CSE2038/2138 Systems Programming
Project 1
Binary Data Converter
Due: 17.04.2023 11:59PM

The purpose of this assignment is to become more familiar with bit-level representations of integers and floating point numbers.

In this project, you will implement an application, in C or Java programming language; that takes

- a file containing hexadecimal numbers as bytes in a memory,
- the byte ordering type (Little Endian or Big Endian),
- the data type to be converted, and
- the size of the given data type,

as input and converts the contents of the file according to the predefined format and gives the converted data as output.

The data type can be any of the following:

- signed integer
- unsigned integer
- floating point number

The size of the data type can be 1, 2, 3, or 4 bytes.

- If the selected data type is signed integer, your program will convert the numbers in the input file using 2's complement representation.
- If the selected data type is unsigned integer, numbers will be converted using unsigned integer representation.
- If the selected data type is floating point number, you will use IEEE-like format. The number of exponent bits according to given data size will be like the following:
    - if 1 byte (i.e., 8 bits), 4 bits will be used for exponent part
    - if 2 bytes (i.e., 16 bits), 6 bits will be used for exponent part
    - if 3 bytes (i.e., 24 bits), 8 bits will be used for exponent part
    - if 4 bytes (i.e., 32 bits), 10 bits will be used for exponent part
    - While calculating the mantissa to get the floating point value, you will only use the first 13 bits of the fraction part. You will use "round to even" method for rounding fraction bits to 13 bits.

Details about the program are listed below:

- At the beginning of the execution, your program will prompt for the input file.
- The format of the input file will be like the following:
    - Each line includes, 12 hexadecimal numbers separated by a single space; each of which correspond to a byte of data.

Example: input.txt

```
f0 90 01 40 03 00 ff ff 00 00 e0 7f
00 00 e0 ff 00 00 00 00 00 00 00 80
00 00 18 80 00 00 00 00 00 00 00 00
```

- After a valid input file is taken as input, the user will be prompted for the byte ordering type, data type and size:

Example:

Byte ordering: Little Endian
Data type: Floating point
Data type size: 4 bytes

- You program will calculate the decimal value of the file content with the given information:

Example:

The byte ordering is Little Endian, and the data type is 4 byte floating point.

- First, your program will read the next 4 bytes of the file. For our "input.txt" the first four bytes is:
  f0 90 01 40
- The byte ordering is Little Endian, so the floating point number is:
  40 01 90 f0, in binary: **0100 0000 0000 0001 1001 0000 1111 0000**
- In the specification, we are given that our 4 byte IEEE-like floating point numbers have 10 bits of exponent part, so Bias = $2^{10-1} - 1 = 511$:
  Sign bit = **0**
  Exponent = **$(1000000000)_2$ = 512**
  Fraction = **00001100100001111110000** rounded fraction = **0000110010001**
  mantissa = **1+1/32+1/64+1/512+1/8192 = 1.0489501953125**

Decimal value = **$(-1)^0 * 1.0489501953125 * 2^{512-511} = 2.097900390625$**
In the output file, the printed value will be: **2.09790**

Example 2:

- If the program reads the following four bytes:
  00 00 18 80
- The byte ordering is Little Endian, so the floating point number is:
  80 18 00 00, in binary: **1000 0000 0001 1000 0000 0000 0000 0000**
- In the specification, we are given that our 4 byte IEEE-like floating point numbers have 10 bits of exponent part, so Bias = $2^{10-1} - 1 = 511$:
  Sign bit = **1**
  Exponent = **0000000000 denormalized number**
  Fraction = **11000000000000000000000** rounded fraction = **1100000000000**
  mantissa = **0+1/2+1/4 = 0.75**

Decimal value = **$(-1)^1 * 0.75 * 2^{1-511} = -2.23750222e{-}154$**
In the output file, the printed value will be: **-2.23750e-154**

- The output file will be:

output.txt

2.09790 NaN ∞
-∞ 0 -0
-2.23750e-154 0 0

- In the output file, the floating point numbers will have precision of maximum 5 digits after the decimal point.

- You cannot use library functions for the binary to decimal conversions.

This is a group project. Each group should have 4 group members. You will submit the source code of your program using the Canvas service. The file name should include your student IDs,s e.g., 150116001_150116002_150116003_150116004.c. If you submit multiple files in a zip file, you need to name both zip file and the file containing your main function/method like this.

(150116001_150116002_150116003_150116004 .zip file containing multiple files one of which has the name 150116001_150116002_150116003_150116004.c)

Keep in mind that the due date will not be extended. So, please start early.

IMPORTANT

- The project will be done in groups of 4 people. (Mandatory!)
- Your project will be graded with an auto grader. Here are the rules you should follow for the auto grader to correctly grade your programs:
    - The name of the output file is always output.txt.
    - The format of the input and output files are exactly given above. (No redundant spaces and newlines! Each line in the output file has to contain (12/size) numbers)
    - The inputs should be taken in the order: inputFileName, byteOrderig (l or b), dataType (float or int or unsigned), size
    - You should not put any spaces in the name of your program or zip file.
    - You should not use Turkish characters anywhere. (Inside programs or as filenames)
    - When extracted, make sure that your program files are directly under the extracted directory. (the extracted directory should not contain any subdirectory)
    - If your program does not follow these rules, then it cannot be graded.
- Project from previous years will get 0 grade!