TP1 C++

Exercice 1 (classe Rmax) : l'*algèbre (max,+)* est l'ensemble des réels muni de **⊕=max et ⊗=+**.
La valeur 0 est neutre pour ⊗ et ε=-∞ (epsilon) est neutre pour ⊕. Par exemple,

$3 \oplus 2 = \max(3,2) = 3, \quad 1 \otimes 1 = 1+1 = 2, \quad 3 \otimes (2 \oplus 1) = (3 \otimes 2) \oplus (3 \otimes 1) = 5 \oplus 4 = 5, \quad 7 \otimes \varepsilon = \varepsilon$

Matrices (max,+) : le calcul matriciel dans cette structure algébrique

$$A = \begin{pmatrix} 1.5 & \varepsilon \\ 2 & 3.2 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 1 \\ \varepsilon & 2 \end{pmatrix} \qquad A \oplus B = \begin{pmatrix} 1.5 & 1 \\ 2 & 3.2 \end{pmatrix} \quad A \otimes B = \begin{pmatrix} 1.5 & 2.5 \\ 2 & 5.2 \end{pmatrix}$$

TODO : écrire la classe `math::Rmax` (diag. UML en fin de texte). Un objet **`math::Rmax`** décrit un réel dont les opérations sont celles de l'algèbre (max,+). Les opérateurs + et * doivent être programmés pour cela.

`valeur numérique de epsilon = std::numeric_limits<double>::lowest() (<limits>)`

```
int main(int argc, char *argv[])
{
	Rmax a(2), b(1), c;  // c vaut epsilon=-oo par défaut
	cout << (a + b) << endl;  // vaut 2 (c'est max(2,1))
	cout << (a*b) << endl;  // vaut 3 (c'est 2+1)
	cout << c << endl;  // vaut eps dans Rmax (-oo)
	cout << "eps=" << Rmax::epsilon() << endl;  // méthode de classe

	// Matrice<T> est fournie
	Matrice<Rmax> M1(2, 2), s(1, 2);
	M1(0, 0) = 2; M1(0, 1) = 3; M1(1, 1) = 4;
	s(0, 0) = 1;  s(0, 1) = 4;
	cout << M1.toString() << endl;         // [2 3;eps 4]
	cout << (s*M1).toString() << endl;     // [3 8]
	cout << (s*M1*M1).toString() << endl;  // [5 12]

	// exemple introductif
	Matrice<Rmax> A(2, 2), B(2, 2);
	A(0, 0) = 1.5; A(1, 0) = 2; A(1,1) = 3.2;
	B(0, 0) =0; B(0, 1) = 1; B(1, 1) = 2;
	cout << (A+B).toString() << endl;
	cout << (A*B).toString() << endl;
}
```

Note: la classe **`math::Matrice<T>`** utilise les opérateurs + et * des éléments de type T pour réaliser les opérations matricielles. En fournissant un type `Rmax` qui implémente ces opérations, le produit matriciel dans (max,+) est pris en charge par `math::Matrice<Rmax>`.

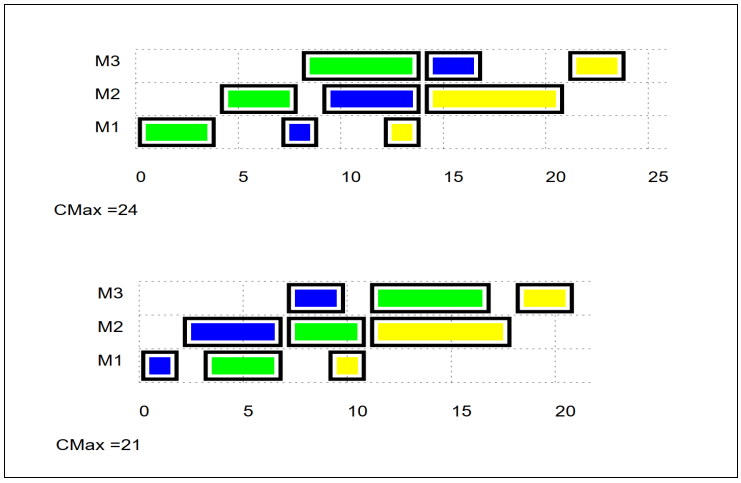
```
//exemple de produit matriciel dans R à l'aide de Matrice<double>
int main(int argc, char *argv[]){
	Matrice<double> M1(2, 2), s(1, 2);   // matrices de réels
	M1(0, 0) = 2;  M1(0, 1) = 3;  M1(1, 1) = 4;
	s(0, 0) = 1;  s(0, 1) = 4;
	cout << (s*M1).toString() << endl;       //[2 19]
	cout << (s*M1*M1).toString() << endl;    //[4 82]
}
```

