Prog1: Multi-base input & unsigned and signed representations (CS220-02, 05)

Develop a C (or C++) program that interprets the 6 bit patterns equivalent to values given as unsigned, signed magnitude, 1s and 2s complement numbers. Please use only a single source file so I can more easily script the downloading and compiling of your submissions.

Your program must require a single command line argument, the name of an input file. This input file will consist of any number of lines, each with two items: a base (2 through 16) and a value in that base that will fit in 6 bits (decimal range of 0 through 63). For each input line, you must output:

1. The unsigned decimal value of the 6 bit pattern (no + or - symbol for this one)
2. A space
3. The signed magnitude decimal value of the 6 bit pattern (+ or - symbol required)
4. A space
5. The 1s complement decimal value of the 6 bit pattern (+ or - symbol required)
6. A space
7. The 2s complement decimal value of the 6 bit pattern (+ or - symbol required)
8. A newline

Example input file		**Expected output to console**

`2 011000`
`8 70`
`7 120`
`14 3D`
		`24 +24 +24 +24`
		`56 -24 -7 -8`
		`63 -31 -0 -1`
		`55 -23 -8 -9`

Your output must precisely match the instructor's so test carefully against the examples.

You can assume I will always test with a properly formatted test file and will always provide the correct command line argument. This way you can focus your code on its primary functions rather than error detection. Your program should nonetheless compile without warning or error.

About command line arguments in C and C++

Some students do not have experience writing code that uses command line arguments at this point, though nearly all have experience running programs using command line arguments. For example, when you compile a C++ program with **`g++`** or use secure-copy (**`scp`**) with commands such as…

```
g++ main.cpp
scp file.pdf amos@server.org:~/file.txt
```

...the operating system executes your program by loading it into memory and calling its ***main*** function. This function has two parameters, ***argc*** and ***argv***. The first is an integer identifying how many arguments were sent via command line/OS (the argument count), and the second is a 0-indexed array (the argument vector) of the actual arguments where each argument is a C-style, NULL terminated character string. In the case of the examples above…

argc	**argv[0]**	**argv[1]**	**argv[2]**	**argv[3]**
2	"g++"	"main.cpp"	*undefined*	*undefined*
3	"scp"	"file.pdf"	"amos@server.org:~/file.txt"	*undefined*

These argument strings can be used in any way the developer likes, but always remember that the user can always type as many or as few or as valid or insane arguments as she likes… so in production software, you should always verify the arguments are valid and meaningful.

On the following pages are two simple example programs that take a single argument (a file name), try to open it, and report whether that was successful. The first is in C. The second is in C++.

The C program

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
        if(argc != 2) {
                fprintf(stderr, "Provide a single filename.\n");
                exit(EXIT_FAILURE);
        }

        FILE *f = fopen(argv[1], "r");
        if(f == NULL) {
                perror("The file didn't open correctly");
                exit(EXIT_FAILURE);
        }

        printf("%s is open/ready for use.\n", argv[1]);
        fclose(f);

        exit(EXIT_SUCCESS);
}
```

The C++ program

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main(int argc, char *argv[]) {
        if(argc != 2) {
                cerr << "Provide a single filename." << endl;
                exit(EXIT_FAILURE);
        }

        ifstream f(argv[1]);
        if(!f.is_open()) {
                perror("The file didn't open correctly");
                exit(EXIT_FAILURE);
        }

        cout << argv[1] << " is open/ready for use." << endl;
        f.close();

        exit(EXIT_SUCCESS);
}
```