

# **Langage C modernisé**

Travaux Pratiques – HAE303E – Licence 2 EEA / 2024

Kada KRIBICH

[kada.kribich@umontpellier.fr](mailto:kada.kribich@umontpellier.fr)

Mikhaël MYARA

[mikhael.myara@umontpellier.fr](mailto:mikhael.myara@umontpellier.fr)



*“La plupart des problèmes d'informatique proviennent de l'interface chaise-clavier.”*

Klaus Klages

**Langage C moderne - Travaux Pratiques - HAE303E - Licence 2 EEA/2024**

29 octobre 2024

Programmation en Langage C "modernisé" : Langage C de base avec des éléments empruntés au C++.  
par Kada Kribich et Mikhaël Myara, pour le Department EEA, Université de Montpellier, France



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0  
Unported License.*

# Table des matières

## Les bases du Langage C/C++

### Séance I – Environnement de travail • Traduction Python vers C/C++ ..... 10

I.1	Environnement de travail : chaîne de compilation	10
I.2	Traduction Python vers C/C++ Problème I.1 : Aire d'un Triangle	13
	Problème I.2 : Produit Scalaire	14
	Problème I.3 : Suite	14
	Problème I.4 : Autre suite	15
I.3	Pour vous entraîner Problème I.5 : Nombres premiers	15

### Séance II – Variables, Tableaux, Chaines de caractères (1) ..... 17

II.1	Variables Problème II.1 : Limites des nombres	17
	Problème II.2 : Jours, Heures, Minutes, Secondes	17
II.2	Tableaux Problème II.3 : Notes	17
II.3	Chaines de caractères Problème II.4 : Palindrome	18
II.4	Pour vous entraîner Problème II.5 : Renversant	18

### Séance III – Variables, Tableaux, Chaines de caractères (2) ..... 19

III.1	Variables Problème III.1 : Table ASCII	19
III.2	Tableaux Problème III.2 : Maximum, Minimum et tri d'un tableau	19
	Problème III.3 : Matrices	20
III.3	Pour vous entraîner Problème III.4 : Cryptage	20
	Problème III.5 : Trinomes	21
	Problème III.6 : Resistance Equivalente	21
	Problème III.7 : Triangle	21
	Problème III.8 : Distance	21
	Problème III.9 : Tables de multiplication	21
	Problème III.10 : Pyramide et Croix	22

<b>Séance IV – Fonctions .....</b>	<b>24</b>
Problème IV.1 : Ranger du code .....	24
Problème IV.2 : Utiliser des fonctions .....	26
Pour vous entraîner .....	28
Problème IV.3 : Prototypes .....	28
Problème IV.4 : Histogramme .....	29
<b>Séance V – Structures .....</b>	<b>32</b>
Problème V.1 : vecteurs dans l'espace .....	32
Problème V.2 : Matrices $2 \times 2$ .....	32
Problème V.3 : Code Morse .....	33
V.1 Pour vous entraîner .....	34
Problème V.4 : Carnet d'Adresses .....	34
Problème V.5 : Tableau de Mendeleev .....	34
<b>Les techniques du Langage C/C++</b>	
<b>Séance VI – Exploration de la mémoire et pointeurs .....</b>	<b>37</b>
Problème VI.1 : Pointeur sur une variable 1 .....	37
Problème VI.2 : Pointeur sur une variable 2 .....	37
Problème VI.3 : Pointeur sur un vector .....	38
VI.1 Pour vous entraîner .....	39
Problème VI.4 : Exercice de base sur les pointeurs .....	39
<b>Séance VII – Passage par adresse .....</b>	<b>40</b>
Problème VII.1 : Jours, heures, minutes, secondes – v2 .....	40
Problème VII.2 : minimum et maximum .....	41
Problème VII.3 : echange .....	41
Problème VII.4 : pair et impair .....	41
VII.1 Pour vous entraîner .....	42
Problème VII.5 : Solutions d'un trinôme .....	42
Problème VII.6 : Tri d'un tableau .....	42
Problème VII.7 : Calcul de Pi .....	44
Problème VII.8 : Histogramme v.2 .....	44
Problème VII.9 : Notes de musique .....	46
<b>Séance VIII – Synthèse : pointeurs, fonctions, tableaux, structures .....</b>	<b>48</b>
Problème VIII.1 : Matrices $2 \times 2$ – V2 .....	48
Problème VIII.2 : Rendre la monnaie .....	50
VIII.1 Pour vous entraîner .....	51
Problème VIII.3 : Stations Météo .....	51
Problème VIII.4 : Jeu de cartes .....	52
Problème VIII.5 : Automate cellulaire 1D .....	54
Problème VIII.6 : Matrices $2 \times 2$ de nombres complexes .....	55
<b>Séance IX – Compilation multi-fichiers et utilisation d' un IDE .....</b>	<b>56</b>
IX.1 Compilation multi-fichiers : Utilité et Difficultés .....	56
IX.2 Compilation multi-fichiers : La bonne méthode .....	62

IX.3 Utilisation d'un IDE – installation et prise en main	64
IX.4 Utilisation d'un IDE – projet multi-fichiers	66
IX.5 Utilisation d'un IDE – autres bénéfices	68
IX.6 Utilisation d'un IDE – le Debugger	68
Problème IX.1 : Code à ranger sous Eclipse .....	70
 <b>Pont entre C "modernisé" et C Standard</b>	
<b>Séance X – Pont entre C "modernisé" et C Standard .....</b>	<b>74</b>
Problème X.1 : Un petit Quizz ... .....	74
Problème X.2 : Tableaux statiques et dynamiques .....	75
Problème X.3 : Chaines de caractères et affichage .....	75
Problème X.4 : Carnet d'Adresses en C standard .....	75
X.1 Pour vous entraîner	76
Problème X.5 : Tableaux à 2 dimensions .....	76
 <b>Annexe A – Bibliothèques : projet "Jeu Video"</b> .....	
A.1 Présentation de Raylib	78
Problème I.1 : Prise en main de Raylib .....	79
A.2 Dessiner le personnage animé	81
Problème I.2 : Animer le Viking .....	81
A.3 Dessiner une carte pour le fond	82
Problème I.3 : La carte .....	82
A.4 Contrôler le personnage	83
Problème I.4 : Touches au clavier .....	83
A.5 Vers un jeu video	84
 <b>Annexe B – Exécuter un code C à partir de Python</b> .....	
B.1 Introduction	85
B.2 Le code que l'on a envie d'écrire en Python	85
B.3 Créer la fonction C/C++	85
B.4 Créer la bibliothèque C/C++	86
B.5 Importer la bibliothèque dans Python	87
B.6 Utiliser la fonction C dans Python	88
B.7 Une écriture plus élégante	89
 <b>Annexe C – Gestion des fichiers</b> .....	
Problème III.1 : Fichier texte ou binaire .....	91
Problème III.2 : Carnet d'adresses "version Fichiers"	92
C.1 Pour vous entraîner	93
Problème III.3 : Analyse d'un fichier WAV .....	93
Problème III.4 : Jeu video avec plusieurs niveaux .....	94

## Préface

On propose ici un ensemble de solutions pour vous permettre de pratiquer le C/C++ chez vous. En effet, l'informatique, c'est un peu de réflexion et beaucoup de pratique. Pour s'entraîner il faut pratiquer sur ordinateur, c'est indispensable. Nous proposons 3 possibilités pour travailler chez vous :

1. Une première possibilité 100% en ligne, avec un simple navigateur web, via <http://repl.it>. C'est une bonne solution si on ne veut pas configurer et installer des logiciels. Mais elle s'éloigne un peu des conditions de travail à l'université (même si le langage C/C++ est exactement identique, bien entendu).
2. Une deuxième possibilité consiste à vous connecter sur les ordinateurs de l'université en utilisant **x2go**. C'est une bonne solution pour être chez soi dans les conditions exactes de travail de l'université, qui correspondent donc aussi aux conditions des contrôles et examens. Mais cela suppose une connexion internet de qualité, et malgré cela vous pouvez remarquer des ralentissements certains jours si les serveurs de l'université sont très sollicités. Et certains TP (ceux qui utilisent la bibliothèque "jeu video" notamment) risquent de ne pas pouvoir se faire par ce moyen.
3. Une troisième solution est dédiée aux ordinateurs sous Windows, elle consiste à installer un ensemble d'outils sur votre ordinateur. C'est une bonne solution qui vous plonge dans un environnement de travail vraiment très proche de l'université, et ne dépend plus d'internet une fois l'installation réalisée. Elle est déclinable sous Linux et Mac, mais n'est pas décrite pour ces machines. Prenez contact avec vos enseignants si vous avez besoin d'aide avec ces plateformes.

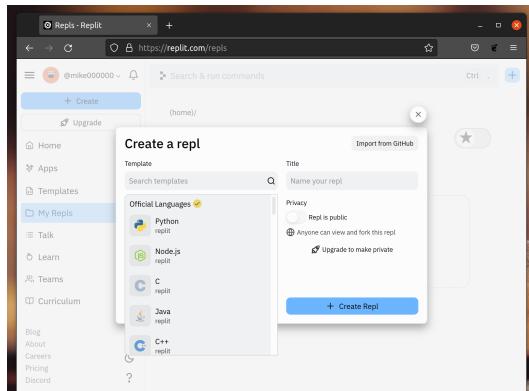
### Solution 1 - <http://repl.it> :

C'est un site web qui donne accès en ligne à beaucoup de langages informatiques, notamment le langage du terminal Linux (**bash**), le langage C/C++, mais aussi beaucoup d'autres.

**Note importante :** Même si le site peut s'utiliser pour programmer en C/C++ sans créer de compte, cette utilisation est limitée et ne donne pas accès à tout. Ainsi, il est **indispensable** de créer un compte sur ce site. Pour créer ce compte, allez simplement sur <http://repl.it> :



Puis allez sur **sign in**, indiquez vos coordonnées et validez le compte en surveillant vos emails. Une fois le compte créé, connectez-vous en retournant sur <http://repl.it>. Vous aboutissez à la fenêtre ci-dessous :



Pour ouvrir le dialogue "Create a repl", on a cliqué sur "+ Create" que vous voyez en arrière-plan, en haut à gauche de la fenêtre. Maintenant, il vous reste à sélectionner "C/C++" et à donner un nom au programme que l'on va créer, puis valider. Et c'est tout. Vous aurez à répéter cette procédure pour chaque nouveau programme.

## Solution 2 - X2GO

X2GO est un logiciel qui vous permet de vous connecter aux ordinateurs de l'université de façon sécurisée. Ce faisant, il vous plonge exactement dans l'environnement que vous utilisez en travaux pratiques, qui correspond donc aussi à celui qui sera utilisé pour les contrôles et examens. C'est un très bon outil de travail global pour vous à l'université, le seul défaut de cette solution est qu'elle suppose une bonne connexion internet. Aussi, les serveurs de l'université ne doivent pas être trop sollicités au moment où vous vous connectez : il arrive qu'ils soient beaucoup, pas forcément souvent, mais cela tombe forcément mal.

La procédure d'installation est disponible pour Windows, Linux et Apple/MacOsX sur le Moodle de l'université :

<https://moodle.umontpellier.fr/mod/page/view.php?id=126345>

Notez que l'article porte bizarrement le nom "Ubuntu 20.04" mais si vous descendez dans la page vous verrez la section **Accès à l'environnement graphique via le client X2go** qui contient toute la partie liée à Windows et Apple/MacOsX également. Une fois l'installation terminée, vous devriez aboutir à votre bureau sur les machines de l'université :



## Solution 3 - Installation de chocolatey, MSYS2 et Eclipse

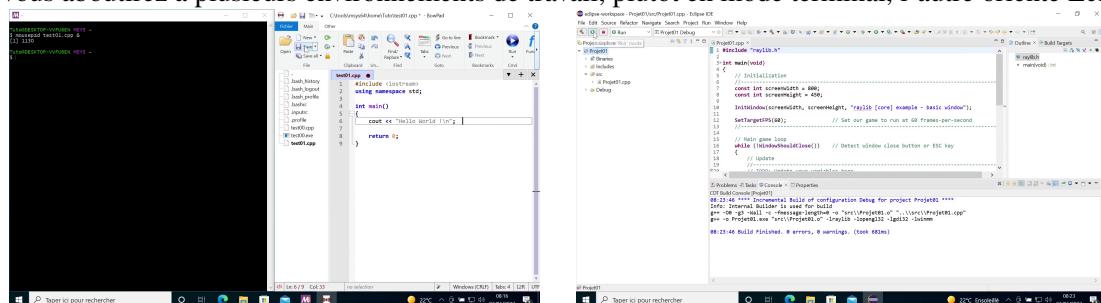
La procédure ci-dessous est dédiée à Windows. Nous avons identifié un ensemble d'outils qui vous permettent malgré tout de travailler chez vous avec des outils adaptés, de façon très similaire à ce que l'on fait sous Linux à l'université. Il y a plusieurs étapes, elles sont toutes décrites dans le tutoriel vidéo :

<http://dl.eea-fds.umontpellier.fr/CppInstall/tuto-install-Cpp.mp4>

Pour suivre ce tutoriel vous aurez besoin d'un package à télécharger :

<http://dl.eea-fds.umontpellier.fr/CppInstall/package.zip>

Vous aboutirez à plusieurs environnements de travail, plutôt en mode terminal, l'autre orienté Eclipse :



Une fois l'installation réussie, suivez ce tutoriel pour utiliser le terminal, C/C++, un éditeur (Bowpad), un IDE (Eclipse) et la bibliothèque Raylib, c'est-à-dire tous les outils nécessaires pour vous entraîner chez

vous avec les outils que nous utiliserons lors de ces TP :

<http://dl.eea-fds.umontpellier.fr/CppInstall/tuto-utilisation-Cpp.mp4>

Enfin, un tutoriel spécifique pour utiliser Raylib sous Eclipse :

<http://dl.eea-fds.umontpellier.fr/CppInstall/tuto-projet-raylib.mp4>

Pour finir : si jamais l'installation se passait mal, il est préférable de tout désinstaller et recommencer plutot qu'essayer de "bricoler". Nous donnons ci-dessous un tutoriel pour tout désinstaller proprement :

<http://dl.eea-fds.umontpellier.fr/CppInstall/tuto-supprimer-Cpp.mp4>

## PARTIE I

---

### **Les bases du Langage C/C++**

# Séance I – Environnement de travail • Traduction Python vers C/C++

## ① Motivations et objectifs

Dans cette séance on va s'intéresser à 2 choses :

- L'environnement de travail, autrement dit : comment, en pratique, j'utilise un compilateur C++ pour transformer un code en programme,
- Ensuite on va prendre en main le langage en "traduisant" des programmes Python en C/C++, pour se faire un peu la main.

## I.1 Environnement de travail : chaîne de compilation

On part d'un code C/C++ simple qui affiche un texte :

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

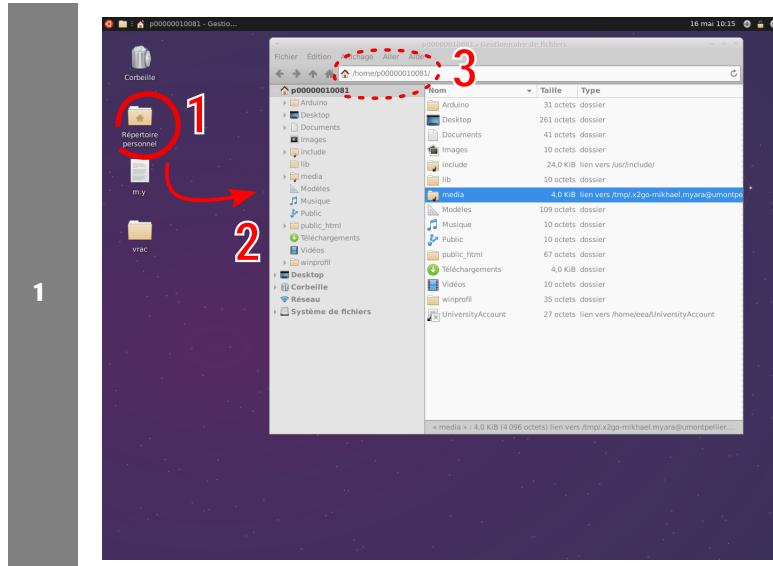
int main()
{
    cout << "Bonjour !\n";

    return 0;
}
```

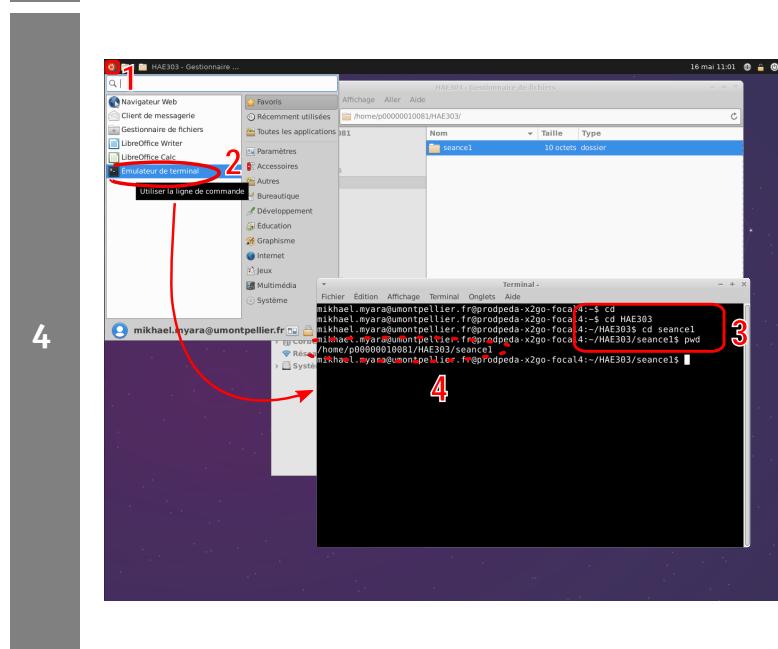
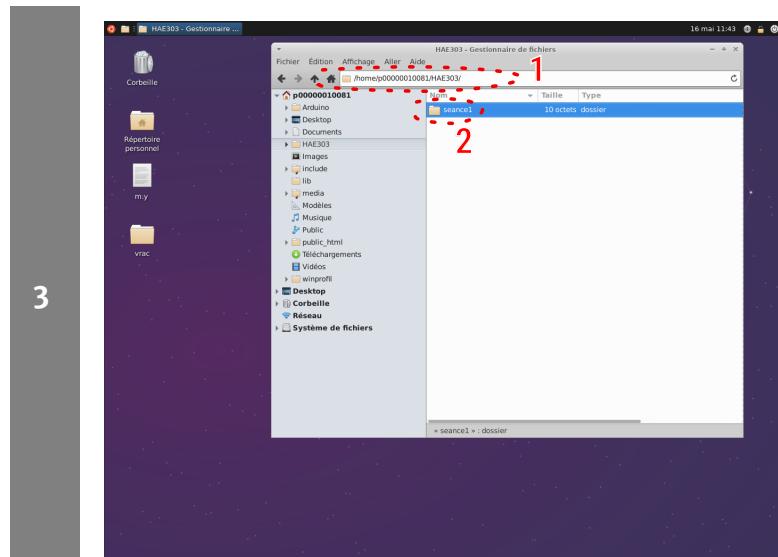
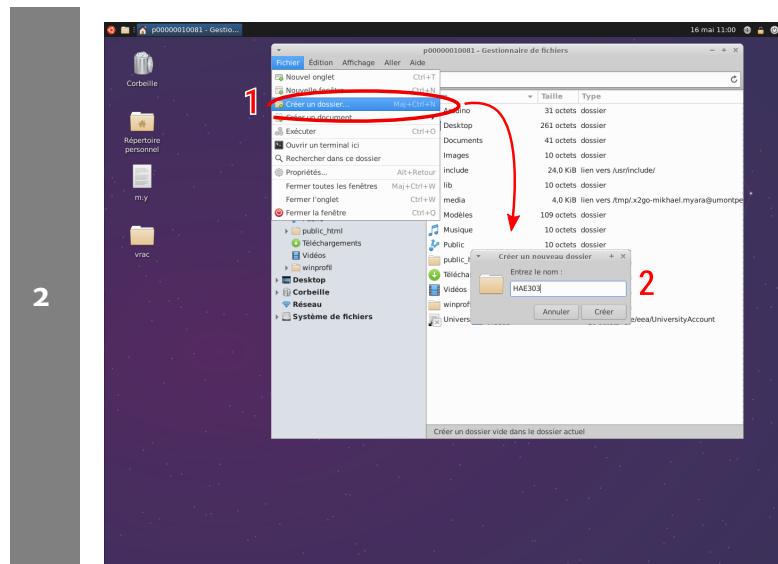
On va suivre un ensemble d'étapes pour aboutir à un programme exécutable :

- un éditeur de texte : ce sera **gedit**
- un compilateur C/C++ : ce sera le compilateur standard **g++**

### Préparation



Avec le système de fichiers, on part de votre répertoire de base "maison" (1), qui a un nom de la forme `/home/e001268759` (3), que l'on représentera par l'icône



Créer un sous-répertoire dans votre espace pour le module (1 et 2). Sur l'exemple son nom est HAE303.

Puis rentrez dedans (1). Vous êtes donc dans le chemin `~/HAE303/`. Répétez le même type d'opérations qu'à l'étape précédente pour créer un sous-répertoire `seance1`. Vous devez le voir comme indiqué en 2.

Maintenant que l'espace de travail est prêt on va se placer dedans avec le terminal. Lancez alors le terminal (1 et 2) puis tapez successivement (3) :

```
cd
cd HAE303
cd seance1
pwd
```

Le terminal est alors placé à l'endroit préparé avec le système de fichiers : la dernière commande a du vous afficher quelque chose de la forme :

```
/home/e001268759/HAE303/
seance1
```

Si vous n'obtenez pas quelque chose qui ressemble à cela demandez à votre enseignant, ce n'est pas normal.

## Ecriture du code et compilation

Ensuite on va vraiment pouvoir commencer à travailler :

1. Lancez l'éditeur **gedit**, pour préparer un premier exemple. Pour cela tapez dans le terminal :

```
gedit exo1.cpp &
```

2. Téléchargez ci-dessous le petit code et copiez/collez le dans **gedit** :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Bonjour !\n";
7
8     return 0;
9 }
```

 [codeTP/codeTP-I-1-2.c]

Puis sauvez le fichier.

3. Vérifiez dans le terminal que le fichier **exo1** est bien présent. Vous pouvez regarder dans le système de fichiers, ou taper dans le terminal (c'est plus direct) :

```
ls
```

qui doit vous afficher **exo1.cpp**. Si ce n'est pas le cas demandez à votre enseignant, ce n'est pas normal.

4. Lancez la compilation. La syntaxe est :

```
g++ exo1.cpp -o exo1
```

où **exo1.cpp** est le nom du fichier C/C++ à compiler, et **exo1** est le nom du programme exécutable qui a été généré par le compilateur. Si tout s'est bien passé, le compilateur n'a simplement rien affiché.

5. Vérifiez que l'exécutable a bien été créé :

```
ls
```

doit vous répondre :

```
exo1 exo1.cpp
```

Vous voyez bien que **exo1** a été créé.

## Exécution du programme

On peut maintenant exécuter le programme Dans le terminal, tapez :

```
./exo1
```

qui doit vous afficher **Bonjour**, c'est-à-dire le résultat du programme.



## Important

Il faut bien se souvenir de l'ensemble de cette procédure, nous allons l'utiliser à chaque fois que nous écrivons un programme et que nous devons le tester.

## Modification du programme

On va faire ça souvent parce que dans l'écriture d'un programme, il y a toujours des phases de tâtonnement, d'essais-erreurs, de corrections, de tests, retests, etc. Par exemple on veut ici faire une modification simple : changer le texte que l'on affiche. Ainsi :

1. Remplacez donc "Bonjour !" par "Bonjour à tous !" dans **gedit**.
2. Pensez à sauver, sinon le compilateur ne verra toujours que la version précédente du code !
3. Dans le terminal, tapez à nouveau :

```
g++ exo1.cpp -o exo1
```

4. Puis lancez à nouveau le programme :

```
./exo1
```

Et regardez ce qui a changé.

### Erreur de compilation

Faisons exprès d'introduire une erreur dans le programme. Par exemple, supprimons le ; à la fin de la ligne 6, puis sauvez et relancez la compilation. Vous obtenez :

```
exo1.cpp: In function 'int main()':
exo1.cpp:6:29: error: expected ';' before 'return'
  6 |   cout << "Bonjour à tous !\n"
    |           ^
    |           ;
  7 |
  8 |   return 0;
  | ~~~~~~
```

Le compilateur vous affiche donc une représentation d'une erreur qu'il perçoit : il a bien détecté qu'il manque un ; à la fin de la ligne. Ça se passe de cette façon, c'est à dire assez simplement, dans le cas des erreurs basiques. Nous aurons l'occasion de discuter de cas plus complexes où il est plus difficile d'interpréter ce que dit le compilateur.



