Program Set #1
Total Points: 30

Three problems must be implemented for full credit. The required (starred) problem marked in the set must be implemented by everyone. See the 2425 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points each)

Note: If coding in C++, use the STL string class for problems involving strings. Do not use C style strings.

Section One- Choose two problems.

1. Something simpler than binary numbers are unary numbers. A unary number, `n > 0`, is coded as `n - 1` one bits followed by a zero bit. Thus, the code for the number 5 is `11110`. Here are some examples of unary numbers.

decimal	unary
1	0
2	10
3	110
4	1110
5	11110
6	111110
7	1111110

Input from the keyboard a positive integer in the range [1,80]. Error check input. For each number, output to the screen the number followed by its unary equivalent with no leading or trailing spaces. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter a positive number: 37

37 in unary is: 11111111111111111111111111111111111110

Run again (Y/N): Y

Enter a positive number: 1

1 in unary is: 0

Run again (Y/N): n
```

Name the program: `UnaryNosXX.cpp` or `UnaryNosXX.java`, where `XX` are your initials.

2. Write a program to convert a decimal value to an eight-bit two's complement binary number. Let the user enter a decimal whole number `X`, either positive or negative, and output the equivalent two's

complement binary number. The value of X will range from [-128,127]. Error check input size. Output to the screen, each X and its eight-bit two's complement equivalent separated by a space, =, and then another space. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Runs:

```
Enter a value: 15

15 = 00001111

Run again (Y/N): Y

Enter a value: -49

-49 = 11001111

Run again (Y/N): n
```

Name the program: TwosComplementXX.cpp or TwosComplementXX.java, where XX are your initials.

3. Write a program, given a positive number n, rotates the base-10 digits m positions rightward. That is, output the result of m-steps of moving the last digit to the start. The rotation count m will be a non-negative integer. For example, n = 100, m = 2 yields 1, first rotate the value to 010, then to 001, then finally drop the leading zeroes to get 1. Input from the keyboard a positive integer n and m the number of positions rightward. Error check. Output to the screen the final number rotated. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter a positive number: 123
Enter the number of positions to rotate: 2

The rotated value is: 231

Run again (Y/N): Y

Enter a positive number: 9998
Enter the number of positions to rotate: 2

The rotated value is: 9899

Run again (Y/N): y

Enter a positive number: 123
Enter the number of positions to rotate: 0
```

```
The rotated value is: 123

Run again (Y/N): n
```

Name the program: `RotateNosXX.cpp` or `RotateNosXX.java`, where XX are your initials.

Required Problem- Comprehensive.

4 (**). Write a program given a number in any base in the range [2,16] and rounds that number to a specified place. Rounding up occurs when the digit to the right of the rounding place divided by the number's base is 0.5 or more. For example, 0.12468 rounded to 3 places is 0.125 because 6 divided by 8 is greater than 0.5; when rounded to 2 places, it's 0.13 since 4 divided by 8 is 0.5; and it's 0.1 when rounded to 1 place because 2 divided by 8 is less than 0.5. Input from the keyboard three (3) values: the number, the base of that number, and the number of places to be rounded. The numbers entered must have one digit to the right of the place to be rounded to and at least one digit to the left of the decimal point. Also, assume the rounding place will never be 0 (round a whole number). For each input, output to the screen the value rounded to the given number of places in the format shown below. Do not print any digits beyond the rounded place. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter a value: 0.11101
Enter the base of the value: 2
Number of decimal places to round: 4

The result in base 2: 0.1111

Run again (Y/N): y

Enter a value: 17.9AAA
Enter the base of the value: 11
Number of decimal places to round: 2

The result in base 11: 17.A0

Run again (Y/N): N
```

Name the program: `BaseRounderXX.cpp` or `BaseRounderXX.java`, where XX are your initials.

Extra Credit: Implement the following problem. See the 2425 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points)

Your professor wants to send a message to his friend but is afraid of the message being read by others. He has a simple encryption scheme to apply to his message. Instead of sending characters, your professor will send numbers. He wants to use the ASCII values of characters but that is a little too obvious. So, he will represent the numbers using bases other than base ten. To make the code harder to break he is going to change the base for each line. The first line of the message will use base two, the second line will use base three, the third line will use base five, the fourth line will use base seven. If there are more than four lines in the message the fifth line will cycle back to base two, the sixth line will be base three and so forth. Here is an example of which base to use on a given line:

Line 1 -- base two
Line 2 -- base three
Line 3 -- base five
Line 4 -- base seven
Line 5 -- base two
Line 6 -- base three
Line 7 -- base five
Line 8 -- base seven
Line 9 -- base two
...

Each character in the message will be replaced by a number, including spaces. The number will be the ASCII value of the character in the appropriate base. The newline characters at the end of the line are not encoded. The number for each character will be followed by a single underscore character, `'_'`, except for the last character on a given line. Write a program to perform this encryption scheme as stated above. The input will come from a data file. The first line of input will contain a single integer `n` indicating the number lines in the message. The following `n` lines will be the plain text version of the message. Output the screen the encoded message using your professor's encryption algorithm, properly labeled. Let the user enter the file name from the keyboard. Refer to the sample output below.

Sample File

```
5
Mik
Hi!
How are u?
"car" is yellow
Bye :)
```

Sample Run:

```
Enter the file name: code.txt

Message:

1001101_1101001_1101011
2200_10220_1020_1012
242_421_434_112_342_424_401_112_432_223
46_201_166_222_46_44_210_223_44_232_203_213_213_216_230
1000010_1111001_1100101_100000_111010_101001
```

Name the program: `LineEncodingXX.cpp` or `LineEncodingXX.java`, where `XX` are your initials.