

# CA146 Introduction to Programming in C++

## CA146 Tutorial 8

### 1 Using one-dimensional arrays for some vector calculations

*Note you must #include <cmath> in your program as you will need to use the sqrt function.*

Type in, compile & run a program to do the following:

- Read the length (n) of 1-D “float” arrays  $x$  and  $y$  (up to a fixed maximum) Note: we can regard  $x$  and  $y$  as vectors with  $x = (x_0, x_1, x_2, \dots, x_{n-1})$  and  $y = (y_0, y_1, y_2, \dots, y_{n-1})$
- Read values for the n elements of  $x$ , read values for the n elements of  $y$
- Calculate & output the length of  $x$ ,  $\text{Length}_x = \sqrt{(x_0^2 + x_1^2 + x_2^2 + \dots + x_{n-1}^2)}$
- Calculate & output the length of  $y$ ,  $\text{Length}_y = \sqrt{(y_0^2 + y_1^2 + y_2^2 + \dots + y_{n-1}^2)}$
- Calculate the unit vector  $\hat{x}$  obtained by dividing each element of  $x$  by  $\text{Length}_x$ .
- Calculate the unit vector  $\hat{y}$  obtained by dividing each element of  $y$  by  $\text{Length}_y$ .
- As a check, calculate & output the length of both  $\hat{x}$  and  $\hat{y}$ . (Both should be 1.0).

You should try out your program with different sets of input data, including:

- 4
- 1 1 1 1
- -1 0 1 2

### 2 Using two-dimensional arrays

2.1 Type in, compile & run the following program which reads data into two 2-D arrays A and B and then prints them out again (in two different ways to emphasise the order in which 2-D arrays are stored in memory).

```
#include <iostream>
using namespace std;
```

```
const int SIZE = 5;
```

```
int
main()
{
    int i, j, n;
    int A[SIZE][SIZE];
    int B[SIZE][SIZE];

    // Initialise all entries to zero
    for (i = 0; i < SIZE; i++) {
        for (j = 0; j < SIZE; j++) {
            A[i][j] = 0;
            B[i][j] = 0;
        }
    }
}
```

```
// A and B are square matrices, read in the number of rows and columns
cout << "How many rows and columns in A and B?" << endl;
cout << "(Note: maximum " << SIZE << ")" << endl;
cin >> n;
```

## CA146 Introduction to Programming in C++

```
// Read values for the elements of A (row by row):
for (i = 0; i < n; i++) {
    cout << "A Row " << i << " (enter " << n << " values)" << endl;
    for (j = 0; j < n; j++) {
        cin >> A[i][j];
    }
}
// Output values for the elements of A (row by row):
for(i = 0; i < n; i++) {
    cout << "A Row " << i << ": ";
    for (j = 0; j < n; j++) {
        cout << A[i][j] << " ";
    }
    cout << endl;
}
// "Memory dump" of A:
for (i = 0; i < SIZE; i++) {
    cout << "A Row " << i << ": ";
    for (j = 0; j < SIZE; j++) {
        cout << A[i][j] << " ";
    }
    cout << endl;
}

// Read values for the elements of B (row by row):
for (i = 0; i < n; i++) {
    cout << "B Row " << i << " (enter " << n << " values)" << endl;
    for (j = 0; j < n; j++) {
        cin >> B[i][j];
    }
}
// Output values for the elements of B (row by row):
for (i = 0; i < n; i++) {
    cout << "B Row " << i << ": ";
    for (j = 0; j < n; j++){
        cout << B[i][j] << " ";
    }
    cout << endl;
}
// "Memory dump" of B:
for (i = 0; i < SIZE; i++) {
    cout << "B Row " << i << ": ";
    for (j = 0; j < SIZE; j++) {
        cout << B[i][j] << " ";
    }
    cout << endl;
}
return (0);
}
```

## CA146 Introduction to Programming in C++

2.2 Using the program of 2.1 as a starting point, write a program to:

- For each of square matrices A and B, read values into the matrix and, as a check, print them out again (*just use the program of 2.1 except remove the “memory dump” part as irrelevant*)
- Declare a new 2-D array C of the same size as A and B (requires adding their declaration to the 2.1 program)
- Calculate the elements of C where  $C = A + B$  (i.e. a matrix sum). In maths notation, this means that  $C_{ij} = A_{ij} + B_{ij}$  for each i and j. So, in the program, you will need to calculate  $C[i][j] = A[i][j] + B[i][j]$  inside nested loops (such as those in the 2.1 program).

### 3 Multiplying matrices

The product of the m x n matrix A and the n x p matrix B is the m x p matrix AB where:

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

Using the above formula as a guide, read the comments in this program and make the changes. Checking your answers is easy since  $AB == B$  for this example:

```
#include <iostream>

using namespace std;

const int M = 3;
const int N = 3;
const int P = 2;

int
main()
{
    int a[M][N] = { {1, 0, 0},
                    {0, 1, 0},
                    {0, 0, 1} };

    int b[N][P] = { {2, 3},
                    {4, 5},
                    {6, 7} };

    int ab[M][P] = { {0, 0},
                     {0, 0},
                     {0, 0} };

    int i, j, k;
```

## CA146 Introduction to Programming in C++

```
// Write the code to compute ab[0][0] (requires 1 for loop)
i = 0;
j = 0;
for (k = 0; k < N; k++) {
    ab[i][j] += a[i][k] * b[k][j];
}

// Write the code to compute ab[2][1] (requires 1 for loop)
// Write the code to compute ab[0][j] (requires 2 for loops)
// Write the code to compute ab[i][1] (requires 2 for loops)
// Write the code to compute ab[i][j] (requires 3 for loops)

return (0);
}
```

You should try out your program with different sets of input data, including for

$$A = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \quad B = \begin{vmatrix} 20 & 20 \\ 20 & 20 \end{vmatrix}$$

for which you should get

$$AB = \begin{vmatrix} 60 & 60 \\ 140 & 140 \end{vmatrix}$$