COMP - MEMORY LAYOUT FOR MULTI-DIMENSIONAL ARRAYS (MIEIC - Compilers - 2021)

* This form will record your name, please fill your name.

1. Each programming language has its own convention to store multi-dimensional arrays. Suppose the following array declaration in the C programming language:
   int A[3][2];
   The way the elements of A are stored in memory is according to the following table :

   Array element - Address position relative to the first array element
   ...
   A[0][0] 0
   A[0][1] 4
   A[1][0] 8
   A[1][1] 12
   A[2][0] 16
   A[2][1] 20
   ... ...

   The expression to access a particular element of array A, A[i][j] is given by A+(i*2+j)*4 (multiplied by 4 considering a representation of 'int' with 4 bytes)

2. What is the error in the following function (consider the body of the function without errors) and what is the rule a C compiler must verify? Justify why in the C programming language this function has an error.
   int f(int A[][]) {...}
   (1 Point)

3. Give an example of a programming language with another memory layout convention for 2D arrays.
   (1 Point)

4. Why the following equivalent Java code does not have the same error?
   int f(int[][] A) {...}
   (1 Point)

5. What is the expression to calculate the address of an array element B[i][j][t] in case of the following array declaration in the C programming language?
   int B[3][2][4];
   (1 Point)

6. Considering C code, what is the generic expression to calculate the address of the element position for an array A (type int) with N dimensions, with sizes S1, S2,..., SN, considering a declaration based on the following one, and an array access like A[i1][i2][i3]...[iN]?
   int A[S1][S2][S3]...[SN];
   (1 Point)

7. The code in the textbox below multiplies each element of matrices A and B and the result is output to matrix C. One possible code optimization that can improve performance and save energy consumption is loop interchange (*). By interchanging the two loops, Calvin found that the execution time reduced about 3x when using the following declarations for arrays A, B, and C, compiling the code with gcc –O3 and executing it in a Windows 7, 64-bit, Intel i5-2467M @1.60 GHz, 4 GB of RAM. What is the main reason for this execution time reduction?

```
#define M1 5000
#define N1 5000

float A3[M1][N1];
float A4[M1][N1];
float C1[M1][N1];

void DotProd(float A[M1][N1], float B[M1][N1],
                                float C[M1][N1], int m, int n) {
        int i, j;

        for (j = 0; j < n; j++) {
                for (i = 0; i < m; i++) {
                        C[i][j] = A[i][j]*B[i][j];
                }
        }
}
```

* In loop interchange the order of two nested loops are exchanged (see, e.g., https://en.wikipedia.org/wiki/Loop_interchange (https://en.wikipedia.org/wiki/Loop_interchange)).
(1 Point)