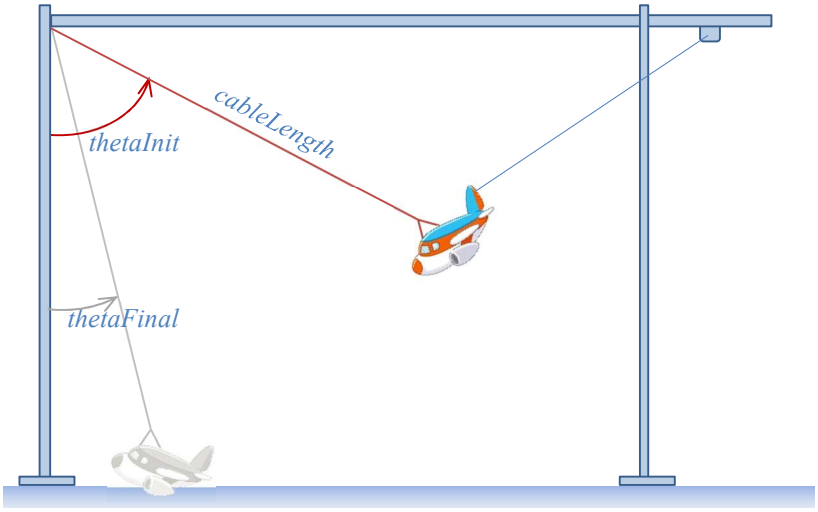
24-780 B—Engineering Computation

Assigned: Wed. Sept. 6, 2020
Due: Tues. Sept. 12, 2023, 11:59pm

Problem Set 2: Crashes & Encryption

PS2-1 Airplane Crashtest Quiz (40 pts)

Write a program that randomly creates a *unique* airplane swing crash simulator problem of the type shown in the attached example. Allow your program to generate reasonable random values for the problem parameters (masses, velocities, direction angles, coefficient of restitution, etc.). Then, present your problem to the user and ask for the resulting final velocities. Assume that the user has access to the following diagram showing all parameter labels:



Where:

- The plane's mass is *massPlane*, it is released at angle *thetaInit*, swinging from a cable with constant length *cableLength*.
- At angle *thetaFinal*, the plane crashes into the ground.
- The plane is not moving when released.

Let the value of the mass of the plane be a uniformly distributed random number varying from 6 to 10 Mg, rounded to one decimal place. Similarly, let *thetaInit* vary from 50 to 80 degrees, *thetaFinal* vary from 5 to 20 degrees, and the cable length vary from 15 to 25 meters, all rounded to zero decimal places. Ask the user to provide crash velocity in m/s and the maximum tension in the cable in kN.

Your program should ask the user to enter each of the 2 answers (i.e., *velFinal*, *maxTension*) and determine if the answers are correct (perhaps also providing a measure of error).

Note that there is an attachment at the end of this document with a solved problem. Feel free to use this example to test the solution part of your program using "hard-wired" parameter values before you start playing around with the random-generated values.

PS2-2 Message Encryption (60 pts)

Recall the island-boat-island message sending thought exercise from Lecture 2 (Aug 30, 2023). We discussed how the secure communication method has similarities to internet communication. In this assignment, you are asked to create a simple encryption/decryption mechanism for text files. We will discuss this task during lecture on Wed., Sept. 6.

The program should do the following:

- Ask the user if they want to encrypt (E), decrypt (D), or exit (X). Expect that they'll use a single letter for their choice.
- For encryption:
    - Ask user for input file with original message. Note that the input may be a file that is already encrypted to simulate the back and forth of our island-boat-island communication.
    - Ask user for name to use for encrypted output file.
    - Ask user for name to use for storing the code file
    - The encrypted version of the input file is created by adding a random number (from 0-255) to each character in the input file. Note that you should use "wrap around" addition (e.g., 187 + 95 = 26, not 282). The series of random numbers (i.e., characters) will be stored in the code file
- For decryption
    - Ask user for input file with encrypted message.
    - Ask user for name to use for decrypted output file.
    - Ask user for code file to use for decryption
    - The decrypted version of the input file is generated by subtracting (wrap-around again) each character in the code file from the corresponding character in the input file.
- For exit
    - Just exit, maybe with a goodbye message.

I have attached a couple of *well-behaved*, not-too-long text files to use for testing, but you may use anything you like. At the start, try encrypting and decrypting a single file. As you get it to work, you may want to do a full island-boat-island test by doing the following:

1. Encrypt the message text file, calling the outputs 1.encr and 1.code (This is the first message sent)
2. Encrypt the 1.encr file again, calling the outputs 2.encr and 2.code (This is the return message, second transmission)
3. Decrypt the 2.encr file, calling the output 3.encr and using 1.code (This is the third transmission)
4. Decrypt the 3.encr file, calling the output Final.txt and using 2.code (This should be the same as the original message, noting that no transmission was unencrypted).

