C Programming: Debugging, Bits, Bytes, Integers

Yipeng Huang

Rutgers University

February 13, 2024
Table of contents

Strategies for correct software & debugging

Announcements
- Canvas timed quiz 3 and programming assignment 2
- Reading assignment: CS:APP Chapters 2.1, 2.2, 2.3

Bits and bytes
- Why binary
- Decimal, binary, octal, and hexadecimal
- Representing characters
- Bitwise operations

Integers and basic arithmetic
- Representing negative and signed integers

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()
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?
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.
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 printf statements.
Table of contents
Strategies for correct software & debugging
Announcements
Canvas timed quiz 3 and programming assignment 2
Reading assignment: CS:APP Chapters 2.1, 2.2, 2.3

Bits and bytes
Why binary
Decimal, binary, octal, and hexadecimal
Representing characters
Bitwise operations

Integers and basic arithmetic
Representing negative and signed integers

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()
Programming assignment 2

1. Due Friday 2/23.
2. Graph algorithms and hash table.
Reading assignment: **CS:APP Chapters 2.1, 2.2, 2.3**

All about integers

1. We will launch in to our chapter on representing data in computers
2. First: all about integers, signs, capacities, operations.
Table of contents

Strategies for correct software & debugging

Announcements
Canvas timed quiz 3 and programming assignment 2
Reading assignment: CS:APP Chapters 2.1, 2.2, 2.3

Bits and bytes
Why binary
Decimal, binary, octal, and hexadecimal
Representing characters
Bitwise operations

Integers and basic arithmetic
Representing negative and signed integers

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()
Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
  - Computers determine what to do (instructions)
  - ... and represent and manipulate numbers, sets, strings, etc...

- Why bits? Electronic Implementation
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires
01001...

Analog → Digital → Quantum

\[ |\frac{1}{2}\rangle_0 + |\frac{1}{2}\rangle_2 \]

1950s → 1960s → 2000s → 2020s
### Decimal, binary, octal, and hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b0000</td>
<td>0o0</td>
<td>0x0</td>
</tr>
<tr>
<td>1</td>
<td>0b0001</td>
<td>0o1</td>
<td>0x1</td>
</tr>
<tr>
<td>2</td>
<td>0b0010</td>
<td>0o2</td>
<td>0x2</td>
</tr>
<tr>
<td>3</td>
<td>0b0011</td>
<td>0o3</td>
<td>0x3</td>
</tr>
<tr>
<td>4</td>
<td>0b0100</td>
<td>0o4</td>
<td>0x4</td>
</tr>
<tr>
<td>5</td>
<td>0b0101</td>
<td>0o5</td>
<td>0x5</td>
</tr>
<tr>
<td>6</td>
<td>0b0110</td>
<td>0o6</td>
<td>0x6</td>
</tr>
<tr>
<td>7</td>
<td>0b0111</td>
<td>0o7</td>
<td>0x7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0b1000</td>
<td>0o10</td>
<td>0x8</td>
</tr>
<tr>
<td>9</td>
<td>0b1001</td>
<td>0o11</td>
<td>0x9</td>
</tr>
<tr>
<td>10</td>
<td>0b1010</td>
<td>0o12</td>
<td>0xA</td>
</tr>
<tr>
<td>11</td>
<td>0b1011</td>
<td>0o13</td>
<td>0xB</td>
</tr>
<tr>
<td>12</td>
<td>0b1100</td>
<td>0o14</td>
<td>0xC</td>
</tr>
<tr>
<td>13</td>
<td>0b1101</td>
<td>0o15</td>
<td>0xD</td>
</tr>
<tr>
<td>14</td>
<td>0b1110</td>
<td>0o16</td>
<td>0xE</td>
</tr>
<tr>
<td>15</td>
<td>0b1111</td>
<td>0o17</td>
<td>0xFF</td>
</tr>
</tbody>
</table>

In C, format specifiers for printf() and fscanf():

1. decimal: `%d`
2. binary: none
3. octal: `%o`
4. hexadecimal: `%x`
Base 10.

\[
\begin{array}{cccccc}
9 & 9 & 9 & 9 & 9 \\
1000 & 100 & 10 & 1
\end{array}
\]

\[9 \times 10^5 + 9 \times 10^4 + 9 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 \]

= 1M - 1

Base 2

\[
\begin{array}{ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
Decimal, binary, octal, and hexadecimal

How to represent the range of unsigned char in each?

Unsigned char is one byte, 8 bits.

1. decimal: 0 to 255
2. binary: 0b0 to 0b11111111
3. octal: 0 to 0o377 (group by 3 bits)
4. hexadecimal: 0x00 to 0xFF (group by 4 bits)
Often encountered use of hexadecimal: RGB colors as opposed to CMYK

<table>
<thead>
<tr>
<th>#000000</th>
<th>#FFFFFF</th>
<th>#6A757C</th>
<th>#CC0033</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0\times16+10) = 106</td>
<td>(0\times75) = 117</td>
<td>(0\times7C) = 124</td>
<td></td>
</tr>
</tbody>
</table>

Red, green, blue values ranging from 0-255

#000000 (0,0,0) = \(0\times16+3\) = 03

#FFFFFF (255,255,255) = \(3\times16+3\) = 51

#6A757C (106,117,124) = \(2\times16+4\) = 304

#CC0033 (204,0,0) = \(12\times16+12\) = 204
Often encountered use of hexadecimal: RGB colors

Red, green, blue values ranging from 0-255

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Hexadecimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#000000</td>
<td>Black</td>
</tr>
<tr>
<td>#FFFFFF</td>
<td>White</td>
</tr>
<tr>
<td>#6A757C</td>
<td>Gray</td>
</tr>
<tr>
<td>#CC0033</td>
<td>Red</td>
</tr>
</tbody>
</table>
Representing characters

- char is a 1-byte, 8-bit data type.
- ASCII is a 7-bit encoding standard.
- "man ascii" to see Linux manual.
- Compile and run ascii.c to see it in action.
- Some interesting characters: 7 (bell), 10 (new line), 27 (escape).

