Lab 0: Arrays, Matrices, Oh My!

Due: January 29th at 11:59pm

Overview

In this lab you will start to program in C++ by implementing a matrix multiplication routine and a simple sorting routine. You will make use of arrays, conditionals, and for-loop control flow to complete this lab. This is an individual lab, and all work must be your own.

Deliverables

Your submission should minimally include the following code and supplemental files:

- `README`
- `matrix_mult.cpp`
- `max_sort.cpp`

The README file should be filled in with appropriate information, and be sure to include a brief description of your program, any problems your encountered, and how long it took you to complete the assignment.

Submission

You will submit using `handin35`. Be sure to place all relevant files in `/cs35/labs/00`.

1 Max Based Sorting

In this part of the lab, you will implement a basic selection sort sorting routine that will sort from max to min. Recall, the purpose of the lab is not to do sorting optimally, but rather familiarize yourself with using C++. You can find the skeleton code in `labs/00/max_sort.cpp` once you run `update35`.

At the heart of the sorting routine is a procedure for finding the maximal value in the input array, `input`. Once the max value (and its index) is identified, you will copy that value to the results array `results`. Additionally, you'll need to print out both `input` and `results`.

Sample Output Below is some sample output to help guide you.

```
bash >./max_sort
9 5 4 3 0 2 8 7 1 8
9 8 8 7 5 4 3 2 1 0
```

2 Matrix Multiplication

In this part of the lab, you will implement a matrix multiplication routine. You will find the skeleton code in `labs/00/matrix_mult.cpp` after you run `update35`. In this lab, you'll see a matrix defined as a multi-dimensional array:

```
int size = 10;
int matrix[size][size]
```

The goal of this lab is to square the matrix `matrix` using matrix multiplication and place the result in another multi-dimesional array that you must define yourself. When you review the code, you'll note that `matrix` is initialized for you using a computation over the indexes. You may You may change this initialization if you wish, and I recommend you adjust the `size` value to test your code with different size arrays.

Indexing the matrix When indexing a multi-dimensional array as a matrix, you should consider the first index in the array to refer to the rows of the matrix, and the second index in the array to refer to the column of the matrix. For example, in the matrix below, `matrix[1]` refers to the second row, $\begin{bmatrix}1&2&3\end{bmatrix}$, and `matrix[1][1]` refers to the second row's second value, that is 2.

$$\begin{bmatrix}0&1&2\\1&2&3\\2&3&4\end{bmatrix}$$

Mathematics of Matrix Multiplication Recall that matrix multiplication is an interative process of performing row-by-collumn multiplication of the two matrixes. More precisely, the value at index (i, j) in the output array is dot-product of multiplying row i of the left-input by column j of the right-input. While this may seem complicated, it's easily understood in an example.

$$\begin{bmatrix}0&1&2\\1&2&3\\2&3&4\end{bmatrix} \times \begin{bmatrix}0&1&2\\1&2&3\\2&3&4\end{bmatrix} = \begin{bmatrix}5&8&11\\8&14&20\\11&20&29\end{bmatrix}$$

The value of the results matrix at index [0][0] is coputed by mutlipling the first row of the left-input- by the first column of the right-input, like so:

$$\begin{bmatrix}0 & 1 & 2\end{bmatrix} \times \begin{bmatrix}0\\1\\2\end{bmatrix} = 0*0+1*1+2*2=5$$

When programming the matrix multiplication routine in C++, you'll find that it will require three for-loops. **In your README file, describe why this is the case.** If you encounter a `SEGFAULT`, this means that you have gone out-of-bounds of your matrix-arrays. To fix this check your bounds in the for-loop carefully and be sure to refer to `size`.

Sample Output Below are some sample output to help guide you:

```
bash >./matrix_mult
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12
4 5 6 7 8 9 10 11 12 13
5 6 7 8 9 10 11 12 13 14
6 7 8 9 10 11 12 13 14 15
7 8 9 10 11 12 13 14 15 16
8 9 10 11 12 13 14 15 16 17
9 10 11 12 13 14 15 16 17 18
Result of matrix^2:
285 330 375 420 465 510 555 600 645 690
330 385 440 495 550 605 660 715 770 825
375 440 505 570 635 700 765 830 895 960
420 495 570 645 720 795 870 945 1020 1095
465 550 635 720 805 890 975 1060 1145 1230
510 605 700 795 890 985 1080 1175 1270 1365
555 660 765 870 975 1080 1185 1290 1395 1500
600 715 830 945 1060 1175 1290 1405 1520 1635
645 770 895 1020 1145 1270 1395 1520 1645 1770
690 825 960 1095 1230 1365 1500 1635 1770 1905
```

REMINDER: Compiling and Running Your Code

To compile your code, you may use either the GNU C++ compiler, `g++`, or the Clang C++ compiler `c++`. I strongly recommend you use Clang because it provides slightly better error output.

Recall from class that you compile a c++ program on the command line like so:

```
bash> c++ code_file.cpp -o binary
```

Where the code for your program is in `code_file.cpp` and the executable output of compilation is written to `binary`. To execute your program, you run the `binary` on the command line like so:

```
bash>./binary
```