▶ Type safety.
- ▶ Interpretation vs. compilation.

BSTNode

```
typedef struct BSTNode BSTNode;  
struct BSTNode {  
    int key;  
    BSTNode* l_child; // nodes with smaller key will be in left s  
    BSTNode* r_child; // nodes with larger key will be in right s  
};
```

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

QueueNode, Queue

```
// queue needed for level order traversal
typedef struct QueueNode QueueNode;
struct QueueNode {
    BSTNode* data;
    QueueNode* next; // pointer to next node in linked list
};
typedef struct Queue {
    QueueNode* front; // front (head) of the queue
    QueueNode* back; // back (tail) of the queue
} Queue;
```

Let's implement `enqueue()`

<https://visualgo.net/en/queue>

- ▶ First, consider if queue is empty.
- ▶ Then, consider if queue is not empty. Only need to touch back (tail) of the queue.

Let's implement `dequeue()`

<https://visualgo.net/en/queue>

- ▶ First, consider if queue will become empty.
- ▶ Then, consider if queue will not not empty. Only need to touch front (head) of the queue.

Subtle point: why are the function signatures (return, parameters) of `enqueue()` and `dequeue()` the way they are?

Data representation: Integers, Fixed Point

Announcements

Canvas timed quiz 4 and programming assignment 2

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

`monteCarloPi.c` Using floating point and random numbers to estimate PI

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

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

matChainMul.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)

matChainMul.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

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

Parenthesization 2: 2+2 = 4 multiplies

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

Matrix Chain Multiplication: The Problem

- ▶ **Associativity:** The order of parenthesization in matrix multiplication does not change the final mathematical result.
- ▶ **Computational Cost:** The order *drastically* changes the number of scalar multiplications required.
- ▶ **Cost Formula:** Combining a left subchain (dimensions $L \times M$) with a right subchain (dimensions $M \times N$) costs $L \times M \times N$ scalar multiplications.

matChainMul.c: Dynamic Programming Approach

- ▶ **Divide and Conquer:** To find the optimal order, decide which multiplication to perform *last* (the split point), then recursively solve the left and right subchains.
- ▶ **The Algorithm:**
 - ▶ Iterate over all possible split points in the chain.
 - ▶ Cost = Cost(Left) + Cost(Right) + Combine Cost ($L \times M \times N$).
 - ▶ Return the minimum cost found.
- ▶ **Base Case:** A single matrix. Its multiplication cost is 0.
- ▶ **PA2 Goal:** Compute the optimal cost *and* perform the actual multiplications in that exact order, tracking the operations globally.

matChainMul.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

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