Dylan Dunsheath HW5 Description

**Problem #1**

This program wasn’t that difficult. I initially started off the program with three headers: string.h (string manipulation); helpful for strlen, ctype.h; helpful for deciding if a character is printable in my program, and stdio.h for simple output (no input excluding command-line). We then start off with the main function..

As we enter the main program, you may notice the “int argc, char \*argv[].” This means that the program takes the command line arguments, where argc is the number of arguments passed (argc – 1 is the ACTUAL number of arguments, since it include the executable name) and argv is an array of pointers to the arguments (array of strings). In the main function, I also have a loop counter called i, a character called c, a char pointer called ptr, and another char pointer called last (which is a pointer to the last character of the last argument). I then initialized a string that will initialize "argv | " to firstOutput and print it out. I then declared a function printHexPtr that essentially print the hexadecimal representation of the address of argv (which takes a void pointer called argv that is passed to the function, now declared/called value). There is a char pointer ‘c’ and a ‘cast’ of a void pointer ‘value’ to the pointer ‘c’ [ie: c = (char \*)value;] After this, I looped from the MSB (bye 8 or 7 🡪 the last byte of the pointer) down to the 0th or first byte, as pointers are 8 bytes (at least on most machines) with a for loop. The initialization was 7 and the end point is EXACTLY 0. If i is 7, I printed a whitespace (won’t run when the loop first starts). Otherwise, it prints a hex representation of the current byte of the pointer with 2 digits.

After the function is finished, the third part of argv that’s printed is simply the address of argv and then newlines the output. After this is done (the address is printed for argv), we essentially need to loop through all arguments (from 0 to N arguments, or argc - 1). Within the loop, we print the current argument index (argv[0], argv[1],… argv[argc – 1]) and similarly with argc, pass the CURRENT argument to printHexPtr to print the hexadecimal representation of said argument and afterwards, the address of the argument.

When the for loop finishes, the value of ‘ptr’ is assigned/given the content of the value of the first argument and masks the lower bits to align it to 8 byte boundary and also finds the pointer to the last character… I then looped through the characters of arguments (iterated through each character with a while loop from the start to the LAST character and printed each character in hex format followed by its corresponding character if it was printable or the code if not. I then just needed to print the address of the current pointer and move to the NEXT 8 bytes of the argument

**Problem 2**

This function begins with various header files: stdio.h for the output (as we are passing command line arguments for parsing, it would just be output), unistd.h for the parsing with getopt(), and stdlib.h for exit(0). The main function takes argc and argv, as previously mentioned are the number of arguments and the pointer to the arguments (or array of strings). Inside the main function, I have an integer called ‘option’ and then the while loop that parses the arguments and assigns it to option which uses getopt(). This method or function. Getopt requires three arguments: argc, argv, and then some string which gives what arguments are accepted. ":cduf:s:” where the ‘:’ at the beginning essentially specifies that some options require additional arguments. This specifies that -c, -d, -u, -f [which requires an additional argument] and -s [also requires an additional argument] are accepted and that we should parse until the end of the option list.

I then used a ‘switch’ statement for each case that corresponds to a different option and printed based on the specifications laid out in the instructions (ie: -c, -d, and -u didn’t require an additional argument) but -f and -s did. I also ensured if an option required an additional argument but wasn’t specified, that the message was printed out that it needed a value (and if an option which is specified with ‘-‘ is unknown, display to the screen. After parsing the options, the program will loop through any remaining options that ARE NOT options and print them out and exit with status of ‘0’, displaying or signifying success.

**Problem 3**

This program only includes two header files, stdio.h for output and string.h for strcmp for strings. For the main function, there is three different parameters: argc, argv, and envp (for environmental variables). Argc pertains to the command-line argument count, argv is the array of strings containing the command-line arguments, and envp is the array of strings containing the env variables.

I started off the main function by declaring a delimiter, called ‘delimiter’ with the string (or character ‘=’ which separates the name from the value). I then declared two loop counters for the for loop (i and j) and a temporary char pointer used for swapping strings. I then used a simple bubble sort algorithm that initialized the outer loop to start at index 0 until we reach the end of the environmental variables (which iterates over each element) and within the outer loop (before the inner loop), I declared strtok to separate the variable name and value using the delimiter ‘=’. Then, I declared an inner loop that similary iterates over the environmental variables, but starts at an index more than the current ‘i’ loop. Within the inner loop, I used strtok to separate the name from the value using the delimiter ‘=’ and checked using strcmp if the first string is greater than the 2nd string (or variable name of the current element is greater than the variable name of the subsequent element). If it is true, it swaps them. Strtok is helpful because we solely want to focus on comparing variable names and not values. This loop is continuously iterated until we reach the end of the environmental variables for the loops.

Afterwards, we need iterate over all environmental variables and print them out. In the loop, we calculate the length of the current env variable string and append the delimiter ‘=’ to the end of the env variable string which effectively separates the name from the value. We then print the key and values out.