LISTE 1

TYPE DE VARIABLES, CONDITIONS ET BOUCLES

Pour me contacter : `m.lejeune@uliege.be`, bureau 1/25.

Nous allons travailler avec certaines fonctions déjà connues, se trouvant dans différentes librairies. On ajoute une librairie au moyen de `#include<...>` où les `...` sont à remplacer par le nom de la librairie. Voici quelques librairies qui sont utiles, à ajouter à chaque projet pour éviter les tracas.

- `stddef.h` et `stdlib.h` contiennent quelques types de variables et autres. Il faut les inclure à chaque fois !
- `stdio.h` permet de travailler avec des fichers externes (utile pour la suite...)
- `math.h` contient bon nombre de fonctions mathématiques : `pow` pour les puissances, `sin`, `cos`...
- ...

Quelques liens utiles :

- `www.cplusplus.com`
- `openclassrooms.com/courses/apprenez-a-programmer-en-c`
- `c.developpez.com`
- et bien évidemment, votre moteur de recherche préféré.

Exercice 1 (Hello world !)**.** Concevoir un programme en langage *C* qui permet d'afficher dans la console le message « Bonjour tout le monde ! ».

Exercice 2. Concevoir un programme en langage *C* dans lequel vous déclarez deux variables : une de type entier et une de type réel. Affecter des valeurs à ces variables et affichez-les dans la console. Que se passe-t-il si vous affectez un nombre réel à la variable de type entier ?

Exercice 3. Concevoir un programme en langage *C* qui demande l'entrée d'un entier et d'un réel. Affecter leur somme dans une nouvelle variable (quel type devez-vous mettre à celle-ci ?) et affichez-la. Que se passe-t-il si vous introduisez un nombre réel dans l'entier ?

Exercice 4 (L'ordinateur : un monde de bits, d'octets, ...)**.** Le mot *bit* est la contraction des mots anglais **bi***nary digi***t**, qui signifient « chiffre binaire ». C'est l'élément de base du système de numération binaire. Ce système est présent dans tous les systèmes informatiques. Un *octet* est un regroupement de 8 bits. Le terme est couramment utilisé comme unité de mesure en informatique pour indiquer la capacité des mémoires informatiques. On utilise souvent des multiples de l'octet, comme le kilooctet (ko) ou le mégaoctet (Mo).

(a) Combien de nombres peut-on représenter avec un octet ?

(b) En sachant qu'un entier de type `int` est, au moins, codé sur 16 bits et que le premier bit code le signe (0 pour + et 1 pour −), quelle est le plus petit entier et le plus grand entier qu'on est sur de pouvoir coder ? Pour un entier de type `unsigned int`, le signe n'est pas codé, i.e. qu'on ne code que des entiers positifs. Le plus petit entier est donc 0. Quel est le plus grand ?

(c) Même question avec les `long` et `unsigned long` en sachant qu'ils sont codés, au moins, sur 32 bits.

Exercice 5. Concevoir un programme en langage *C* qui permute la valeur de deux nombres réels, d'abord intuitivement. Dans un second temps, essayer de supprimer la variable temporaire introduite.

Exercice 6. Concevoir un programme en langage *C* qui demande à l'utilisateur deux nombres réels et affiche le plus grand des deux. Dans un premier temps, utiliser la condition `if ... else` ; puis dans un second temps, utiliser l'opérateur ternaire.

Exercice 7. Concevoir un programme en langage *C* qui demande l'entrée de la taille de l'utilisateur (en cm), de son poids, et qui détermine si la personne est en surpoids ou pas (selon la règle que le poids idéal est égale à la taille à laquelle on soustrait 100). N'oubliez pas de gérer les erreurs éventuelles lors de l'encodage par l'utilisateur.

Exercice 8. Concevoir un programme en langage *C* calculant les racines réelles d'un polynôme du second degré du type $ax^2 + bx + c$ avec $a, b, c \in \mathbb{R}$. Dans un second temps, le programme affichera dans la console la factorisation réelle du polynôme.

