

TP 11 Transitions

Objectifs

L'objectif de ce TP est de faire le lien avec les cours de S6 "Programmation Avancée" (section 1), et "Analyse Numérique" (section 2). Passer 1H sur chaque section !

1 Utilisation de la librairie `string.h`

Taille, concaténation : `strlen`, `strcat` Dans `tp11.c` :

- Déclarer une chaîne de caractère dans un tableau de taille statique N, N étant une constante symbolique fixée à 100, et sur la même ligne l'initialiser à "turlututu".
- Imprimer cette chaîne. (`printf` avec la chaîne de formatage `%s`)
- Après avoir inclus la librairie `string.h`, utiliser la fonction `strlen` pour calculer, puis imprimer la taille de la chaîne.
- Dans un terminal, observer la documentation de la librairie `string.h` (`man 3 string`) et y retrouver la signature de la fonction `strlen`. Qu'en déduisez-vous ?
- Déclarer une autre chaîne de taille statique, l'initialiser, puis imprimer directement le résultat de la concaténation des deux chaînes (`strcat`) :

```
printf(concat de %s et %s = %s, ch1, ch2, strcat(ch1, ch2));
```
- Déclarer une troisième chaîne statique, et essayer de récupérer le résultat de la concaténation, observer le message d'erreur de compilation. Supprimer cette déclaration.
- Récupérer le résultat de la concaténation à l'aide d'une variable de type `char*`, puis imprimer.

► Conclusion, différence entre `char*` et `char[N]` ? une question de taille !

`char*`, `fgets` et `segmentation fault` Toujours dans le même `.c` :

- Déclarer une chaîne statique `ch6` de taille N, et l'initialiser avec `fgets` (`man 3 fgets`) :

```
char* varquisertarien = fgets(ch6, 5, stdin);
```

(demande sur le terminal/l'entrée standard, une chaîne, et stocke dans `ch6` les 4 premiers caractères). Imprimer `varquisertarien` et `ch6` pour vérifier.
- Que se passe-t-il si vous déclarez `ch6` comme étant un `char*`, à la compilation ? à l'exécution ? Pourquoi ?

► Il manque une façon de "donner une taille" à un `char*`. (cf PA, deuxième semestre)

2 Matrices statiques, pivot de Gauss

On vous fournit un fichier `tp11_mat.c` contenant des fonctions de base pour les matrices de taille statique.

Pivot de Gauss - Résolution de $Ax=b$ Adapté d'une partie du chapitre 1 d'Analyse Numérique (les formules sont exactement les mêmes, les indices aussi!). On cherche à résoudre un système $Ax = b$ avec A matrice $n \times n$, b vecteur de taille n :

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n = b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n = b_2, \\ \dots \\ a_{n,1}x_1 + a_{n,2}x_2 + \dots + a_{n,n}x_n = b_n, \end{cases} \quad (1)$$

où les coefficients $a_{i,j}$, pour $i, j \in \{1, \dots, n\}$ sont les coefficients de la matrice A . On suppose que A est inversible, donc que le système n'a qu'une unique solution. **Attention!** en maths les indices vont de 1 à n compris, en algo de 0 à $n-1$!

Étape 1 : résolution dans le cas triangulaire En prenant un exemple simple, se convaincre que si A est triangulaire supérieure, alors on peut calculer aisément l'unique solution du système à l'aide des formules suivantes :

$$\begin{cases} x_n = \frac{b_n}{a_{n,n}}, \\ x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}, \\ \dots \\ x_1 = \frac{b_1 - a_{1,2}x_2 - \dots - a_{1,n}x_n}{a_{1,1}}. \end{cases} \quad (2)$$

1. Vérifier que $x_i = \frac{b_i - \sum_{k=i+1}^n a_{i,k}x_k}{a_{i,i}}$.

2. i varie de quoi à quoi ?

3. Écrire donc une fonction C (attention aux indices), de signature :

```
void solve_triangular_system(float A[N][N], float b[N], float result[N]);
```

qui étant donnés A triangulaire supérieure, b un vecteur, calcule et stocke dans $result$ l'unique solution (obtenue par les formules que l'on vient d'obtenir).

4. Tester votre fonction avec $A = \begin{pmatrix} 2 & 3 \\ 0 & -5.5 \end{pmatrix}$ et $b = \begin{pmatrix} 3 \\ 2.5 \end{pmatrix}$. On doit trouver $\begin{pmatrix} 2.181818 \\ -0.454545 \end{pmatrix}$.