### Important

Quand le compilateur produit une erreur, il ne recrée pas un nouvel exécutable ! Ainsi, l'exécutable qui est disponible est celui qui a été créé par la dernière version du code qui a été compilée sans erreur ! ■

## I.2 Traduction Python vers C/C++

On va donner quelques petits codes Python que vous avez à traduire en C/C++.



### Problème I.1 : Aire d'un Triangle

On donne le code Python ci-dessous :

```
base = 3.5
hauteur = 8.2

aireTriangle = base*hauteur/2

print("L'aire d'un triangle de base {} et de hauteur {} vaut {}".format(
    base, hauteur, aireTriangle))
🔗 [codeTP/codeTP-I-2-1.py]
```

- A. Téléchargez le, sauvegardez le dans /HAE303/seance1 en tant que "triangle.py", puis exécutez le en lançant :

```
python3 triangle.py
```

- B. Réécrivez le en langage C/C++, appelez le fichier "triangle.cpp" puis compilez le et exécutez le.

Comparez les résultats.

*Note :* Une fois le fichier cpp créé et rempli, la compilation peut se faire en tapant :

```
g++ triangle.cpp -o triangle
```

Et alors, si la compilation a fonctionné, l'exécution se fait ainsi :

```
./triangle
```

*Remarque :* Les résultats affichés donnent l'impression que l'erreur d'arrondi est différente en C/C++ et en Python. Il n'en n'est en réalité rien : c'est purement un effet de l'affichage. ■

## ② Problème I.2 : Produit Scalaire

On donne le code Python ci-dessous :

```
from math import cos,pi

AB = 5.2
AC = 3.5
A = 60

PS = AB*AC*cos(A/180*pi)

print("Le produit scalaire de deux vecteurs AB et AC de normes {} et {}
      et tels que l'angle BAC vaut {} degres est {}".format(AB,AC,A,PS))
\[codeTP/codeTP-I-2-2.py\]
```

- A. Téléchargez le, sauvegardez le dans /HAE303/seance1 en tant que "ps.py", puis exécutez le.
- B. Réécrivez le en langage C/C++, appelez le fichier "ps.cpp" puis compilez le et exécutez le. Comparez les résultats.

*Note :* Vous trouverez la valeur de  $\pi$  dans la bibliothèque mathématique, il se nomme **M\_PI**. Vous pourrez inclure la bibliothèque mathématique avec :

```
#include<cmath>
\[codeTP/codeTP-I-2-3.c\]
```

## ③ Problème I.3 : Suite

On donne le code Python ci-dessous :

```
U0=0.01
n=200

k=0
U=U0

while k<n:
    print("U{} = {}".format(k,U))
    U = U**0.5 + U
    k=k+1
\[codeTP/codeTP-I-2-3.py\]
```

- A. Téléchargez le, sauvegardez le dans /HAE303/seance1 en tant que "suite.py", puis exécutez le.
- B. Réécrivez le en langage C/C++, appelez le fichier "suite.cpp" puis compilez le et exécutez le. Comparez les résultats. Vous conserverez un **while** en C/C++ dans votre adaptation.

*Note :* Vous trouverez la fonction puissance dans la bibliothèque mathématique, elle se nomme **pow**. Par

exemple : `pow(5,2)` calcule  $5^2$ .

- C. Modifiez votre code C/C++ pour utiliser une boucle `for` au lieu du `while`

### ② Problème I.4 : Autre suite

On donne le code Python ci-dessous :

```
U0=27
n=100

k=0
U=U0
while k<n:
    print("{} : {}".format(k,U))
    if U%2 == 0:
        U = U//2
    else:
        U = 3*U+1
    k=k+1
```

 [codeTP/codeTP-I-2-4.py]

- A. Téléchargez le, sauvegardez le dans /HAE303/seance1 en tant que "suite.py", puis exécutez le.
- B. Réécrivez le en langage C/C++, appelez le fichier "suite.cpp" puis compilez le et exécutez le. Comparez les résultats. Vous conserverez un `while` en C/C++ dans votre adaptation.
- C. Modifiez vos codes Python et C/C++ pour n'afficher que le dernier terme
- D. Modifiez les programmes Python et C/C++ précédents pour avoir  $n = 100\,000\,000$
- E. Comparez les temps de calcul en tapant :

```
time python3 suite.py
time ./suite
```

Quelle différence et quel rapport de temps trouvez-vous ?

- F. On va noter  $n_0 = 100\,000\,000$ , c'est à dire la dernière valeur que l'on a testé. En changeant la valeur de  $n$  dans le code, remplissez le tableau suivant :

valeur de $n$	$n_0/10$	$n_0/2$	$n_0$	$n_0*2$	$n_0*5$
temps de calcul en Python					
temps de calcul en C/C++					

Tracez dans un script `matplotlib` (en Python donc) les courbes qui donnent les temps en fonction de  $n$ , pour chacun des 2 langages. Est-ce que les tendances sont similaires ? est-ce normal ?

- G. Tracez maintenant la différence et le rapport des temps entre les deux langages en fonction de  $n$ . Que constatez-vous ?

## I.3 Pour vous entraîner

### ② Problème I.5 : Nombres premiers

On donne le code Python ci-dessous :

```
kmax = 1000
```

```

k=0
while k <= kmax :
    n=2
    d=0
    while n < k :
        if k % n == 0:
            d = d+1
        n=n+1
    if d==0:
        print(n)
    k = k+1

```

 [codeTP/codeTP-I-3-5.py]

- A. Téléchargez le, sauvegardez le dans /HAE303/seance1 en tant que "premiers.py", puis exécutez le.
- B. Réécrivez le en langage C/C++, appelez le fichier "premiers.cpp" puis compilez le et exécutez le. Comparez les résultats.
- C. Modifiez vos codes Python et C/C++ pour n'afficher que le dernier terme
- D. Comparez les temps de calcul en tapant :

```

time python3 premiers.py
time ./premiers

```

Quelle différence et quel rapport de temps trouvez-vous ?

- E. On va noter **k0** = 1000, c'est à dire la dernière valeur que l'on a testé. En changeant la valeur de **kmax** dans le code, remplissez le tableau suivant :

valeur de <b>kmax</b>	<b>k0</b>	<b>k0*2</b>	<b>k0*5</b>	<b>k0*10</b>	<b>k0*20</b>	<b>k0*30</b>
temps de calcul en Python						
temps de calcul en C/C++						

Tracez dans un script **matplotlib** (en Python donc) les courbes qui donnent les temps en fonction de **n**, pour chacun des 2 langages. Est-ce que les tendances sont similaires ? est-ce normal ?

- F. Tracez maintenant la différence et le rapport des temps entre les deux langages en fonction de **n**. Que constatez-vous ?

- G. Comparons maintenant les tendances vues dans **matplotlib** sur les temps entre cet exercice et l'exercice I.4 : on constate qu'elles sont assez différentes. Pouvez-vous expliquer pourquoi en comparant les codes des deux exercices ? ■

# Séance II – Variables, Tableaux, Chaines de caractères (1)

## ⌚ Motivations et objectifs

Ce TP concerne les chapitres II, III et IV du cours

## II.1 Variables

### (?) Problème II.1 : Limites des nombres

- A. Dans un programme C/C++, déclarez 2 variables :

```
1 | short s;  
2 | char c;
```

📎🌐 [codeTP/codeTP-II-1-4.c]

Initialisez-les à 1. Puis multipliez les variables par 3 dans une boucle, de façon à réaliser l'opération 25 fois. A chaque "tour" de la boucle, vous afficherez les valeurs de **c** et **s**.

Note : On rappelle que **cout** n'affiche pas les **char** en tant que valeurs par défaut, et qu'il faut réaliser une conversion en **int** par exemple pour en voir la valeur. Cela vous amène à utiliser l'expression : **(int)c**

- B. Expliquez ce que vous observez.

- C. Recommencez le même exercice en vous basant sur 2 variables différentes :

```
1 | unsigned short s1;  
2 | short s2;
```

📎🌐 [codeTP/codeTP-II-1-5.c]

- D. Recommencez le même exercice en vous basant sur 2 variables différentes :

```
1 | long l;  
2 | short s;
```

📎🌐 [codeTP/codeTP-II-1-6.c]

### (?) Problème II.2 : Jours, Heures, Minutes, Secondes

On veut décomposer un temps, donné en secondes, en jours, heures minutes et secondes. En utilisant des divisions successives et les restes successifs, écrivez un tel programme en C/C++, et essayez le sur différentes valeurs.

## II.2 Tableaux

### (?) Problème II.3 : Notes

On donne le petit code ci-dessous :

```
1 | vector<float> tab =  
| {12.5, 8, 15.25, 19.5, 2, 0.5, 13.5, 15, 12, 9, 7, 18.5, 17.5, 5, 3, 13, 12, 11, 18.5, 12.3};  
📎 [codeTP/codeTP-II-2-7.c]
```

Il crée simplement un tableau contenant quelques valeurs, ces valeurs étant ici des notes d'examen.

A. En utilisant ce tableau, écrivez un programme qui affiche les valeurs qu'il contient,

B. Calculez la moyenne et affichez la.

C. Complétez le programme pour qu'il calcule le pourcentage de notes supérieures ou égales à 10. ■

## II.3 Chaines de caractères

### ② Problème II.4 : Palindrome

A. Ecrivez un programme qui demande la saisie d'un mot au clavier, puis qui détermine si le mot est un palindrome. Par exemple, BOB et RADAR sont des palindromes. Par simplicité on supposera que le mot est tapé tout en majuscules ou tout en minuscules. ■

## II.4 Pour vous entraîner

### ② Problème II.5 : Renversant

On donne le petit code ci-dessous :

```
1 | string s1 = "Bonjour a tous !";  
📎 [codeTP/codeTP-II-4-8.c]
```

A. Ecrivez un programme qui renverse la chaîne de caractères dans une nouvelle chaîne de caractères **s2**. Cela signifie que vous devez obtenir :

"! sout a rounjnoB"

Note : Si vous utilisez un texte à vous, ne mettez pas de caractères accentués. C'est lié au codage "UTF-8", qui utilise, pour certains caractères, plus que 1 octet, et les caractères accentués en font partie.

B. Ecrivez un programme qui renverse la chaîne de caractères directement dans **s1**. ■

## Séance III – Variables, Tableaux, Chaines de caractères (2)

### ⌚ Motivations et objectifs

Ce TP concerne les chapitres II, III et IV du cours

### III.1 Variables

#### (?) Problème III.1 : Table ASCII

On veut afficher la table ASCII pour les caractères dont les codes sont entre **32** et **126** inclus, et on veut que cet affichage se fasse comme ci-dessous :

:	20	!	:	21	"	:	22	#	:	23	\$	:	24	%	:	25	&	:	26	
'	27	(	:	28	)	:	29	*	:	2a	+	:	2b	,	:	2c	-	:	2d	
.	2e	/	:	2f	0	:	30	1	:	31	2	:	32	3	:	33	4	:	34	
5	35	6	:	36	7	:	37	8	:	38	9	:	39	:	:	3a	;	:	3b	
<	3c	=	:	3d	>	:	3e	?	:	3f	@	:	40	A	:	41	B	:	42	
C	43		D	44		E	45		F	46		G	47		H	48		I	49	
J	4a		K	4b		L	4c		M	4d		N	4e		O	4f		P	50	
Q	51		R	52		S	53		T	54		U	55		V	56		W	57	
X	58		Y	59		Z	5a		[	5b		\	5c		]	5d		^	5e	
_	5f		'	60		a	61		b	62		c	63		d	64		e	65	
f	66		g	67		h	68		i	69		j	6a		k	6b		l	6c	
m	6d		n	6e		o	6f		p	70		q	71		r	72		s	73	
t	74		u	75		v	76		w	77		x	78		y	79		z	7a	
{	7b			7c		}	7d		~	7e										

C'est à dire sous la forme de 7 colonnes et avec les valeurs en hexadécimal.

**A.** Ecrivez un code pour cela.

Note : Pour afficher une valeur en hexadécimal, il faut associer la commande **hex** à **cout**. Par exemple :  
**cout << hex << 1357;**  
affichera **54d**.

### III.2 Tableaux

#### (?) Problème III.2 : Maximum, Minimum et tri d'un tableau

On donne le petit code ci-dessous :

```
1 |vector<int> tab = {5,8,1,6,2,12,15,9,3,20,11,17,19,7};  
2 | [codeTP/codeTP-III-2-9.c]
```

Il crée simplement un tableau contenant quelques valeurs.

**A.** En utilisant ce tableau, écrivez un programme qui affiche les valeurs qu'il contient,

**B.** Ecrivez maintenant un programme qui détermine le maximum de ce tableau. Il n'existe pas de fonction `max` comme en Python, vous devez programmer quelque chose (quelques lignes) pour y parvenir. Affichez la valeur trouvée pour vérifier.

**C.** Copiez/coller la partie du code qui sert à déterminer le maximum, et transformez la pour déterminer le minimum. Affichez maintenant le maximum et le minimum du tableau.

**D.** Modifiez le code ci-dessus pour déterminer aussi le numero de la case du tableau qui contient le minimum (c'est à dire la position du minimum dans le tableau). Affichez la également pour vérifier

**E.** A partir de là on peut programmer le tri du tableau par ordre croissant. Pour cela il faut quelques étapes :

1. Déterminez le maximum de `tab` et gardez le dans une variable `max_tab`.
2. Créez un nouveau tableau qui fait la même taille que `tab`.
3. Trouvez le minimum de `tab` ainsi que sa position.
4. Ecrivez la valeur du minimum dans le nouveau tableau dans la case `0`, puis remplacez le minimum dans `tab` par le maximum.
5. recommencez à l'étape 3 mais en plaçant le nouveau minimum trouvé à chaque étape aux successivement aux positions 1, 2, 3, etc.

Commencez par vous convaincre sur papier et sur un tableau de 4 cases que cet algorithme a l'air de fonctionner.

**F.** Ecrivez le en C/C++.

### ② Problème III.3 : Matrices

On propose de représenter des matrices carrées par des vector de vector, en utilisant la syntaxe ci-dessous vue en cours :

```
1 |vector<vector<float>> M1;
 [codeTP/codeTP-III-2-10.c]
```

**A.** Créez un entier `N` qui représente la taille de la matrice. Pour commencer, donnez lui la valeur 3.

**B.** Créez maintenant 3 matrices `M1`, `M2`, `M3`. Faites en sorte que les 3 matrices vaillent les valeurs ci-dessous :

$$M1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \quad M3 = \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix}$$

Vous devriez remplir assez facilement `M1` et `M2` en gérant correctement des boucles. Ensuite, affichez les 3 matrices (n'hésitez pas à copier/coller du code que vous avez déjà écrit en l'adaptant, au lieu de réécrire).

**C.** Complétez le programme pour que `M3` contienne le produit  $M1 \times M2$ . Attention :  $\times$  est l'opérateur "produit de matrices", qui n'est pas le produit "terme à terme". Si vous avez oublié la technique, retrouvez la sur internet. Ensuite, affichez le résultat.

**D.** Remplacez `N` par 4, puis 5, puis 6. Si vous devez retoucher à votre code à chaque fois pour que cela fonctionne, c'est que vous avez raté quelque chose : si c'est le cas, corrigez le pour qu'il continue de fonctionner si on change seulement la valeur de `N` à un seul endroit du code.

## III.3 Pour vous entraîner

### ② Problème III.4 : Cryptage

On donne le petit code ci-dessous :

```
1 |string mdp = "MOTDEPASSE";
 [codeTP/codeTP-III-3-11.c]
```

On veut réaliser un cryptage dans lequel on remplace une lettre par son symétrique dans l'alphabet. Ainsi, on va remplacer A par Z, B par Y, C par W, et ainsi de suite.

A. Ecrivez un programme qui réalise ce cryptage, et affichez le résultat.

B. Appliquez deux fois ce cryptage en cascade. Que constatez-vous ? Est-ce "normal" ? ■

## ② Problème III.5 : Trinomes

A. Ecrivez un programme qui calcule les solutions réelles d'une équation du second degré :

$$ax^2 + bx + c = 0$$

en discutant la formule :

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} \quad \text{avec} \quad \Delta = b^2 - 4ac.$$

Utilisez une variable d'aide D pour la valeur du discriminant et décidez à l'aide de D, si l'équation a une, deux ou aucune solution réelle. Utilisez des variables du type int pour A, B et C.

Considérez aussi les cas où l'utilisateur entre des valeurs nulles pour A ; pour A et B ; pour A, B et C. Affichez les résultats et les messages nécessaires sur l'écran. ■

## ② Problème III.6 : Resistance Equivalente

Ecrire un programme qui affiche la résistance équivalente à trois résistances R1, R2, R3 (type double),

— pour les résistances sont branchées en série :

$$R_{ser} = R_1 + R_2 + R_3$$

— pour les résistances sont branchées en parallèle :

$$R_{par} = \frac{R_1 R_2 R_3}{R_1 R_2 + R_1 R_3 + R_2 R_3}$$

## ② Problème III.7 : Triangle

Ecrire un programme qui calcule et affiche l'aire d'un triangle dont il faut entrer les longueurs des trois côtés au clavier. Utilisez la formule :

$$A = \sqrt{P(P - A)(P - B)(P - C)}$$

où A, B, C sont les longueurs des trois côtés et P le demi-périmètre du triangle. ■

## ② Problème III.8 : Distance

Ecrire un programme qui calcule et affiche la distance en type **double** entre deux points A et B du plan dont les coordonnées (XA, YA) et (XB, YB) sont entrées au clavier comme entiers. ■

## ② Problème III.9 : Tables de multiplication

**A.** Ecrivez un programme qui demande un entier entre 1 et 12 à l'utilisateur. Si la valeur n'est pas correcte le programme doit s'entêter à redemander, autant de fois que nécessaire. Ensuite, le programme devra afficher la table de multiplication de la valeur saisie.

**B.** Ecrivez un programme qui affiche un tableau global des tables de multiplication pour N variant de 1 à 10 :

X*Y	I	0	1	2	3	4	5	6	7	8	9	10
0	I	0	0	0	0	0	0	0	0	0	0	0
1	I	0	1	2	3	4	5	6	7	8	9	10
2	I	0	2	4	6	8	10	12	14	16	18	20
3	I	0	3	6	9	12	15	18	21	24	27	30
4	I	0	4	8	12	16	20	24	28	32	36	40
5	I	0	5	10	15	20	25	30	35	40	45	50
6	I	0	6	12	18	24	30	36	42	48	54	60
7	I	0	7	14	21	28	35	42	49	56	63	70
8	I	0	8	16	24	32	40	48	56	64	72	80
9	I	0	9	18	27	36	45	54	63	72	81	90
10	I	0	10	20	30	40	50	60	70	80	90	100

## ② Problème III.10 : Pyramide et Croix

On veut faire afficher à un programme des pyramides dessinées avec des caractères dans le terminal. L'idée est de donner au programme le nombre de lignes sur laquelle elle s'étendra, et de la générer ensuite. Par exemple, si on donne n=5, le programme doit générer :

```
*  
***  
*****  
*****  
*****
```

**A.** Ecrivez une fonction qui affiche *m* espaces. Justifiez son prototype.

**B.** Ecrivez une fonction qui affiche *p* étoiles. Justifiez son prototype.

**C.** Si on note *k* le numéro de la ligne :

- déterminez combien il faut afficher d'étoiles en fonction de *k*,
- déterminez combien il faut afficher de caractères "espace" avant d'afficher la première étoile de chaque ligne en fonction de la valeur de *k*.

A partir de ces remarques, écrivez une fonction qui affiche une ligne de la pyramide en fonction de la valeur de *k*. Cette fonction fera appel aux deux fonctions écrites à la question précédente.

**D.** Ecrivez enfin une fonction :

`void pyramide(int n);` qui affiche la pyramide complète.

**E.** Ecrivez un main qui propose à l'utilisateur de saisir la hauteur de la pyramide puis l'affiche.

**F.** Sur le même principe, écrivez des fonctions et un programme qui affiche un "V", puis un "X", comme suit (ici exemple pour 4 dans les deux cas).

Pour le V :

```
*      *  
*      *
```

\* \*  
\*

Pour le X :

\* \*  
\* \*  
\* \*  
\*  
\* \*  
\* \*  
\* \*

■

# Séance IV – Fonctions

## ⌚ Motivations et objectifs

Ce TP concerne le chapitre V du cours.

## (?) Problème IV.1 : Ranger du code

On donne le code (long) ci-dessous :

```
1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4
5 using namespace std;
6
7 int main() {
8     vector<vector<double>> M1, M2, M3;
9
10    // saisie au clavier de la taille de travail
11    int N;
12    cout << "Entrez la taille des matrices à traiter :\n";
13    cin >> N;
14
15    // Creation et saisie de M1
16    M1.resize(N);
17    int i, j;
18    for(i=0; i<N; i++) {
19        M1[i].resize(N);
20    }
21    cout << "Entrez les valeurs :\n";
22    for(i=0; i<N; i++) {
23        for(j=0; j<N; j++) {
24            cout << "M(" << i+1 << ", " << j+1 << ") = ";
25            cin >> M1[i][j];
26        }
27    }
28
29
30    // Creation et saisie de M2
31    M2.resize(N);
32    for(i=0; i<N; i++) {
33        M2[i].resize(N);
34    }
35    cout << "Entrez les valeurs :\n";
36    for(i=0; i<N; i++) {
37        for(j=0; j<N; j++) {
38            cout << "M(" << i+1 << ", " << j+1 << ") = ";
```

```
39     cin >> M2[i][j];
40 }
41 }
42
43
44 // Creation de M3
45 M3.resize(N);
46 for(i=0;i<N;i++) {
47     M3[i].resize(N);
48 }
49
50
51
52
53 // parametrage du style d'affichage des double
54 cout.precision(5);
55
56 // affichage de M1
57 cout << "\n";
58 for(i=0;i<N;i++) {
59     for(j=0;j<N;j++) {
60         cout << setw(4) << M1[i][j] << " ";
61     }
62     cout << "\n";
63 }
64
65
66 // affichage de M2
67 cout << "\n";
68 for(i=0;i<N;i++) {
69     for(j=0;j<N;j++) {
70         cout << setw(4) << M2[i][j] << " ";
71     }
72     cout << "\n";
73 }
74
75
76 // calcul du produit
77 int k;
78 double prod;
79
80 for(i=0;i<N;i++) {
81     for(j=0;j<N;j++) {
82         prod=0;
83         for(k=0;k<N;k++) {
84             prod = prod + M1[i][k]*M2[k][j];
85         }
86         M3[i][j]=prod;
87     }
88 }
89
90 // affichage de M3
91 cout << "\n";
92 for(i=0;i<N;i++) {
93     for(j=0;j<N;j++) {
94         cout << setw(4) << M3[i][j] << " ";
95     }
```

```

96     cout << "\n";
97 }
98
99 return 0;
100 }
```

 [codeTP/codeTP-IV-0-12.c]

**A.** Commencez par compiler et lancer ce code, et assurez vous sur le fait que vous faites la relation entre le code et ce que le programme exécute.

**B.** Maintenant observez ce code, et constatez qu'il y a de nombreux "copier/coller" adaptés à différents endroits, ce que l'on veut absolument éviter : c'est une mauvaise pratique de la programmation, autrement dit, même si ce code fonctionne, on peut dire qu'il n'est pas de bonne qualité.

**C.** Pour améliorer le code, commencez par créer une fonction dont le prototype est :

```
vector<vector<double> > creerMatrice(int N);
```

qui crée la matrice, au sens où elle prépare les tableaux pour qu'ils soient assez grands. Remplacez, dans le code, les lignes qui peuvent l'être, par un appel à cette nouvelle fonction.

**D.** Ensuite, commencez par créer une fonction dont le prototype est :

```
vector<vector<double> > saisieMatrice(int N);
```

qui réalise la création et la saisie de la matrice au clavier. Remplacez, dans le code, les lignes qui peuvent l'être, par un appel à cette nouvelle fonction.

*Note :* Ce n'est pas obligatoire mais il serait élégant de faire appel à la fonction **creerMatrice** dans **saisieMatrice** plutôt que de recopier le code.

**E.** Ecrivez maintenant la fonction :

```
void afficheMatrice(vector<vector<double> > M);
```

qui affiche une matrice, puis remplacez dans le code les lignes qui peuvent l'être par un appel à cette nouvelle fonction.

**F.** Ecrivez maintenant une fonction pour le produit de matrices. Déterminez par vous-même le prototype que devrait prendre la fonction produit. Remplacez alors les lignes qui servent au produit de matrices par un appel à cette fonction.

**G.** Vérifiez que tout fonctionne exactement comme le programme d'origine.

**H.** Normalement, vous avez créé 4 fonctions et rendu le **main** plus lisible. Le constatez-vous ? Regardez aussi combien de lignes fait l'ancien code et comparez le avec le nouveau. Vous constatez que le programme, bien qu'il produise exactement les mêmes résultats, est écrit avec moins de ligne et ne comporte pas de "copier/coller", ce qui en facilite la lecture et donc l'évolution : il est "mieux organisé" que le code d'origine. Ce n'est pas un détail, c'est un enjeu important de la programmation.

