Unless you are thinking in base-2 or base-16, converting between decimal, hexadecimal, and binary integer representations is extremely valuable when reasoning and discussing how data is stored by a computer. The conversion is algorithmic, but it is tedious. Is there a way we could automate the conversion of a decimal integer to its hex or binary equivalent?

For this assignment, you will write a C program, *convert*, that reads decimal integers from standard input (a.k.a., the keyboard) and prints the representation of each value in either hex or binary. The program will continue to read values until reaching the end-of-file signal. What is the end-of-file for the keyboard? Control-D. We use control characters so often, “control” is usually written with a carrot. For example, ^D.

The program convert must take a single command-line argument that is either "-x" or "-b", where "-x" indicates that the program should print the input value in hexadecimal, and "-b" indicates that it should print binary.

Outcomes

- Gain more experience with C programming, the C library, compiling C programs, etc.
- Properly use some of the C bit-wise operators
- Produce well formatted code.

Example invocations

```
$./convert -x
1234
0x4d2
879
0x36F
^D
```

```
$./convert -b
1234
100 1101 0010
4321
1 0000 1110 0001
^D
```

```
$./convert
Usage: ./convert [-x|-b]
```

The dollar sign, $, is a generic shell prompt. Your prompt will probably be different. Carrot-D (^D) is shorthand for control-D.

Implementation Details

Name the source file *convert.c*. Create the file with your favorite text editor (nano was mentioned in lab 1, but there are many others).

If you develop your code on your personal computer, make sure you're your code compiles correctly on the CS linux machines. You program much check to make sure the correct flag (-x or -b) was issued during invocation.

To get started, here is a skeleton of a program:

```
#include <stdio.h>

/* Main
 * argc, argv   Command line arguments.
 * return       zero for success, non-zero for failure
 *
 * This is the main entry point for the program.
 */
int main(int argc, char* argv[]) {
  printf("%s says, \"Hello, World!\"\n", argv[0]);
  return 0;
}
```

This program doesn't convert anything, but it is a starting point. To verify that it works, compile it and run it (just like you've been shown in lab 1):

```
$ gcc -o convert convert.c
$./convert
./convert says, "Hello, World!"
```

Remember to use the on-line manual: `man`. For example, C-strings or char*s are really weird. The normal comparison operators do not do what you expect, so you have to use the `strcmp` function. How is it used?

Hint: You may want to read the following man-pages: *scanf*, *printf*, *strstr*, and *strcmp*.

You are not allowed to use `printf`'s `%x` format character. In other words, you must do the conversion manually.

Coding Style

Coding style is almost as important (some say more so) than correct code logic. Code should be easy to read, both for your sake when you return to a code file after a long absence, and for the benefit of others because in the real world (post graduation, industry), multiple people work on and share the same code base, so everything should be tidy, well organized, and clean.

For this and all future labs, homework assignments, and projects, please adhere to the following guidelines.

1) Use meaningful names that give the reader a clue as to the purpose of the thing being named.

2) Avoid the repeated use of numeric constants. For any numeric constants used in your program, define a static constant:

   ```
   static const float PI = 3.141592653589793;
   ```

   From then on, use the symbol in place of the value. There is a C convention to use ALL CAPS for constants.

3) Use comments at the start of the program to identify the purpose of the program and include your name. If you are working in groups, BOTH names must be included, but there is only a single submission. For example:

   ```
   /*
    * Name: Marie Curie
    * Description: This program converts decimal to either
    * binary or hexadecimal...
   */
   ```

4) Use comments at the start of each function to describe the purpose of the function, the purpose of each parameter to the function, and the return value from the function.

   ```
   /* Read Decimal
    * This reads and returns a single, decimal integer from stdin.
    * NOTE: Most of the parsing work is performed by scanf.
    */
   int read_decimal()
   ```

5) Avoid in-line comments in the code itself.

6) Use comments at the start of each section of the program to explain what that part of the program does.

7) Use consistent indentation.

8) Write functions (methods). Do not compose a .c file with just a `main` function, with all the code inside of the `main` function.

These are general guidelines to help you produce superior code. However, as an up-and-coming software professional, it is your job to know when to deviate from these rules to produce code that is as easy to read and understand as possible.

Rubric and Submission

Upload your *convert.c* file to Canvas.

Correctness	Points
• Checks for proper invocation	5
• Correctly converts to and prints binary value	5
• Correctly converts to and prints hexadecimal value	5
• Correctly prints values without using `printf`'s %x format character	5
Total	20 points

Be sure to test your program thoroughly. Partial credit is given

And, adhere to correct formatting. This is the quality component of code. You will be deducted points if your code is not of high quality.

Quality
• Programming style (organization, indenting, comments, symbol names, etc.)
• Organization (Intro to file includes comments, meaningful grouping, etc.)
• Use of multiple functions