AUBURN UNIVERSITY

CPSC 3350 Project Programming

Instructions For this project, you must write 3 programs in C. These exercises are **extensions** of the exercises assigned for the previous module. For the first two exercises, the extension consists of going from a unique character to a string of characters. For the third exercise, you must get "closer" to the computer and process the numbers as 0s and 1s using **ONLY logic bitwise operations**, rather than arithmetic operations. Do not hesitate to ask the instructor or the TA if you encounter obstacles or have doubts.

Your C programs must compile and execute on the Unix machines on the Engineering Network Systems (**ENS**). If not, no credit will be awarded. These ENS machines can be accessed through the host **gate.eng.auburn.edu**. We will demo in class how to access these machines.

Efficient code is expected.

For all these exercises, you cannot use built-in functions that perform these operations. In case of a doubt, check with your instructor.

Objectives of this project:

- to manipulate the variables as 0s and 1s using logic bitwise operation
- to practice conversion from a base (decimal, hexadecimal, or binary) to another base
- to distinguish between "numbers" and characters
-

What you need to do:

Programming Exercise 1 (30 points): (Call the Source code Exercise1.c)

*This exercise is an extension of Programming Exercise 1 from the previous module's assignment. The extension consists of transitioning from one character to a **string** of characters.*

Write a program that prompts the user to enter a **string *s*** (representing a 16-bit binary integer ***n*)**. Assuming that the 16-bit binary number ***n*** is unsigned, you must write a **function** that takes ***s*** as input and returns the equivalent **integer *n***. Your main program must print out ***n*** in decimal. **It is expected that you will use the routine/functions you developed in the previous programming assignment.**

Example: If the user enters the string "01000010", your program must print out 66.

Programming Exercise 2 (30 points): (Call the Source code Exercise2.c)

This exercise is an extension of Programming Exercise 2 from the previous module's assignment.

Write a program that prompts the user to enter a **string *s*** representing a 4 digit hexadecimal number ***n***. Assuming that the number ***n*** is unsigned, you must write a **function** that takes ***s*** as input and returns the equivalent **integer *n***. Your main program must print out *n* in decimal. **It is expected that you will use the routine/functions you developed in the previous programming assignment.**

Example: If the user enters the string "02AE", your program must print out 686.

Programming Exercise 3 (60 points): (Call the Source code Exercise3.c)

*This exercise is the **same** as the Programming Exercise 3 from the previous module, except that you are not allowed anymore to use arithmetic operations such as `division` (/), `multiplication`, or `modulo` (%) to extract the bits. **In this exercise use only logic bit-wise operations.*** **Do not hesitate** to ask for hints or guidance if you do not see how to start or achieve this project without using arithmetic operations. Some hints are provided below.

Write a program that prompts the user to enter a positive **integer *n*** (0 up to 2^{32}-1). You must write a function that takes as input ***n*** and returns a **string *s*** representing the number ***n*** in binary. For this assignment,

you **CANNOT use the arithmetic division by 2 or the modulo operation** to convert the number to binary. Your main program must print out ***s***.
Example: If the user enters the number 66, your program must print out 1000010.

Hints for Programming Exercise 3:

Hint 1: Recall that the number ***n*** is **already** in binary inside the memory! All you need is to "extract" or "read" the bits individually. Read the next hints to know how.
Hint 2: Consider the number n=66. In the memory, 66 is 00000000010000**10**. I can isolate the least significant bit (**red** rightmost bit) by using the logic operation **AND** (**&**). Compute n & 1 = 66 & 1=**0**. Try the operation x & 1 with x taking different values to find out the effect: the operation "& 1" returns the value of the least significant bit!
Hint 3: Well, you read the rightmost bit with the operation " & 1". How to read the bit that is to the left of the least significant bit (i.e., the **blue** bit to the left of the red bit)?

The hint is to *push* all the bits to the right **after** I extracted the rightmost bit. To push to the right, you can **shift right** (**>>**) the number n to the right.
n >> 1 = 66 >> 1 = 00000000001000**01**: all bits are pushed to the right. Now, that bit became the rightmost bit…. And you know how to read the rightmost bit. Now, you just need to repeat the operation until no 1 is left into the number you are reading.

What you need to turn in:

- Turn in 1) Electronic copy of your report (standalone) and 2) the source code of your programs (**standalone**). **In addition,** all programming files (source code) must be put in a zipped folder named mX-name, where *name* is your lastname and **X** is the module number. Do not use any space to name the directory (there is no space between 'm' and 'X'). Recall that you must name the source programs Exercise1.c, Exercise2.c, and Exercise3.c respectively and include them in the folder. Zip the folder and post it on Canvas.

  In summary, your source code must be submitted twice: standalone AND in a zipped folder. Submit **separately** (not inside the zipped folder) the report as a Microsoft Word or PDF.
- Your report must:
  - **State** whether your code works (One sentence). If it does not work, state clearly what is wrong with it.
  - Clearly explain how to compile and execute your code.
- Good writing and presentation are expected.

How this assignment will be graded:

Your code will be compiled, run, and tested **EXCLUSIVELY** on Engineering Unix machines. **It is your responsibility to run your code and test it on those machines.**

1. The program compiles but does not produce any meaningful output (5%).
2. The program compiles and executes with major bugs (40% credit).
3. The program compiles and executes with minor bugs. The code is well-designed and commented (90% credit).
4. The program compiles and executes correctly without any apparent bugs. The code is well-designed and commented (100%).
5. If the instructor needs additional communications/actions to compile and execute your code, then a **30% penalty** will be applied.
6. **If the turn-in instructions are not correctly followed, 25% will be deducted.**