**I.** Modifiez le code pour avoir la fonction **main** le plus haut possible dans le fichier .cpp. Vous utiliserez pour cela les déclarations de prototypes des fonctions que vous avez créé. ■

## ?) Problème IV.2 : Utiliser des fonctions

On donne le code ci-dessous ; c'est le code qui fait le tri d'un tableau par ordre croissant, tel qu'on l'a établi à la séance précédente :

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5
6 int main()
7 {
8     vector<int> tab = {5, 8, 1, 6, 2, 12, 15, 9, 3, 20, 11, 17, 19, 7};
```

```
9   int k=0;
10  cout << "\n";
11  while(k<tab.size()) {
12      cout << tab[k] << " ";
13      k=k+1;
14  }
15  cout << "\n";
16
17  vector<int> tab2;
18  tab2.resize(tab.size());
19
20  int max ,posMax;
21  k=0;
22  max = tab[0];
23  while(k<tab.size()) {
24      if(tab[k]>max) {
25          max = tab[k];
26          posMax = k;
27      }
28      k=k+1;
29  }
30
31  int min ,posMin;
32  k=0;
33  posMin = 0;
34  min = tab[0];
35  while(k<tab.size()) {
36      if(tab[k]<min) {
37          min = tab[k];
38          posMin = k;
39      }
40      k=k+1;
41  }
42
43
44  int m;
45  k=0;
46  while(k<tab2.size()) {
47      min = tab[0];
48      posMin = 0;
49      m=0;
50      while(m<tab.size()) {
51          if(tab[m]<min) {
52              min = tab[m];
53              posMin = m;
54          }
55          m=m+1;
56      }
57      tab2[k] = min;
58      tab[posMin] = max;
59      k=k+1;
60  }
61
62  k=0;
63  cout << "\n";
64  while(k<tab2.size()) {
```

```

66         cout << tab2[k] << " ";
67         k=k+1;
68     }
69     cout << "\n";
70
71     return 0;
72 }
```

 [codeTP/codeTP-IV-0-13.c]

Ainsi que les fonctions ci-dessous :

```

1 // Fonction qui trouve la position du minimum dans un vector<int>
2 int positionMin(vector<int> T)
3 {
4     int k=0, pos=0;
5     while(k<T.size()) {
6         if(T[k]<T[pos]) {
7             pos = k;
8         }
9         k=k+1;
10    }
11    return pos;
12 }

13
14 // Fonction qui trouve la position du maximum dans un vector<int>
15 int positionMax(vector<int> T)
16 {
17     int k=0, pos=0;
18     while(k<T.size()) {
19         if(T[k]>T[pos]) {
20             pos = k;
21         }
22         k=k+1;
23     }
24     return pos;
25 }
```

 [codeTP/codeTP-IV-0-14.c]

**A.** Réécrivez la fonction `main` pour utiliser au maximum les 2 fonctions `positionMaximum` et `positionMinimum`

Remarque : Normalement, pour les fonctions données (sauf pour le `main`), vous n'avez pas spécifiquement à comprendre leur contenu : le prototype de la fonction ainsi que son nom et les commentaires autour devraient suffire à savoir comment l'utiliser (si c'est bien fait, et là c'est le cas). Bien entendu vous avez tout à fait le droit de regarder leur code malgré tout, si cela aide. Mais ça n'est pas la démarche de premier abord que l'on a en programmation.

**B.** Ecrivez une fonction qui affiche les valeurs du tableau, et utilisez la pour remplacer les sections du code qui affichent les valeurs.

**C.** Modifiez le code pour avoir la fonction `main` le plus haut possible dans le fichier `.cpp`. Vous utiliserez pour cela les déclarations de prototypes des fonctions que vous avez créé.

**D.** Ecrivez maintenant le tri sous forme d'une fonction. ■

## Pour vous entraîner

### (?) Problème IV.3 : Prototypes

**A.** Quel serait le prototype de la fonction qui affiche la table ASCII programmée au problème III.1 ? Si vous avez toujours le code, transformez le en fonction et écrivez une fonction **main** pour tester.

**B.** Quel serait le prototype d'une fonction qui réalise le cryptage d'une chaîne de caractères, comme déjà programmé en §III.4 ? Vous donnerez votre réponse dans deux cas :

- Si la chaîne à transformer est donnée en entrée et le résultat est affiché par la fonction,
- Si la chaîne à transformer est donnée en entrée et le résultat renvoyé par la fonction.

Vous programmerez les deux et écrirez un **main** à chaque fois pour tester.



## ② Problème IV.4 : Histogramme

On donne le code ci-dessous. Peu importe que vous en compreniez les détails, il remplit simplement le tableau **val** avec 10000 valeurs aléatoires selon une distribution qui suit la loi binomiale.

```
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <iostream>
#include <vector>
using namespace std;

float ranf(); /* ranf() is uniform in 0..1 */
float ranNorm(float m, float s); /* box-muller method */

int main()
{
    srand(time(NULL));

    int Nval;
    int i;
    Nval = 10000;
    vector<float> val;
    val.resize(Nval);
    for(i=0; i<Nval; i++)
    {
        val[i]=ranNorm(5,5);
    }

// A completer

    for(i=0; i<Nval; i++)
    {
        cout << val[i] << " ";
    }

    return 0;
}

float ranf() /* ranf() is uniform in 0..1 */
{
    return ((float)rand()) / RAND_MAX;
}
```

```

// moyenne m, ecart type s
// methode dite de box-muller
float ranNorm(float m, float s)
{

    float x1, x2, w, y1;
    static float y2;
    static int use_last = 0;

    if (use_last)
    {
        y1 = y2;
        use_last = 0;
    }
    else
    {
        do {
            x1 = 2.0 * ranf() - 1.0;
            x2 = 2.0 * ranf() - 1.0;
            w = x1 * x1 + x2 * x2;
        } while ( w >= 1.0 );

        w = sqrt( (-2.0 * log( w ) ) / w );
        y1 = x1 * w;
        y2 = x2 * w;
        use_last = 1;
    }
    return( m + y1 * s );
}

```

 [codeTP/codeTP-IV-0-15.c]

A partir de ces données, on veut calculer un histogramme.

**A.** Complétez ce programme pour qu'il calcule l'histogramme du tableau val sur  $N$  tranches,  $N$  étant choisi par l'utilisateur du programme. Cet histogramme sera calculé par une fonction dont le prototype sera :

```
vector<float> histogramme(vector<float> val, int nval, int N);
```

Vous reprendrez et adapterez les fonctions de calcul du maximum et minimum vues à l'exercice IV.2 pour déterminer l'étendue des valeurs.

**B.** Ajoutez maintenant une fonction qui représente l'histogramme obtenu en mode texte, horizontalement, comme représenté ci-dessous.

```
1 *
2 *
3 *
4 **
5 *****
6 *****
7 *****
8 *****
9 *****
10 *****
11 *****
12 *****
13 *****
14 *****
15 *****
16 *****
17 ****
18 **
19 *
20 *
```

# Séance V – Structures

## ① Motivations et objectifs

Ce TP concerne le chapitre VI du cours.

## ② Problème V.1 : vecteurs dans l'espace

On ne parle pas ici des "vector" du C/C++ mais de vecteurs au sens géométrique. On veut gérer des vecteurs à 3 coordonnées et réaliser un ensemble d'opérations avec.

- A. Proposer une structure qui permette de représenter un vecteur à 3 dimensions. Vous nommerez cette structure **vect**.
- B. Créez trois vecteurs **V1**, **V2** et **V3**, et attribuez à **V1** et **V2** les coordonnées non nulles que vous voudrez.
- C. Affichez le contenu de **V1** et **V2**
- D. Calculez le produit scalaire de **V1** et **V2** et affichez le résultat.
- E. Calculez le produit vectoriel de **V1** et **V2** dans **V3** puis affichez **V3** (si vous avez oublié ce que c'est regardez sur internet).

## ③ Problème V.2 : Matrices $2 \times 2$

On veut gérer des matrices  $2 \times 2$ . Ces matrices sont souvent appelées "matrices ABCD" parce qu'il y a 4 coefficients. Il est donc classique de dire qu'une matrice  $2 \times 2$  est simplement un ensemble de 4 valeurs.

Ainsi, on propose la structure ci-dessous pour représenter les matrices  $2 \times 2$  :

```
1 typedef struct {  
2     double A,B,C,D;  
3 } matrice22;
```

 [codeTP/codeTP-V-0-16.c]

- A. Ecrivez une fonction :

```
void afficheM22(matrice22 M);
```

qui affiche une matrice  $2 \times 2$ . Ecrivez un **main** minimaliste pour tester votre fonction.

- B. Ecrivez une fonction **saisieM22** (dont vous déciderez le prototype) qui propose la saisie au clavier avec **cin** d'une matrice  $2 \times 2$ . Ecrivez un **main** minimaliste pour tester votre fonction.
- C. Ecrivez une fonction **produitM22** qui calcule le produit de 2 matrices et **renvoie** le résultat. Ecrivez un **main** minimaliste pour tester votre fonction.

D. Créez dans le **main** un **vector** de 4 **matrice22**, que vous appellerez **tab**, et écrivez, toujours dans le **main** le code utile pour en faire la saisie au clavier.

E. Ecrivez maintenant une fonction **transfert** qui prend en paramètre un **vector** de **matrice22** et qui renvoie la **matrice22** qui est le produit de la totalité des matrices contenues dans le **vector**, dans l'ordre, de gauche à droite.

**F.** Utilisez cette fonction dans le **main** pour calculer le produit des matrices contenues dans la variable **tab** du **main**, puis affichez le résultat.

*Note :* Ce type de calcul existe dans un grand nombre de logiciels scientifiques dans des domaines très variés (optique, électronique, ...)

## ② Problème V.3 : Code Morse

On donne le code ci-dessous :

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4 using namespace std;
5
6 typedef struct
7 {
8     char lettre;
9     string morse;
10 }code;
11
12
13 int main()
14 {
15     vector<code> alphabet=
16     { {'A', ".-"}, {"B", "-..."}, {"C", "-.-."}, {"D", "-.."}, {"E", "."}, {"F", ".---"}, {"G", "--."}, {"H", "...."}, {"I", ". ."}, {"J", ".---"}, {"K", "-.-"}, {"L", ".-.."}, {"M", "--"}, {"N", "-."}, {"O", "---"}, {"P", ".--."}, {"Q", "--.-"}, {"R", ".-."}, {"S", "..."}, {"T", "-"}, {"U", ".--"}, {"V", "...-"}, {"W", ".--"}, {"X", "-..-"}, {"Y", "-.--"}, {"Z", "--.."}};
17
18
19     return 0;
20 }
```



Trois remarques :

- Ce code ne fait qu'initialiser un tableau de structures en y plaçant le code Morse de chaque lettre de l'alphabet.
- Dans la mesure où les lettres sont dans l'ordre alphabétique, le champ **lettre** de la structure ne sert fonctionnellement à peu près à rien. Il participe surtout à la lisibilité du code et au déboggage si besoin,
- Si on suppose que le code ASCII de la lettre dont on veut connaître le codage Morse s'appelle **maLettre**, il suffit de taper :

```
|alphabet[maLettre - 'A'].morse
```



En effet, cette expression retranche à "**maLettre**" le code ASCII du '**A**', ce qui revient à obtenir une valeur entre 0 et 26, soit l'indice de la bonne case dans le tableau **alphabet**.

- A.** Affichez le tableau de code Morse sous la forme :

```

A  .-
B  -...
C  -..
D  -..
E  .
F  ...-
G  --.

```

etc.

**B.** Ecrivez une fonction **strToMorse** qui renvoie une chaîne de caractères contenant le code Morse correspondant à une chaîne de caractères donnée en paramètre. Dans cette nouvelle chaîne, chaque code Morse sera séparé du précédent par un espace. Son prototype sera :

```
|string strToMorse(string s, vector<code> alpha);
```


[\[codeTP/codeTP-V-0-19.c\]](#)

Vous ferez en sorte que si un caractère est en minuscule dans la chaîne fournie, alors votre fonction ira chercher la lettre majuscule correspondante. Pour cela, vous avez la fonction **toupper** à votre disposition, qui convertit une lettre en sa correspondance en majuscule.

**C.** Créez un **main** qui :

- propose la saisie au clavier d'un mot,
- transforme en code Morse,
- Affiche la chaîne correspondant en code Morse.

## V.1 Pour vous entraîner

### (?) Problème V.4 : Carnet d'Adresses

On souhaite gérer un carnet d'adresses de 10 contacts au total dans lequel les coordonnées des différents contacts seront stockées. Pour chaque contact, on souhaite stocker :

- le Nom, sous forme d'une **string**
- le Prénom, sous forme d'une **string**
- l'âge, sous forme d'un **long**.

Pour celà, on propose les étapes suivantes :

- A.** Ecrivez une fonction permettant la saisie d'un contact et une permettant l'affichage d'un contact.
- B.** Ecrivez une fonction permettant d'ajouter un contact à la liste des contacts.
- C.** Ecrivez une fonction permettant de supprimer un contact à la liste des contacts.
- D.** Faites une gestion complète avec un menu permettant d'ajouter/supprimer un contact, et d'afficher la liste complète des contacts.

### (?) Problème V.5 : Tableau de Mendeleev

On donne un code qui crée une partie du tableau de Mendeleev :

```

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

typedef struct {
    string nom, symbole;
    int numero;
}

```

```
float masse;
}atome;

int main()
{

vector<atome> mendeleev = { {"Hydrogene", "H", 1, 1.008}, {"Helium",
    "He", 2, 4.0026}, {"Lithium", "Li", 3, 6.94},
    {"Beryllium", "Be", 4, 9.0122}, {"Bore", "B", 5, 10.81}, {"Carbone",
        "C", 6, 12.011}, {"Azote", "N", 7, 14.007},
    {"Oxygene", "O", 8, 15.999}, {"Fluor", "F", 9, 18.998}, {"Neon",
        "Ne", 10, 20.180},
    {"Sodium", "Na", 11, 22.990}, {"Magnesium", "Mg", 12, 24.305}, {"Aluminium",
        "Al", 13, 26.982},
    {"Silicium", "Si", 14, 28.085}, {"Phosphore", "P", 15, 30.974}, {"Soufre",
        "S", 16, 32.06},
    {"Chlore", "Cl", 17, 35.45}, {"Argon", "Ar", 18, 39.948}, {"Potassium",
        "K", 19, 39.098},
    {"Calcium", "Ca", 20, 40.078} };

return 0;
}
```

 [codeTP/codeTP-V-1-20.c]

- A. Ecrivez un code qui affiche l'ensemble des données, avec un atome par ligne.
- B. Ecrivez une fonction qui trouve un atome à partir de son symbole. Par exemple, si on donne "C" à cette fonction, elle doit renvoyer la structure de l'atome de carbone. Son prototype sera :

```
atome trouver_atome(string symb);
```

- C. On veut calculer le poids de molécules, en faisant la somme des masses atomiques. On va décrire une molécule sous la forme d'une chaîne de caractères qui contient l'ensemble des atomes qui la forment. Par exemple, le Méthane CH<sub>4</sub> serait représenté par une chaîne contenant CHHHH.

Ecrivez une fonction :

```
double calcul_masse(string molecule);
```



## **PARTIE II**

---

### **Les techniques du Langage C/C++**

# Séance VI – Exploration de la mémoire et pointeurs

## ⌚ Motivations et objectifs

Ce TP concerne les chapitres VIII et IX du cours.

## (?) Problème VI.1 : Pointeur sur une variable 1

On donne le petit code suivant :

```
1 int i = 12345;  
2 int* p;  
🌐 [codeTP/codeTP-VI-0-21.c]
```

- A. Modifiez ce petit code pour que la valeur de **i** devienne **3869** mais en utilisant uniquement **p**, jamais **i**.  
B. Affichez alors **i**.

## (?) Problème VI.2 : Pointeur sur une variable 2

On donne le petit code suivant :

```
1 unsigned long a = 0x11BB55DD99FF33AA;  
2 unsigned char* pc = (unsigned char*) &a;  
3 unsigned short* ps = (unsigned short*) &a;  
🌐 [codeTP/codeTP-VI-0-22.c]
```

On a tout mis en **unsigned** pour ne pas avoir à se poser de questions sur le codage des nombres négatifs, c'est plus simple.

A. Dessinez sur un papier ce que cette situation signifie intuitivement. Vous représenterez la RAM sous forme d'un tableau d'octets, dans lequel vous placerez la variable **a** décomposée octet par octet, et les pointeurs **pc** et **ps** sous forme de flèches.

B. Affichez les adresses contenues dans **pc** et **ps**. Qu'observez-vous ? Est-ce logique ?

Note : Attention au pointeur **pc**, vous devez le convertir en **void\*** au moment de l'affichage, parce que **cout** fait une interprétation particulière des pointeurs **char\***, ce qu'il faut contourner.

C. Affichez maintenant la valeur pointée par **pc** et celle pointée par **ps**. Pour simplifier, vous afficherez les valeurs en hexadécimal. On redonne un exemple pour cela :

**cout << hex << 1357;** affichera **54d**

Comprenez-vous ce qui est affiché à travers chacun des pointeurs ? Est-ce que cela dit que les nombres sont codés en big endian ou en little endian ? Corrigez, si besoin, le dessin que vous avez fait à la première question, suite à cette constatation.

Note : Vous constaterez sûrement que l'affichage des valeurs pointées par le pointeur **char\*** prend une forme inexploitable. Pour que l'affichage soit utilisable, il faut utiliser l'opérateur de conversion (**int**) quand vous regardez les valeurs sur lesquelles il pointe.

D. Complétez votre code en augmentant de 1 **pc** et **ps**, puis affichez les nouvelles adresses des pointeurs,

et à nouveau les valeurs pointées. Que constatez vous au niveau des nouvelles adresses ? Pourquoi les adresses ne sont plus identiques entre elles ? Les valeurs pointées sont-elles cohérentes avec les adresses affichées ?

**E.** Ajoutez maintenant :

```
*ps = 0x2244;
 [codeTP/codeTP-VI-0-23.c]
```

puis affichez la valeur de **a**. Est-ce cohérent ?

**F.** En faisant un petit dessin, expliquez pourquoi on peut augmenter **pc** 7 fois au maximum par pas de 1 ? A l'inverse, pourquoi ce serait une source d'erreur de le déplacer 4 fois **ps**, toujours par pas de 1. ■

## ② Problème VI.3 : Pointeur sur un vector

On donne programme suivant :

```
1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4 using namespace std;
5
6 int main()
7 {
8     int i;
9     vector<unsigned int> T;
10    T.resize(20);
11    unsigned int* pL;
12    unsigned short* pS;
13    unsigned char* pC;
14
15    pL=&T[0];
16    pS=(unsigned short*)&T[0];
17    pC=(unsigned char*)&T[0];
18
19
20
21    for(i=0;i<5;i++)
22    {
23        T[i]=253+i;
24    }
25
26    cout << "\nResultats :\n";
27    for(i=0;i<20;i++)
28    {
29        cout << pL + i << " " << pS + i << " " << (void*) (pC + i) << "\n";
30    }
31
32 }
```

```
 [codeTP/codeTP-VI-0-24.c]
```

**A.** Expliquez l'utilité de (**unsigned short\***) et (**unsigned char\***) lignes 16 et 17. Expliquez pourquoi on n'a pas de (**unsigned int\***) ligne 15.

**B.** Expliquez et commentez son fonctionnement à travers ce qui est affiché à l'écran.

**C.** On remplace les ligne 27 à 29 par :

```
for(i=0;i<20;i++)
{
    cout << setw(10)<< *(pL+i) << " " << setw(5)<< *(pS+i) << " " <<
        setw(3)<< (int) *(pC+i) << "\n";
```



Note : les `setw` servent juste à fixer la largeur d'affichage des nombres, pour avoir une représentation "type" colonne.

Expliquez ce qui est affiché.

■

## VI.1 Pour vous entraîner

### ② Problème VI.4 : Exercice de base sur les pointeurs

A. Soit P un pointeur qui "pointe" sur un "tableau" A :

```
vector<int> A = {12, 23, 34, 45, 56, 67, 78, 89, 90};  
int *P;  
P = &A[0];
```

Quelles valeurs ou adresses (relatives à A) fournissent ces expressions (certaines n'ont aucun sens) :

1.  $(*P)+2$
2.  $*(P+2)$
3.  $P+1$
4.  $\&A[4]-3$
5.  $A+3$
6.  $P+((*P)-10)$

**Vous cherchez d'abord, en réfléchissant et dessinant le tableau, quelles sont les réponses théoriques, puis vérifiez en testant dans un `main` que votre idée est juste.** Bien entendu, en ce qui concerne les réponses théoriques, les déductions que vous ferez en matière d'adresses seront relatives à l'adresse de A, alors que les résultats que vous obtiendrez sur machine, eux, seront fixés à des adresses absolues.

■

# Séance VII – Passage par adresse

## ⌚ Motivations et objectifs

Ce TP concerne le chapitre X du cours. Il utilise massivement le chapitre précédent et concerne une technique **extrêmement importante** à savoir utiliser en C/C++.

## (?) Problème VII.1 : Jours, heures, minutes, secondes – v2

On reprend l'exercice §II.2, dont on redonne le code :

```
1 int temps, j, h, m, s, r1, r2;
2 cout << "Donnez le temps à décomposer, en secondes : ";
3 cin >> temps;
4 j = temps / (24*60*60);
5 r1 = temps % (24*60*60);
6 h = r1 / (60*60);
7 r2 = r1 % (60*60);
8 m = r2 / 60;
9 s = r2 % 60;
10
11 cout << "\n" << temps << "secondes se convertissent en " << j << " jours,
   " << h << " heure, " << m << " minutes et " << s << " secondes\n";
```

🌐 [codeTP/codeTP-VII-0-26.c]

A. Commencez par compiler et essayer ce code.

B. Ecrivez une fonction appelée `jhms` qui remplace toute la partie calcul de ce code. Mais cette fonction n'affichera pas les résultats : elle prend en paramètre le nombre de secondes ("`temps`" dans le code ci-dessus) et aura pour sorties, en utilisant le passage par adresses, le nombre de jours, d'heures de minutes et de secondes.

C. Modifiez la fonction `main` pour qu'elle utilise maintenant cette fonction `jhms` pour réaliser le calcul.

D. Expliquez pourquoi une fonction `jhms2`, qui ferait le même travail et dont le prototype serait :

`void jhms2(int temps, int jours, int heures, int minutes, int secondes);`  
ne peut PAS fonctionner. Vous pouvez la programmer et observer ce qui se passe.

Note : on ne parle pas de problème de compilation, on suppose que le `main` serait corrigé pour permettre l'utilisation de cette fonction. On parle uniquement de ce qui se passe à l'exécution.

E. Avec votre fonction `jhms`, utilisez le main ci-dessous :

```
1 int main() {
2     int t;
3     cout << "Donnez le temps à décomposer, en secondes : ";
4     cin >> temps;
5     int* j;    int* h;    int* m; int* s;
6     jhms(temps, j, h, m, s);
7
8     cout << "\n" << temps << "secondes se convertissent en " << *j << " jours
```

```

    , " << *h << " heure , " << *m << " minutes et " << *s << " secondes \
n";
9
10 return 0;
11 }
```

 [codeTP/codeTP-VII-0-27.c]

Constatez que le programme se compile mais plante à l'exécution. Expliquez pourquoi.



## ② Problème VII.2 : minimum et maximum

A. Ecrivez une fonction qui fournit en sortie le minimum et le maximum d'un tableau de float. Son prototype sera :

```
void minmax(vector<float> T, float* min, float* max);
```

 [codeTP/codeTP-VII-0-28.c]

B. Ecrivez une fonction main qui crée le tableau :

```
vector<float> tab =
{1.2, 0.8, 5.3, 6.15, 0.2, 12.4, 15.12, 9.89, 3.1, 0.7, 1.1, 1.7, 19.1, 2.05};
```

 [codeTP/codeTP-VII-0-29.c]

et en calcule le min et le max.



## ② Problème VII.3 : echange

On donne le code suivant :

```

1 int a,b;
2 a=3;
3 b=5;
4 cout << "avant : a=" << a << " b=" << b << "\n";
5
6 int c;
7 c = a;
8 a = b;
9 b = c;
10 cout << "apres : a=" << a << " b=" << b << "\n";
```

 [codeTP/codeTP-VII-0-30.c]

A. Exécutez le et constatez ce qu'il fait.

B. On va donc écrire une fonction **echange**. qui ... échange ... les valeurs de **a** et **b**. Notez que cette fonction a forcément 2 sorties puisqu'il faut modifier 2 variables. Alors son prototype sera :

```
void echange(int* pa, int* pb);
```

Ecrivez la fonciton **echange** pour qu'elle réalise le travail fait au niveau des lignes 6 à 9 du code donné ci-dessus.

C. Modifiez le **main** pour qu'elle utilise la fonction **echange** en lieu et place des lignes 6 à 9.

D. Expliquez pourquoi **pa** et **pb** ne sont pas juste des "sorties" de la fonction, mais plutot des "entrées / sorties".



