PES Project 1 – 50 Points

This first PES project is intended to let you refresh your C skills a bit before we start on specific embedded systems development. It can be performed individually or by a team of two students. Teams will receive a single grade for submissions. The project is due on Tuesday 9/17 prior to class and will be submitted on Canvas. The Canvas submission will consist of two parts:

Part 1 is a single GitHub repo URL which will be accessed by graders to review code and documentation. This will consist of any C code or include files, any captured output files requested, and a README Markdown document that includes:

- A title (PES Project 1 Readme)
- Names of your team
- A description of the repo contents
- And any installation/execution notes for others using the code

Part 2 will be a PDF containing all C code and README documentation – the PDF is used specifically for plagiarism checks: your code should be your team's alone, developed by your team. You should provide a URL for any significant code taken from a third party source, and you should not take code artifacts from other student teams. However, you may consult with other teams, the SAs, and the instructor in reviewing concepts and crafting solutions to problems (and may wish to credit them in documentation).

Development environment for the project:

For this project, you can use any GCC C99 compiler. If you're not directly using a Linux system, I would recommend using a VirtualBox to create a GCC development environment in a common recent Linux distribution, like Ubuntu. As an alternative, you can also use GCC tools for a Mac (which is available in the terminal for registered Apple Developers, and is also free) or a Windows PC (using Cygwin or MinGW). See the SAs for any assistance you need in establishing a development environment.

Your code should follow the ESE C Style Guide as closely as possible.

When compiling with GCC, use -Wall and -Werror compiler flags. Your code should have no compilation errors or warnings.

Points will be awarded as follows: half for the correctness of output and half for the construction of the code (including following style guide elements and the quality of solution). There is no extra credit in this submission.

Project 1 is due on 9/17 at 3:30 PM.

Assignments will be accepted late for one week, at a penalty of 15% of the grade. After that point, assignments will not be accepted.

Project exercises follow:

Program One (20 points): Create a C program that will take as input a numeric value, a radix, and an operand size. Your program will need to error check for invalid inputs. Radix values are limited to 8, 10, 16. Operand size values are limited to bit sizes of 4,8,16. Numeric input must be valid for the operand size (that is, a number larger than the operand size should throw an error). For each input line read, return the following output to the console (for capture into an output file) – the following output is for the input -6, 10, 4 (numeric value, radix of the value, operand bit size). Any values that cannot be calculated should show the word Error for a result in the table. Note that the first four outputs in the table display absolute values, the last three are signed.

Input: Value -6 Radix 10 Operand Size 4

Output:	Value	Maximum	Minimum
Binary (abs)	0b0110	0b1111	0b0000
Octal (abs)	06	017	0
Decimal (abs)	6	15	0
Hexadecimal (abs)	0x6	0xF	0x0
Signed One's Compliment	0b1001	0b0111	0b1000
Signed Two's Compliment	0b1010	0b0111	0b1001
Sign-Magnitude	0b1110	0b0111	0b1111

Your C program should run against the following data:

Input Set {Value, Radix, Operand Size}

{-6, 10, 4}, {-6, 9, 4}, {-6, 10, 5}, {-9, 10, 4}, {237, 10, 8}, {0354, 8, 8}, {0xEB, 16, 8}, {-125, 10, 8}, {65400, 10, 8}, {65400, 10, 16}, {-32701, 10, 16}

Capture all output into a file called ProgramOne.out and save your final version in your repository to turn in. It does not have to be included in the PDF.

Program Two (15 points): Write a program that uses a logical expression that tests whether a given character code is a

- lower case
- upper case
- digit
- white space (like null, backspace, space, tabs, etc.)
- or a special character (like ! or >) in ASCII.

You can document your decisions of segregating the ASCII codes into the categories.

For each test input print the input code, the type of character, and the ASCII character to the console (for capture to ProgramTwo.out). A typical printed line would look like:

Code: 66 Type: Upper Case ASCII Char: B

Test it on the following ASCII codes:
{66,114,117,99,101,32,83,97,121,115,32,72,105,33,7,9,50,48,49,57}

Program Three (15 points): Given the starting integer value 0xCAFE, perform each of these operations in series, that is, each operation should be performed on the result of the previous function. Print the results of each function to the command line (to capture as ProgramThree.out).

- Print the original input in hexadecimal
- Test if 3 of last 4 bits are on, and print the value in binary (along with the result of the test – true/false)
- Reverse the byte order, print the value in hexadecimal
- Test if 3 of last 4 bits are on, and print the value in binary (along with the result of the test – true/false)
- Rotate the value by four bits to the left, print the value in hexadecimal
- Test if 3 of last 4 bits are on, and print the value in binary (along with the result of the test – true/false)
- Rotate the value by eight bits to the right, print the value in hexadecimal
- Test if 3 of last 4 bits are on, and print the value in binary (along with the result of the test – true/false)

For all programs, captured out files should be placed in your GitHub repo. You do not need to put the output file content in the PDF submission – the PDF should contain only code and readme documentation.