Figure: ASCII character set. Image credit Wikimedia
Bitwise operations

Why are bitwise operations important?

- Network and UNIX settings using bit masks (e.g., umask)
- Hardware and microcontroller programming (e.g., Arduinos)
- Instruction set architecture encodings (e.g., ARM, x86)
Bitwise operations

\[ \sim: \text{bitwise NOT} \]

unsigned char \( a = 128 \)

\[ \begin{array}{c|c}
\hline
b & \sim b \\
\hline
0 & 1 \\
1 & 0 \\
\hline
\end{array} \]

\[ \sim a = \sim 0b1000_0000 = 0b0111_1111 = 127 \]
Bitwise operations

&: bitwise AND

3 & 1 = 0b11 & 0b01
  = 0b01
  = 1

\begin{verbatim}
printf("%d/n", 3 & 1);
\end{verbatim}

\[ 3 \rightarrow 0b00011 \]
\[ \& 1 \rightarrow 0b00001 \]
\[ 0b00001 \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a &amp; b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ a \]
Bitwise operations

| : bitwise OR

\[
3 | 1 = 0b11 | 0b01 \\
= 0b11 \\
= 3
\]

\[
2 | 1 = 0b10 | 0b01 \\
= 0b11 \\
= 3
\]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

printf("0b\"%d\"\" 3 | 1\")

3 => 0b0011
1 => 0b0001

0b0011

2 => 0b0010
1 => 0b0001

0b0011
Bitwise operations

`printf("\%d", 3 ^ 1);`

This is NOT 3!

`: bitwise XOR

\[ a \land 1 = \text{0b}10 \text{\land 0b}01 \]

\[ = \text{0b}10 \]

\[ = 2 \]

\[ 3 \Rightarrow \text{0b}0011 \]

\[ 1 \Rightarrow \text{0b}0001 \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a ^ b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**inplaceSwap.c**: Swapping variables without temp variables.

\[
y = (x \land y);
\Rightarrow x = x \land \frac{y}{y} = x \land (x \land y) = (x \land x) \land y
\Rightarrow y = x \land y;
\]

How does it work?

\[
y = y \land (x \land y);
\]

\[
y = y \land (y \land x);
\]

\[
= (y \land y) \land x
\]

\[
= 0 \land x
\]

\[
= x
\]

\[
x \leq y;
\]

\[
y \leq x;
\]
Don’t confuse bitwise operators with logical operators

Bitwise operators

- ~
- &
- |
- ^

Logical operators

- !
- &&
- ||
- !=(for bool type)
Table of contents

Strategies for correct software & debugging

Announcements
- Canvas timed quiz 3 and programming assignment 2
- Reading assignment: CS:APP Chapters 2.1, 2.2, 2.3

Bits and bytes
- Why binary
- Decimal, binary, octal, and hexadecimal
- Representing characters
- Bitwise operations

Integers and basic arithmetic
- Representing negative and signed integers

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()
Representing negative and signed integers

Ways to represent negative numbers

1. Sign magnitude
2. 1s’ complement
3. 2’s complement
Representing negative and signed integers

Sign magnitude
Flip leading bit.
Representing negative and signed integers

1s’ complement

- Flip all bits
- Addition in 1s’ complement is sound
- In this encoding there are 2 encodings for 0
  - -0: 0b1111
  - +0: 0b0000
Representing negative and signed integers

2’s complement

<table>
<thead>
<tr>
<th>signed char</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
</tr>
<tr>
<td>00100000</td>
<td>32</td>
</tr>
<tr>
<td>01000000</td>
<td>64</td>
</tr>
<tr>
<td>10000000</td>
<td>-128</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>signed char</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
</tr>
<tr>
<td>00100000</td>
<td>32</td>
</tr>
<tr>
<td>01000000</td>
<td>64</td>
</tr>
<tr>
<td>10000000</td>
<td>-128</td>
</tr>
</tbody>
</table>

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

Table of contents
Strategies for correct software & debugging

Announcements
  Canvas timed quiz 3 and programming assignment 2
  Reading assignment: CS:APP Chapters 2.1, 2.2, 2.3

Bits and bytes
  Why binary
  Decimal, binary, octal, and hexadecimal
  Representing characters
  Bitwise operations

Integers and basic arithmetic
  Representing negative and signed integers

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()
Programming assignment 2: Graphs, trees, queues, hashes

Programming Assignment 2 parts

1. edgelist: loading and printing a graph
2. isTree: needs either DFS (stack) or BFS (queue)
3. mst: a greedy algorithm
4. solveMaze: needs either DFS (stack) or BFS (queue)
5. findCycle: needs either DFS (stack) or BFS (queue)
6. hashTable: a separate chaining hash table
Using `graphutils.h`

- The adjacency list representation
- The edgelist representation
- The query
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).
Why types are important

- Natural language has nouns, verbs, adjectives, adverbs.
- Type safety.
- Interpretation vs. compilation.
typedef struct BSTNode BSTNode;
struct BSTNode {
    int key;
    BSTNode* l_child; // nodes with smaller key will be in left subtree
    BSTNode* r_child; // nodes with larger key will be in right subtree
};
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 be 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?