Exercice 9. Un des deux codes suivants est faux. Déterminer lequel et pourquoi.
Code 1 :

```c
int main () {
    int a;

    do{
        printf("Entrer un entier positif : ");
        scanf("%d",&a);
    }
    while(a<0);
}
```

Code 2 :

```c
int main () {
    int a;

    while(a<0){
        printf("Entrer un entier positif : ");
        scanf("%d",&a);
    }
}
```

Exercice 10. Concevoir un programme en langage *C* demandant à l'utilisateur d'entrer un nombre entier strictement positif `n`, `n` nombres réels et affiche le plus grand nombre entré par l'utilisateur.

Exercice 11. Concevoir un programme en langage *C* demandant à l'utilisateur d'entrer un nombre entier strictement positif `n`, `n` nombres réels et affiche le nombre d'occurrences du plus grand nombre saisi par l'utilisateur.

Exercice 12. Concevoir un programme en langage *C* demandant à l'utilisateur d'entrer un nombre entier strictement positif `n`, `n` nombres réels et affiche la moyenne arithmétique de ces nombres

Exercice 13. Concevoir un programme en langage C demandant à l'utilisateur d'entrer un nombre entier strictement positif `n` et détermine s'il est premier ou pas.

Exercice 14. Concevoir un programme en langage C qui calcule la somme suivante :

$$\sum_{i=4}^{20} \sum_{j=3}^{i} (i+j)^3.$$

Recalculer cette somme en inversant l'ordre des indices.

Exercice 15. Une intégrale peut se calculer comme la limite d'une somme de Riemann :

$$\int_0^1 x^2 \, dx = \lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n} \left(\frac{i}{n} \right)^2.$$

Concevoir un programme en langage C qui calcule la somme de Riemann ci-dessus pour les valeurs de $n = 10$ et 100.

LISTE 2 — CRÉER SES PROPRES FONCTIONS

Dans la suite du cours, chaque programme doit être découpé en fonctions et celles-ci doivent être spécifiées correctement. Une fonction non spécifiée ne sera pas lue ! Une spécification correcte d'une fonction comprend les informations suivantes :

- une explication de ses paramètres en disant ce qu'ils représentent en « vrai » (ex : taille d'une matrice, un mot, une phrase, ...), leurs restrictions éventuelles (ex : entier entre -5 et 5, ...) et leurs types.
- une explication du but de la fonction. Ce but est précisé de la manière suivante :
  - si la fonction retourne une valeur, il faut expliquer le lien entre cette valeur et les paramètres de la fonction,
  - si certains paramètres sont des pointeurs, il faut expliquer leurs éventuelles modifications,
  - si la fonction affiche un message dans la console, il faut le préciser.

Voici une liste non-exhaustive de conseils :

- la fonction `main` contient uniquement des initialisations de variables, des appels à des fonctions et des affichages dans la console ; cette fonction ne doit donc pas être trop longue ; cependant, elle est nécessaire pour que votre programme fasse quelque chose,
- une fonction n'a qu'un seul but,
- si dans votre code, plusieurs lignes se répètent, il est conseillé d'isoler ces lignes dans une ou plusieurs fonctions,
- si lors de la spécification d'une fonction, un argument n'est pas utilisé dans l'explication du but de la fonction, posez-vous la question de son utilité.

