UMass Boston CS 444
Homework 1
Reposted Saturday, September 12, 2026
Part1: Due Tuesday, September 15, 2026 by section time
Part2: Due Thursday, September 17, 2026 by end of class
Part3: Due Saturday, September 19, 2026 at 11:59 pm

J.H.DeBlois

0 General Instructions

For hw1, the use of AI is not allowed, so be sure to handwrite the items that say handwrite, take photos of your work and include them in your pdf file.
The three parts of hw1 are:

H1-1: (15 points). Login to the CS server and create the subdirectory hw1 in your course directory. Make sure name and location are correct. (Type ls -la.)

H1-2: (20 points total). (15 points) Do the handwritten exercise (bits and bytes, huff and grammar) in class on Thurs, Sept 10th. (5 points) Do the on-server exercise (bits and bytes and huff) also in class on Thurs, Sept 10th. The graders will analyze all the submissions (and some prior versions) using Measure of Software Similarity (MOSS) at Stanford University. If your MOSS number is greater than 45

H1-3: (65 points total). Answer the questions in sections 1 - 6 below.

Guidelines for Submission. Your H1-3 work must be typeset and converted to PDF format. Each drawing must be drawn by hand, photographed and included in your file. Include name of file, submitted location, and full name at the top of page1. Use this exact format:

```
Filename: hw1.pdf
Server location: /home/<cs username>/cs444/hw1/
Your Name: <full name>
```

To submit your homework, prepare a PDF file called `hw1.pdf` — the filename must be exactly `hw1.pdf` because a script will collect it. Common mistakes for filenames include Hw1.pdf, HW1.pdf, hw1.hw1.pdf, which loses the points. It must be an actual pdf, not just a file with extension .pdf.
Upload the file to the `cs444` folder in your home directory on the CS Linux server. Use secure copy command with the -p switch to preserve timestamp. Practice uploading early. Use ls -la to see your file with permissions, timestamp and byte count. Copy each submitted version before you upload again. Use cp -p to preserve the timestamp. For example, type: `cp -p hw1.pdf hw1v1.pdf`.
If you have trouble uploading, email operator@cs.umb.edu for help and copy jane.deblois@umb.edu. Not being able to upload to the server is not a valid excuse for an extension because this is a course in operating systems!

Topics Covered. Homework 1 covers computer bits and bytes; ASCII and other codes; reading T&B OS ch1 Introduction pp 1-81; slides for ch1 (slides 1-22 and 34-40); huffman slides (slides 1-47) covering file compression, the Huffman file compression algorithm, C code functions for bit manipulation and commands for writing bits and bytes to files; and C code grammar for C code constants, operators, functions and basic programs.

1 Metric prefixes from T&B Ch1

a. How many nanoseconds in one second of time? Please handwrite out all digits of the number.

b. How many bits per second is a 1 gigabit per second channel running at? Please handwrite out all the digits of the number.

c. Kilo as a metric prefix means 1000 or 10^3 except if it is measuring memory it means 1024 or 2^{10}. Suppose you have one kilobyte of memory drawn as a rectangle that is 64 bits or 8 bytes wide. This is where the data goes. Assume the addresses are on the left, but do not include addresses in this problem.

1) Draw by hand the rectangle for memory.

2) Handwrite the width in bytes (8 bytes) and as a power of 2.

3) Handwrite your answer for the height in bytes and as a power of 2.

2 ASCII Codes and Extended Code Sets

Handwriting uses an alphabet and numbers. Computers show letters and numbers on the screen, but of course must use bits to store them. Any ASCII code takes 8 bits or one byte of memory for a letter, number or non-printing character. (See the ASCII code set on the class website.)

a. For each of three characters, 'A', '9' and '?', handwrite the codes three ways: in decimal, binary and hex.

There are 128 ASCII codes numbered 0 to 127. The most significant bit of the byte (usually the one farthest to the left) is not used, so it is always zero. The least significant bit of the byte is called bit 0. Number the bits of the byte from right to left as bits 0, 1, 2, 3, 4, 5, 6, 7 (or 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7).

b. Given hex code 0x55, handwrite the bit numbers that are set to one.

c. Compute the decimal number of the code by handwriting the addition of the place values of the set bits.

Typically, unicode requires 2 bytes per code. Extended codes require 1-4 bytes.

d. Handwrite the symbol for unicode U+03AC.

