Assignment 4 Data Conversion & Bit Operation

Due on Sunday April 8, 11:59pm

To solidify your understanding of memory: arrays, pointers, typecasts, parameter-passing, compile-time language implementation: the layout and organization of memory, what kind of code is generated from a compiler, how the **runtime memory structures (stack and heap) are managed**, and so on.

It's the chance to do some in-depth experimentation with the compiler and the debugger in order to dissect your programs.

Problem 1: Binary numbers and Data conversion

A. Since computers work entirely in binary, learning how to manipulate numbers in the base two system can sometimes come in handy.
   1) Convert decimal numbers into binary form:
      58
      47
      100
      27
      20
      3

   2) Do binary arithmetic with binary form, and convert them back to decimal
      Note: list the calculation steps, e.g. 123-12

```
      1111011
    -    1100
   ----------
      1101111  ==> 111 (decimal)
```

      58+47
      100-27
      20*3

B. Write a program to examine the results of following operations; explain how the results are related to what the desired answers would have been. (Include code, execution results and explanation in your submission)
   1) Adding one to the maximum signed short (32767)
   2) Adding one to the maximum unsigned short (65535)
   3) Assigning the short 1025 to a char
   4) Assigning char 125 to short

Problem 2: Bit operations

Bit manipulations (such as bit shifting) are used in a variety of situations (graphics, robotics, cryptography, etc.) especially when you need to work with a form of packed data. In addition to the usual logical AND, OR, and NOT connectives, there are bitwise versions of these operations available operations in C.

- The bitwise **AND** operator (expressed with **single &**) compares its two operands bit-by-bit and reports which bits were 1 in both, 0 otherwise.

  For example:

  unsigned char a = 12, b = 5, c;
  c = a & b;
  Doing a bitwise AND on 00001100 (12 in binary) and 00000101 (5 in binary) gives the result 00000100, since the two patterns have only one bit in common.

- The bitwise **OR** operator (**single** |) works similarly to logical OR, but operates at the bit level.
- The bitwise exclusive **XOR** (^) reports which bits are on in one operand, but not both.
- The bitwise **NOT** (~) is a unary operator that just inverts all the bits in its operand.
- The right-shift operator (>>) moves the bits of shift_expression to the right.
- The left-shift operator (<<) moves the bits of shift_expression to the left.

A. Write a program using bit operations to fulfill the following requirements (Include code, execution results and explanation in your submission):

1) Shifting the signed short (32767) 3 times to left and right respectively, and explain the results

2) Shifting the unsigned short (65535) 3 times to left and right respectively, and explain the results

3) Use a bit approach (return *Yes* or *No* for any given integer) to determine if an integer would lose data when assigned to a short and a char (hint: check if upper bytes of integers are zeros or not).

4) Use bit operations to convert a number from negative to positive or vice versa (Refer back to the "two's complement" representation for negative numbers we showed in class and try to work through what the patterns are in terms of bits.) (hint: Inverting all the bits can be done with a bitwise XOR with that number that has all bits on.)

B. One neat feature of the XOR operation is that it is completely invertible. If XOR *a* with *b* and then XOR the result with *b* again, you get back *a* (trace this out for yourself). This makes this operation useful for encryption and decryption. Write a simple program to encrypt and decrypt an input file:

1) Encrypt a file using XOR and a specified key

2) Then decrypt the file with the same key

3) Program execution format:
encryptool -e filename key // encrypt the file with the key value
encryptool -d filename key // decrypt the file with the key value
//here, *encrpyool*: your exe file; *-e*: parameter indicating encrypt action; *-d*: parameter indication decrypt action; *filename*: the input file; *key*: the value used in XOR