Exercice 1. Voici deux fonctions écrites en langage *C* :

```
int fonction1(int a, int b){

        return a++ +b;

}
```

```
int fonction2(int a, int b){

        return ++a +b;

}
```

(a) Que renvoient ces fonctions si on leur donne comme paramètres `(2,3)` ?
(b) Spécifier chacune de ces fonctions.
(c) Simuler sur feuille l'exécution du programme suivant :

```
int main(){

        int a=1,b=3,f1,f2;

        printf("a=%d, b=%d\n",a,b);
```

```
        f1=fonction1(a,b);
        f2=fonction2(a,b);

        printf("a=%d, b=%d\n f1=%d, f2=%d",a,b,f1,f2);

        return 0;
}
```

Exercice 2. Spécifier la fonction suivante :

```
double fonction (double x) {

        return x>=0 ? x : -x ;

}
```

Exercice 3. Spécifier la fonction suivante :

```
double fonction (int n){

        int i;
        double val,sum=0;

        for(i=1 ; i<=n ; i++){
                printf("Entrer un nombre reel : ");
                scanf("%lf",&val);
                sum=sum+val;
        }

        return sum/n;

}
```

Exercice 4. Voici une fonction écrite en langage C :

```
int fonction(unsigned int n){
        unsigned int m=0;

        if (n != 0){
                while(n != 1){
                        if (n & 1)              //raccourci de ((n & 1) != 0)
                                m++;
                        n = n>>1;
                }
                m++;
        }

        return m;
}
```

(a) Que renvoie cette fonction si on lui donne comme paramètre `5` ? Même question avec `14` et `17`.
(b) Spécifier cette fonction.

Exercice 5. Ecrire en langage *C* de deux façons différentes une fonction prenant en argument un entier strictement positif n et renvoyant sa factorielle. Rédiger vos codes dans un premier temps sur papier. Vous pouvez utiliser l'ordinateur pour le vérifier par après.
Laquelle des méthodes est préférable ? Pourquoi ?

Exercice 6. Ecrire en langage *C* de deux façons différentes une fonction prenant en argument un entier positif n et renvoyant le $n^{\text{ème}}$ nombre de Fibonacci. L'une des méthodes est-elle plus rapide que l'autre ? (Essayer de calculer le $50^{\text{ème}}$ nombre de Fibonacci avec les deux méthodes) Expliquer. Rédiger vos codes dans un premier temps sur papier.
Rappel des nombres de Fibonacci : $u_0 = u_1 = 1$ et $\forall n \geq 2, u_n = u_{n-1} + u_{n-2}$.

Exercice 7 (Problème de Collatz)**.** Concevoir un programme en langage *C* qui étudie les itérations successives de la fonction $f : \mathbb{N} \to \mathbb{N}$ définie par

$$\begin{cases} f(n) = 3n+1 & \text{si } n \text{ est impair} \\ f(n) = n/2 & \text{sinon.} \end{cases}$$

Jusqu'à quel nombre n pouvez-vous vérifier que les itérations successives de la fonction f atteignent le nombre 1 ?

Exercice 8 (Le jeu *Plus ou moins*)**.** Concevoir un programme en langage *C* permettant de jouer au jeu *Plus ou moins*. Le principe de ce jeu est le suivant :
— L'ordinateur tire au sort un nombre entre 1 et 100.
— Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
— L'ordinateur compare le nombre que vous avez entré avec le nombre « mystère » qu'il a tiré au sort. Il vous dit si le nombre mystère est supérieur ou inférieur à celui que vous avez entré.
— L'ordinateur vous demande à nouveau un nombre, et ainsi de suite, jusqu'à ce que vous ayez trouvé le nombre mystère.
— L'ordinateur vous annonce alors votre victoire et vous dit le nombre d'essais qui vous ont été nécessaires.

Voici un exemple du déroulement d'un jeu :

```
Entrer un nombre entre 1 et 100 : 50
Le nombre est plus grand
Entrer un nombre entre 1 et 100 : 750
Je t'ai dit entre 1 et 100 !!! Recommence
Entrer un nombre entre 1 et 100 : 75
Le nombre est plus grand
Entrer un nombre entre 1 et 100 : 88
Le nombre est plus petit
Entrer un nombre entre 1 et 100 : 80
Le nombre est plus grand
Entrer un nombre entre 1 et 100 : 83
Gagné en 5 étapes !!
```

Pour tirer un nombre au sort, ajoutez (si ce n'est déjà fait) dans un premier temps les librairies `stdlib.h` et `time.h`.