## ② Problème VII.4 : pair et impair

On donne un vector :

```
vector<int> tab = {1,8,5,6,2,12,15,9,3,7,11,17,19,20};
```

 [codeTP/codeTP-VII-0-31.c]

A. Ecrivez une fonction **pairImpair** qui prend ce tableau en entrée et fournit deux tableaux en sortie, un contenant uniquement les valeurs paires, l'autre uniquement les valeurs impaires du tableau d'origine.



## VII.1 Pour vous entraîner

### ② Problème VII.5 : Solutions d'un trinôme

A. Commencez par écrire un `main` qui calcule les solutions  $x_1$  et  $x_2 \in \mathbb{R}$  d'un trinôme :

$$ax^2 + bx + c = 0$$

avec  $a, b, c \in \mathbb{R}$ . Le programme devra faire la distinction entre les cas où il y a 0, 1 ou 2 solutions réelles.

On rappelle :

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} \quad \text{avec} \quad \Delta = b^2 - 4ac.$$

B. Transformez votre programme pour que le calcul des solutions soit fait dans une fonction :

```
int trinome(double a, double b, double c, double* x1, double* x2);
```

Les deux solutions, lorsqu'elles existent, fonctionneront en passage par adresse. La fonction renverra le nombre de solutions. Elle n'affichera rien. Vous écrirez aussi un `main` qui fait appel à votre fonction et affiche les résultats.

C. Réécrivez la fonction sous la forme :

```
void trinome(double a, double b, double c, double* x1, double* x2, int* nbsol);
```

D. Réécrivez la fonction sous la forme :

```
void trinome(double a, double b, double c, vector<double>*> solutions);
```



### ② Problème VII.6 : Tri d'un tableau

On redonne le code de l'algorithme de tri :

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5
6 int main()
7 {
8     vector<int> tab = {5, 8, 1, 6, 2, 12, 15, 9, 3, 20, 11, 17, 19, 7};
9     int k=0;
10    cout << "\n";
11    while(k<tab.size()) {
12        cout << tab[k] << " ";
13        k=k+1;
14    }
15    cout << "\n";
16
17    vector<int> tab2;
18    tab2.resize(tab.size());
19
20    int max, posMax;
21    k=0;
22    max = tab[0];

```

```

23     while(k<tab.size()) {
24         if(tab[k]>max) {
25             max = tab[k];
26             posMax = k;
27         }
28         k=k+1;
29     }
30
31     int min, posMin;
32     k=0;
33     posMin = 0;
34     min = tab[0];
35     while(k<tab.size()) {
36         if(tab[k]<min) {
37             min = tab[k];
38             posMin = k;
39         }
40         k=k+1;
41     }
42
43
44
45     int m;
46     k=0;
47     while(k<tab2.size()) {
48         min = tab[0];
49         posMin = 0;
50         m=0;
51         while(m<tab.size()) {
52             if(tab[m]<min) {
53                 min = tab[m];
54                 posMin = m;
55             }
56             m=m+1;
57         }
58         tab2[k] = min;
59         tab[posMin] = max;
60         k=k+1;
61     }
62
63     k=0;
64     cout << "\n";
65     while(k<tab2.size()) {
66         cout << tab2[k] << " ";
67         k=k+1;
68     }
69     cout << "\n";
70
71     return 0;
72 }
```



[codeTP/codeTP-VII-1-32.c]

**A.** Ecrivez deux fonctions :

```

void max(vector<int> tab, int* max, int* posMax);
void min(vector<int> tab, int* min, int* posMin);
```

qui déterminent le maximum (resp. le minimum) d'un tableau, ainsi que sa position.

**B.** Réécrivez l'ensemble du code pour utiliser ces fonctions au maximum.

C. Ecrivez maintenant une fonction de tri dont le prototype est :

```
void tri(vector<int> tab, vector<int>* tabTri);
```

Et modifiez le **main** pour l'utiliser.



## ② Problème VII.7 : Calcul de Pi

A. Commencez par écrire un **main** qui calcule la valeur de  $\pi$  en utilisant la formule :

$$\pi \approx 4 \times \left( \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^k}{2k+1} \right)$$

Après calcul, vous afficherez la valeur. Votre approximation de  $\pi$  utilisera les 100 premiers termes.

B. Déplacez maintenant le code du calcul dans une fonction que vous appellerez **calcPi** qui reverra la valeur de *pi* calculée en utilisant les 100 premiers termes. Vous testerez votre fonction **calcPi** en y faisant appel dans le **main**.

C. Généralisez maintenant votre fonction pour qu'elle prenne en paramètre le nombre de termes à calculer. Affichez le résultat pour 2, 5, 10, 20, 50 et 100 termes dans le **main**.

D. Transformez maintenant votre fonction pour qu'elle prenne pour prototype :

```
void calcPi(int nbTermes, double* valeurPi);
```

Et utilisez la dans le **main**. Vérifiez que vous obtenez les mêmes résultats que précédemment.



## ② Problème VII.8 : Histogramme v.2

On donne le code ci-dessous. Peu importe que vous en compreniez les détails, il remplit simplement le tableau **val** avec 10000 valeurs aléatoires selon une distribution qui suit la loi binomiale.

```
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <iostream>
#include <vector>
using namespace std;

float ranf(); /* ranf() is uniform in 0..1 */
float ranNorm(float m, float s); /* box-muller method */

int main()
{
    srand(time(NULL));

    int Nval;
    int i;
    Nval = 10000;
    vector<float> val;
    val.resize(Nval);
    for(i=0; i<Nval; i++)
    {
        val[i]=ranNorm(5,5);
    }

    // a completer
```

```

        return 0;
    }

float ranf()           /* ranf() is uniform in 0..1 */
{
    return ((float)rand()) / RAND_MAX;
}

// moyenne m, ecart type s
// methode dite de box-muller
float ranNorm(float m, float s)
{

    float x1, x2, w, y1;
    static float y2;
    static int use_last = 0;

    if (use_last)
    {
        y1 = y2;
        use_last = 0;
    }
    else
    {
        do {
            x1 = 2.0 * ranf() - 1.0;
            x2 = 2.0 * ranf() - 1.0;
            w = x1 * x1 + x2 * x2;
        } while (w >= 1.0);

        w = sqrt( (-2.0 * log(w)) / w );
        y1 = x1 * w;
        y2 = x2 * w;
        use_last = 1;
    }
    return( m + y1 * s );
}

```

 [codeTP/codeTP-VII-1-33.c]

A partir de ces données, on veut calculer un histogramme.

**A.** Complétez ce programme pour qu'il calcule l'histogramme du tableau val sur  $N$  tranches,  $N$  étant choisi par l'utilisateur du programme. Cet histogramme sera calculé par une fonction dont le prototype sera :

```
void histogramme(vector<float> val, int nval, int N, vector<float>*& histo);
```

Vous reprendrez et adaperez les fonctions de calcul du maximum et minimum vues à l'exercice [IV.2](#) pour déterminer l'étendue des valeurs.

**B.** Ajoutez maintenant une fonction qui calcule le tableau de chaînes de caractères permettant d'obtenir la représentation ci-dessous. Son prototype sera :

```
void dessineHistogramme(vector<float> histo, vector<string>* dessin);
```

```

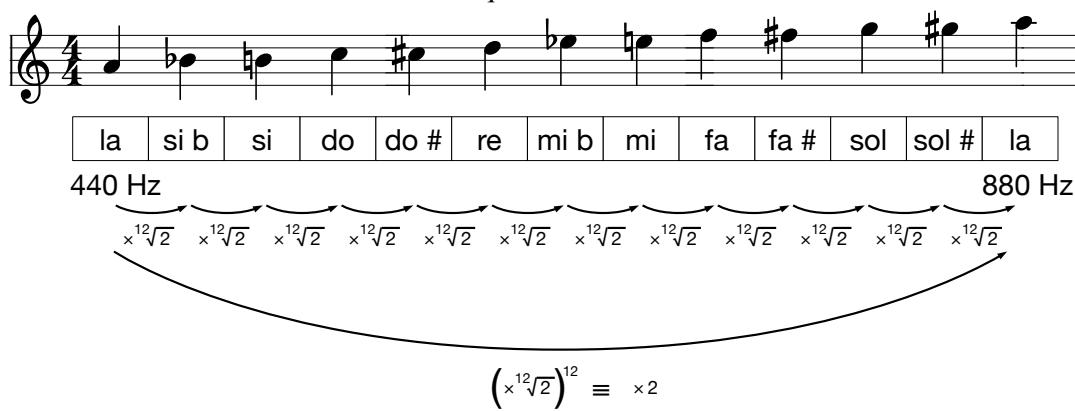
1 *
2 *
3 *
4 **
5 *****
6 *****
7 *****
8 *****
9 *****
10 *****
11 *****
12 *****
13 *****
14 *****
15 *****
16 *****
17 ****
18 **
19 *
20 *

```

C. Modifiez le `main` pour faire appel à ces deux fonctions, puis tracez l'histogramme avec le tableau de chaînes de caractères obtenu avec la fonction `dessineHistogramme`. ■

### Problème VII.9 : Notes de musique

Ce que nous percevons comme des notes de musique sont physiquement des fréquences : chaque note correspond à une fréquence. Les notes s'obtiennent par une progression géométrique (multiplicative) de raison  $\sqrt[12]{2} = 2^{1/12}$ . Un point de référence possible est le "la" à 440Hz. On représente ci-dessous la séquence de base pour toutes les notes entre deux "la". Cet écart entre deux "la" successifs correspond à une "octave", c'est à dire l'écart entre une fréquence et son double :



*Exemple à partir du la 440 Hz*

Cette séquence se répète à l'identique pour tous les "la" : on pourrait la reproduire exactement pour l'intervalle 110Hz – 220Hz ou encore 880Hz – 1760Hz (et en deçà et au-delà).

*Remarque : en C/C++ on peut obtenir  $\sqrt[12]{2}$  en utilisant `pow` de `cmath` :*

```
v=pow(2,1.0/12);
```

A. Ecrivez une fonction `freq1` qui prend en paramètre le nom d'une note et donne la valeur de sa fréquence dans l'intervalle 440Hz – 880Hz. Essayez la dans un `main`.

**B.** On appelle "la0" le la à  $55\text{Hz}$ , "la1" le la à  $110\text{Hz}$ , le "la2" le la à  $220\text{Hz}$ , et ainsi de suite : c'est le "la" de base pour l'octave considérée. Ecrivez une fonction `freq2` similaire à la fonction précédente, sauf qu'elle prendra en paramètre supplémentaire le "numéro" du "la" de base de l'octave considérée. Par exemple, pour le "do2", l'appel à la fonction devra pouvoir se faire comme suit :

```
f = freq2("do", 2);
```

**C.** Plus compliqué : Ecrivez une fonction qui prend en paramètre la valeur d'une fréquence et renvoie le nom de la note qui correspond. La fréquence donnée pourra ne pas être exactement celle d'une note, aussi il faudra donner la note la plus proche de la fréquence considérée. Vous ajouterez une sortie à la fonction pour fournir en sortie l'erreur relative sur la fréquence par rapport à la fréquence de la note exacte.

**D.** Sachant que les sons émis par les instruments de musique sont des fonctions périodiques (ou presque) en fonction du temps, déduisez-en les fréquences des 10 premières harmoniques. Ecrire une fonction qui renvoie un tableau de chaînes contenant les notes associées aux 10 premières harmoniques d'une note dont on donne la fréquence en paramètre. ■

# Séance VIII – Synthèse : pointeurs, fonctions, tableaux, structures

## ⌚ Motivations et objectifs

La programmation en C/C++ repose complètement sur l'utilisation combinée de ces 4 concepts du langage. On fait donc un TP de synthèse pour voir comment tout ça s'articule ensemble.

Ce TP concerne tous les chapitres précédents du cours, en mettant particulièrement l'accent sur les chapitres V, VI, IX et X.

## ❓ Problème VIII.1 : Matrices 2x2 – V2

On redonne le code des matrices  $2 \times 2$ . L'objectif est de réécrire en utilisant le passage par adresse :

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 typedef struct {
6     double A,B,C,D;
7 } matrice22;
8
9 void afficheM22 ( matrice22 M );
10 matrice22 saisieM22();
11 matrice22 produitM22(matrice22 M1, matrice22 M2);
12 matrice22 transfert(vector<matrice22> tab);
13
14 int main() {
15     vector<matrice22> T;
16     int k,N=4;
17     T.resize(N);
18
19     cout << "Entrez les matrices :\n";
20     for(k=0; k<N; k++) {
21         cout << "Matrice n." << k+1 << " :\n";
22         T[k] = saisieM22();
23     }
24
25     matrice22 prod;
26     prod = transfert(T);
27     afficheM22(prod);
28
29     return 0;
30 }
31
32
33
34 void afficheM22 ( matrice22 M ) {
35     cout << M.A << " " << M.B << "\n";
```

```

36     cout << M.C << " " << M.D << "\n";
37 }
38
39 matrice22 saisieM22() {
40     matrice22 M;
41     cout << "Entrez les valeurs :\n";
42     cout << "A : ";
43     cin >> M.A;
44     cout << "B : ";
45     cin >> M.B;
46     cout << "C : ";
47     cin >> M.C;
48     cout << "D : ";
49     cin >> M.D;
50
51     return M;
52 }
53
54 matrice22 produitM22(matrice22 M1, matrice22 M2)
55 {
56     matrice22 R;
57     R.A = M1.A*M2.A + M1.B*M2.C;
58     R.B = M1.A*M2.B + M1.B*M2.D;
59     R.C = M1.C*M2.A + M1.D*M2.C;
60     R.D = M1.C*M2.B + M1.D*M2.D;
61
62     return R;
63 }
64
65 matrice22 transfert(vector<matrice22> tab)
66 {
67     matrice22 R=tab[0];
68     int k;
69     for(k=1; k<tab.size(); k=k+1) {
70         R = produitM22(R,tab[k]);
71     }
72
73     return R;
74 }
```

 [codeTP/codeTP-VIII-0-34.c]

- Commencez par tester le programme en l'état.
- Réécrivez la fonction `saisieM22v2` pour qu'elle utilise le passage par adresse. Remplacez aussi l'appel qui est fait ligne 22 pour l'adapter. Pensez aussi à la déclaration de prototype ligne 10. Vérifiez que le programme fonctionne toujours et que le résultat du calcul ne change pas après cette adaptation.
- Réécrivez la fonction `produitM22v2` pour qu'elle utilise le passage par adresse pour le résultat. Remplacez aussi l'appel qui est fait ligne 70 pour l'adapter. Pensez aussi à la déclaration de prototype ligne 11. Vérifiez que le programme fonctionne toujours et que le résultat du calcul ne change pas après cette adaptation.
- Réécrivez à nouveau la fonction `produitM22v3` pour qu'elle utilise du passage par adresse à la fois pour le résultat, pour `M1` et pour `M2`. Remplacez aussi l'appel qui est fait ligne 70 pour l'adapter. Pensez aussi à la déclaration de prototype ligne 11. Expliquez ce que l'on est supposé gagner en utilisant cette écriture.
- Vérifiez que le programme fonctionne toujours. Vous constatez que le résultat n'est plus correct. Expliquez pourquoi.

**E.** Trouvez une solution à ce problème, tout en utilisant `produitM22v3` exactement comme vous venez de l'écrire.

**G.** Que peut-on conclure sur les bénéfices et les inconvénients du passage par adresse ?

**H.** La seule fonction que l'on n'a pas traité en matière de passage par adresse est la fonction `transfert`. A-t-on intérêt à écrire cette fonction par adresse ? Posez vous la question non seulement sur les sorties, mais aussi sur les entrées, en réfléchissant à ce qui est recopié ou pas. Si vous pensez que oui : écrivez la sous le nom `transfertv2` et faites appel dans le `main`. Si vous pensez que non, expliquez pourquoi. ■

## Problème VIII.2 : Rendre la monnaie

On veut écrire un programme qui calcule la meilleure façon de rendre la monnaie. De façon stéréotypée, la meilleure façon de rendre la monnaie signifie en général (et c'est ce que l'on va faire) de suivre un algorithme dit "glouton" : cela consiste simplement à donner la priorité aux plus grosses coupures. Pour faire cela correctement, on a besoin, pour chaque pièce ou billet, de deux informations : sa valeur, en euros, et le nombre que l'on en a dans la caisse. On va donc partir de la situation ci-dessous :

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

typedef struct {
    double valeur;
    int nombre;
} monnaie;

int main() {
    vector<monnaie> caisse = {{500,3},{100,7},{50,12},{20,22},{10,15},
                                {5,25},{2,15},{1,25},{0.5,20}};

    // à compléter

    return 0;
}
```

 [codeTP/codeTP-VIII-0-35.c]

**A.** Ecrivez une fonction qui affiche le contenu complet et détaillé (valeur et quantité), de la caisse. Testez la dans le `main`.

**B.** Ecrivez une fonction `total` qui renvoie le total de la caisse et testez la dans le `main`.

**C.** Ecrivez maintenant une fonction dont le prototype est :

```
vector<monnaie> rendreMonnaie(double donne, double prix, vector<monnaie>*
                                caisse);
```

 [codeTP/codeTP-VIII-0-36.c]

Qui calcule la monnaie à rendre en utilisant l'algorithme "glouton" décrit en introduction de cet exercice. Cette fonction :

- prend en paramètre la somme donnée par le client pour payer ce qu'il doit payer,
- prend aussi en paramètre le prix de l'objet à payer,
- mais aussi l'état de la caisse, en passage par adresse.
- Elle renvoie aussi, sous la forme d'un `vector<monnaie>`, l'ensemble des billets et leur nombre.

A la fin de la fonction, la caisse aura été mise à jour.

*Note : On ne demande pas d'ajouter à la caisse la somme encaissée. La caisse sert ici uniquement à rendre la monnaie. On pourrait traiter aussi l'encaissement mais la fonction, telle qu'elle est proposée,*

*ne le permet pas, dans un soucis de simplification*

Essayez votre fonction dans le `main`, et affichez l'état de la caisse avant et après le paiement. Vous vérifieriez, de façon préliminaire, que le total contenu dans la caisse suffit à rendre la monnaie, en utilisant la fonction `total`.

**D.** Modifiez votre code pour que la fonction `rendreMonnaie` prenne le prototype suivant :

```
void rendreMonnaie(double donne, double prix, vector<monnaie>*> caisse,
                    vector<monnaie>*> rendu);
🌐 [codeTP/codeTP-VIII-0-37.c]
```

**E.** Comment traiteriez-vous le cas où il faut aussi que la caisse encaisse la somme donnée par le client ? Quelle information manque dans ce qui a déjà été traité ? Faites le si vous avez le temps. ■

## VIII.1 Pour vous entraîner

### ➊ Problème VIII.3 : Stations Météo

On veut représenter les données issues de plusieurs stations météo, dans plusieurs villes. Chaque station meteo relève 3 températures par jour, chaque jour de la semaine. On propose la structure ci-dessous pour représenter une station meteo :

```
typedef struct {
    string ville;
    vector<vector<float>> temperatures;
} station;
```

Dans la structure, le tableau des températures comporte 7 lignes, une par jour de la semaine, et chaque ligne comporte 3 valeurs, une par température pour chaque jour.

**A.** Ecrivez une fonction :

```
station initialise(string nom);
```

qui renvoie une structure initialisée en utilisant le paramètre `nom` pour la ville, et fait ce qu'il faut pour que le tableau des températures fasse la bonne taille pour stocker les températures de chaque jour.

**B.** On devrait normalement donner des valeurs à chaque température de nous même, mais dans le cadre d'un exercice c'est trop long. Alors on va faire remplir automatiquement les températures avec des valeurs aléatoires. Ecrivez une fonction :

```
void temperatures_aleatoires(station* s);
```

qui remplit la totalité des températures du tableau avec des valeurs aléatoires entre -20 et 50. Pour générer des nombres aléatoires, on donne un code d'exemple ci dessous. Vous devrez vous en inspirer pour l'écriture de la fonction `temperatures_aleatoires`.

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

int main()
{
    double valeur;
    // initialisation de l'aleatoire
    srand(time(NULL));

    int k=0;
    while(k < 100) {
        // genere un nombre aleatoire entre 0 et 1
        valeur = ((double)rand()) / RAND_MAX;
```

```

    // affichage
    cout << valeur << " " << "\n";
    k = k + 1;
}
return 0;
}

```

 [codeTP/codeTP-VIII-1-38.c]

C. Créez une fonction :

```
void afficher(station* s)
```

qui affiche la ville et l'ensemble des températures sur 3 lignes. Réfléchissez à pourquoi le passage par adresse n'était pas obligatoire mais possible. Est-ce mieux ou pas ?

D. Ecrivez une fonction :

```
double temperature_moyenne(station* s)
```

Qui calcule la température moyenne pour une station.

E. On donne le début de code suivant :

```

int main()
{vector<string> villes = {"Paris", "Lyon", "Marseille", "Toulouse", "
    Montpellier", "Nimes", "Lille", "Rennes", "Clermont-Ferrand", "Grenoble"};
}

return 0;
}

```

 [codeTP/codeTP-VIII-1-39.c]

En utilisant le tableau villes, écrivez un code qui remplit automatiquement un tableau de station avec la totalité des villes et des températures prises au hasard. Ecrivez votre code comme si il y avait des milliers de villes et pas juste une dizaine. Réalisez l'affichage de l'ensemble des données obtenu. Pour chaque ville, vous ajouterez l'affichage de sa température moyenne.

■

## ② Problème VIII.4 : Jeu de cartes

On veut gérer un jeu de 54 cartes. On part d'une structure pour gérer chaque carte :

```

typedef struct {
    int valeur;
    string couleur;
}carte;

```

 [codeTP/codeTP-VIII-1-40.c]

Dans cette structure, on considèrera que le champ "valeur" contient la valeur de la carte, soit une valeur entre 1 et 10 pour les simples valeurs, puis 11 pour le valet, 12 pour la dame, 13 pour le roi. Le champ "couleur" est une chaîne de caractères qui peut être au choix "Coeur", "Carreau", "Trèfle", "Pique", ou "Jocker". Si la couleur est "Jocker", alors la valeur doit être 0.

A. On donne le code de base ci-dessous :

```

#include <iostream>
#include <vector>
#include <string>
using namespace std;

typedef struct {
    int valeur;
    string couleur;
}carte;

int main() {

```

```

vector<string> couleursPossibles={"Coeur","Carreau","Trefle","Pique"};
vector<carte> jeu;
vector<carte> joueur1, joueur2;

return 0;
}

 [codeTP/codeTP-VIII-1-41.c]

```

**B.** Ecrivez un code qui remplit le tableau jeu avec toutes les cartes possibles, toutes combinaisons de valeurs et couleurs possibles, en utilisant une boucle. Vous pourrez ajouter ensuite les 2 jocker sans faire de boucle.

**C.** Ecrivez une fonction qui retire une carte du jeu et la place dans un autre tableau de cartes. La carte à retirer est identifiée par sa position dans le tableau. Le prototype de la fonction sera :

```
int déplacerCarte(vector<carte>* origine, vector<carte>* destination, int numero);
```

La fonction doit renvoyer si l'opération pu être réalisée ou pas.

Réfléchissez aussi à pourquoi cette fonction ne marcherait pas si on l'écrivait sans pointeurs.

**D.** Ecrivez une fonction qui mélange les cartes d'un tableau de cartes. Une façon simple est de procéder par échange : tirer 2 nombres aléatoires pour permute les 2 cartes qui sont à ces deux positions. Le prototype sera :

```
void melanger(vector<carte>* amelanger);
```

Pour tirer aléatoirement des numeros de cartes, vous pouvez vous inspirer du code ci-dessous :

```

#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;

int main()
{
    double valeur;
    // initialisation de l'aleatoire
    srand(time(NULL));

    int k=0;
    while(k < 100) {
        // genere un nombre aleatoire entre 0 et 1
        valeur = ((double)rand()) / RAND_MAX;
        // affichage
        cout << valeur << " " << "\n";
        k = k + 1;
    }
    return 0;
}

 [codeTP/codeTP-VIII-1-42.c]

```

**E.** Ajoutez dans le main ce qu'il faut pour mélanger les cartes du tableau jeu, puis qui distribue 5 cartes à chaque joueur, à partir du jeu mélangé.

**F.** Ecrivez maintenant un programme qui joue au jeu de Bataille tout seul entre les 2 joueurs : Le jeu doit consister à :

1. tirer aléatoirement une carte dans le tas de cartes de chaque joueur
2. comparer les valeurs des deux cartes. Celui qui a la plus grande valeur marque 1 point. Si les

deux ont la même valeur, chacun marche 1 point. Si le jocker sort, il est considéré comme la carte la plus forte.