Étape 2 : pivotage pour arriver triangulaire supérieure On cherche donc, étant donné A inversible quelconque, et b vecteur, à rendre A triangulaire supérieure, et à modifier b en conséquence pour que le système obtenu soit équivalent au système initial. On va modifier A et b , par étapes, de façon à ce qu'à l'étape i la matrice A soit de la forme :

$$\begin{pmatrix} a_{1,1}^{(1)} & \dots & a_{1,n}^{(1)} \\ 0 & \ddots & & & & & & & \vdots \\ \vdots & \ddots & \ddots & & & & & & \vdots \\ \vdots & & 0 & a_{i,i}^{(i)} & & & & & \vdots \\ \vdots & & \vdots & a_{i+1,i}^{(i)} & \ddots & & & & \vdots \\ \vdots & & \vdots & \vdots & & \ddots & & & \vdots \\ \vdots & & \vdots & \vdots & & & \ddots & & \vdots \\ 0 & \dots & 0 & a_{n,i}^{(i)} & \dots & \dots & \dots & \dots & a_{n,n}^{(i)} \end{pmatrix}.$$

Après l'étape $n - 1$ la matrice A sera donc triangulaire supérieure.

1. *En prenant un exemple de taille 3×3* , vérifier que les formules suivantes réalisent l'étape i de l'algorithme de Gauss :
 - Pour $k \leq i$ et pour $j = 1, \dots, n$, la matrice A et le vecteur b sont inchangés.
 - Pour $k > i$, **en supposant que $a_{i,i} \neq 0$** ,¹ on modifie A et b de la façon suivante :

$$a_{k,j} \leftarrow a_{k,j} - \frac{a_{k,i}}{a_{i,i}} a_{i,j}, \text{ pour } j = 1, \dots, n \quad (3)$$

$$b_k \leftarrow b_k - \frac{a_{k,i}}{a_{i,i}} b_i. \quad (4)$$

Attention, il s'agit des $a_{,*}$ de l'étape précédente !*

2. i varie de quoi à quoi ? Justifier le fait que j peut varier uniquement de i à n .
3. Écrire donc une fonction C (attention aux indices), de signature :

```
void gauss_pivot_simple(float A[N][N], float b[N])
```

qui transforme A et b de façon à rendre A triangulaire supérieure en gardant le système $Ax = b$ équivalent à l'initial.

4. Tester votre fonction avec $A = \begin{pmatrix} 2 & 3 \\ 5 & 2 \end{pmatrix}$ et $b = \begin{pmatrix} 3 \\ 10 \end{pmatrix}$. On doit trouver après modifications

$$A = \begin{pmatrix} 2 & 3 \\ 0 & -5.5 \end{pmatrix} \text{ et } b = \begin{pmatrix} 3 \\ 2.5 \end{pmatrix}.$$

Les notions acquises en Programmation Structurée, notamment concernant la conception d'algorithmes et leur réalisation en C, sont un pré-requis du cours d'Analyse Numérique du S6. Le langage algorithmique a été normalisé entre les deux enseignements. Les choix d'implémentation seront quelquefois différents en Matlab, qui possède des fonctions de traitement mathématiques spécialisées. En bonus, on vous fournit un tableau récap C/Matlab !

1. Le cours d'Analyse numérique vous dira quoi faire dans le cas contraire !

Syntaxe Algorithmique, C, Matlab

Syntaxe de base

Nom	Syntaxe Algo	Syntaxe C	Syntaxe MATLAB
Affectation exemple	← x ← 42	= x=42	= x=42
Constante numérique	Constante N:10	#define N 10	N=10 Pas une véritable constante (on peut changer la valeur)
Type entier déclaration d'entier	Entier x:Entier	int int x	Pas de déclaration de variable : inférence automatique de type
Type réel déclaration de réel	Réel r:Réel	float float r	
Type caractère déclaration de réel	Caractère c:Caractère	char char c	
Type booléen déclaration de réel	Booléen b:Booléens	bool (inclure <code>stdbool</code>) bool b	
Caractères Booléens constants	'a', 'b' Vrai, Faux	'a','b' true,false	'a', 'b' ?? true,false
expressions booléennes expressions numériques	a et b, a ou b, non(a), $x \geq 2$ $2x+21$	a b, a && b, !a, $x \geq 2$ $2*x+21$	a && b, ~a, $x \geq 2$ $2*x+21$
fonctions maths de base	cos,sin,exp	idem (inclure <code>stdmath.h</code> et lier avec <code>-lm</code>)	fonctions maths "avancées" déjà écrites (dérivée, équas diff, ...)
Tableau Exemple	Vecteur[taille] de type toto:Vecteur[10] d'entiers	type t[taille] int toto[10]	Tout est matrice. Pas besoin de spécifier la taille
Matrice Exemple	Matrice[n][m] de type titi:Matrice[10][12] d'entiers	type t[n][m] int titi[10][2]	
Initialisation tableau à 0 Initialisation matrice à 0	(boucle pour) (boucle pour)	int t[4]={0} (uniquement à la déclaration); int t[4][8]={0} (idem);	zeros(4) si matrice carrée
Type chaîne exemple Chaînes constantes	Chaîne s : chaîne "toto"	char[N] ou char* char s[10] ou char* s "toto"	Pas de déclaration idem 'toto'

Structures de contrôle Dans tous les tests si dessous, les *conditions* sont des expressions booléennes (qui s'évaluent donc en vrai/true ou faux/false).