Ensuite, à la première fois où vous tirez un nombre aléatoire, initialisez le processus grâce à la commande

```
srand(time(NULL));
```

Cette ligne ne doit être exécutée qu'une seule fois !
Enfin, la ligne de code que vous devez taper pour tirer un nombre aléatoirement entre `MIN` et `MAX` est la suivante :

```
int MAX=100, MIN=1, nombreMystere;
nombreMystere = (rand() % (MAX-MIN+1)) + MIN ;
```

Exercice 9. Le développement de Taylor de la fonction sinus est donné par

$$\sin x = \sum_{k=0}^{n} \frac{(-1)^k}{(2k+1)!} x^{2k+1} + R_n(x) \quad \text{où} \quad |R_n(x)| \leq \frac{|x|^{n+1}}{(n+1)!}$$

pour tout $x \in \mathbb{R}$ et tout $n \in \mathbb{N}_0$. Concevoir un programme en langage C contenant la fonction `taylor_sin(x,e)` qui calcule une approximation de $\sin(x)$ avec une erreur majorée par `e`.
Votre programme est-il efficace ? Si non, améliorez-le.

Exercices supplémentaires

Exercice 10. Concevoir un programme dans lequel l'utilisateur entre, un par un, une suite d'entiers strictement positifs. Dans cette suite, seul le dernier élément est 1. On demande de retourner la moyenne géométrique de tous les nombres entrés.

Exercice 11. Concevoir un programme demandant l'entrée de deux nombres entiers a et b, et renvoyant le quotient et le reste de la division entière de a par b.

Exercice 12. Concevoir un programme demandant à l'utilisateur deux naturels n et p et calculant, si cela a du sens, le coefficient binomial $\binom{n}{p}$. Est-il efficace ? Comment pourriez-vous l'améliorer ?

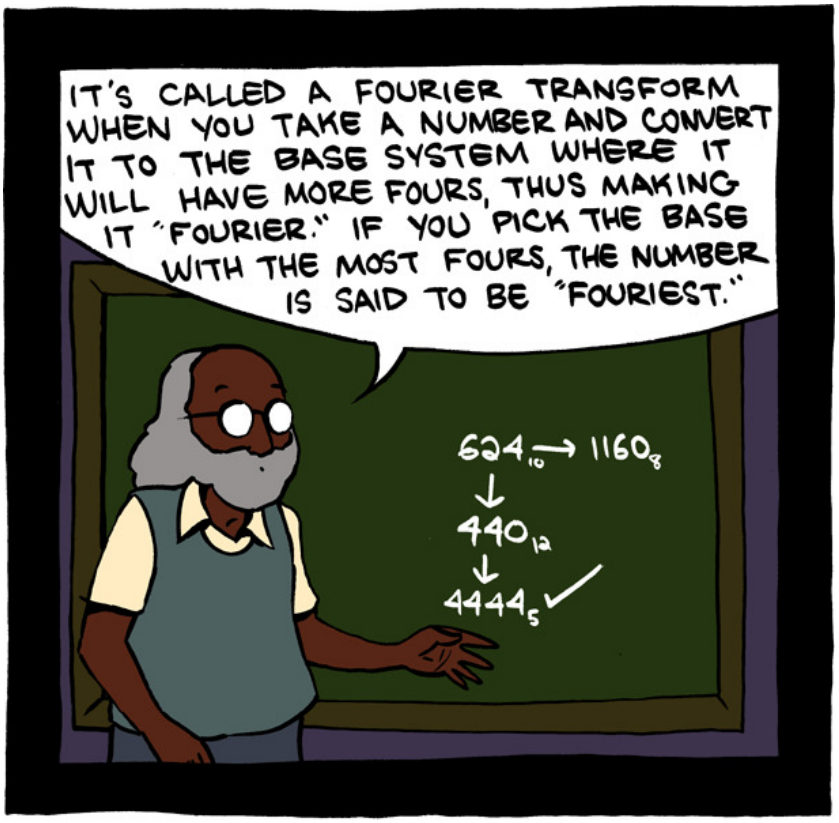
Exercice 13. Ecrire un programme qui permet de trouver le PGCD de deux nombres a et b.