### 3. distribuer à chaque joueur une carte prise dans le jeu principal

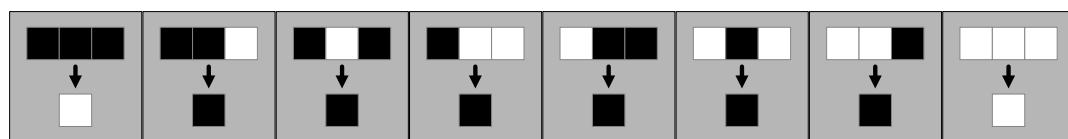
Bien sûr il faut afficher à chaque étape ce qu'il se passe pour suivre le jeu. On recommence ces étapes tant qu'il y a des cartes dans le jeu principal. Afficher les scores et dire qui, du joueur 1 ou du joueur 2, a gagné. ■

## ② Problème VIII.5 : Automate cellulaire 1D

*Note :* Je vous encourage à aller regarder une vidéo youtube sur ce sujet : [Sciences étonnantes #49 : le jeu de la vie](#).

On veut programmer un automate cellulaire. Il s'agit d'un programme qui part d'un tableau de valeurs contenant uniquement des 0 et des 1, et qui va, en utilisant des règles d'évolution, définir un nouvel état du tableau. A partir de ce nouveau état, on pourra appliquer à nouveau les règles, et ainsi de suite. C'est ce que l'on propose de programmer.

Dans ce que l'on propose ici, l'état de chaque case de l'étape suivante sera toujours défini par une règle sur l'état de 3 cases connexes du tableau. Voici par exemple une série de règles (appelées "règle 126") que l'on peut appliquer :



Avec ces règles et un tableau de 80 cases contenant uniquement des 0 et un seul 1 au milieu, et en calculant l'évolution du tableau sur 24 étapes, on obtient (les 0 sont des espaces et les 1 sont des \*) :

```

01 *
02 ***
03 ** **
04 *****
05 **   **
06 ****  ****
07 **  **  **  **
08 *********
09 **      **
10 ****      ****
11 **  **      **  **
12 *****      *****
13 **  **  **  **  **
14 ****  ****  ****  ****
15 **  **  **  **  **  **  **
16 *********
17 **          **
18 ****          ****
19 **  **          **  **
20 *****          *****
21 **  **          **  **
22 ****  ****          ****  ****
23 **  **  **  **          **  **  **  **
24 *********

```

- A. Expliquez pourquoi la structure ci-dessous modélise correctement une règle :

```
typedef struct {
    vector<int> avant;
    int resultat;
} regle;
```

 [codeTP/codeTP-VIII-1-43.c]

Vous justifierez votre réponse en donnant notamment combien de valeurs contient le champ **avant** dans la structure.

- B. Ecrivez une fonction qui permet de remplir une structure règle :

```
|regle init(vector<int> avant, int resultat);
```

 [codeTP/codeTP-VIII-1-44.c]

- C. Créez dans le **main** un tableau pour contenir toutes les règles énoncées dans le dessin proposé plus haut. Remplissez ce tableau avec les règles en question.

- D. Ecrivez enfin le programme qui permet de générer la figure ci-dessus en utilisant ce tableau. Vous aurez aussi besoin d'un autre tableau de 80 cases dont le contenu représente la condition initiale (ligne **01**) de l'illustration au départ du programme. ■

## ② Problème VIII.6 : Matrices 2x2 de nombres complexes

On veut gérer des matrices  $2 \times 2$  de nombres complexes.

- A. Ecrivez un type complexe et les fonctions minimum pour les utiliser :

```
typedef struct {
    double Re;
    double Im;
} complexe;

void afficheCplx ( complexe z ) ;
void saisieCplx ( complexe* z );
complexe sommeCplx ( complexe g , complexe d ) ;
complexe produitCplx ( complexe g , complexe d ) ;
```

 [codeTP/codeTP-VIII-1-45.c]

- B. Réécrivez complètement l'exercice VIII.1 où les nombres de la matrice sont des nombres complexes définis par le type complexe évoqué ci-dessus. ■

# Séance IX – Compilation multi-fichiers et utilisation d'un IDE

## ⌚ Motivations et objectifs

Ce TP concerne la première moitié du chapitre XI du cours.

### IX.1 Compilation multi-fichiers : Utilité et Difficultés

On commence à écrire des codes de plus en plus gros, et il n'est pas commode de garder tout ce code dans un seul fichier. Il est donc usuel en C/C++ de séparer un gros code en plusieurs sous-parties, en plusieurs fichiers. Mais quand on fait ça, il faut pouvoir maintenir la capacité des différents fichiers à "communiquer" entre eux. C'est ce que l'on va regarder ici.

On va partir d'un code C un peu gros, que vous pouvez télécharger ci-dessous :

```
1 #include<vector>
2 #include<string>
3 #include<iostream>
4 #include<cstdlib>
5 #include<ctime>
6 using namespace std;
7
8
9 vector<vector<int> > creerGrille(unsigned int N) ;
10 void effacerGrille(vector<vector<int >>* G) ;
11 int estAligne(int joueur,int x0,int y0,int nb,int dx, int dy,vector<vector<int >> G);
12 int grillePleine(vector<vector<int >> G) ;
13 void afficherGrille(vector<vector<int >> G);
14 int placePion(int x,int y,int joueur,vector<vector<int >>* G);
15 int aGagne(int joueur,vector<vector<int >> G);
16 int partieTerminee(vector<vector<int >> G);
17 void joueurIA(int joueur,vector<vector<int >>* G);
18
19
20
21 // programme principal
22 int main() {
23
24     srand( time( NULL ) );
25
26     vector<vector<int >> grille = creerGrille(3);
27     effacerGrille(&grille);
28
29     int x,y,joueur=1;
30     int fini = 0;
31     while(fini == 0)
32     {
33         afficherGrille(grille);
```

```
34     joueur= 1;
35     cout << "# C'est a Joueur " << joueur << "\n";
36     int place = 0;
37     while(place != 1) {
38         cout << "ou jouez vous ? (x,y) \nx : ";
39         cin >> x;
40         cout << "y : ";
41         cin >> y;
42         place = placePion(x,y,joueur,&grille);
43         if(place==0)
44         {
45             cout << "\n(" << x << ", " << y << ") est deja pris. Proposez autre
46             chose.";
47         }
48     }
49
50     fini = partieTerminee(grille);
51     afficherGrille(grille);
52     if(fini==0)
53     {
54         joueur = 2;
55         cout << "# C'est a Joueur " << joueur << "\n";
56         joueurIA(joueur,&grille);
57         fini = partieTerminee(grille);
58     }
59     cout << "Jeu Termine\n";
60     afficherGrille(grille);
61     return 0;
62 }
63
64
65
66
67
68 // Partie du code qui gere la grille
69 vector<vector<int> > creerGrille(unsigned int N) {
70     vector<vector<int> > G;
71     G.resize(N);
72     for(unsigned int k=0; k<N; k++) {
73         G[k].resize(N);
74     }
75     return G;
76 }
77
78 void effacerGrille(vector<vector<int>>* G) {
79     for(unsigned int y=0; y<G->size(); y++) {
80         for(unsigned int x=0; x<(*G)[y].size(); x++) {
81             (*G)[y][x]=0;
82         }
83     }
84 }
85
86
87
88 int estAligne(int joueur,int x0,int y0,int nb,int dx, int dy,vector<vector<
89 int >> G) {
```

```
90     for(int k=0; k<nb; k++)  
91     {  
92         if(G[y0+k*dy][x0+k*dx]!=joueur)  
93             {    return 0;  
94             }  
95     }  
96  
97     return 1;  
98 }  
99  
100  
101 int grillePleine(vector<vector<int >> G) {  
102     for(unsigned int y=0; y<G.size(); y++) {  
103         for(unsigned int x=0; x<G[y].size(); x++) {  
104             if(G[y][x]==0) return 0;  
105         }  
106     }  
107     return 1;  
108 }  
109  
110  
111 // partie du code qui gère le jeu  
112 void afficherGrille(vector<vector<int >> G) {  
113     cout <<"\n";  
114     for(unsigned int y=0; y<G.size(); y++) {  
115         for(unsigned int x=0; x<G[y].size(); x++) {  
116             cout << " ";  
117             if(G[y][x]==0) {  
118                 cout << " ";  
119             }  
120             if(G[y][x]==1) {  
121                 cout << "X";  
122             }  
123             if(G[y][x]==2) {  
124                 cout << "O";  
125             }  
126             if(x!=G[y].size() -1) {  
127                 cout << " |";  
128             }  
129         }  
130     }  
131     if(y!=G.size()-1)  
132     {    cout << "\n";  
133         for(unsigned int x=0; x<G[y].size()*3+2; x++) {  
134             cout << "-";  
135         }  
136     }  
137     cout << "\n";  
138 }  
139     cout << "\n";  
140 }  
141  
142  
143  
144 int placePion(int x, int y,int joueur,vector<vector<int >>*> G)  
145 {  
146     if((*G)[y][x]==0)
```

```
148     {
149         (*G)[y][x] = joueur;
150         return 1;
151     }
152 else return 0;
153 }
154
155
156
157 int aGagne(int joueur, vector<vector<int>> G)
158 { // test diagonales, puis lignes, puis colonnes
159     int diags = estAligne(joueur, 0, 0, 3, 1, 1, G)==1 || estAligne(joueur
160     , 2, 0, 3, -1, 1, G)==1 ;
161     int lines = 0;
162     int cols = 0;
163     for(unsigned int i=0; i<3; i++)
164     {   lines = lines || estAligne(joueur, i, 0, 3, 0, 1, G);
165         cols = cols || estAligne(joueur, 0, i, 3, 1, 0, G);
166     }
167     if( diags || lines || cols) {
168         return 1;
169     }
170
171     return 0;
172 }
173
174 int partieTerminee(vector<vector<int>> G)
175 {
176     if(aGagne(1, G)==1) {
177         cout << "\nJoueur 1 a gagne !\n";
178         return 1;
179     }
180     if(aGagne(2, G)==1) {
181         cout << "\nJoueur 2 a gagne !\n";
182         return 1;
183     }
184     if(grillePleine(G)==1) {
185         cout << "Aucun joueur n'a gagne.\n";
186         return 1;
187     }
188     return 0;
189 }
190
191 // une IA idiote qui joue au hasard
192 void joueurIA(int joueur, vector<vector<int>>* G) {
193     vector<int> X, Y;
194     X.resize((*G).size() * (*G)[0].size());
195     Y.resize((*G).size() * (*G)[0].size());
196     int k=0;
197     for(unsigned int y=0; y<(*G).size(); y++) {
198         for(unsigned int x=0; x<(*G)[y].size(); x++) {
199             if((*G)[y][x] == 0)
200             {
201                 X[k] = x;
202                 Y[k] = y;
203                 k++;
204             }
205         }
206     }
207 }
```

```

205     }
206     int IA = rand() % k;
207     (*G)[Y[IA]][X[IA]] = joueur;
208 }
209 }
```

 [codeTP/codeTP-IX-1-46.c]

**A.** Commencez par le récupérer dans un fichier .cpp, compilez-le, essayez le. C'est un code qui gère un jeu de "morpion", avec une "intelligence artificielle" idiote qui joue simplement au hasard (pour éviter d'avoir, malgré tout, trop de code à gérer pour cet exemple).

**B.** Bien que ce code soit écrit sous la forme d'un seul fichier, il est plutôt correctement organisé. On va vouloir le séparer en 3 fichiers :

- **main.cpp** qui contiendra essentiellement la fonction principale : il s'agit des lignes 22 à 62. Ainsi : créez un nouveau fichier **main.cpp** et copiez/collez dedans les lignes 22 à 62. Si vous parcourez le code de cette partie, vous voyez que l'on a besoin des bibliothèques **vector**, **iostream**, **cstdlib** et **ctime**. Aussi il faudra ajouter ces **include** en haut de **main.cpp**.
- **grille.cpp** qui contiendra le code qui sert à gérer la grille de jeu. Dans le code, cela est identifié entre les lignes 68 et 108. Alors faites un copier coller des lignes 68 à 108 dans un nouveau fichier **grille.cpp**. Vous voyez que vous avez besoin seulement de la bibliothèque **vector** pour ces lignes de codes, alors ajoutez juste l'**include** qui correspond en haut de ce fichier **cpp**.
- **jeu.cpp** qui contiendra le code associé au jeu lui même, c'est à dire les lignes 112 à 209. On voit qu'on a besoin de **vector** et **iostream** et rien d'autre, alors ajoutez les **includes** qui correspondent à ces bibliothèques.

Pour chaque fichier qui contient des **includes**, n'oubliez pas le **using namespace std;**

**C.** Une fois que ce travail est fait, on peut essayer de compiler tout ça ensemble :

```
g++ main.cpp grille.cpp jeu.cpp -o morpion
```

Et regardons le résultat. On obtient tout un tas d'erreurs :

```

main.cpp: In function ‘int main()’:
main.cpp:13:35: error: ‘creerGrille’ was not declared in this scope
  13 |     vector<vector<int >> grille = creerGrille(3);
                 ^~~~~~
main.cpp:14:5: error: ‘effacerGrille’ was not declared in this scope
  14 |     effacerGrille(&grille);
                 ^~~~~~
main.cpp:20:9: error: ‘afficherGrille’ was not declared in this scope
  20 |     afficherGrille(grille);
                 ^~~~~~
main.cpp:29:21: error: ‘placePion’ was not declared in this scope
  29 |         place = placePion(x,y,joueur,&grille);
                 ^~~~~~
main.cpp:36:16: error: ‘partieTerminee’ was not declared in this scope
  36 |         fini = partieTerminee(grille);
                 ^~~~~~
main.cpp:42:13: error: ‘joueurIA’ was not declared in this scope; did you mean ‘joueur’?
  42 |             joueurIA(joueur,&grille);
                 ^~~~~~
                 |             joueur
main.cpp:47:5: error: ‘afficherGrille’ was not declared in this scope
  47 |     afficherGrille(grille);
```

```

|      ~~~~~
jeu.cpp: In function 'int aGagne(int, std::vector<std::vector<int> >)':
jeu.cpp:52:18: error: 'estAligne' was not declared in this scope
52 |     int diags = estAligne(joueur,0,0,3,1,1,G)==1 || estAligne(joueur,2,0,3,-1,1,G)==1 ;
|           ~~~~~
jeu.cpp: In function 'int partieTerminee(std::vector<std::vector<int> >)':
jeu.cpp:77:8: error: 'grillePleine' was not declared in this scope
77 |     if(grillePleine(G)==1) {
|           ~~~~~

```

On voit qu'il y a malgré tout bien 2 parties : la plus grande part est consacrée à `main.cpp`, et la dernière partie est consacrée à `jeu.cpp`. Cela signifie que le compilateur a commencé par compiler `main.cpp` (ça correspond à l'ordre que l'on a donné pour la compilation), puis il a sûrement compilé `grille.cpp` sans trouver d'erreur, puis il a compilé `jeu.cpp` où il a trouvé quelques erreurs.

Et quel est le problème ? Il s'agit, pour toutes les erreurs, de "fonctions non déclarées". Vous pouvez vérifier, les problèmes sont `creerGrille`, `afficherGrille`, etc, qu'il ne trouve pas. Il ne s'agit pas de variables manquantes par exemple.

Ce qui se passe, c'est que le compilateur compile en réalité les fichiers séparément : le fait de les compiler ensemble ne change pas le fait que, quand le compilateur compile `main.cpp`, il a à faire à des fonctions qu'il ne connaît pas : le fichier `main.cpp` est insuffisant.

En réalité, le compilateur a surtout besoin de savoir que les fonctions que `main.cpp` utilise existent, il faut les lui déclarer. Et ça on sait le faire, il suffit de recopier les prototypes qui manquent. De par les erreurs du compilateur, on voit qu'il manque :

- dans `main.cpp` : `creerGrille`, `effacerGrille`, `afficherGrille`, `placePion`, `partieTerminee`, `joueurIA`,
- et dans `jeu.cpp` : `estAligne`, et `grillePleine`.

Alors, ajoutons les prototypes des fonctions qui correspondent en haut de `main.cpp` et dans `jeu.cpp`. Ainsi, le début de `main.cpp` va ressembler à :

```

#include<vector>
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;

vector<vector<int> > creerGrille(unsigned int N) ;
void effacerGrille(vector<vector<int >>* G) ;
void afficherGrille(vector<vector<int >> G);
int placePion(unsigned int x, unsigned int y, int joueur, vector<vector<int
>>* G);
int partieTerminee(vector<vector<int >> G);
void joueurIA(int joueur, vector<vector<int >>* G);
int estAligne(int joueur, int x0, int y0, int nb, int dx, int dy, vector<vector<
int >> G);

// puis le code de la fonction main

```

et le début de `jeu.cpp` sera :

```

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

int estAligne(int joueur, int x0, int y0, int nb, int dx, int dy, vector<vector<
int >> G);
int grillePleine(vector<vector<int >> G) ;

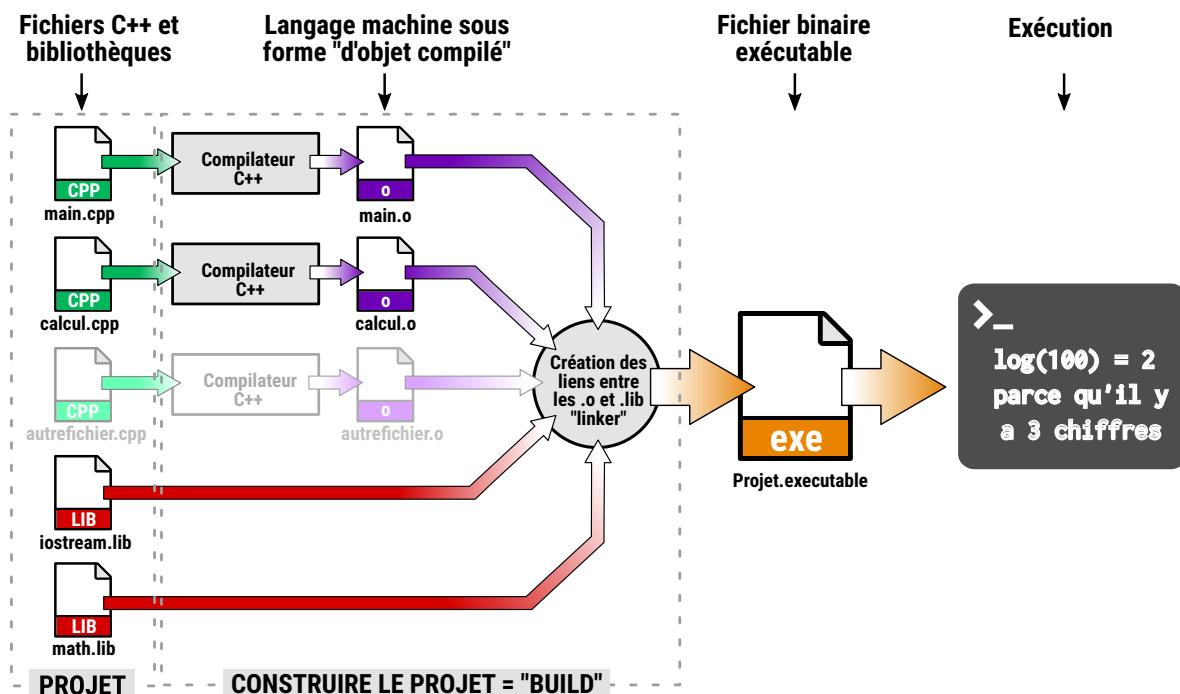
// puis les fonctions de jeu.cpp

```

Et maintenant si vous recompilez :

```
g++ main.cpp grille.cpp jeu.cpp -o morpion
```

vous ne voyez plus d'erreur, et si vous lancez le programme il fonctionne. La leçon à retenir de ça c'est que pour compiler un fichier, le C/C++ n'a pas besoin des fonctions elles-mêmes mais juste de connaître leur prototype. C'est dans une phase supplémentaire, l'édition des liens, que les différentes fonctions vont être vraiment mises ensemble. la commande `g++` fait tout cela en une fois. On redonne ci-dessous le schéma de compilation du cours, pour que vous puissiez voir les étapes :



## IX.2 Compilation multi-fichiers : La bonne méthode

Ce que l'on vient de faire est un peu empirique : on a attendu d'avoir des problèmes pour mettre des "rustines" pour que la compilation passe. Ca n'est pas très méthodique. Il y a une méthode bien plus rigoureuse et universelle pour faire cela, et c'est ce que l'on décrit maintenant. Pour cela, commencez par effacer les fichiers `main.cpp`, `grille.cpp` et `jeu.cpp`, et repartons sur une base propre. Puis on va refaire exactement ce que l'on a fait au début :

- **main.cpp** qui contiendra essentiellement la fonction principale : il s'agit des lignes 22 à 62. Ainsi : créez un nouveau fichier `main.cpp` et copiez/collez dedans les lignes 22 à 62. Si vous parcourrez le code de cette partie, vous voyez que l'on a besoin des bibliothèques `vector`, `iostream`, `cstdlib` et `ctime`. Aussi il faudra ajouter ces `include` en haut de `main.cpp`.
- **grille.cpp** qui contiendra le code qui sert à gérer la grille de jeu. Dans le code, cela est identifié entre les lignes 68 et 108. Alors faites un copier coller des lignes 68 à 108 dans un nouveau fichier `grille.cpp`. Vous voyez que vous avez besoin seulement de la bibliothèque `vector` pour ces lignes de codes, alors ajoutez juste l'`include` qui correspond en haut de ce fichier `cpp`.
- **jeu.cpp** qui contiendra le code associé au jeu lui-même, c'est à dire les lignes 112 à 209. On voit qu'on a besoin de `vector` et `iostream` et rien d'autre, alors ajoutez les `includes` qui correspondent à ce fichier.

Pour chaque fichier qui contient des `includes`, n'oubliez pas le `using namespace std;`

Mais à partir de là on va travailler différemment : on n'ajoute pas de prototypes de fonction. Au lieu d'attendre d'avoir des problèmes de compilation, on va les anticiper.

Ce que l'on va faire : maintenant, on va faire en sorte que **chaque fichier .cpp déclare les fonctions qu'il a besoin d'exporter vers d'autres fichiers**. Autrement dit c'est la **démarche inverse de tout à l'heure** : nous avions plutot ajouté dans un fichier .cpp ce dont il avait besoin. Pour mettre en oeuvre cette démarche, on va créer, en supplément, 2 nouveaux fichiers :

- **grille.h** qui contiendra les prototypes de toutes les fonctions contenues dans **grille.cpp**
- **jeu.h** qui contiendra les prototypes de toutes les fonctions contenues dans **jeu.cpp**

Et on va ajouter une sécurité en plus. Alors voila à quoi vont ressembler ces deux fichiers.

Pour **grille.h** :

```
#ifndef __GRILLE_H__
#define __GRILLE_H__

#include<vector>
using namespace std;

// prototypes de toutes les fonctions de grille.cpp :
vector<vector<int> > creerGrille(int N);
void effacerGrille(vector<vector<int>>* G);
int estAligne(int joueur, int x0, int y0, int nb, int dx, int dy, vector<vector<int>>* G);
int grillePleine(vector<vector<int>> G);

#endif
```

 [codeTP/codeTP-IX-2-47.c]

Pour **jeu.h** :

```
#ifndef __JEU_H__
#define __JEU_H__

#include<vector>
using namespace std;

void afficherGrille(vector<vector<int>> G);
int placePion(int x, int y, int joueur, vector<vector<int>>* G);
int aGagne(int joueur, vector<vector<int>> G);
int partieTerminee(vector<vector<int>> G);
void joueurIA(int joueur, vector<vector<int>>* G);

#endif
```

 [codeTP/codeTP-IX-2-48.c]

Notez que dans chacun de ces fichiers, on a du ajouter `#include<vector>` parce que les prototypes eux-mêmes utilisent le type `vector`.

Ensuite, on a ajouté les séquences `#ifndef`, `#define` et `#endif` pour éviter les inclusions cycliques. On ne va pas rentrer dans le détail de ça, c'est une protection à ajouter systématiquement, et sous une forme qui ressemble à celle qui est donnée ici, inspirée par le nom du fichier .h

Et maintenant il reste juste à indiquer aux fichiers qui en ont besoin les .h qui sont utiles. Ainsi on va ajouter :

- à **main.cpp** l'inclusion de **jeu.h** et de **grille.h** parce que le **main** utilise des fonctions issues de **jeu.cpp** et **grille.cpp**. Cela va donner, en haut de **main.cpp** :

```
#include<vector>
#include<iostream>
#include<cstdlib>
#include<ctime>
```

```
#include "grille.h"
#include "jeu.h"
using namespace std;
```

- à **grille.cpp** l'inclusion de seulement **grille.h**, parce qu'il est habituel (pour des raisons pas décrites ici) d'ajouter le **.h** dans le **.cpp** du même nom. Cela va donner, en haut de **grille.cpp** :

```
#include<vector>
#include "grille.h"
using namespace std;
```

- à **jeu.cpp** l'inclusion de **jeu.h** et de **grille.h**, parce que **grille.h** contient des prototypes utilisés dans **jeu.cpp**. Cela va donner, en haut de **jeu.cpp** :

```
#include<vector>
#include<iostream>
#include"grille.h"
#include"jeu.h"
using namespace std;
```

Il reste à recompiler exactement comme précédemment :

```
g++ main.cpp grille.cpp jeu.cpp -o morpion
```

Et tout est bon.

