
Instructions: Follow instructions *carefully*, failure to do so may result in points being deducted. You may discuss problems with your classmates, but all work must be your own. The CSE academic integrity policy is in effect (see http://cse.unl.edu/ugrad/resources/academic_integrity.php).

Test Case: For each problem, you may be required to submit two non-trivial *test cases* which include a valid plain text input file and a plain text output file for each test case. The contents of these test cases are simply the input (command line arguments) and expected output. A test case is an input-output pair that is known to be correct – the solution should be worked out by hand or by use of a "known" correct mechanism. Simply using the output of your program as the valid output is *not* a test case. Points will be awarded based on the items outlined in the rubric.

Hand in your source files via the webhandin and be sure to verify that your programs work with the webgrader. Submit the rubric via handin as well with your name and login on it.

1. **Program 1 – Affine Cipher**

   A cipher is an algorithm that will encrypt (and decrypt) data, most commonly text. Several types of cryptographic ciphers exist, but one of the simplest (albeit least secure) is the *affine cipher*. In an affine cipher, each letter in the unencrypted message is given a numeric representation (such as its ASCII value) and modified using a linear function of the form $(ax + b) \bmod m$, where

   - x is a numeric form of the letter to encrypt
   - a and b are the cryptographic keys
   - m is the size of the alphabet

   The values for a and b are called the *keys* of the cipher. The only constraint on these values is that a and m must be co-prime; that is, the greatest common divisor of a and m must be 1.

   Your task will be to write a program that will both encrypt any text file containing spaces, uppercase letters, and lowercase letters (you should leave all punctuation other than spaces unencrypted). You should map the input letters as follows (lower-case letters from 0 to 25; upper-case letters from 26 to 51; and whitespace to 52):

```
a - 0
b - 1
c - 2
...
z - 25
A - 26
B - 27
...
Z - 51
[space] - 52
```

The input will be a text document containing both upper-case and lower-case letters, and the values for a and b. The value of m should be 53. The output of your program should be the encrypted contents of the input file displayed on **standard output**.

For example, given $a = 3$ and $b = 4$ and the text "Hello" (note: $m = 53$):

1. Translate the input string to its numeric equivalent: 33 4 11 11 14
2. Perform the encryption on each of the numbers above, using the given encryption keys: 50 16 37 37 46
3. Translate the transformed numbers back to their alpha-numeric equivalent: "YqLLU".

Java: Place your code in a Java class called `AffineCipher` (the source file will be `AffineCipher.java`) and have it accept the value for a, the value for b, and the input file name from the command line. That is, your program should be executable from the command line by calling:
`~>java AffineCipher 3 4 inputFile.txt`
which will output to the standard output.

Test Cases Design four test cases and hand them in. Make sure your test case files have the following names: `cipher_input_001.txt`, `cipher_output_001.txt`, and `cipher_input_002.txt`, `cipher_output_002.txt`.

2. **Program 2 – Anagrams**

An anagram is a word formed by rearranging the letters of another word. For this program, we will extend that definition to include any words formed by rearranging any *subsets* of letters of another word.

Write a program that takes two arguments: a length n and a string, and outputs all english words of length n that appear in the given string. For example, if the $n = 4$ and the string is "caret", the output would be similar to the following:

```
n = 4, word=Caret
acer
```

```
3   acre
4   aret
5   care
6   cart
7   cate
8   cert
9   race
10  rate
11  rect
12  tace
13  tare
14  tear
```

The input should accept both upper-case and lower-case letters in the string, and should ignore the case when finding the anagrams. An American-english dictionary can be found at `/usr/share/dict/american` on CSE.

Java: if using Java, name your class `Anagrams`. You must use command line arguments to read the input; so the command line usage for the example above would be:

```
~>java Anagrams 4 caret
```

Test Cases Design four test cases and hand them in. Make sure your test case files have the following names: `anagrams_input_001.txt`, `anagrams_output_001.txt`, and `anagrams_input_002.txt, anagrams_output_002.txt`.

3. **Program 3 – Population Growth**:

   Many types of populations undergo exponential growth, including animals, humans, and bacteria. The size of a given population after a certain amount of time can be estimated by using the formula $P = P_0 e^{rt}$, where

   - P_0 is the initial size of the population (at time $t = 0$)
   - e is the Napier's constant, with an approximate value of 2.71828
   - r is the rate of growth for the population
   - t is some amount of time (typically given in hours or years) and
   - P is the estimate for the final population

   You will write a program that takes the following input via command line arguments:

   - P_0, the initial population size
   - r, the rate of growth for the population
   - t, an integer denoting the amount of time elapsed

Your program will then produce a table showing the estimated size of the population at each integer time step between 0 and t.

For example, using your program on a population with an initial size of 10, a rate of growth of 10.5%, and a time of 30, would produce a table something like:

```
1 Elapsed Years     Amount
2 -------------------------
3 -                  10
4 1                  11
5 2                  12
6 3                  13
7 ...
8 30                 233
```

Java: if using Java, name your class `PopulationGrowth`. You must use command line arguments to read the input; so the command line usage for the example above would be:

```
~>java PopulationGrowth 10 10.5 30
```

Test Cases Design four test cases and hand them in. Make sure your test case files have the following names: `population_input_001.txt`, `population_output_001.txt`, and `population_input_002.txt, population_output_002.txt`.