Exercice 14. (Représentation en base b d'un entier)

(a) Concevoir un programme demandant à l'utilisateur d'entrer un nombre entier n et affichant le nombre de chiffres composant cet entier.

(b) En plus du nombre entier n à entrer, on demande à l'utilisateur un naturel $b \geq 2$ et cette fois, le programme doit renvoyer le nombre de chiffres composant la représentation du nombre n en base b (notée $\text{rep}_b(n)$).

Exercice 15. (Fourier transform)
Concevoir le programme suivant : l'utilisateur doit dans les deux cas entrer un entier n. Dans un premier temps, il entre aussi deux bases b_1 et b_2 comprises entre 2 et 16. On lui demande de déterminer si le passage de la base b_1 à la base b_2 est une *Fourier transform* (voir illustration), c'est-à-dire si on augmente le nombre de 4 en passant de $\text{rep}_{b_1}(n)$ à $\text{rep}_{b_2}(n)$.
Ensuite, on demandera au programme de déterminer la *Fouriest transform* du nombre n (voir illustration) : la base $b \in \{2, \dots, 16\}$ dans laquelle $\text{rep}_b(n)$ contient le plus de 4.



Teaching math was way more fun after tenure.

LISTE 3 — LES STRUCTURES

Exercice 1. Quelles seront les valeurs affichées à l'écran après l'exécution de ce programme ? Justifier.

```c
typedef struct account account;

struct account{
   int account_number;
};

void modifV (account z){
       z.account_number=z.account_number+2;
}

account modif (account z){
    z.account_number=z.account_number+2;
    return z;
}

void modifPoint(account *z){
    z->account_number=z->account_number+2;
}

int main(){
    account s,t;                // objets de la structure
    account *a;               // pointeur vers un objet de la structure
    a = &t;                     // on pointe le pointeur vers un objet

    s.account_number=3;         // assignation de valeurs
    a->account_number=4;
    printf("%d\n",s.account_number);
    printf("%d\n",t.account_number);

    modifV(s);
    (*a).account_number=6;
    printf("%d\n",s.account_number);
    printf("%d\n",t.account_number);

    s=modif(s);
    printf("%d\n",s.account_number);
    printf("%d\n",t.account_number);

    modifPoint(a);
    printf("%d\n",s.account_number);
    printf("%d\n",t.account_number);
    return 0;
}
```

Exercice 2. Concevoir un programme en langage *C* dans lequel on définit une structure `NbCompl` qui représente un nombre complexe. Ce programme doit contenir, au minimum, les fonctions suivantes :

- `void affiche(NbCompl z)` : fonction prenant en argument un nombre complexe z et l'affiche dans la console sous la forme $\Re(z) + i\Im(z)$,
- `NbCompl creer()` : fonction qui demande, dans la console, à l'utilisateur d'encoder un nombre complexe et renvoie ce nombre,
- `NbCompl somme(NbCompl z1 , NbCompl z2)` : fonction prenant en argument deux nombres complexes et renvoyant leur somme,
- `NbCompl oppose(NbCompl z1)` : fonction prenant en argument un nombre complexe et renvoyant son oppposé,
- `NbCompl conjugue(NbCompl z1)` : fonction prenant en argument un nombre complexe et renvoyant son conjugué,
- `NbCompl produit(NbCompl z1 , NbCompl z2)` : fonction prenant en argument deux nombres complexes et renvoyant leur produit,
- `double module(NbCompl z1)` : fonction prenant en argument un nombre complexe et renvoyant son module,
- `NbCompl inverse(NbCompl z1)` : fonction prenant en argument un nombre complexe et renvoyant son inverse ; si le nombre n'est pas inversible, la fonction renvoie le nombre complexe 0.