Vous avez pu remarquer les "" au lieu des <> au niveau des **#include**. C'est simplement parce que les **.h** que nous avons créé sont stockés dans le même répertoire que les **.cpp** que nous avons, alors que les <> indique d'aller chercher dans les répertoires où le compilateur est installé.

## IX.3 Utilisation d'un IDE – installation et prise en main

On l'a vu, nos programmes se complexifient au point que l'on a besoin de les découper en plusieurs fichiers. Quand on arrive à un certain niveau de complexité, on peut avoir des dizaines ou des centaines de fichiers, eux mêmes organisés dans des sous-répertoires, et juste utiliser la commande **g++** avec tous les noms de fichiers devient rébarbatif et peu pratique.

Alors il existe des outils pour gérer ce que l'on appelle des **projets C/C++**. Certains sont simplement en ligne de commande (**Make**, **Cmake**, et bien d'autres), d'autres utilisent un environnement graphique.

Au second semestre nous utiliserons un environnement graphique pour gérer cela, alors c'est ce que nous allons faire ici. Ce genre d'outil graphique porte le nom de **IDE**, pour **Integrated Development Environment**, soit **Environnement de Développement Intégré**. Ces outils se présentent comme une sorte de traitement de texte dédié à la programmation, et qui comportent des outils pour gérer des **projets C/C++** et lancer leur exécution : ils facilitent les choses lorsque les programmes deviennent complexes (mais sont souvent inutiles voire encombrants pour écrire des programmes simples). C'est ce dont on va parler ici.

Il existe des dizaines d'outils pour cela, et c'est vrai pour à peu près tous les langages, et le C/C++ ne fait pas exception. Nous allons utiliser ici l'un des plus répandus (et qui ne sert pas qu'au C/C++ mais pour énormément de langages) : il s'agit d'**Eclipse**, qui est gratuit, opensource et fonctionne sous Linux-Mac-Windows. C'est ce même environnement qui sert au développement pour les microcontrôleurs par exemple, mais ici on va simplement l'utiliser pour faire du C/C++ standard.

### Installation de Eclipse à l'Université

Eclipse n'est pas installé par défaut sur les machines de l'université, mais on peut l'installer nous-mêmes. **Suivez le tutoriel vidéo ci-dessous pour installer Eclipse.**

Commencez par télécharger Eclipse :

<http://dl.eea-fds.umontpellier.fr/CppInstall/eclipse-inst-jre-linux64.tar.gz>

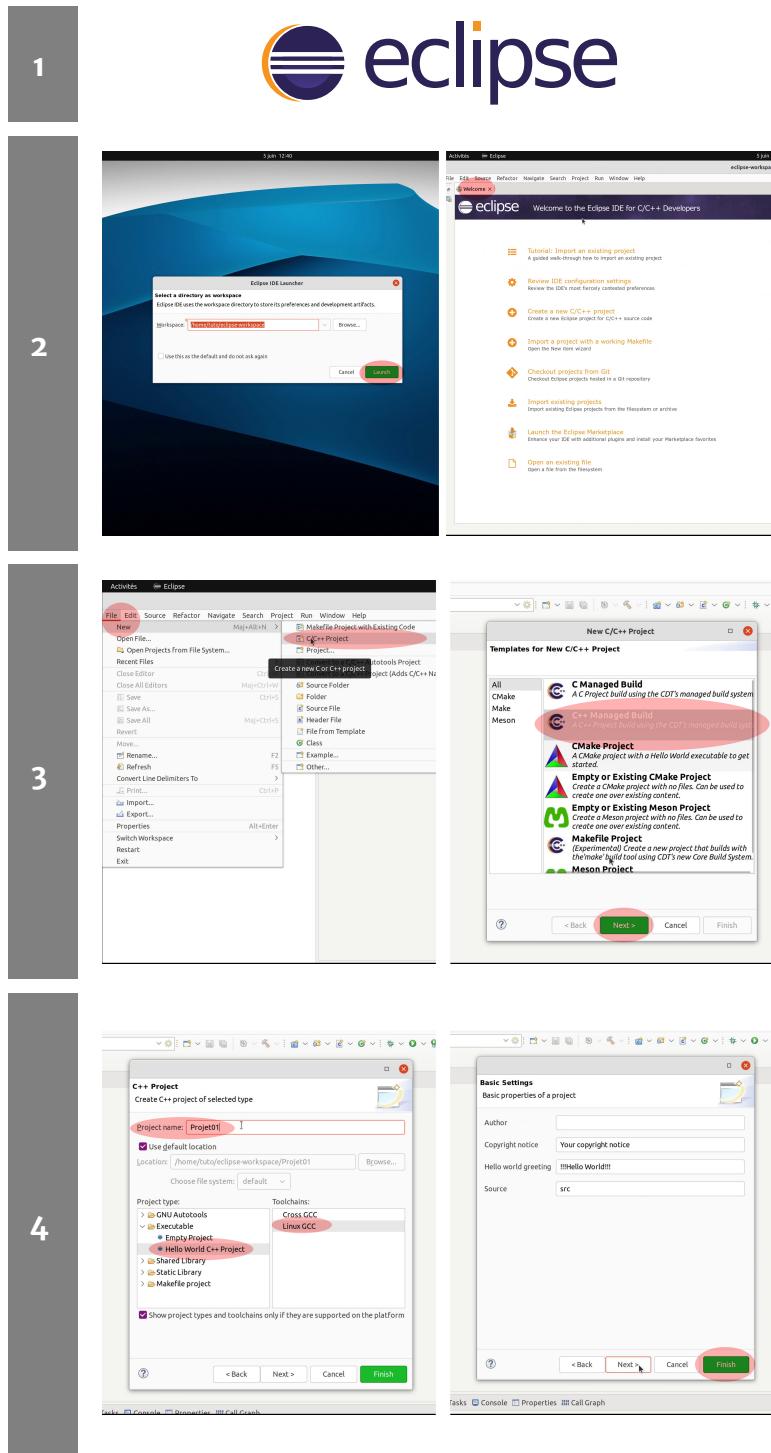
Puis suivez ce tutoriel (avec sous-titrage) :

<http://dl.eea-fds.umontpellier.fr/CppInstall/tuto-eclipse-linux.mp4>

Note : vous pouvez disposer de Eclipse sous Windows pour votre ordinateur personnel, il fait partie des outils installés avec la procédure d'installation proposée dans la préface de ce document.

## Utilisation de Eclipse

Voici comment créer un projet minimal avec Eclipse :



Lancez Eclipse à partir du menu représenté par un logo en forme de cercle rouge en haut à gauche de l'écran.

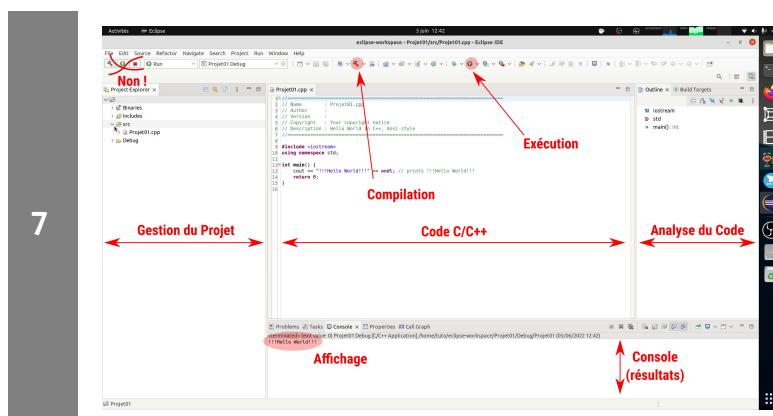
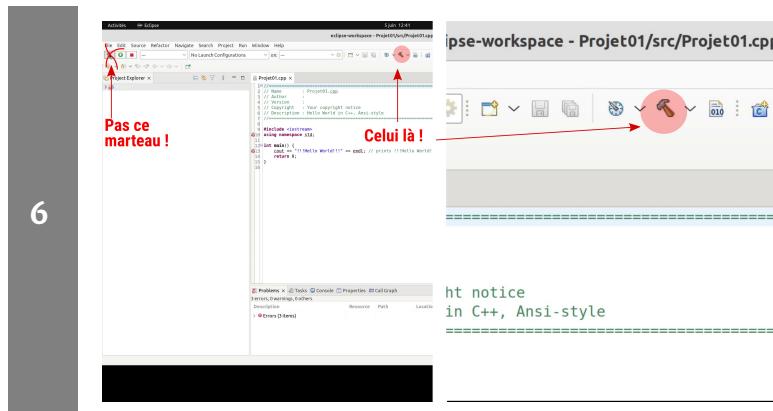
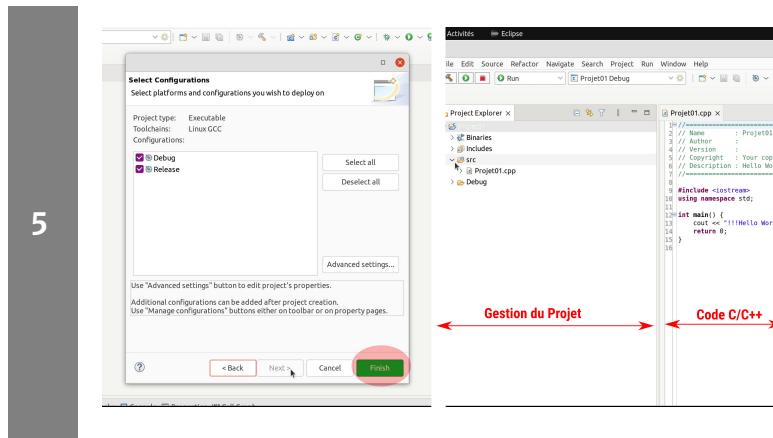
**2.a - Image de gauche :** Apparaît alors un dialogue qui demande dans quelle partie du disque dur il faut travailler. La proposition par défaut convient, validez simplement.

**2.b - Image de droite :** Après un peu d'attente, la page de bienvenue d'Eclipse s'ouvre. Cliquez sur la croix de fermeture à l'endroit indiqué sur l'image.

**3.a - Image de gauche :** Créez un nouveau projet. Pour cela, allez dans **File ▶ New ▶ C/C++ Project**

**3.b - Image de droite :** Un dialogue s'ouvre qui vous demande quel mode de gestion du projet doit être utilisé. Cliquez sur **C++ Managed Build**, puis **Next**.

**4.a - Image de gauche :** Un nouveau dialogue demande le nom du projet ainsi que quelques paramètres. Pour le nom on a choisi **Projet01** mais ça n'est pas obligatoire. Par contre, il est utile de choisir un nom simple, sans caractères spéciaux (pas de caractères accentués par exemple) ni espace. Ensuite, il faut sélectionner "**Hello World C++ Project**" et **Linux GCC** au niveau des options, puis **Next**.  
**4.b - Image de droite :** Un nouveau dialogue demande des informations facultatives, cliquez simplement sur **Next**.



**5.a - Image de gauche :** Un nouveau dialogue pose des questions sur le type de compilation qu'il faut réaliser, laissez tout par défaut, puis **Finish**.

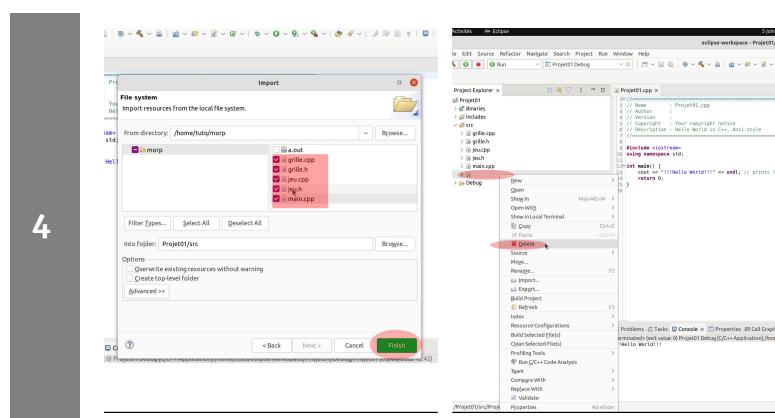
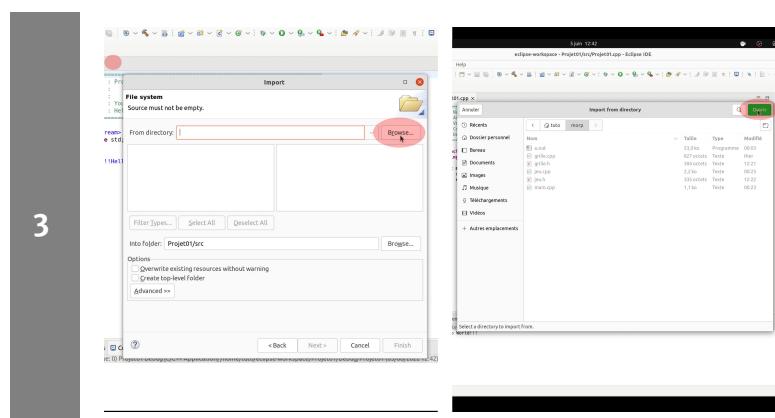
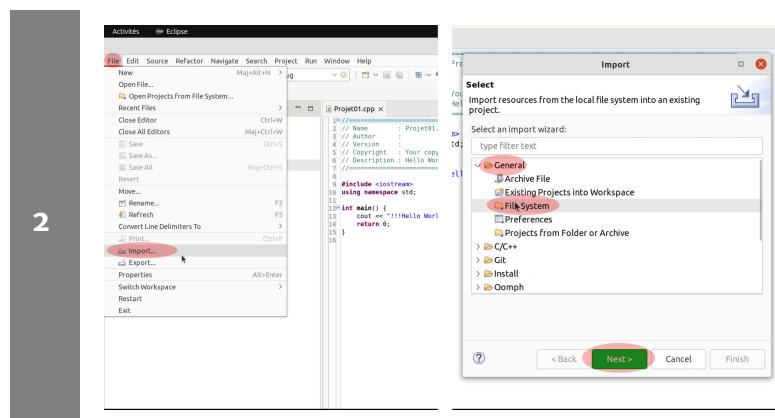
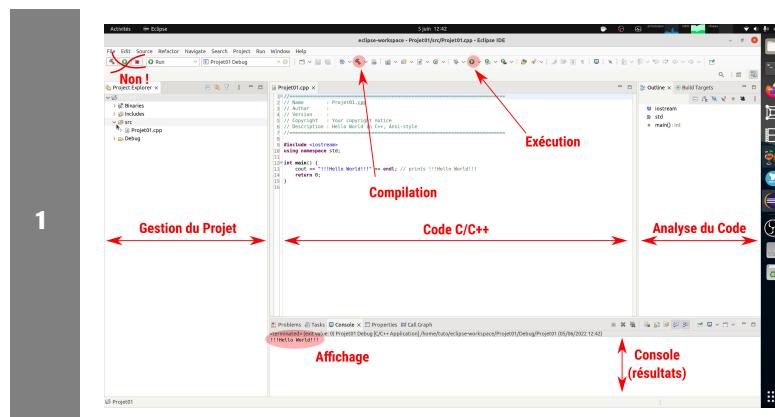
**5.b - Image de droite :** Le projet est créé. Vous êtes alors dans le mode de fonctionnement principal d'Eclipse, avec à gauche tout ce qui concerne le projet, avec la liste des fichiers .cpp qu'il contient (ici un seul, comme on le voit), et à droite la partie "traitement de texte" qui affiche le code.

**6.a et 6.b :** Lancez une première compilation du projet. C'est l'icône avec un marteau, comme montré sur l'image. Ne pas utiliser l'autre icône marteau à gauche !

Le projet est prêt. Utilisez l'icône "marteau" pour compiler, et l'icône "play" pour lancer le programme, en veillant bien de ne pas utiliser les icônes du côté gauche, comme montré sur l'image. Les résultats apparaissent en bas, dans la partie "console".

## IX.4 Utilisation d'un IDE - projet multi-fichiers

On va maintenant reprendre le projet multi-fichiers que l'on a créé tout à l'heure, mais sous Eclipse. A partir du projet que l'on vient de créer, on va importer les fichiers déjà créés :



La compilation du projet se fait maintenant simplement en cliquant sur le marteau, on n'a plus besoin de spécifier la liste des fichiers à compiler à chaque fois : par rapport à l'approche en ligne de commande rudimentaire que nous avons vue (il y a beaucoup beaucoup mieux), c'est plus simple si on a beaucoup de fichiers.

On repart du projet précédent.

**2.a - Image de gauche :** Allez dans **File ▶ Import...**.

**2.b - Image de droite :** Un dialogue s'ouvre, sélectionnez **General ▶ File System**, puis **Next >**.

**3.a - Image de gauche :** Dans le dialogue qui s'ouvre, sélectionnez "Browse".

**3.b - Image de droite :** Un sélecteur de fichiers s'ouvre. **Allez jusqu'au répertoire qui contient main.cpp, grille.cpp, jeu.cpp, grille.h, et jeu.h**, puis **Ouvrir**.

**4.a - Image de gauche :** Dans la partie droite, ne sélectionner **que main.cpp, grille.cpp, jeu.cpp, grille.h, et jeu.h**, puis **Finish**.

**4.b - Image de droite :** Puis il faut supprimer le fichier qui était présent dans le projet à sa création, puisqu'il contient une fonction **main** et qu'on en a déjà une. Pour cela, cliquez avec le bouton droit sur le fichier **Projet01.cpp** et faites "Delete", puis validez.

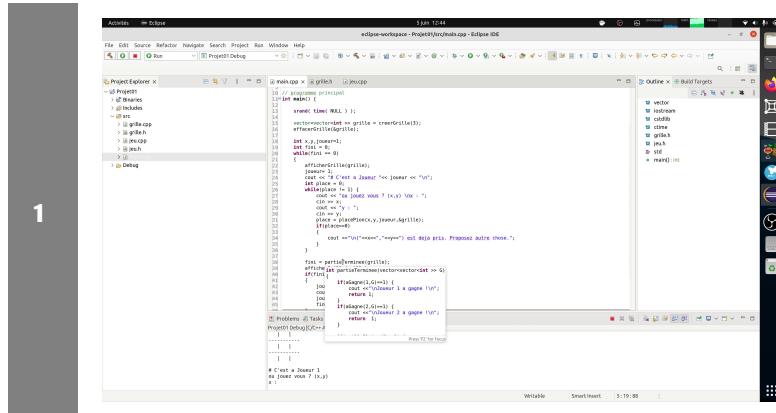
Si vous vous êtes perdus en chemin, il y a un tutoriel vidéo :

<https://dl.eea-fds.umontpellier.fr/CppL2/mat/9/eclipse-multifile.mp4>

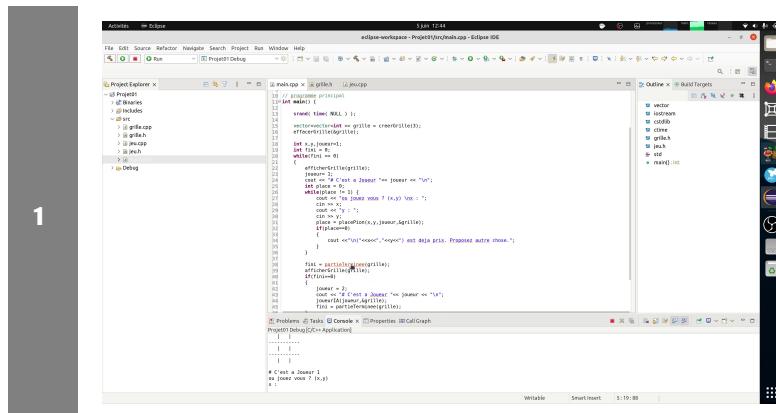
## IX.5 Utilisation d'un IDE - autres bénéfices

Les bénéfices d'un IDE vont bien plus loin. Par exemple, il y a un ensemble d'outils plus ou moins sophistiqués pour faciliter le développement.

### Exemple 1 :



### Exemple 2 :



Il y aurait encore des dizaines d'exemples, comme l'assistance pendant que vous tapez du code (si vous commencez à taper le nom d'une variable, structure ou fonction qui existe déjà, Eclipse peut vous aider à la compléter automatiquement). Mais on va voir que, en plus de tout ça, les IDE contiennent en général un outil très pratique : les Debuggers.

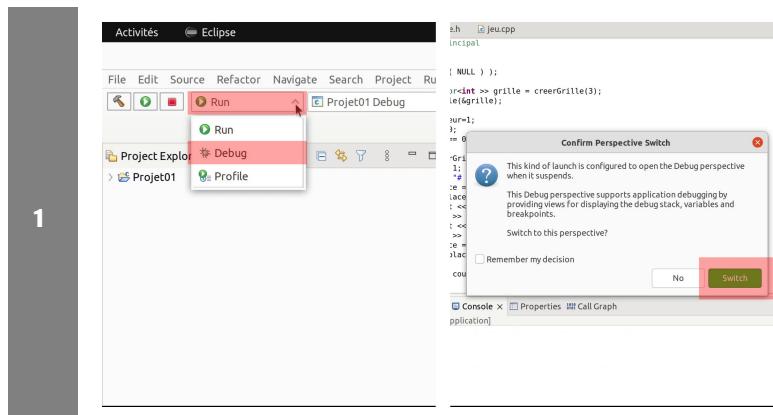
## IX.6 Utilisation d'un IDE - le Debugger

Un Debugger est un outil qui porte bien son nom : c'est une aide pour enlever les "Bugs", c'est à dire les dysfonctionnements d'un code. Pour cela, un Debugger permet de contrôler l'exécution d'un code, ligne par ligne, en ayant la possibilité de connaître, à chaque étape, l'état des variables. Ca évite notamment d'avoir à mettre des **cout** partout pour comprendre à quel moment le code ne fait plus ce que l'on croit, mais en beaucoup mieux.

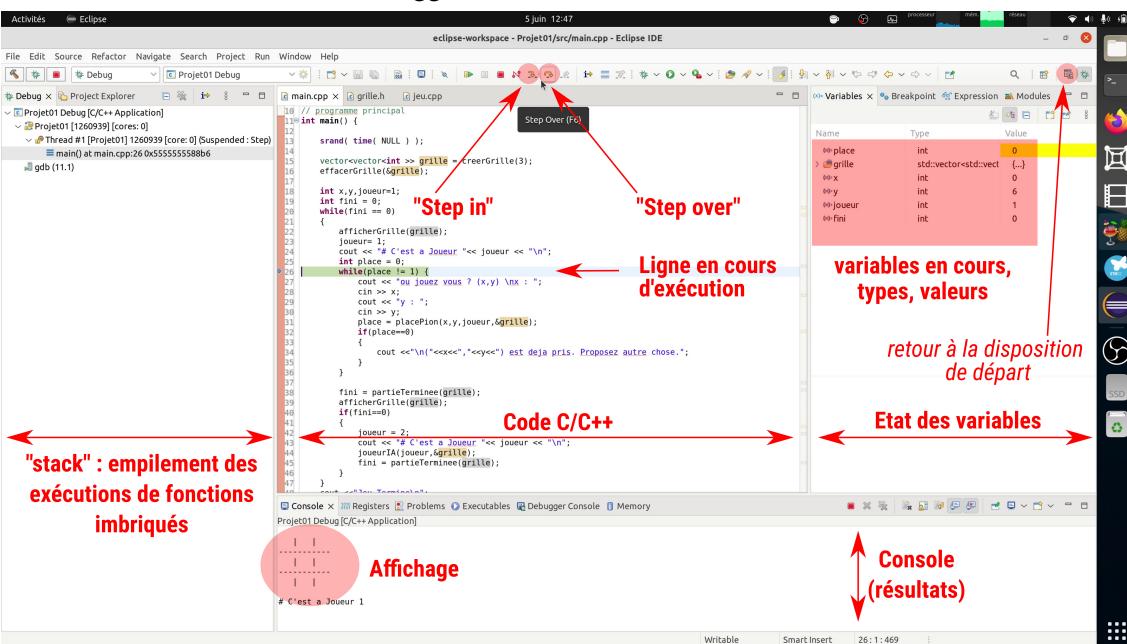
Voici sur un exemple :

Si on laisse le pointeur de la souris au niveau du nom d'une fonction à laquelle on fait appel, on voit un aperçu de son code, peu importe qu'elle soit ou pas dans le même fichier. C'est très pratique, ça permet de se rassurer notamment sur ses entrées et sorties en regardant son prototype. Un appui sur **F2** vous amène à zoomer sur le code

Si on appuie sur **ctrl** et que l'on va, avec la souris, sur le nom d'une fonction, on peut alors cliquer dessus et Eclipse nous ouvre automatiquement le fichier qui contient cette fonction et nous place à l'endroit où elle est définie. C'est très pratique si on se rend compte qu'un problème provient d'une fonction particulière, on y a accès directement.