Note: The following advice will save you about 10 hours of effort:

- In C/C++, the most efficient way to store an integer from 0 to 255 is to store it as type *unsigned char*. This has the added benefit that you can easily read/write to permanent storage without worrying about delimiters and conversions.
- Be careful with the wrap-around addition and subtraction. Be 100% certain that the integer arithmetic is correct before you start testing for other things.
- When you create the file streams for input/output, be sure to do so in "binary mode" (`ios::binary`). This will ensure that special characters (like carriage return and end-of-file) are NOT processed, but are instead just taken at "face value"
- You want to include any whitespace (blanks, newlines, tabs, etc.) in the file you read, so be sure to include `inputFileStreamName >> noskipws;` before you use the stream.
- (Less important) Use a naming scheme to keep track of your files. I use *.txt* for text files, *.encr* for encrypted files, and *.code* for code files

Deliverables

2 files, very appropriately named:

ps02crash_yourAndrewID.cpp

ps02encryption_yourAndrewID.cpp

Upload the files to the class Canvas page before the deadline (Tuesday, Sept.12, 11:59pm).

Hint: Even if you name your file appropriately, be sure to include your full name within the code itself (perhaps as a comment block at the top of the file). It is also appropriate to include date and course info, plus a short description of what the program does. (Think about what part of the text in the assignment write-up can be copy/pasted or adapted for this purpose.)

Learning Objectives

Making use of console input/output

Math on floating points and integers

Developing simple algorithms

File input and output

Using functions effectively

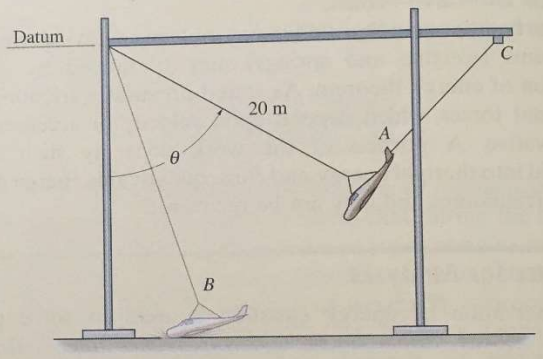
Searching references (online and/or textbook) for C++ library functions.

This is the text in the image (including variable names), in case you want to use it as part of problem prompt (copy/paste/edit):

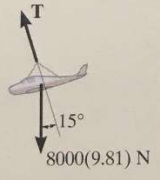
The gantry structure in the photo is used to test the response of an airplane during a crash. As shown, the plane, having a *massPlane* of ???, is hoisted back until *thetaInit* is ???, and then the pull-back cable is released when the plane is at rest. Determine the speed *velFinal* of the plane just before crashing into the ground when *thetaFinal* = ???. Also, what is the maximum tension *maxTension* developed in the supporting cable (which has length of *cableLength*) during the motion? Neglect the effect of lift caused by the wings during the motion and the size of the airplane.

EXAMPLE 14–9

The gantry structure in the photo is used to test the response of an airplane during a crash. As shown in Fig. 14–21*a*, the plane, having a mass of 8 Mg, is hoisted back until $\theta = 60°$, and then the pull-back cable AC is released when the plane is at rest. Determine the speed of the plane just before crashing into the ground, $\theta = 15°$. Also, what is the maximum tension developed in the supporting cable during the motion? Neglect the effect of lift caused by the wings during the motion and the size of the airplane.



(a)



(b)

Fig. 14–21

Solution

Since the force of the cable does *no work* on the plane, it must be obtained using the equation of motion. First, however, we must determine the plane's speed at B.

Potential Energy. For convenience, the datum has been established at the top of the gantry.

Conservation of Energy

$$T_A + V_A = T_B + V_B$$

$$0 - 8000\ \text{kg}\,(9.81\ \text{m/s}^2)(20\cos 60°\ \text{m}) = \tfrac{1}{2}(8000\ \text{kg})v_B^2 - 8000\ \text{kg}\,(9.81\ \text{m/s}^2)(20\cos 15°\ \text{m})$$

$$v_B = 13.5\ \text{m/s} \qquad \textit{Ans.}$$

Equation of Motion. Using the data tabulated on the free-body diagram when the plane is at B, Fig. 14–21*b*, we have

$$+\nwarrow \Sigma F_n = ma_n;$$

$$T - 8000\,(9.81)\text{N}\cos 15° = (8000\ \text{kg})\frac{(13.5\ \text{m/s})^2}{20\ \text{m}}$$

$$T = 149\ \text{kN} \qquad \textit{Ans.}$$