Computer Systems Organization

CSCI 463-1

Spring 2023

Homework #8

# I. Endianness

Consider the following program:

#include <iostream>

#include <iomanip>

#include <fstream>

using namespace std;

int main()

{

int i = 0x87654321;

char u[5] = "unix";

ofstream myfile;

myfile.open("fileout.txt", ios::binary);

myfile.write((char \*)&i, sizeof(i));

myfile.write((char \*)u, 4);

myfile.close();

}

1. Run this program on turing/hopper.

a) List the characters in the output file in hex in increasing order. Here is one way to see the characters in order; there are surely others:

emacs -nw fileout.txt // start emacs

<esc>x hexl-mode // after you type <esc>x, the cursor will drop to the bottom line for

// entry of the rest of the command

<cntl>x<cntl>c // exit emacs

These commands use the hex mode of the emacs editor. Remember that <esc>x means to press the escape key, then the letter ‘x’.

Emacs hex mode uses a three-column display. The top line of the display gives the header for each column. The first column contains the hex address of the beginning of the line. The header ‘87654321’ indicates that the address could contain up to 8 hex digits. The second column contains 16 bytes of data represented in hex. The heading ‘0011...’ indicates that each byte contains two hex digits and therefore requires two characters to print. The third column represents the same data in ASCII and therefore only requires one character per byte.

b) What does this tell you about turing/hopper: are they big-endian or little-endian? How do you know?

c) Run the following command: od -t cx1 fileout.txt  
Explain the result, i.e., what do you see, and why? (Use man od if you need more information.)

d) Same as previous question using the following command: od -t cx2 fileout.txt

e) Same as previous question using the following command: od -t cx4 fileout.txt

f) Run the following command: od -t cx fileout.txt  
What does this tell you about the default value for the x value of the -t operand of od?

# II. Character representation

2. Show the representation of the string 'BKbz09' in each of the formats listed.

|  |  |
| --- | --- |
| Coding system | Representation |
| Unicode (code points) | a) |
| Ascii (hex) | b) |
| UTF‑8 (hex) | c) |

3. How many bytes would the string 'BKbz09' would occupy in each of the following formats?

|  |  |
| --- | --- |
| Coding scheme | # of bytes |
| Ascii | a) |
| UTF‑8 | b) |
| UTF‑16 | c) |
| UTF‑32 | d) |

4. Show the representation of each of the following values as a single hex byte.

|  |  |
| --- | --- |
| Value | hex byte |
| unsigned binary 3 | a) |
| binary +3 (two’s complement form) | b) |
| binary -3 (two’s complement form) | c) |
| Ascii '3' | d) |
| UTF‑8 '3' | e) |

Fill in the following charts (on the answer sheet). The first chart shows the number of Unicode code points in the basic multilingual plane (BMP, i.e., code page 0) that need 1 byte, 2 bytes, etc. for their representation in UTF-8. The second and third charts show the same thing for UTF‑16 and UTF‑32.

Express code points as U+hhhh, where hhhh is a 4-digit hex number, e.g., U+0000 is the first code point in the Unicode system.

Use ISO-8859-1 for “extended Ascii”.

Hints:

a) The BMP extends from U+0000 to U+FFFF, which is 65536 entries. But U+D800 through U+DFFF don’t represent characters because they are used as control characters, so the total size of the BMP is 65536 ‑ 2048 or 63488 characters. Therefore the numbers in each chart should sum to 63488.

b) Many of the entries in each table are 0. Study the charts in the slides and fill in the 0’s first.

5. UTF‑8

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Category | Range in decimal | First code point | Last code point | 1 byte | 2 bytes | 3 bytes | 4 bytes |
| ASCII | 0-127 |  |  |  |  |  |  |
| extended ASCII | 128-255 |  |  |  |  |  |  |
| rest of BMP | 256-65535 |  |  |  |  |  |  |

6. UTF-16

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Category | Range in decimal | First code point | Last code point | 1 byte | 2 bytes | 3 bytes | 4 bytes |
| ASCII | 0-127 | (same) | (same) |  |  |  |  |
| extended ASCII | 128-255 | (same) | (same) |  |  |  |  |
| rest of BMP | 256-65535 | (same) | (same) |  |  |  |  |

7. UTF-32

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Category | Range in decimal | First code point | Last code point | 1 byte | 2 bytes | 3 bytes | 4 bytes |
| ASCII | 0-127 | (same) | (same) |  |  |  |  |
| extended ASCII | 128-255 | (same) | (same) |  |  |  |  |
| rest of BMP | 256-65535 | (same) | (same) |  |  |  |  |

# Part III: Floating point

8. Fill in the following chart (on the answer sheet). All of the table entries refer to positive numbers. Use powers of 2 for the last four rows. For each entry, either the answer or a formula to calculate it can be found in the slides. (Either way, for purposes of studying for the final, make sure you understand how the numbers were derived!)

|  |  |  |
| --- | --- | --- |
| Category | IEEE single precision f.p. | IEEE double precision f.p. |
| a) No. of bits in exponent |  |  |
| b) No. of bits in significand |  |  |
| c) Bias |  |  |
| d) Smallest non-special exponent  (value without bias) |  |  |
| e) Largest non-special exponent  (value without bias) |  |  |
| f) Smallest normalized number |  |  |
| g) Largest normalized number (approximate by nearest power of 2) |  |  |
| h) Smallest denormalized number |  |  |
| i) Largest denormalized number (approximate by nearest power of 2) |  |  |

9. a) For a given standard, the smallest normalized number is approximately equal to the largest denormalized number. Why?

b) If you don’t round off, which of the two is always larger?