Exercice 2 (Job dans un FlowShop à 3 machines)

Dans un atelier *Flow Shop*, chaque job J_i fait l'objet d'une opération sur 3 machines M_1 à M_3. Les durées sont stockées dans un objet `Job3M`. On souhaite calculer la durée d'exécution (ou Cmax) d'une séquence de jobs quand il n'y a pas d'attente entre deux machines.

Exemple : Gantt J1 (vert) puis J2 (bleu) puis J3(jaune) (Cmax=24)
Gantt J2 puis J1 puis J3 (CMax=21)

Jobs	J1	J2	J3
M1	4	2	2
M2	4	5	7
M3	6	3	3



On peut calculer le Cmax grâce à des produits de matrices `math::Matrice<Rmax>` (exercice 1). A chaque job (durées [d1,d2,d3]) est associée une matrice (max,+) : (+ = somme classique)

[d1 d1+d2 d1+d2+d3;
0 d2 d2+d3;
-d2 0 d3]

Pour l'exemple ci-dessus, les matrices (3×3) associées aux jobs J1 à J3 sont :

J1 = [4 8 14; 0 4 10; -4 0 6] **J2=[2 7 10; 0 5 8; -5 0 3]** **J3=[2 9 12; 0 7 10; -7 0 3]**

Pour obtenir le Cmax, il suffit de multiplier les matrices (max,+) associées aux jobs. En ajoutant la matrice s=[0 0 0], on obtient le Cmax de J1,J2,J3 et J2,J1,J3 comme suit :

s⊗J1⊗J2⊗J3=[14 21 24] fin à 14 sur M1, 21 sur M2 et 24 sur M3 (cf. Gantt)

s⊗J2⊗J1⊗J3=[11 18 21] fin à 11 sur M1, 18 sur M2 et 21 sur M3.

La matrice ligne ainsi obtenue donne "le profil" de l'empilement, c'est-à-dire l'heure de fin du dernier job sur les trois machines M1,M2,M3. Le ***Cmax est la valeur de fin sur M3.***

TODO : écrire la classe `Job3M` ainsi qu'un programme de calcul de makespan

Classe `flowshop::Job3M`: les objets Job3M doivent stocker les durées opératoires et fournir la matrice (max,+) caractéristique.

```
// utilisation Job3M
#include <iostream>
#include "Rmax.h"

using namespace std;
using namespace flowshop;
using namespace math;

int main(int argc, char *argv[]){
	Job3M J1(2,4.5,5);
	Job3M J2(3,1,7);
	cout << "J1=" << J1.toString() << endl; // <2,4.5,5>
	cout << "J2=" <<J2.toString() << endl; // <3,1,7>
	cout << J1.getDureeMachine(0) << endl; //2
	cout << J1[1] << endl; //4.5
	Matrice<Rmax> mJ1=J1.getMatrice();
	cout << "mJ1=" << mJ1.toString() << endl; //[2 6.5 11.5;0 4.5 9.5;-4.5 0 5]
}
```

```
J1=<2,4.5,5>
J2=<3,1,7>
2
4.5
mJ1=[2 6.5 11.5;0 4.5 9.5;-4.5 0 5]
```

Exercice 3 (Programme de calcul de Cmax)

Classe `flowshop::Parser`: la classe Parser doit contenir un parser (fonction membre statique) **`Parser::toJobs(string)`** pour créer des jobs à partir d'une chaîne de caractères. Notez que l'expression régulière `\d+(\.\d+)?` permet de reconnaître le format des nombres à virgule.

```
	"<1,2.4,3><3,4,7.5>"  -> 2 objets Job3M dans un vecteur

int main(int argc, char *argv[]){
	vector<Job3M> v = Parser::toJobs("<1,2.4,3><3,4,7.5>");
	for (Job3M j : v){ cout << j.toString() << endl;}
}
```

```
<1,2.4,3>
<3,4,7.5>
```

