

General information for all the projects:

- .1) **You pulled a projects. You can switch it with somebody else, if you don't like it, or propose something else to do to me and we will see if it is appropriate for our class.**
- .2) **You are to present working program on May 18th** . If you don't present, 10 points will be taken from a 100 points grade. Therefore, make sure that you have a working program by then. Start working on the project as soon as possible!
- .3) **The due date to submit your project is the Final Exam date** (TBA). Don't forget that the program should be well commented, with specifications and internal comments. Give precise instructions on how to run your program.

CSI33: Final Project Submission Guidelines

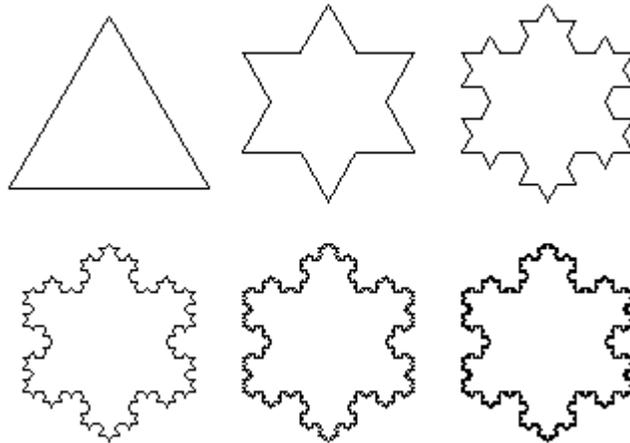
1. Program should be well commented :
 1. plenty of internal comments,
 2. function specification should be given for every method/function you defined
2. Project should contain class definitions in separate files, standalone methods (in case of sorting algorithms) should be stored in separate files as well.
3. The program demonstrating the use of classes and methods you defined should be stored in a separate file as well.
4. If you have any restrictions on input: state them in the comments at the top of the file.
For example, one of the projects was work with polynomials. There are many ways to enter a polynomial, so person who will be using it later – should know how to use it.
5. In general, describe someplace how to use your project. It can be done in a separate **readme** file.

Please, be aware that I won't be able to suggest any corrections to be done to the project (as we are limited in time for the Final Grades submission). Therefore make sure that you did your best, and then send me your work.

Project 1 (Recursion):

Kosh Snowflake

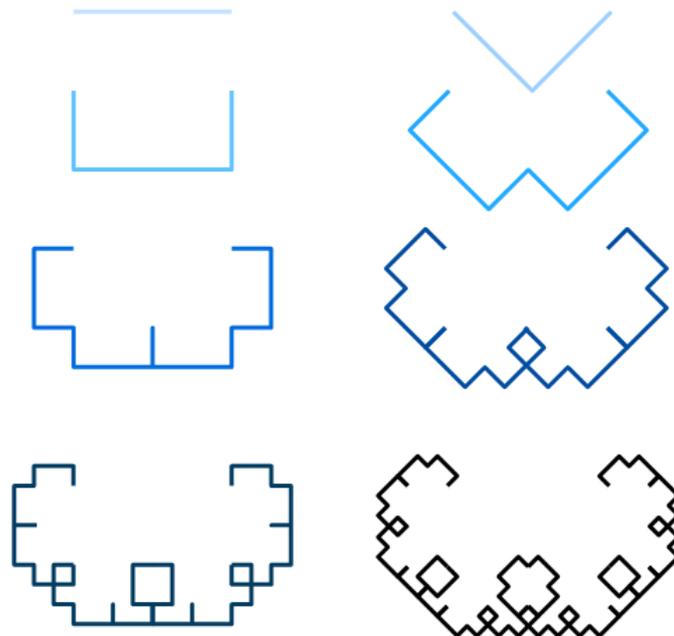
(see our book, page 219 exercise 10)



Project 2 (Recursion):

C-curve

(see our book, page 220 exercise 11)



Project 3 (Cryptography):

Encoding and decoding messages using the Vignere Cipher.

Here is the information about it (in the beginning the general introduction along with description of two easier algorithms is given, and the Vignere Cipher's description is given at the end):

www.natna.info/CSI33/cryptography.pdf

Project 4 (Polynomials, ADTs):

Write a class in Python to represent a polynomial. The class should store a list of the coefficients and the degree of the polynomial. You may assume a maximum degree of 100 for the polynomial. Write the methods for the addition, subtraction, and multiplication operators, input and output operators for the class. Also provide a method for evaluating the polynomial at a specific value. Write unit tests for your **Polynomial class**, as well as a program that plays with the **Polynomial class**.

(see page 352, exercise 6 + page 400, exercise 3)

Project 5 (Polynomials, ADTs):

Write a class in C++ to represent a polynomial in one variable, using dynamic arrays. The class should store an array of the coefficients and the degree of the polynomial. You may assume a maximum degree of 100 for the polynomial. Write the methods for the addition, subtraction, and multiplication operators, input and output operators for the class. Also provide a method for evaluating the polynomial at a specific value. Write program that tests your **Polynomial class**.

(see page 352, exercise 6 + page 400, exercise 3)

Project 6 (Algorithms):

Read about and implement *Randomized Quicksort* using C++ (using the provided algorithm)

The worst-case running time for quicksort is still $\Theta(n^2)$, but on average it runs in $\Theta(n \log n)$ time. It's randomized version works even better. You are asked to implement the randomized version.

