SYDE 121
Lab Number 8

Exercise 1: Summing two numbers stored in arrays

Learning Objectives: Practice working with 1-dimensional (1D) arrays and single character entry of data. Further practice using modular programming.

Read this first

This problem is taken from your Savitch textbook. An array can be used to store large integers one digit at a time. For example, the integer 1234 could be stored in the array 'a' by setting a[0] to 1, a[1] to 2, a[2] to 3, and a[3] to 4. However, here you might find it more useful to store the digits backward, that is, place 4 in a[0], 3 in a[1], 2 in a [2], and 1 in a[3].

In this exercise, you will write a program that reads in two positive integers that are 20 or fewer digits in length and then outputs the sum of the two numbers. Your program will read the digits as values of data type ***char*** so that the number 1234 is read as the four characters '1', '2', '3', and '4'. After they are read into the program, the characters are changed to values of type *int*. The digits will be read into a partially filled array, and you might find it useful to reverse the order of the elements in the array after the array is filled with data from the keyboard. (Whether or not you reverse the order of the elements in the array is up to you. It can be done either way, and each way has its advantages and disadvantages.)

Your program will perform addition by implementing the usual paper-and-pencil addition algorithm. The result of the addition is stored in an array of size 20, and the result is then written to the screen. If the result of the addition is an integer with more than the maximum number of digits (that is, more than 20 digits), then your program should issue a message saying that it has encountered "integer overflow". You should be able to change the maximum length of the integers by changing only one **globally defined constant**. Include a loop that allows the user to continue to do more additions until the user says the program should end.

What to do

As in Lab 7, Exercise 1, create three different source files for your program named *sum.cpp*, *sumtwonums.cpp*, and *sumtwonums.h*.

In this case, since you are entering single chars. Use the `get` cin member function to do this i.e., `cin.get( )` (see Section 6.3 in Savitch text for details of this function). To represent a longer integer number, it is not necessary to output the single char each time one is entered. Once the full number is entered and stored in a numerical array, then the entire number should be echoed to the user.

You are expected to complete **all** of the directions provided to you in the problem, and a few more listed below.

Basic Program Requirements

- create 3 separate source files (as in Lab 7, Exercise #1)
- read in two numbers one character at a time into numerical arrays
- echo both numbers to the user
- determine and display proper sum of the two numbers
- act appropriately if user enters too many digits for a single number (exit is ok)
- only numerical chars are entered into the arrays i.e., for all data entered, only allow 0, 1, 2, … 9 to be entered into the arrays
- disregard leading zeros e.g., 00123 is displayed as 123
- overflow check following the summation (do not exit program if overflow encountered)
- allow user to loop through process as many times as desired
- As always, use of proper commenting, code is neat, concise, and readable

Bonus: Additional Features

- display numbers using commas e.g., 12,345,567
- allow user to quit at any time (even midway through data entry)
- handle addition of two numbers of type double (constrain the user to entering numbers with a small number of decimal places, e.g. 1 or 2)

Create a "*README_sum.txt"* file, and create detailed instructions on how to run your *sum.cpp* program, including what input should be entered. Make sure to indicate in your readme file which requirements/features work for your program and which features do not. Include the above list of requirements/features in the readme file and indicate which ones you were able to complete. This information will assist the TA when grading your submission.

Note that there are different ways to implement the given problem. Ensure that your code is properly documented so that the TA is aware of how you solved the problem.

Hint: Design your code **prior** to going to the computer. Then, once you make a certain amount of progress on the computer, you will probably find that your original design is not working as planned. It is advisable to leave the computer and redesign your plan. **This is not easy to do** since the tendency is to stay at the computer. In terms of minimizing time spent solving the problem, knowing when to redesign the algorithm away from the computer is an asset.

What to submit

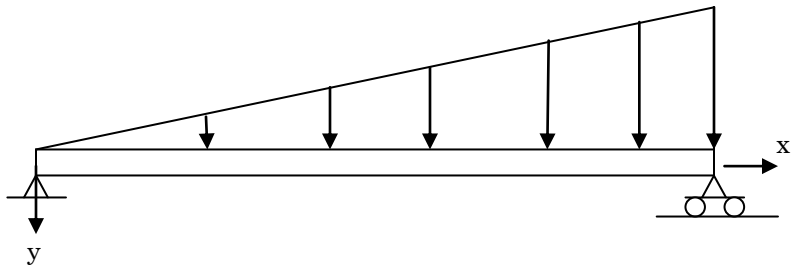
Submit your three source files *sumtwonums.h*, *sumtwonums.cpp*, and *sum.cpp* and *README_sum.txt* files to the Lab Assignment 8 LEARN dropbox. Clearly indicate in your README file, header documentation, and in your code whether you have implemented the bonus features.

