CIS 2107 Lab 5

This lab has 2 parts: Converting data types and a safe calculator (150 points)

Part 1: Converting data types (50 points)

In this project, you will write code to convert strings containing hexadecimal. Decimal, octal and binary strings to an unsigned integer. Some of the code is completed for you. Open the Convert source code file and look at the completed code and comments. The description and comments in each function should help you complete this project.

To convert from hex string to unsigned int, accumulate the position-wise values of each character in the string. Remember that you can perform math operations on char data because they are numbers and have specific positions in the ASCII table. This method is like the one shown in lecture. Convert 0xFFFF to a unsigned char, considering that F = 15 and each hex char a multiple of 16:

15 * 1 + 15 * 16 + 15 * 256 + 15 * 4096 = 65,535

This process is similar with decimal, octal and binary. To from an unsigned int 16,535 convert to hex:

1. 65,535 / 16 = 4095 and 65,535 % 16 = 15
2. 4,095 / 16 = 255 and 4095 % 16 = 15
3. 255 / 16 = 15 and 255 % 16 = 15
4. 15 / 16 = 0 and 15 % 16 = 15

Collect remainders for the hex values in reverse of these steps.

15 15 15 15 -> 0xFFFF

This is a similar process for base-10, base-8 and base-2 conversions.

Part 2: A safe calculator (100 points)

In this project, you will write a safe signed integer calculator. As with Part 1, some of the code is completed for you and there are descriptions and comments to help. You should not use any arithmetic operators, except simple assignment, or increment and decrement operators in your code. The _add(), an add function that only uses bitwise operators, and add(), a safe add function that detects and reports overflow and underflow errors, are crucial to the other functions. The seven remaining functions either directly or indirectly depend on _add() and add(). Code these functions in order as they appear below to maintain functional dependencies. Use the main function to test each function before continuing to the next function.

1. Negation is defined using a bitwise operator and the add() function
    a. Remember two's compliment
2. Subtraction is defined using Negation and add() functions
    a. For a – b, add negation of b to a.

3. Multiplication is defined with successive add() function and Subtraction function calls
    a. Should perform with positive integers and keep track of which sign the product has
4. Division is defined with successive Subtraction and add() function calls
    a. Should perform with positive integers and keep track of which sign the result has
5. Modulus is defined as successive calls to Subtraction to return a remainder
    a. Should perform with positive integers
6. Power is defined as successive calls to Multiply and Subtraction
    a. Only use positive exponents
7. Convert converts an input string to an integer value
    a. Pay attention to comments – This is the longest function
    b. You may be able to modify your dec_to_uint() from Part 1