UNIVERSITE DE RENNES
Département de Mathématiques
PSc - Programmation scientifique

2025–2026
M1 Mathématiques et applications
Calcul scientifique et modélisation
Simulation numérique pour la santé

TD - TP 5

Objectifs : programmation en langage C

- Structure

Références

- **Chapter 8 : Structure**

A. Structure of matrix stored in a vector

1. Consider the header file `matrix.h`

```
// Structure of a 2D-matrix as a vector for efficiency

typedef struct
{
  int nrow;
  int ncol;
  double *mat;
} Matrix;

void mat_create(Matrix *, int, int);
void mat_free(Matrix);
void mat_init(Matrix *);
void mat_print(Matrix);
void matsum(Matrix , Matrix , Matrix *);
```

Structure/matrix.h

defining a structure `Matrix` of `nrow` rows and `ncol` columns stored in a vector: the element A_{ij} corresponds to the element of the vector `A[i*ncol+j]`.

2. Write the functions to respectively
   - dynamically allocate a matrix with `nrow` rows and `ncol` columns,
   - free a matrix,
   - initialize a matrix by reading the element values from standard input (using `scanf`),
   - print a matrix to standard output (using `printf`),
   - sum two matrices.

3. Test the functions by calling them from a `main` function that reads the dimensions of two matrices `A` and `B`, initializes them, calculates their sum and prints the results.

4. Create a library `libmatrix.a` containing the 5 functions handling the matrices.

5. Create the executable using `libmatrix.a` and the `main` function in the separate file `test_matrix.c`.

B. Structure of 2D-array

1. Consider the header file `M_matrix.h`

```c
// structure of a 2D-matrix

typedef struct
{
  int nrow;
  int ncol;
  double *mat_alloc;
  double **mat;
} Matrix;

void mat_create(Matrix *, int, int);
void mat_free(Matrix);
void mat_init(Matrix *);
void mat_print(Matrix);
Matrix matmul(Matrix,Matrix);
```

Structure/M_matrix.h

where `Matrix` is a structure corresponding to a 2D-array with members

- `nrow` = numbers of rows,
- `ncol` = number of columns,
- matrix of **contiguous** `nrow` × `ncol` doubles.

2. Write the functions to respectively

- dynamically allocate `A`
- free `A`
- initialize the matrix as: $A_{ij} = \cos(i * \mathrm{M\_PI}/2) + \sin(j * \mathrm{M\_PI}/2) + i + j$
- print matrix `A`
- multiply 2 matrices `A` and `B`.

3. Write a function `main` that

- reads the dimensions of matrix `A[nrowA][ncolA]`,
- allocates and initializes `A`,
- reads the dimensions of matrix `B[nrowB][ncolB]`,
- allocates and initializes `B`,
- multiplies `A` by `B`,
- prints the result.

4. Create a library `libmatrix.a` containing the 5 functions handling the matrices.

5. Create the executable using `libmatrix.a` and the `main` function in the separate file `test_matrix.c`.