3 Writing Bits and Bytes in Binary, Octal and Hexadecimal

On a 32-bit word machine, messages have 32 bits and both addresses and data in memory have 32 bits each.

a. Handwrite the ones and zeroes for your choice of 32-bit word binary number that has 11 ones and 21 zeroes.

b. Start at the right end, translate each 3 bits to octal (base 8) and handwrite the answer. Pad the last character with zeroes.

c. Using the same binary number, start at the right end, translate each 4 bits to hexadecimal (base 16, so the digits are: 0, 1, 2, ..., 9, A, B, C, D, E, F) and handwrite the answer.

4 Huffman coding

The following lists the character frequencies in a file:

Char:	1	2	3	4	5	6	7	8	9
Freq:	32	29	19	17	13	11	7	3	2

(a) Apply Huffman's algorithm to the above data and draw the prefix code tree in steps. The leaves should be annotated with the characters and their frequencies, and internal nodes should be annotate with the total frequencies of their subtrees. When merging two subtrees, put the lower-frequency subtree on the LEFT. Make a series of drawings by hand that show all your work from forest to complete tree. Please do not use a graphics program.
(b) Fill out the following table with the Huffman codes for each character. On the prefix code tree, a left branch is a 0 and a right branch is a 1. Assemble the code for each leaf node character by reading the branch numbers from top to leaf node. Make a chart of the characters and their codes.

Char:	1	2	3	4	5	6	7	8	9
Code:									

You can draw the Huffman tree by hand, take a picture and include the image in the PDF file of your submission. Be sure to take a picture at each step of your work, starting from individual frequencies for the characters. After you draw all the steps combining the frequencies, sorting as you go, then draw the tree again with the total frequency at the top.

5 Compiling Tokens

T&B ch1 explains C code on pp. 150-175. Two examples are given. The compile process is summarized in Figure 1-25, page 170.

K&R explains C code, the compile process and the grammar. The C code grammar is presented five ways:
as a tutorial, in ch1;
as the basics of C, in ch2-ch6 (ch1-ch6 was already studied in cs240);
as library extensions to interact with a specific operating system (OS), now Linux not Unix, in ch7-ch8;
as grammar in compile process order, Appendix A.1-A.12; and
as a topdown grammar syntactic categories list, in A.13, which has 33 categories written in italics.

Tokens in C code. There are six types of tokens written in typewriter style (see A2.1):
1-identifiers (see A2.3),
2-keywords (32 of them, see A2.4),
3-constants: integer, character, floating and enumeration (see A2.5),
4-string literals (see A2.6),
5-operators (see A7.4 for unary operators and A7.6 to A7.18 for others), and
6-other separators, which include: [ ] ( ) ;

In the C program below, the tokens are separated by two spaces. Actual full code for printf in GNU glibc is not shown (it would add about 20 more tokens). Its source code is not normally on the server. It is linked in as an object module.

Handwrite the program as tokens with spaces. Then circle each token and label its type by its number 1-6 explained above.

```
int  main  (  )  {  extern  int  printf  (  "Hello World"  )  ;  extern  int  printf  (  "\n"
)  ;  return  0  ;  }
```

6 C Code Bit Manipulation Programs

In the Huffman slides, in addition to explaining compression algorithms, we look at Bitwise Operations in C code. These are the operators:

`~, &, |, ^, <<, >>.`

Operators can be unary or binary, meaning they work with one or two operands. In the above list, only & as address operator and õne's complement are binary operators.

Handwrite the math in the last phrase with the answer in place of '?'.

```
a. ~ Flip each bit in the operand.  Operand: 10101010.  So: ~10101010 == ?

b. & Compute bitwise AND.  Operands: 01100110, 11110000.  Copies a bit if it exists in both.
So: 01100110 & 11110000 == ?

c. | Compute bitwise Inclusive OR.  Same operands.  Copies a bit if it exists in one but
not in the other.  So: 01100110 | 11110000 == ?

d. ^ Compute bitwise Exclusive XOR.  Same operands.  Copies a bit if it exists in one
but not both.  So: 01100110 ^ 11110000 == ?

e. << Left Shift.  Syntax: operand << numOfBitPositions.  Multiplies by a power of 2.
So: 0x55 << 1 == ?

f. >> Right Shift.  Syntax: operand >> numOfBitPositions.  Can divide by a power of 2.
So: 0x55 >> 1 == ?
```