Nom	Syntaxe Algo	Syntaxe C	Syntaxe MATLAB
Conditionnelle I	<p><u>Si</u> <i>condition</i> <u>alors</u> instructions si vrai <u>Fsi</u></p>	<pre>if (condition) { instructionssivrai }</pre>	<pre>if condition instructionssivrai end</pre>
Conditionnelle II	<p><u>Si</u> <i>condition</i> <u>alors</u> instructions si vrai <u>Sinon</u> instructions si faux <u>Fsi</u></p>	<pre>if (condition) { instructionssivrai } else { instructionssifaux }</pre> <p>Les accolades ne sont pas nécessaires en cas d'instruction unique.</p>	<pre>if condition instructionssivrai else instructionssifaux end</pre>
Conditionnelle III	<p><u>Si</u> <i>cond1</i> <u>alors</u> instructions si cond1 vraie <u>Sinon</u> <u>Si</u> <i>cond2</i> <u>alors</u> instructions si cond1 fausse et cond2 vraie <u>Sinon</u> instructions si cond1 et cond2 fausses <u>Fsi</u></p>	<pre>if (condition) { instructionssivrai } else { if (cond2) { instructionssifaux } }</pre>	<pre>if condition inst1 elseif cond2 inst2 else inst3 end</pre> <p>inst1 faites si cond1 vraie, inst2 si cond1 fausse et cond2 vraie, inst3 sinon.</p>
Boucle Pour	<p><u>Pour</u> <i>i de inf à sup</i> <u>Faire</u> instructions <u>Fpour</u> (incrément de 1 implicite)</p>	<pre>int i; for (i=inf; i<=sup; i++){ instructions }</pre> <p>(i++ raccourci pour i=i+1) (on peut aussi décrémenter)</p>	<pre>for i=inf:sup instructions end</pre> <p>Par défaut, l'incrément est 1. Pour spécifier un autre incrément <i>k</i> on écrit : for i=inf:k:sup <i>k</i> peut être positif ou négatif.</p>
Boucle Tant Que	<p><u>Tq</u> <i>condition</i> <u>faire</u> instructions <u>Ftq</u></p>	<pre>while (condition){ instructions }</pre>	<pre>while condition instructions end</pre>

Fonctions, programmes

Nom	Syntaxe Algo	Syntaxe C	Syntaxe MATLAB
Programme principal	<p>Programme Main</p> <pre> Retourner 0 FProgramme </pre>	<pre> int main(){ return 0; } </pre>	<pre> </pre> <p>Pas de fonction "d'entrée" pour un script Matlab. Le fichier est exécuté directement, du début à la fin.</p>
Fonction	<p>Fonction <i>fonct(c)</i> :</p> <p><i>Entier</i></p> <pre> D: c:Caractère L: s:Entier Retourner s FFonction </pre>	<pre> int fonct(char c){ int s; return s; } </pre> <p>return obligatoire.</p>	<pre> function [s]=fonct(c) instruction(s); s=instruction; end </pre> <p>Remarques :</p> <ol style="list-style-type: none"> 1. Les points virgule (optionnels) empêchent l'affichage de toutes les instructions dans la fenêtre de commande. 2. Il est important que le nom du fichier contenant la fonction soit exactement le même que celui de la fonction. 3. Le return est facultatif, mais utile si on veut interrompre le flot d'exécution.
Action	<p>Action <i>monact(a,b,c)</i></p> <pre> D: a : Entier D/R: b : Entier R: c : Caractère FAction </pre> <p>(les modifications faites à b et c sont enregistrées)</p>	<pre> void monact(int a, int* pb, int* pc){ } </pre> <p>(pointeurs : passage par adresse)</p>	<p>Pas de passage par adresse, mais on peut retourner plusieurs résultats :</p> <pre> fonction [b,c]= monact(a,b,c) end </pre>

Caractéristiques comparées

Caractéristique	C	MATLAB
But des progs	généralistes	calcul scientifique
Langage	compilé	interprété (script)
Passage de paramètres standard	valeur (copie)	valeur
Accès aux adresses	oui (&x)	seulement pour les fonctions (@nomfonction)