Ensuite s'ouvre la fenêtre du Debugger :



Il faut commenter un ensemble de choses :

- La colonne centrale contient toujours le code. La ligne en cours d'exécution est surlignée en bleu.
- Pour arriver à cette ligne, on a du cliquer 5 ou 6 fois sur le bouton "step-over" : c'est le bouton qui fait avancer le programme d'un pas, et on est à 5 ou 6 lignes du début du `main`.
- En bas on voit toujours ce qui est affiché par le programme, comme précédemment.
- A gauche on voit la "stack" (l'empilement) des appels de fonction imbriqués qui ont permis d'arriver jusqu'à la ligne où on est. Comme la ligne en question est dans le `main`, la `stack` du debugger ne montre que le `main`.
- A droite une des choses les plus utiles avec le debugger : l'état des variables. On peut lire par exemple que la variable `y` vaut **6** alors que c'est impossible (il vaut forcément une valeur entre 0 et 2) : c'est simplement lié au fait que lorsque le programme atteint la ligne où nous sommes, la variable `y` n'a encore jamais été affectée, et donc elle vaut "quelque chose au hasard", ce que correspond bien au fonctionnement du C/C++. On voit aussi que `joueur` vaut 1, et c'est bien ce qui est écrit quelques lignes au dessus, sur laquelle on est forcément passé pour arriver où on est.

On a vu les fonctions essentielles du debugger et c'est déjà très puissant comme ça. Il reste 2 options supplémentaires à discuter :

- Le bouton `step-over` n'est pas le seul qui permet de faire avancer le programme. Il y a aussi `step-in`, qui est lui aussi représenté. La différence se fait lorsque la ligne à exécuter contient des appels de

**1.a - Image de gauche :** Pour basculer dans le mode "Debug", cliquez sur "Run" comme montré sur l'image, puis sélectionnez "Debug".

**1.b - Image de droite :** Un dialogue s'ouvre, Eclipse vous prévient que la disposition des éléments à l'écran va changer, cliquez sur "Switch".

fonction ou pas : **step-over** va exécuter la fonction comme "une seule ligne" de code, alors que **step-in** va nous amener à l'intérieur de la fonction pour en exécuter chaque ligne. C'est un choix qui dépend de ce dont on a besoin : si on soupçonne que le problème se situe à l'intérieur de la fonction sur laquelle on se situe, alors on fait **step-in**. Mais si on n'a aucun doute sur la fonction en question, on fait **step-over**.

- Enfin, il reste le bouton "retour à la disposition de départ" qui permet de remettre l'environnement de développement dans la disposition dans laquelle il était au lancement. C'est utile lorsque l'on a fini de débugguer et que l'on veut recommencer à simplement écrire du code.

Si vous vous êtes perdus en chemin, il y a un exemple en video :

<https://dl.eea-fds.umontpellier.fr/CppL2/mat/9/eclipse-debug.mp4>

## ② Problème IX.1 : Code à ranger sous Eclipse

On donne le code qui est la correction du problème VIII.5 :

```
1 #include <vector>
2 #include <iostream>
3 using namespace std;
4
5
6 typedef struct {
7     vector<int> avant;
8     int resultat;
9 } regle;
10
11
12 regle init(vector<int> a, int r);
13 vector<regle> toutesRegles();
14 void affLigne(vector<int> m) ;
15 int appliquerRegles(vector<int> m, int position, vector<regle> regles);
16 vector<int> calcLigne(vector<int> m, vector<regle> r) ;
17
18
19
20 // fonction principale
21 int main()
22 {
23     vector<regle> regles = toutesRegles();
24
25     vector<int> ligne;
26     ligne.resize(80);
27     int k;
28     for( k=0; k<ligne.size(); k++)
29     {
30         ligne[k]=0;
31     }
32     ligne[40]=1;
33
34     int l;
35     for(l=0; l<32; l++) {
36         affLigne(ligne);
37         ligne = calcLigne(ligne, regles);
38     }
39
40     return 0;
41 }
```

```
42
43
44 // gestion des règles
45 regle init(vector<int> a, int r)
46 {
47
48     regle R;
49     R.avant = a;
50     R.resultat = r;
51
52     return R;
53 }
54
55 vector<regle> toutesRegles()
56 {
57     vector<regle> T;
58     T.resize(8);
59     T[0] = init({0,0,0},0);
60     T[1] = init({0,0,1},1);
61     T[2] = init({0,1,0},1);
62     T[3] = init({0,1,1},1);
63     T[4] = init({1,0,0},1);
64     T[5] = init({1,0,1},1);
65     T[6] = init({1,1,0},1);
66     T[7] = init({1,1,1},0);
67
68     return T;
69 }
70
71
72 // gestion du calcul
73 void affLigne(vector<int> m) {
74     int k;
75     for(k=0; k<m.size(); k++)
76     {
77         if(m[k]==0) {
78             cout << " ";
79         }
80         else {
81             cout << "*";
82         }
83     }
84     cout <<"\n";
85 }
86
87
88 int appliquerRegles(vector<int> m, int position, vector<regle> regles) {
89     for(unsigned int r=0; r<regles.size(); r++) {
90         if ( m[position-1] == regles[r].avant[0]
91             && m[position] == regles[r].avant[1]
92             && m[position+1]==regles[r].avant[2]
93             )
94         {
95             return regles[r].resultat;
96         }
97     }
98 }
```

```
99     return -1; // ne devrait jamais arriver
100 }
101
102 vector<int> calcLigne(vector<int> m, vector<regle> r) {
103
104     vector<int> n;
105     n.resize(m.size());
106
107     for(unsigned int k=1; k<n.size()-1; k++)
108     {
109         n[k] = appliquerRegles(m,k,r);
110     }
111
112     n[0]=0;
113     n[n.size()-1]=0;
114
115     return n;
116 }
```

 [codeTP/codeTP-IX-6-49.c]

- A. Avec Eclipse, créez un nouveau projet C/C++ que vous appellerez **automcell**
- B. Vérifiez qu'il se compile, regardez ce qu'il affiche (agrandissez la fenêtre de la console).
- C. En vous basant sur les commentaires de ce programme, découpez le programme en 5 fichiers : **regles.h**, **regles.cpp**, **calcul.h**, **calcul.cpp** et **main.cpp**  
*Note : La structure devra être dans **regles.h** et seulement là.*  
*Remarque :* Pour créer un nouveau fichier dans Eclipse, allez dans le menu principal **File** puis **New ▶ Source File** pour un fichier **.cpp**, et **New ▶ Header File** pour un **.h**
- D. Vérifiez que tout se compile et s'exécute exactement comme avant. ■

## PARTIE III

---

### **Pont entre C "modernisé" et C Standard**

# Séance X – Pont entre C "modernisé" et C Standard

## ⌚ Motivations et objectifs

Le "C Standard" est le langage le plus utilisé sur microcontrôleurs encore aujourd'hui, même si le C "modernisé", voire même le C++ complet, sont utilisables sur les plus puissants d'entre eux. Il est donc utile de connaître les différences d'approches entre ces 2 langages, en vue du cours de microcontrôleurs.

## ❓ Problème X.1 : Un petit Quizz ...

Soit les deux programmes, l'un C/C++ l'autre C standard :

### Code C/C++

```
1 #include <vector>
2 using namespace std;
3
4 int main() {
5     vector<double> v;
6     int k;
7     v.resize(100);
8     double* p = &v[0];
9
10    return 0;
11 }
```

### Code C standard

```
1 #include <stdlib.h>
2
3 int main() {
4     int T1[100];
5     double* T2;
6     T2 = malloc(100*sizeof(double));
7     int m;
8     int* q = T1;
9     double* r = T2;
10
11    return 0;
12 }
```

A. Rappelez ce que signifie "allocation statique" et "allocation dynamique".

B. Sur l'ensemble de ces 2 programmes :

- Quelles variables sont allouées en mode statique ?
- Quelles variables sont allouées en mode dynamique ?

C. Donnez le type de toutes les variables des deux programmes. Par type, on n'entend pas "entier" ou "pointeur", mais une expression du C comme par exemple "short\*".

**D.** Parmi les expressions ci-dessous, lesquelles se compilent/ne se compilent pas, et lesquelles sont dangereuses ou pas :

<code>v[3] = 2;</code>	<code>T1[3] = 2;</code>	<code>T2[3] = 2;</code>
<code>*v = 2;</code>	<code>*T1 = 2;</code>	<code>*T2 = 2;</code>
<code>*(v+3) = 2;</code>	<code>*(T1+3) = 2;</code>	<code>*(T2+3) = 2;</code>
<code>p[5] = 8;</code>	<code>q[5] = 8;</code>	<code>r[5] = 8;</code>
<code>q = &amp;T1</code>		

## ② Problème X.2 : Tableaux statiques et dynamiques

On donne le code ci-dessous :

 [codeTP/codeTP-X-0-52.c]

**A.** Convertissez le en C standard et vérifiez qu'il fonctionne toujours.

## ③ Problème X.3 : Chaines de caractères et affichage

On part du code de base ci-dessous :

```

1 #include <stdio.h>
2
3 int main() {
4     char s1[]="Bonjour";
5     char s2[]="a tous !";
6
7     return 0;
8 }
```

 [codeTP/codeTP-X-0-53.c]

**A.** Ecrivez une fonction :

`int tailleChaine(char* s);`

qui renvoie le nombre de caractères contenus dans la chaîne. On rappelle qu'une chaîne de caractère en C se termine par la valeur 0.

**B.** Ecrivez une fonction capable de prendre en paramètre 2 chaînes, et de renvoyer une nouvelle chaîne qui est la concaténation des deux chaînes passées en entrée.

Par exemple, si on passe "Hello" et "World", la fonction doit renvoyer (et non afficher) "HelloWorld".

**C.** Faites appel à chacune des deux fonctions dans le `main` ci dessus en utilisant `s1` et `s2`. Utilisez ensuite `printf` de `stdio.h` pour afficher la chaîne concaténée et sa taille.

**D.** Regardez sur internet le contenu de la bibliothèque `string.h` du C standard et regardez si vous ne retrouvez pas ces fonctions, déjà toutes faites, dans les bibliothèques. Utilisez les alors dans le `main`.

## ④ Problème X.4 : Carnet d'Adresses en C standard

On souhaite gérer un carnet d'adresses de 10 contacts maximum dans lequel les coordonnées des différents contacts seront stockées. Pour chaque contact, on souhaite stocker :

- le Nom, sur 30 caractères au maximum,
- le Prénom, sur 30 caractères au maximum,
- l'âge, sous forme d'un `long`.

Pour cela, on propose les étapes suivantes :

**A.** Proposez une structure pour représenter un contact unique.

**B.** Ecrivez une fonction permettant la saisie d'un contact et une permettant l'affichage d'un contact.

- C. Ecrivez une fonction permettant d'ajouter un contact à la liste des contacts.
- D. Ecrivez une fonction permettant de supprimer un contact de la liste des contacts. La suppression se fera en connaissant le nom du contact.  
*Note : vous aurez besoin de la fonction strcmp de string.h. Documentez vous dessus et réfléchissez à pourquoi son usage est nécessaire ici.*
- E. Faites une gestion complète avec un menu permettant d'ajouter/supprimer un contact, et d'afficher la liste complète des contacts.
- F. Réécrivez ce code pour qu'il n'y ait pas de limite au nombre de caractères stockés pour le nom et pour le prénom. ■

## X.1 Pour vous entraîner

### ② Problème X.5 : Tableaux à 2 dimensions

 [codeTP/codeTP-X-1-54.c]

- A. Convertissez le en C standard et vérifiez qu'il fonctionne toujours. ■

## *Annexes*

---

## Annexe A – Bibliothèques : projet "Jeu Video"

### ⌚ Motivations et objectifs

Les bibliothèques utilisent presque toujours les ingrédients principaux du C/C++ : structures, pointeurs et fonctions. C'est ce à quoi on va se confronter avec une bibliothèque qui sert à programmer des jeux video : **raylib**.

Ce TP concerne la deuxième moitié du chapitre XI du cours. On impose ici l'utilisation d'Eclipse.

### A.1 Présentation de Raylib

On donne un code de base qui sert à afficher une image (cet exemple est issu des exemples officiels de la bibliothèque) :

```
1 #include "raylib.h"
2
3 int main(void)
4 {
5     // Initialization
6     //-----
7     const int screenWidth = 800;
8     const int screenHeight = 450;
9
10    InitWindow(screenWidth, screenHeight, "Raylib Example");
11
12    // Texture loading
13    Texture2D texture = LoadTexture("resources/raylib_logo.png");
14    //-----
15
16    // Main game loop
17    while (!WindowShouldClose())      // Detect window close button or ESC key
18    {
19        // Update
20        //-----
21        // TODO: Update your variables here
22        //-----
23
24        // Draw
25        //-----
26        BeginDrawing();
27
28            ClearBackground(RAYWHITE);
29
30            DrawTexture(texture, screenWidth/2 - texture.width/2,
31                         screenHeight/2 - texture.height/2, WHITE);
32
33            DrawText("this IS a texture!", 360, 370, 10, GRAY);
```

```

33         EndDrawing();
34         // -----
35     }
36
37     // De-Initialization
38     // -----
39     UnloadTexture(texture);           // Texture unloading
40
41     CloseWindow();                  // Close window
42     // -----
43
44
45     return 0;
46 }
```

 [codeTP/codeTP-A-1-55.c]

Le concept d'image est appelé "**Texture**" dans Raylib<sup>1</sup>. Et donc pour parler "d'image" on parlera de "texture".

Dans le code, on voit que le mot "texture" apparait ligne 13 :

```
Texture2D texture = LoadTexture("resources/raylib_logo.png");
```

Analysons un peu cette ligne :

- **Texture2D texture** est une déclaration de variable (deux mots simplement séparés par un espace, en C/C++ c'est presque forcément ça). Dans cette déclaration, **Texture2D** est forcément le nom du type, et **texture** forcément le nom de la variable. Ca signifie que **Texture2D** est un type défini par la bibliothèque Raylib (sûrement une structure).
- La fin de la ligne est un appel de fonction : un mot suivi de parenthèses, en C/C++ c'est presque forcément une fonction. Et ici c'est forcément un appel puisqu'on est déjà dans une fonction (le **main**), et qu'on n'indique pas, entre les parenthèses, le type des paramètres, mais on donne leur valeur.
- Le paramètre qui est donné est un nom de fichier **.png**.

En résumé, avec quelques bribes d'anglais et un peu d'analyse du langage, on comprend, même sans se renseigner trop sur la bibliothèque, que cette ligne charge un fichier **resources/raylib\_logo.png**, et que cette image va être stockée dans la variable **texture**.

## ② Problème I.1 : Prise en main de Raylib

On va donc adapter ce code pour charger une image de notre choix.

**A.** Commencez par créer un projet nommé **jeuVideo** avec **Eclipse** et copiez/collez le code d'exemple donné dans ce projet.

**B.** Si vous compilez le projet, vous vous rendez compte qu'il ne fonctionne pas, il indique un ensemble d'erreurs :

```
/home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:10 :
référence indéfinie vers « InitWindow »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:13 :
référence indéfinie vers « LoadTexture »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:26 :
référence indéfinie vers « BeginDrawing »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:28 :
```

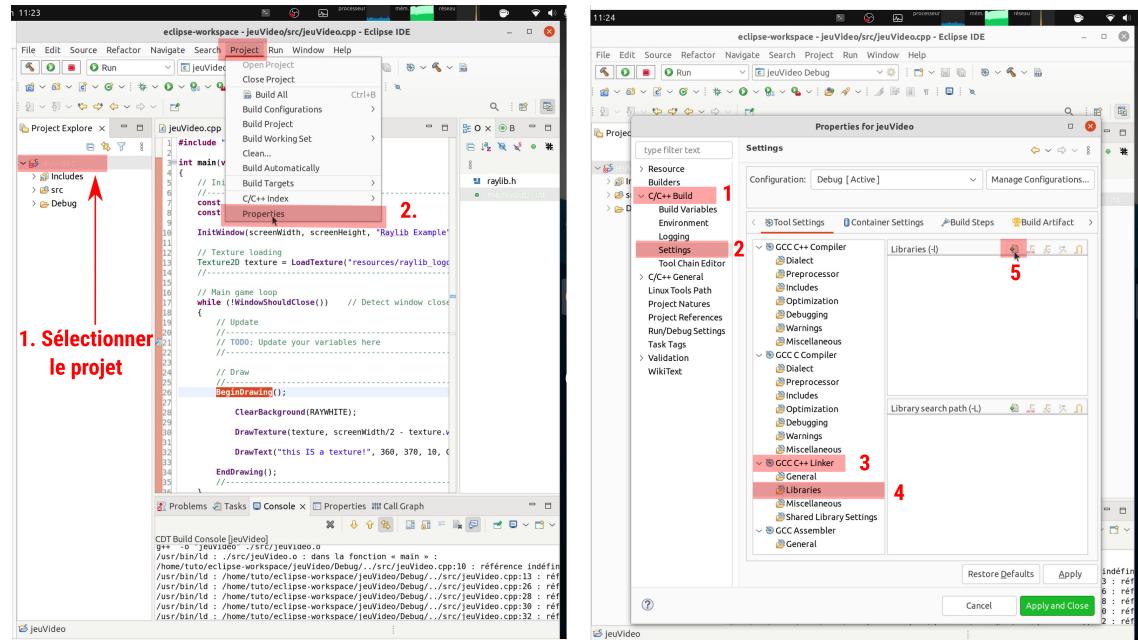
1. C'est lié au fait que Raylib est aussi capable de gérer des objets en 3 dimensions et que dans ce contexte précis, une image est appelée texture, parce qu'elle permet de donner à un objet à 3 dimensions une apparence liée à une matière donnée (bois, pierre, etc), et donc on parle de la "texture" de sa surface.

```

référence indéfinie vers « ClearBackground »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:30 :
référence indéfinie vers « DrawTexture »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:32 :
référence indéfinie vers « DrawText »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:34 :
référence indéfinie vers « EndDrawing »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:17 :
référence indéfinie vers « WindowShouldClose »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:40 :
référence indéfinie vers « UnloadTexture »
/usr/bin/ld : /home/tuto/eclipse-workspace/jeuVideo/Debug/..../src/jeuVideo.cpp:42 :
référence indéfinie vers « CloseWindow »
collect2: error: ld returned 1 exit status
make: *** [makefile:60 : jeuVideo] Erreur 1
"make all" terminated with exit code 2. Build might be incomplete.

```

On remarque qu'il y a une erreur pour chacune des fonctions que l'on ne connaît pas, et qui proviennent certainement de raylib. Et c'est normal : on n'a pas ajouté la bibliothèque Raylib au projet, on a juste mentionné l'inclusion de **raylib.h** dans le programme. Mais, vous avez pu comprendre au TP précédent que les fichiers .h ne contiennent pas de code, juste des déclarations et des prototypes. Mais on a aussi besoin du code de **rayLib**. Pour cela il faut paramétriser le projet pour ajouter la bibliothèque. Voici comment faire en deux étapes (image de gauche et image de droite) :



Une fois arrivé à l'étape 5 :

- **sous Linux** tapez simplement le nom de la bibliothèque à intégrer au projet, c'est à dire **raylib**, puis validez. Vous devez voir apparaître alors le nom de la bibliothèque apparaître dans la section "Libraries".
- **sous Windows**, commencez par taper le nom de la bibliothèque à intégrer au projet, c'est à dire **raylib**, puis validez. Vous devez voir apparaître alors le nom de la bibliothèque dans la section "Libraries". Ensuite, vous devez répéter l'étape 5, pour chacune des 3 bibliothèques ci-dessous :

opengl32

```
gdi32
winmm
```

Compilez alors le projet, et constatez que la compilation fonctionne.

- C.** Maintenant le projet se compile, mais à l'exécution il ouvre bien une fenêtre, mais il n'y a pas d'image. Et dans la console, on remarque une erreur, et la bibliothèque nous informe que :

```
WARNING: FILEIO: [resources/raylib_logo.png] Failed to open file
```

Et en effet, on ne lui a donné aucune image à afficher. Alors commençons par télécharger une image que l'on va utiliser, par exemple<sup>a</sup> : <https://dl.eea-fds.umontpellier.fr/CppL2/mat/10/paysage.png>

Une fois téléchargée, copiez la dans le répertoire de votre projet. Si vous avez suivi rigoureusement ce que l'on a fait jusqu'ici, l'image `paysage.png` devrait se trouver maintenant dans le répertoire "`~/eclipse-workspace/jeuVideo/`". Puis modifiez le code en remplaçant, ligne 13 la chaîne de caractères complète par "`paysage.png`". Recompilez et relancez le programme, vous devez voir votre image.

**D.** Jouez maintenant un peu avec les paramètres et regardez ce que cela fait, en recompilant et réexécutant à chaque fois. Par exemple changez `screenWidth` ou les paramètres de `DrawTexture`, ou ceux de `DrawText`. Faites vous une idée de ce que changent ces paramètres. N'hésitez pas à utiliser le `ctrl+click` (à la souris) pour aller chercher de la "documentation" sur Raylib au sujet des fonctions et structures, c'est très instructif.

Si vous êtes perdus, la démarche complète est montrée en vidéo ici :

<https://dl.eea-fds.umontpellier.fr/CppL2/mat/10/eclipse-raylib.mp4>

- - a. Ça peut être ce que vous voulez, il suffit que ce soit une `png`

## A.2 Dessiner le personnage animé

On donne un ensemble d'images pour un personnage animé<sup>2</sup>. On résume l'ensemble de ces images ci-dessous :



On propose une archive à télécharger :

<https://dl.eea-fds.umontpellier.fr/CppL2/mat/10/images-128.zip>

### ② Problème I.2 : Animer le Viking

---

2. Sous licence opensource . Réalisé avec les logiciels opensource **inkscape** pour le dessin et **OpenToonz** pour l'animation.

- A.** Dans cette archive, le dessin de chacun des personnages ci-dessus est contenu dans une image **.png**. Téléchargez la, ouvrez la, puis extrayez le répertoire **images-128** (par copier/coller par exemple), et copiez le dans votre projet, comme précédemment.
- B.** Modifiez votre code pour afficher le fichier **viking-face-attend-0001.png** au centre de l'écran, par dessus l'image de fond que l'on a dessiné précédemment.
- C.** Pour faire une animation, il faut regarder de plus près les fichiers contenus dans l'archive. Vous remarquez qu'il y a 24 fichiers de la forme **viking-face-attend-xxxx.png**, et si vous les regardez un par un vous visualisez bien qu'il s'agit des étapes d'une animation. Ecrivez une fonction :

```
|string chaineEtape(string nom, int etape);
 [codeTP/codeTP-A-2-56.c]
```

pour laquelle on pourrait faire un appel comme suit dans le **main** :

```
|string nom = chaineEtape("viking-face-attend", 3);
 [codeTP/codeTP-A-2-57.c]
```

Et qui renverrait dans **nom** la chaîne :

```
viking-face-attend-0003.png
```

*Note* : pour imposer au C/C++ qu'un nombre fasse une longueur fixe avec des 0 qui le précédent, il faut une syntaxe du type :

```
cout << setfill('0') << setw(5) << 25;
```

qui affichera **00025** sur cet exemple. Vous aurez besoin d'inclure **iomanip** dans votre code.

Vérifiez que la chaîne est correctement créée en l'affichant dans la console, puis utilisez la fonction **chaineEtape** avec **LoadTexture** pour charger la 7ième étape du Viking de face qui attend.

**D.** Chargez maintenant toutes les étapes du Viking de face qui attend dans un **vector** de **Texture2D**.

**E.** Enfin, faites en sorte que l'animation du personnage qui attend se fasse à l'écran : il suffit d'afficher la bonne image à chaque étape, en la récupérant du tableau précédent.

**F.** Normalement vous n'avez presque rien à faire pour voir le Viking marcher "sur place", il suffit de modifier une chaîne de caractères dans votre code. Faites le.

**G.** Maintenant, faites en sorte que le Viking marche vraiment en descendant (puisque il est de face). Peu importe si il sort de l'écran à un moment, c'est juste pour voir.

**H.** Maintenant, et en prévision de la suite, remodifiez votre code pour que le personnage reste au centre de l'écran, tout en restant animé. ■

### A.3 Dessiner une carte pour le fond

Pour dessiner la carte de fond, on a, dans la même archive, un ensemble d'images que l'on va pouvoir combiner : **carre-arbre.png**, **carre-herbe.png**, et **carre-montagne.png**<sup>3</sup>. Avec ces simples 3 éléments, on peut dessiner une carte aussi grande que l'on veut.

