PES Project 1 – 50 Points

Introduction

This first PES project is intended to let you refresh your Make, Git, and 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 2/4 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 the makefile, C source or include files, any captured output files requested, and a README Markdown document.

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 from this or prior semesters. 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 (this is posted in Canvas class files).

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

Assignment details/due date

Points will be awarded as follows: half for demonstrated 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 Tuesday 2/4 at 2 PM.

Assignments will be accepted late for one week. There is no late penalty within 4 hours of the due date/time. In the next 24 hours, the penalty for a late submission is 5%. After that the late penalty increases to 15% of the grade. After the one week point, assignments will not be accepted. The team may submit the assignment using a late pass from each of the team members. Only one late pass can be submitted per project, and it will extend the due date/time 24 hours.

Project exercise elements follow:

README Markdown File (10 points): The markdown file should include the following sections.

- **Title** ("PES Project 1 Readme")
- **Names** of your team members
- **Repository Comments** – A description of the repo contents
- **Project Comments** (issues encountered, suggestions for the project in future)
- **Installation/Execution Notes** for others using the code (including what your development environment was)

GNU Make makefile (10 points): Create a single makefile to build the following programs. The makefile should allow for the following build targets:

- "all" (i.e. make all) – compile and build the executables for all three C programs listed
- "one", "two", "three" – compile and build an individual executable for program one, two, or three
- "clean" – delete all files in the make directories that are normally created by building the program(s); this should allow the targets above to rebuild if make is called for them
- Just as with code elements, your makefile should include appropriate comments and should be well structured
- You should be able to demonstrate these make targets, and show that if called make is called a second time (with all files up to date), no changes are made

Program One (10 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. Since printf does not directly print formatted binary values as strings, you will need to create and use a function for displaying binary value strings for the different operand sizes.

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}

{-7, 10, 4}, {-7, 9, 4}, {-7, 10, 5}, {-10, 10, 4}, {236, 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. The output file does not have to be included in the PDF of code and README.

Program Two (10 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 (10 points): Given the starting integer value 0xFACE, 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). Use the same binary print function you designed for program one to display values in binary.

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

For all deliverables, individual source files, makefiles, captured out files, etc. 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 your makefile, code, and README documentation.