Ecrire un programme console qui exploite une liste de jobs (format texte) puis affiche le Cmax de la séquence. Ci-dessous un exemple d'exécution de ce programme.

```
F:\Temp>Cmax <4.5,5,6><1,2.5,3><7,8,9.5><10,12.5,3><2,9.1,3>
Jobs lus=<4.5,5,6><1,2.5,3><7,8,9.5><10,12.5,3><2,9.1,3>
calcul du Cmax:
profil=[42.5 51.6 54.6]
Cmax=54.6
```

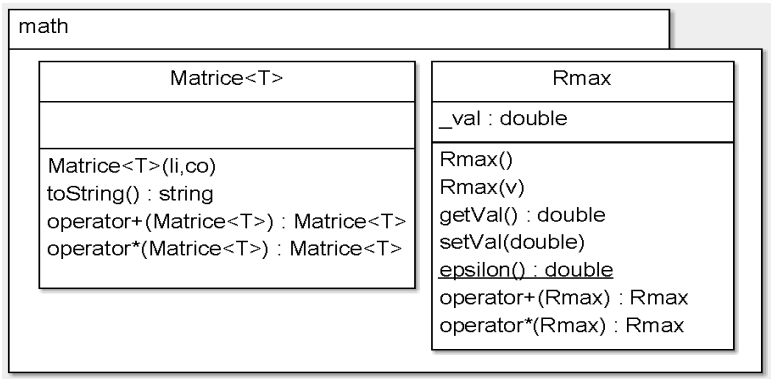
Exercice 4 (option, selon avancement)

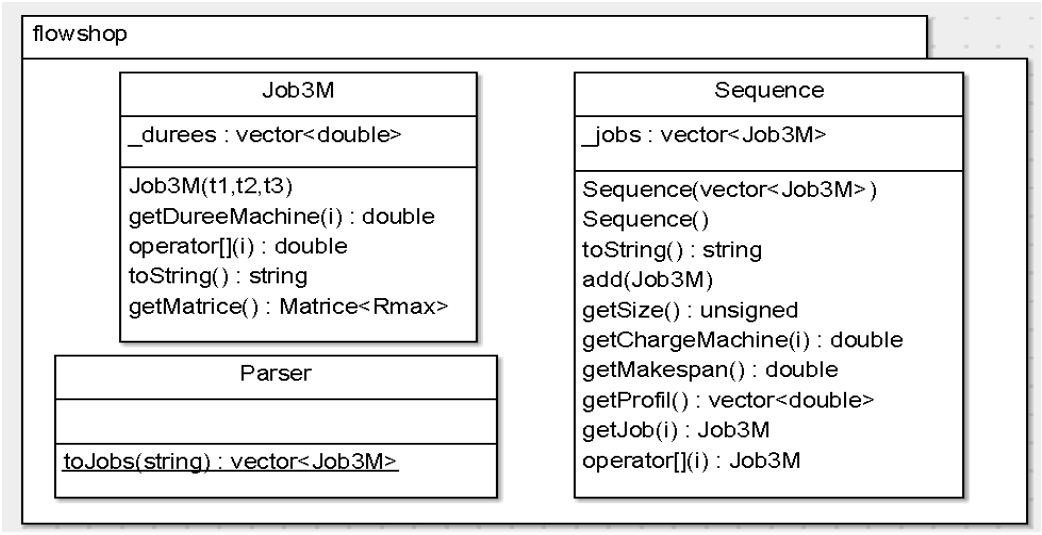
Ecrire un programme console qui exploite une séquence de jobs et un nombre d'itérations (nbIt). Le programme cherche, parmi nbIt permutations aléatoires de la séquence, celle qui minimise le makespan

Note : utiliser random_shuffle pour mélanger/permuter aléatoirement un vecteur

```
F:\Temp>optim 50
saisir : <4.5,5,6><1,2.5,3><7,8,9.5><10,12.5,3><2,9.1,3>
Meilleure sequence=<2,9.1,3><4.5,5,6><7,8,9.5><10,12.5,3><1,2.5,3>
Cmax=46.6
```

Diagramme UML





Exercice 5 (option, selon avancement)

Ecrire un programme console permettant de multiplier des matrices de réels (algèbre classique). Les deux matrices sont données dans la ligne de commande

```
F:\Temp>matrix_mul [1 2;3 4] [2 3;4 5]
resultat:[10 13;22 29]
```