#### ② Problème I.3 : La carte

- A.** Retirez du code le dessin de l'image de fond que l'on a mis au début de ce TP.
- B.** Dans un **vector** de **Texture2D** que vous nommerez **carreaux**, chargez chacune des 3 images **carre-herbe.png**, **carre-arbre.png**, et **carre-montagne.png**. Maintenez cet ordre, c'est à dire que l'herbe doit être dans la case 0, l'arbre dans la case 1, la montagne dans la case 2.
- C.** Ajoutez dans votre **main** :

```
|vector<vector <int> > carte= { {2,2,2,2,2,2,2,2,2,2,2},
```

3. On a aussi un ensemble de **carre-Eau-xx.png** mais il faut les agencer correctement pour faire des lacs ou des rivières

```

{2,2,1,1,0,0,0,2,2,2,2},
{2,1,0,0,0,1,0,0,0,1,2},
{2,0,0,0,1,1,1,0,0,2,2},
{2,0,0,1,1,1,1,0,0,0,2},
{2,0,1,1,1,0,0,0,0,0,2},
{2,0,1,0,0,0,0,0,0,0,2},
{2,1,0,0,0,0,0,0,2,2,2},
{2,2,2,2,2,2,2,2,2,2,2},
];

```



[codeTP/codeTP-A-3-58.c]

Si on considère que les 0,1 et 2 correspondent aux positions des images dans le tableau **carreaux**, pouvez-vous imaginer (sans grande rigueur) le dessin de cette "carte" ? ■

- D.** Ecrivez un code qui permet de dessiner cette **carte** en utilisant les **carreaux** du tableau à l'arrière plan du viking.

## A.4 Contrôler le personnage

On veut maintenant pouvoir faire bouger le personnage selon les touches que l'on appuie sur le clavier.

### ② Problème I.4 : Touches au clavier

On ne peut plus utiliser **cin**, parce que **cin** attend que l'on valide la commande avec . Pendant tout le temps que **cin** attend, l'animation ne peut plus se faire, puisque ... pendant que **cin** attend, on ne peut rien faire. Alors on a besoin de regarder comment interroger le clavier avec **raylib**.

- A.** On donne cet exemple issu de la documentation de la bibliothèque :

```

#include "raylib.h"

int main(void)
{
    // Initialization
    //-----
    const int screenWidth = 800;
    const int screenHeight = 450;

    InitWindow(screenWidth, screenHeight, "raylib [core] example - 
        keyboard input");

    Vector2 ballPosition = { (float)screenWidth/2, (float)screenHeight/2
    };

    SetTargetFPS(60);    // Set our game to run at 60 frames-per-second
    //-----

    // Main game loop
    while (!WindowShouldClose())    // Detect window close button or ESC key
    {
        // Update
        //-----
        if (IsKeyDown(KEY_RIGHT)) ballPosition.x += 2.0f;
        if (IsKeyDown(KEY_LEFT)) ballPosition.x -= 2.0f;
        if (IsKeyDown(KEY_UP)) ballPosition.y -= 2.0f;
    }
}

```

```

    if (IsKeyDown(KEY_DOWN)) ballPosition.y += 2.0f;
//-----


    // Draw
//-----
BeginDrawing();

    ClearBackground(RAYWHITE);

    DrawText("move the ball with arrow keys", 10, 10, 20,
        DARKGRAY);

    DrawCircleV(ballPosition, 50, MAROON);

    EndDrawing();
//-----


}

// De-Initialization
//-----
CloseWindow();           // Close window and OpenGL context
//-----


return 0;
}

```

 [codeTP/codeTP-A-4-59.c]

Téléchargez le et essayez le dans un nouveau projet Eclipse. **Attention :** gardez bien votre projet précédent, ne le supprimez pas, ici on ne fait qu'un test avec ce programme.

Essayez de comprendre comment Raylib récupère les touches au clavier. Puis, reprenez le code du Viking, et faites en sorte de le faire bouger dans les 4 directions. Bien entendu, l'animation sera toujours la même pour le moment.

Maintenant, généralisez votre code et faites en sorte de charger toutes les possibilités d'animation (attente face,dos,gauche,droite et marche face,dos,gauche,droite) pour que le Viking puisse marcher ou attendre dans toutes les directions possibles.

Note : C'est assez long, il est probable que vous ne le terminiez pas pendant la séance de TP, mais vous pouvez terminer chez vous. ■

]

## A.5 Vers un jeu video

Pour faire un vrai jeu video, il faudrait aussi gérer des collisions avec la carte. Par exemple, bien que notre héros soit un Viking, il est probable qu'il soit ralenti ou bloqué par les arbres et par les montagnes.

Pour gérer cela, vous devez faire la correspondance entre la position du viking à l'écran et la position du viking sur dans le **vector carte** que nous avons créé plus haut : vous devez écrire un code qui convertit les coordonnées du viking en coordonnées dans la carte pour regarder si la case sur laquelle il est est de l'herbe ou autre chose. Vous pourrez alors mettre des conditions pour ralentir ou interdire le déplacement du Viking. Tout ça peut être un peu subtil pour que le déplacement soit fluide.

Note : il est plus simple d'écrire le code du Viking qui "ralentit" plutôt que le code du Viking qui est "bloqué" par les obstacles. Il est du coup intéressant de commencer par écrire un code où le Viking est simplement ralenti.

## Annexe B – Exécuter un code C à partir de Python

### B.1 Introduction

Le principe est le suivant :

1. créer une bibliothèque C/C++, c'est à dire un fichier **déjà compilé** qui contient des fonctions dont on va avoir besoin sous Python
2. importer la bibliothèque C/C++ dans un code Python
3. utiliser la bibliothèque C dans le code Python

#### Remarque

Il existe plusieurs façons pour faire des échanges entre Python et C/C++. La méthode que l'on propose ici n'est pas universelle, elle ne permet pas de "tout" faire, mais elle permet de faire déjà beaucoup de choses et a l'avantage d'être assez simple (comparativement aux autres).

De plus, ce qui est montré dessus ne fonctionne, de façon exacte, que sous Linux et probablement sur Mac, mais peut s'adapter assez facilement pour fonctionner sous Windows.

### B.2 Le code que l'on a envie d'écrire en Python

Tout débute ici : il faut décider, à partir de Python, de comment on veut utiliser la fonction que l'on va créer en C/C++. Cette fonction doit permettre d'avoir les entrées utiles (c'est à dire  $U_0$  et  $N$ , le nombre de termes à calculer), et fournir les valeurs en sortie.

Dans cette méthode, l'utilisation de **return** est limitée à un résultat avec un seul entier, on ne peut rien faire d'autre. Donc les résultats passeront par les paramètres, à la façon d'un passage par adresse, en C/C++. Autrement dit, on va passer à la fonction C/C++ (que l'on créera plus tard) des variables Python qu'elle devra modifier.

Compte tenu de cette remarque, on va utiliser un code Python qui ressemble au code ci-dessous pour faire appel à la fonction C/C++ que l'on créera ensuite :

```
U0 = 27
N = 120
sortie = np.zeros((N,), dtype=np.uint32)
syraccuse_du_c(U0,N,sortie)
```

 [codeTP/codeTP-II-2-6.py]

Ca n'est qu'une "maquette", pas un code définitif, mais il sert à situer les choses. En Python, on fournira donc à la fonction **syraccuse\_du\_c** la valeur de  $U_0$  puis  $N$ , qui sont des entrées. Le paramètre suivant est un **np.array** Python de  $N$  valeurs de type **uint32** (qui correspond au **unsigned long** ou **unsigned int** du C/C++), que la bibliothèque C/C++ devra remplir avec les valeurs de la suite de Syracuse. Voyons maintenant les choses côté C/C++.

### B.3 Créer la fonction C/C++

On va partir de l'exemple de la conjecture de Syracuse. On donne un exemple de code C/C++ qui calcule les valeurs :

```

#include <vector>
#include <fstream>
using namespace std;

// calcule N elements de la suite de syracuse, debutant a U0
int syracuse(unsigned int U0, unsigned int N, unsigned int* output)
{
    unsigned int U;
    unsigned int k;

    U = U0;
    k = 0;
    output[k]=U;
    k++;

    while (k < N)
    {
        if (U % 2 == 0)
        {
            U = U / 2;
        }
        else
        {
            U = 3*U+1;
        }

        output[k]=U;
        k = k+1;
    }
    return 0; // toujours renvoyer 0 avec cette methode
}

```

 [codeTP/codeTP-B-3-60.c]

Commentons :

- On a créé une fonction **syracuse** qui prend en paramètres des entiers pour  $U_0$  et  $N$
- Pour faire sortir les valeurs, comme on l'a montré en Python, on a un 3ieme paramètre de type **unsigned int\***, qui est un pointeur qui va pointer sur les valeurs du **np.array** créé dans Python.

## B.4 Créer la bibliothèque C/C++

Pour que cette fonction soit "comprise" comme une bibliothèque C/C++, il faut ajouter encore des petites choses :

```

1 #include <vector>
2 #include <fstream>
3 using namespace std;
4
5
6 // force l'utilisation de la convention d'appel C
7 extern "C" {
8 int syracuse(unsigned int U0, unsigned int N, unsigned int* output);
9 }
10
11
12 // calcule N elements de la suite de syracuse, debutant a U0

```

```

13 int syracuse(unsigned int U0, unsigned int N, unsigned int* output)
14 {
15
16     unsigned int U;
17     unsigned int k;
18
19     U = U0;
20     k = 0;
21     output[k]=U;
22     k++;
23
24     while (k < N)
25     {
26         if (U % 2 == 0)
27         {
28             U = U / 2;
29         }
30         else
31         {
32             U = 3*U+1;
33         }
34
35         output[k]=U;
36         k = k+1;
37     }
38     return 0; // corresponds to python assumed return type by default
39 }
```

 [codeTP/codeTP-B-4-61.c]

On a juste ajouté les lignes 6 à 9, qui forcent la "convention d'appel" type "C" plutôt que "C++". Ca n'est pas facile à décrire rapidement. Disons pour faire simple que le protocole qui permet de faire appel à une fonction en C n'est pas identique à celui que l'on utilise en C++ (en apparence sur le code que l'on écrit il n'y a pas de différence, mais le compilateur fait les choses différemment). Et pour une bibliothèque, on doit utiliser le protocole type "C" pour les fonctions que Python va voir.

Maintenant, si on compile de code de la même façon que d'habitude, on n'obtiendra pas une bibliothèque<sup>1</sup>. Il faut demander au compilateur de ne pas générer un exécutable mais une bibliothèque, et cela se fait en ajoutant simplement quelques options. Voici :

```
g++ -fPIC -shared syracuse.cpp -o syracuse.so
```

*Remarque* : Ceci est valable sous Linux et fonctionne sûrement sur Mac. Sous Windows il faudrait adapter un peu.

On obtient alors le fichier **syracuse.so**, de type **.so**, qui est le format des bibliothèques pour Linux (et sûrement compréhensible pour un Mac).

## B.5 Importer la bibliothèque dans Python

On va maintenant importer la bibliothèque dans Python. La difficulté principale est que C/C++ est type explicitement (c'est à dire qu'on doit déclarer les variables), alors que Python non. On va donc devoir utiliser un certain protocole pour indiquer à Python quels sont les types à utiliser avec nos fonctions C/C++. Voici comment cela s'écrit :

```

1 import ctypes
2 from ctypes import c_uint, POINTER, cast
3 import numpy as np
4
5 # chargement de la bibliotheque
```

1. En fait on n'obtiendra rien parce qu'il manquerait un **main**

```

6 lib = ctypes.cdll.LoadLibrary('./syraccuse.so')
7 # on recuperere la fonction "syraccuse" de la bibliotheque
8 syraccuse_du_c = lib.syraccuse #
9 # on declare a Python les parametres de la fonction C
10 # soit 3 parameters : 2 uint and 1 uint* (uint = unsigned int pour ctypes)
11 syraccuse_du_c.argtypes=[c_uint,c_uint,POINTER(c_uint)]

```

 [codeTP/codeTP-II-5-7.py]

A partir de là, on a fait le travail qu'il fallait faire : on a importé la bibliothèque, récupéré la fonction C, et décrit à Python les paramètres. A partir de maintenant, on peut utiliser `syraccuse_du_c` comme une fonction Python.

## B.6 Utiliser la fonction C dans Python

On revient à quelque chose de très proche de ce que l'on avait écrit au tout début de ce chapitre : on va indiquer à la fonction importée les paramètres à prendre, et utiliser les résultats. Cela démarre comme on l'a déjà indiqué :

```

1 import ctypes
2 from ctypes import c_uint,POINTER,cast
3 import numpy as np
4
5 # chargement de la bibliotheque
6 lib = ctypes.cdll.LoadLibrary('./syraccuse.so')
7 # on recuperere la fonction "syraccuse" de la bibliotheque
8 syraccuse_du_c = lib.syraccuse #
9 # on declare a Python les parametres de la fonction C
10 # soit 3 parameters : 2 uint and 1 uint* (uint = unsigned int pour ctypes)
11 syraccuse_du_c.argtypes=[c_uint,c_uint,POINTER(c_uint)]
12
13
14 # on prepare l'appel a la fonction : on definit les valeurs des parametres
15 U0 = 27
16 N = 120
17 resultats = np.zeros((N,), dtype=np.uint32)
18 # et voici l'appel lui meme
19 syraccuse_du_c(U0,N,cast(resultats.ctypes.data,POINTER(c_uint)))
20
21
22 # et on affiche le resultat
23 import matplotlib.pyplot as plt
24 plt.plot(resultats,'r.')
25 plt.show()

```

 [codeTP/codeTP-II-6-8.py]

Jusqu'à la ligne 12, c'est exactement ce que l'on a déjà écrit pour importer la fonction. Les lignes 15 à 19 donnent des valeurs aux paramètres que l'on va utiliser dans la fonction.

La ligne 20 est plus complexe à comprendre : elle fait appel à la fonction `syraccuse_du_c` en fournissant 3 paramètres. Pour les deux premiers, `U0` et `N`, c'est simple. Mais le 3ième vaut :

`cast(resultats.ctypes.data,POINTER(c_uint))`

Et ça semble très obscur ! En fait, cette ligne consiste d'abord à récupérer l'adresse des valeurs du `np.array` appelé `resultat` : c'est le sens à donner à `resultats.ctypes.data`. Mais cette adresse n'est pas du bon type : elle est par défaut de type C `void*`. Et il faut donc la convertir en `unsigned int*` pour que cela soit accepté par la fonction. Et c'est le sens à donner à `cast` et à `POINTER(c_uint)`. En résumé, tout ce paramètre s'écrirait, en C, `(unsigned int*)(resultats.ctypes.data)`, sauf qu'on est en Python. On voit bien ici l'aspect orienté "bas niveau" du C/C++, qui décrit facilement les adresses et les pointeurs, et la complexité que

l'on a à décrire cela avec Python, qui n'a, la plupart du temps, rien à faire avec le concept d'adresse ou de pointeur.

Au final, on trace dans un **plot** les données.

## B.7 Une écriture plus élégante

Si on doit faire plusieurs fois des appels à la fonction **syraccuse\_du\_c**, on va devoir réécrire à chaque fois la conversion du pointeur, et c'est pénible. Alors, à la place, on peut réécrire le code ci-dessus comme ceci :

```

1 import ctypes
2 from ctypes import c_uint, POINTER, cast
3 import numpy as np
4
5 # chargement de la bibliotheque
6 lib = ctypes.cdll.LoadLibrary('./syraccuse.so')
7 # on recupere la fonction "syraccuse" de la bibliotheque
8 syraccuse_du_c = lib.syraccuse #
9 # on declare a Python les parametres de la fonction C
10 # soit 3 parameters : 2 uint and 1 uint* (uint = unsigned int pour ctypes)
11 syraccuse_du_c.argtypes=[c_uint,c_uint,POINTER(c_uint)]
12
13
14 # definition d'une fonction Python qui utilise syraccuse_du_c :
15 def syraccuse(U0,N):
16     r = np.zeros((N,), dtype=np.uint32)
17     syraccuse_du_c(U0,N,cast(r.ctypes.data,POINTER(c_uint)))
18     return r
19
20 # maintenant on peut ecrire facilement les appels a syraccuse_du_c :
21 resultats = syraccuse(27,120)
22
23 # et on affiche le resultat
24 import matplotlib.pyplot as plt
25 plt.plot(resultats,'r.')
26 plt.show()

```



On a juste créé une fonction Python qui gère le protocole d'utilisation de **syraccuse\_du\_c**

### Remarque

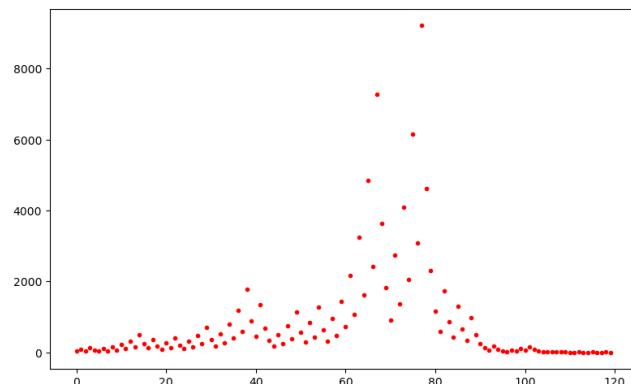
A ce stade si vous êtes un peu perdus, ce qu'il vous faut faire pour faire fonctionner ça :

- Récupérer le code de §B.4, le coller dans un fichier nommé par exemple **syraccuse.cpp**, puis le compiler comme bibliothèque avec le terminal, en faisant :
 

```
g++ -fPIC -shared syraccuse.cpp -o syraccuse.so
```
- Récupérer le code de §B.7, le collé dans un fichier nommé par exemple **main.py**, puis l'exécuter en tapant dans le terminal :
 

```
python3 main.py
```

Et vous obtiendrez :



## Annexe C – Gestion des fichiers

### ⌚ Motivations et objectifs

Ce TP concerne le chapitre XII du cours.

### ❓ Problème III.1 : Fichier texte ou binaire

On veut sauver dans un fichier les valeurs de la suite de Syracuse. On redonne le code qui permet de générer les valeurs de cette suite :

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

int main ()
{
    int U;
    int k;

    U = 27;
    k = 0;

    while (k < 120)
    {
        if (U % 2 == 0)
        {
            U = U / 2;
        }
        else
        {
            U = 3*U+1;
        }
        k = k+1;
        cout << "U" << k << " = " << U << "\n";
    }

    return 0;
}
```

🌐 [codeTP/codeTP-C-0-62.c]

A. Modifiez ce code pour stocker dans un `vector<int>` les séquences  $(k, U_k)$  dans l'ordre des  $k$  croissants. Autrement dit, les cases paires contiendront la valeur de  $k$ , et les valeurs impaires les valeurs de la suite de Syracuse. Votre tableau doit donc contenir :

1 82 2 41 3 124 etc.

B. Vérifiez que cela fonctionne en écrivant une nouvelle boucle qui affiche les valeurs du tableau dans l'ordre.

C. Utilisez maintenant les fonctions "mode binaire" vues dans le cours pour la gestion des fichiers

pour enregistrer la totalité du tableau dans un fichier "**syraccuse.txt**".

*Note :* L'extension ".txt" n'est pas adaptée pour le fichier que nous créons, mais c'est ce qui est voulu ici.

**D.** Ouvrez le fichier avec un éditeur de texte tel que **gedit**. Vous observez n'importe quoi. Est-ce normal ?

**E.** Regardez maintenant le fichier **syraccuse.txt** avec le terminal : placez vous dans le répertoire où ce fichier a été créé, puis faites :

```
hexdump -C syraccuse.txt
```

Vous allez alors voir la décomposition binaire de votre fichier. Identifiez les valeurs, essayez de voir si vous pouvez comprendre ce que contient le fichier, et pourquoi c'est logique.

**F.** Ce que l'on vient de faire, c'est créer un fichier Binaire. Mais il serait pratique, dans beaucoup de cas, de disposer d'un fichier Texte, c'est à dire lisible par **gedit**. Transformez alors votre programme pour qu'il stocke les nombres sous forme de texte dans un nouveau fichier. La valeur de  $k$  et de  $U_k$  seront séparées par une virgule pour ne pas les confondre, et après une séquence  $k, U_k$  vous ajouterez un retour à la ligne.

**G.** Le type de fichier que vous venez de créer est un fichier "CSV", c'est à dire un fichier qui contient des valeurs en tant que texte, mais séparées par des virgules ("Comma Separated Values"). Vous avez déjà vu l'an dernier qu'on pouvait charger ce genre de fichier avec Python, mais on peut aussi le charger avec Excel ou LibreOffice Calc. En utilisant l'une de ces 3 solutions, tracez la courbe  $U_k = f(k)$  à partir des données du fichier CSV. ■

## ② Problème III.2 : Carnet d'adresses "version Fichiers"

On veut stocker des contacts dans un carnet d'adresses avec un fichier, pour pouvoir conserver les informations. L'idée est de stocker les contacts les uns à la suite des autres dans le fichier. Les informations que l'on veut stocker sont :

- Le nom, sur 20 caractères maximum,
- Le prénom, sur 20 caractères maximum,
- la date de naissance, sous forme de 3 entiers (jour, mois et année)

**A.** Réfléchissez à "pourquoi" stocker ces informations en utilisant un "codage binaire" plutôt qu'un "codage texte" est préférable.

*Une piste de réflexion :* la réponse est liée au fait que si on écrit des données c'est pour pouvoir les relire le plus facilement possible. Mais pourquoi exactement ?

**B.** Ecrivez un **main** qui ouvre un fichier **carnet.dat** en écriture seulement.

**C.** Complétez le **main** pour écrire 3 contacts dans le fichiers, les 3 contacts étant saisis au clavier. Vous éviterez les caractères accentués lors de votre saisie. Regardez ensuite, avec **hexdump**, comme à l'exercice précédent, le contenu du fichier, regardez ce que vous pouvez reconnaître.

**D.** Ecrivez un nouveau programme qui ouvre le fichier **carnet.dat** en lecture seulement, puis qui lit les 3 contacts.

**E.** Modifiez votre programme maintenant pour qu'il ne lise que le dernier contact du carnet. Vous utiliserez le "curseur" sur fichiers vu en cours pour cela.

**F.** Ecrivez maintenant un programme complet qui propose un menu :

1. Ajouter un contact
2. Afficher tous les contacts
3. Afficher un contact par sa position
4. Afficher un contact connaissant son nom
5. Sortir du programme

| **G.** Pourquoi est-il difficile de supprimer un contact dans le fichier ? Proposez une solution. ■

## C.1 Pour vous entraîner

### (?) Problème III.3 : Analyse d'un fichier WAV

On donne un fichier son au format WAV<sup>a</sup> :

<https://dl.eea-fds.umontpellier.fr/CppL2/mat/11/Applause.wav>

On va le télécharger et essayer de comprendre ce que l'on peut comprendre. La première chose à comprendre dans un fichier c'est ce que l'on appelle le "header" ou "en-tête" : il s'agit de l'ensemble des informations placées au tout début du fichier. Pour faire son analyse, on a récupéré quelque part sur internet la documentation ci-dessous :

Offset in Bytes	Example Value	Description
0 - 3	"RIFF"	Marks the file as a riff file. Characters are each 1 byte long.
4 - 7	File size (integer)	Size of the overall file - 8 bytes, in bytes (32-bit integer). Typically, you'd fill this in after creation.
8-11	"WAVE"	File Type Header. For our purposes, it always equals "WAVE".
12-15	"fmt "	Format chunk marker. Includes trailing null
16-19	16	Length of format data as listed above
20-21	1	Type of format (1 is PCM) - 2 byte integer
22-23	2	Number of Channels - 2 byte integer
24-27	44100	Sample Rate - 32 byte integer. Common values are 44100 (CD), 48000 (DAT). Sample Rate = Number of Samples per second, or Hertz.
28-31	176400	(Sample Rate * BitsPerSample * Channels) / 8.
32-33	4	(BitsPerSample * Channels) / 8 1 - 8 bit mono 2 - 8 bit stereo/16 bit mono 4 - 16 bit stereo
34-35	16	Bits per sample
36-39	"data"	"data" chunk header. Marks the beginning of the data section.
40-43	File size (data)	Size of the data section.
Sample values are given above for a 16-bit stereo source.		

On donne un peu d'aide pour les termes :

- Channel : voie. Par exemple, 1 voie signifie mono, 2 voies signifie stereo
- Sample Rate : fréquence d'échantillonnage
- BitsPerSample : nombre de bits par échantillon (=par amplitude)

**A.** Regardez le contenu du fichier à l'aide du terminal en tapant<sup>b</sup> :

```
hexdump -C -n100 Applause.wav
```

et regardez si vous arrivez à :

- dire si le fichier est bien un WAV
- si le codage est "PCM" (Pulse Code Modulation),
- Le nombre de bits utilisé pour les amplitudes,
- Si c'est un signal mono ou stereo,

**B.** Ecrivez maintenant un programme C/C++ qui sait dire si un fichier est bien un fichier WAV, si son codage est PCM, quel est le nombre de bits utilisés pour l'amplitude, et combien de "channels" il

contient.

- a. licence OpenSource , d'après <https://bigsoundbank.com/detail-2363-applause-1.html>
- b. Cette syntaxe limite aux 100 premiers octets l'affichage de **hexdump**.

### ② Problème III.4 : Jeu video avec plusieurs niveaux

A. Reprenez le code du Viking et proposez un format de fichier texte pour stocker la carte de jeu pour 1 niveau du viking. Le fichier devra également contenir la position du Viking sur la carte au démarrage du jeu.

Written with open-source software only



LINUX



the LATEX Project



INKSCAPE



KRITA