Here is the original QuickSort Algorithm:

www.natna.info/CSI33/QuickSort-page1.jpg,
www.natna.info/CSI33/QuickSort-page2.jpg,
www.natna.info/CSI33/QuickSort-page3.jpg, and

Here is the Randomized QuickSort:

www.natna.info/CSI33/RandomizedQuickSort-page1.jpg,
www.natna.info/CSI33/RandomizedQuickSort-page2.jpg

After you programmed the Randomized QuickSort and tested it, run the algorithm by hand on the following list: [1,6,3,4,8,2,5,7]. Show all the steps. Whenever a random choice is needed, write up the indices on pieces of paper. Put them all into a hat/container, mix them thoroughly and pull out a number – here is a random number. Submit it with your project submission.

Project 7 (Algorithms):

Read about and implement Radix sort using C++ (using the given algorithm)

Radix sort is called a linear sort (you can read about it in the information provided in the pictures).

Here is the information and algorithm for the Radix Sort:

www.natna.info/CSI33/RadixSort-page1.jpg,
www.natna.info/CSI33/RadixSort-page2.jpg, and

Here is the algorithm for one of the stable sorts:

www.natna.info/CSI33/CountingSort-page1.jpg,
www.natna.info/CSI33/CountingSort-page2.jpg,
www.natna.info/CSI33/CountingSort-page3.jpg

After you programmed the Radix Sort and tested it, run the algorithm by hand on the following list: [1,6,3,4,8,2,5,7]. Show all the steps. Submit it with your project submission.

Project 8 (Data Structures):

Write an AVL class using C++, add ability to display the tree.

Project 9 (Data Structures):

Complete the AVL tree class (the one that we wrote in Python) with the rest of operations our BST class has, and add the ability to display the tree either using graphics or using shell nicely.

Project 10 (Statistical Analysis):

Read about two-variable data, linear regression line and correlation coefficient.

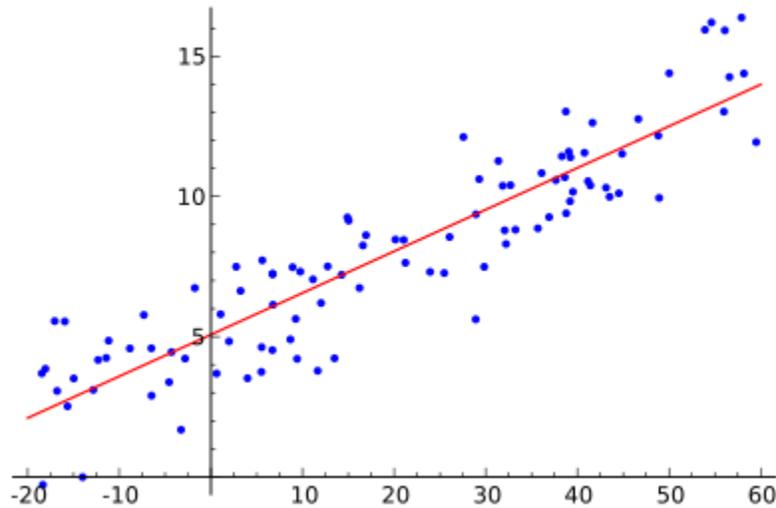
Here are few of the useful links:

<https://www.youtube.com/watch?v=T7LauV6AnRg>

<http://www.dummies.com/how-to/content/how-to-calculate-a-regression-line.html>

<http://onlinestatbook.com/2/regression/intro.html>

Write a program that accepts two variable data from a data file, which is formatted in such a way that each line has an even number of values separated by white space, and value of first variable, say x , is followed by the value of the second variable, say y . The program should process the data, plot points and draw regression line, if possible, in a graphical window, displaying the axis and the scale.



Project 11 (Propositional Logic, Truth Tables):

Write a program that given a compound proposition builds the truth table for it.

For example,

p	q	r	$p \vee q$	$r \rightarrow (p \vee q)$	$q \wedge (r \rightarrow (p \vee q))$	$(q \wedge (r \rightarrow (p \vee q))) \rightarrow p$
0	0	0	0	1	0	1
0	0	1	0	0	0	1
0	1	0	1	1	1	0
0	1	1	1	1	1	0
1	0	0	1	1	0	1
1	0	1	1	1	0	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

You can read about Propositional Logic in your Discrete Math I (CSI30) book. If you don't have it then let me know.

Project 12 (Calculus I) Newton's method

Write a program that will be using Newton's method to approximate the indicated root of an equation.

Input: equation in one variable, in the form $polynomial(x) = 0$, where for x^6 will be used for x^6 ;

initial approximation x_1 ; and

the n , which is the n^{th} approximation that needs to be found

Output: the n^{th} approximation

See the pdf file with method's description (note that the first page is flipped, sorry):

www.natna.info/CSI33/CSI33NewtonsMethod.pdf

Project 13 (Simulation - stacks, queues):

Consider a real life situation: *airplanes taking off and landing on a runway*. Research about it on the internet, then formulate a question (for example it may be: "Do we need one more runway?" or "Do we need a larger crew in control tower?"), then design a simulation that can help you to answer this question.