CS-102 Project 2 – Base Conversion

Professor: Brian Frost

Summer 2019

Specifications

In this project, you will use your working knowledge of standard input and output, functions, and loops to create a very useful tool: a binary/decimal converter.

Create two functions – one which converts a decimal integer between 0 and 255 to a binary byte and one which converts a binary byte to a decimal integer between 0 and 255. I have given some background in how to do this in the following section.

I want the main program to use these two functions to handle input in the following way:

- The first input is a single letter 'd' or 'b'. An input of 'd' means the user will input decimal values to be converted to binary. A 'b' means thee user is inputting bytes to be converted to decimal.
- After this first input, the program should line-for-line convert the numbers, printing them to standard output line-for-line.
- If a line is an invalid input (number above 255, negative, a string, binary string too long), print the line 'INVALID INPUT' and continue to the next number.
- The program should stop when EOF is reached.

I want bytes printed as 8 bits even if there are a number of leading zeros. It should be assumed that bytes are fed in this way as well.

For example, if my input is a file that looks like

```
d
8
255
-1
1
```

My output should be

```
00001000
11111111
INVALID INPUT
00000001
```

I will test both directions with input FILES of each type (decimal-tobinary and binary-decimal). Simply getting this correct is worth the majority of points here!

Some suggestions: Use character arrays for binary bytes rather than integers (you can use integers but it is a bit sloppy). Use getchar/putchar rather than scanf as you don't know if honest integers are being input. Use ASCII arithmetic where it could make things easier. Test each function separately first, before implementing the input/output overhead.

Lastly, don't go too crazy on catching bad inputs. I would be upset if it couldn't catch a number that is too large or a string, but if it's something really crazy I probably won't test for it.

Extra Credit

Do the same but for hexadecimal conversion – an input of D means I want to convert decimal numbers to hexadecimal, and an input of x means I want to convert from hexadecimal to decimal. Again, integers here are between 0 and 255 – how many hex digits do you need to represent that?

A Brief Explanation of Base Conversion

We can represent any decimal integer as a sum of its digits multiplied by powers of ten, e.g. $253 = (2 \times 10^2) + (5 \times 10^1) + (3 \times 10^0)$. This is an arbitrary choice of representation based on our having 10 fingers. As we covered in class, computers eventually operate through binary instructions, corresponding to the on/off "binary" nature of electronic switches.

In binary, each number is represented as a sum of its digits multiplied by corresponding powers of two, and each digit is either a zero or a one. For example, the binary number 1101 can be converted to decimal by writing $(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 13$. This gives an easy way to go from binary to decimal in the case of a positive integer. Negative integers are more complicated, and will not be covered in this course, although if you are interested you can look up "two's complement".

For this assignment, we do not use fractional parts, but they work the same way with negative powers of two. The difficult part is encoding where the decimal point is!

So still assuming positive integral values, how do we go the other way? I'll show by example with the number 113.

1. First, find the highest power p of two which is smaller than or equal to your value. $2^6 = 64$ and $2^7 = 128$, so $p = 6$.

2. Your first '1' bit goes in the $p + 1^{\text{st}}$ place. If you are trying to represent the number as a byte, you must thereby lead with a 0 in the highest place (i.e. a 0 in the 128s place). You also know there is a 1 in the 64s place – the second-most significant bit. Thus, your number looks like 01______.

3. Subtract 2^p from your number and repeat step 1 to find a new p. $113 - 64 = 49$, and $2^5 = 32$, so $p = 5$. Put a one in the $p + 1 = 6^{\text{th}}$ place. Now your number is 011_____

4. Repeat again, subtracting 2^p from your new number. $49 - 32 = 17$. Find p again; here it is 4 as $2^4 = 16$. Put a one in the fifth place, for 0111____

5. And again! Now it is a bit more interesting – $17 - 16 = 1$. We know $1 = 2^0$, so $p = 0$. That means we put a one in the first place, and fill in all higher places with zeros! Our final answer is 01110001.

Try to play the game in reverse and see if you get 113 back. Do this by hand both ways if you are unfamiliar. Note that this “algorithm” could just as easily be performed on a computer with slight modifications. For example, instead of finding the highest power of two that is smaller than or equal o your number at each step, you might check if the next power of two satisfies this condition. If it does, then place a 1 and subtract. If not, place a zero.

This can be tricky if it is your first time seeing this stuff so don’t worry if you’re having a bit of trouble.

For those interested in the extra credit, hexadecimal works thee same way, but the base is 16! Thus, we must add new digits to our number system. Decimal 10 is ‘A’ in hex, 11 is ‘B’ and so on to 15 which is ‘F’. Here’s an example of hexadecimal-to-decimal conversion: I have A4 in hex, so in decimal this is $A \times 16^1 + 4 \times 16^0 = 10 \times 16 + 4 = 164$. Just the same as binary! Decimal-to-hexadecimal is not too different either, but you must account for the fact that digits are not always one of 1 or 0.

Submission and Grading

Please submit your assignment as a .c file to b.frost@columbia.edu by midnight on Wednesday, July 24. Remember that lateness is penalized! Please state which operating system, text editor and compiler you used in your email. Also, please don’t hesitate to ask as many questions as you need through email, or by coming to office hours.

Grading is broken down according to **Correctness** and **Style**. Your total grade is out of 100 points.

Correctness (80 points): This is simply how well your program fits the exact above-stated specifications.

Style (20 points): This is composed of efficient commenting, intuitive spacing and indentation, intuitive variable names, and overall elegant code. Unnecessary statements, poor placement of variable instantiations and otherwise difficult-to-read code will result in a loss of points here. Remeber to put a brief comment at the top of your code explaining the program’s operation.

Extra Credit (10 points): If you complete the extra credit, you can earn up to ten points back. The highest possible grade is still a 100, however.