Exercise 2: Determining deflection and moment of a beam under loading

Learning Objectives: Practice working with 1-d arrays. Further practice using modular programming.

Read this first

Later in your SYDE curriculum, you will take a course in deformable solids. You will be asked to determine equations that describe the behaviour of various bodies under load. Unlike your first year statics course where rigid bodies are assumed, deformable solids better approximate the real world by recognizing that bodies under load will deflect. Beams are an important part of the engineering world and determining their deflection under certain loading conditions is a necessity. Here's an example:



For this beam, both ends are allowed to rotate (indicated by the triangles), however, the left end is fixed whereas the right end can move freely in the horizontal direction. The arrows pointing downwards describe the direction of load on the beam and the length of the arrow described the magnitude of that load at that point. In this case, the load continuously and linearly increases from left to right along the beam. This is helpful to understand, but what you really need to implement follows next.

The equation for the deflection (y) along the length of the beam (x) is (note that positive y deflection is defined as downwards):

$$y(x) = \left[\frac{2QL^4}{\pi^5 EI}\right]\sum_{n=1}^{\infty}\left[\frac{(-1)^{n+1}}{n^5}\right]\sin\left(\frac{n\pi x}{L}\right)$$

Although this seems like a messy equation, most of the terms are constants, where

- E is the "modulus of elasticity" (i.e. the material stiffness),
- I is the moment of inertia of the cross-section (i.e. the resistance to bending due to the cross-sectional shape),
- Q is the loading (as a function of length; in this case, the loading is triangular), and
- L is the length of the beam.
- y represents the deflection as a function of x (position along the length of the beam). You will require an array to store this information.

You can use the following constants in your header file:

```
const double ELASTICITY = 200e6; // kiloNewtons / m^2
const double INERTIA = 8.3e-6;   // m^4
const double LOAD = 100.0;       // kN / m
const double LENGTH = 1.0;       // m
const double PI = 4.0*atan(1.0);
const double EPSILON = 1e-30;    // tolerance
const int STEPS = 10;
```

where `STEPS` is the number of discrete steps along the length of the beam where the deflection (y) and bending moment (M) are to be calculated. We calculate the displacement at discrete locations because we are not calculating a formula to calculate the deflection at any real number along the axis. If we select enough discrete points, then we have enough information to interpolate between these points.

You will have to use a tolerance (const EPSILON) to terminate the series summation.

The bending moment is given by:

$$M(x) = EI\frac{d^2y(x)}{dx^2}$$

What to do

As in Exercise 1, create three different source files for your program: *beambend.cpp*, *beam.cpp*, and *beam.h*. The main program (contained in *beambend.cpp*) will create the necessary arrays (initialize them) and pass them down into separate functions to determine the deflection (y) and the bending moment (M) along the beam at intervals of L/10.

You will need to calculate the second derivative. This is not difficult. In calculus, you learn about the continuous definition for a derivative. Here, since the computer works in a discrete environment, you need a discrete approximation for a derivative. Here, we will use the forward discrete approximation:

```
y'[i] = ( y[i+1] - y[i] ) / d
```

where `d` is the spatial displacement (namely the spacing along the length of the beam). Note that this just measures the slope of the deflection along the beam. The second derivative is determined by taking the derivative of the first derivative – you just have to create one function that determines the derivative and then apply it twice!

Don't worry about the physics and mathematics – these are concepts that are eventually covered in the SYDE curriculum.

You will probably want to create a generic function to display an array to the screen. This function would accept the size of the array as well as the array itself.

You should allow the size of the arrays to be a global constant; however, you should pass the size of the arrays into the functions as arguments along with the array itself.

Note that you will see some odd behaviour at the end of the beam due to the calculation of the derivatives at the end of the beam. Don't worry about this.

Optional

If you want to view the deflections (this is not part of the lab submission requirements), you can download the "output_to_file.cpp" program on the course website and integrate it into your program. This function accepts three parameters, the name of the array (just a string), the array itself, and the size of the array. The array will be sent to a file "out.txt" in the directory where the beam program is stored. You can take this output, cut and paste it to an Excel spreadsheet, and use Excel's plotting functions to plot the result. You can plot the deflection, its first and second derivative, as well as the bending moment. Note that there will be a numerical error near the end of the beam, but do not be concerned about this. Enjoy!

Note: you will soon learn how to send data to and from files.

Create a "*README_beambend.txt"* file, and create detailed instructions on how to run your *sum.cpp* program, including what input should be entered.

What to submit

Submit your three source files *beambend.cpp*, *beam.cpp*, and *beam.h* and *README_beambend.txt* files to the Lab Assignment 8 LEARN dropbox.

Due Date

All materials for this lab are due **Friday, November 14** by **1:30pm.**

Reminders

Make sure that all your function prototypes are documented properly. See the Style Guide for an indication of how to do this.

Ensure that you are using `const` modifiers when necessary.