Le programme doit être capable de :

1. demander à l'utilisateur s'il veut entrer 1 ou 2 nombres complexes,
2. permettre à l'utilisateur de le(s) encoder,
3. si l'utilisateur a encodé un nombre complexe, le programme affiche son opposé, son module et son inverse (si cela a un sens),
4. si l'utilisateur a encodé deux nombres complexes, le programme affiche leur somme, leur différence, leur produit et leur quotient (si cela a un sens).

Dans un second temps, le programme peut être modifié pour demander à l'utilisateur ce qu'il veut afficher (l'opposé ou l'inverse ou la somme ou ...) à la place de tout afficher directement.

Exercice 3. A l'aide de l'exercice précédent, concevoir un programme calculant les racines d'un polynôme du second degré du type ax^2+bx+c avec $a, b, c \in \mathbb{C}$. Le programme affichera également la factorisation de ce polynôme dans $\mathbb{C}$.

Exercices supplémentaires

Exercice 4. Concevoir un programme en langage *C* contenant une fonction qui prend en argument deux points de l'espace $\mathbb{R}^2$ et qui renvoie une équation cartésienne de la droite passant par ces points. Un point et une droite seront modélisés par des structures adéquates.

Pour tester votre fonction, votre programme doit demander à l'utilisateur deux points de l'espace et affiche l'équation cartésienne de la droite. Celle-ci doit être affichée, si c'est possible, sous la forme $ax + by + c = 0$ et sous la forme $y = mx + p$.

Exercice 5. Concevoir un programme dans lequel on définit une structure `Rat` qui représente un nombre rationnel. Ce programme doit contenir les fonctions suivantes :

- `void affiche (Rat q)` , une fonction prenant en argument un nombre rationnel et l'affichant dans la console sous la forme p/q.

— `Rat creer()`, une fonction demandant à l'utilisateur d'entrer un nombre rationnel valide et le renvoie.
— `Rat simpl (Rat q)`, une fonction prenant en argument un nombre rationnel et renvoyant le nombre rationnel simplifié au maximum, avec la convention que le dénominateur est toujours positif.
— `Rat somme (Rat q1, Rat q2)`, une fonction prenant en argument deux nombres rationnels et renvoyant leur somme.
— `Rat produit (Rat q1, Rat q2)`, une fonction prenant en argument deux nombres rationnels et renvoyant leur produit.
— `Rat inverse (Rat q)`, une fonction prenant un argument un nombre rationnel et renvoyant son inverse. On renverra le nombre rationnel 0 si le nombre n'est pas inversible.

Le programme doit ensuite être capable d'effectuer les tâches suivantes.

1. Demander à l'utilisateur s'il veut entrer 1 ou 2 nombres rationnels.
2. Permettre à l'utilisateur de le(s) encoder.
3. Si l'utilisateur n'a entré qu'un seul nombre rationnel, afficher la valeur simplifiée de ce rationnel ainsi que, si cela a un sens, son inverse.
4. Si l'utilisateur a entré deux nombres rationnels, afficher leur somme, leur différence, leur produit et leur quotient (si cela a un sens), tous les résultats étant simplifiés au maximum.

Dans un second temps, modifier le programme pour qu'il demande à l'utilisateur ce qu'il veut afficher (produit, somme, ...).

LISTE 4 — LES TABLEAUX STATIQUES

Les exercices suivants sont à faire dans un premier temps par écrit !

Exercice 1. Concevoir un programme en langage *C* qui demande à l'utilisateur d'entrer 10 nombres réels. Ceux-ci sont stockés dans un tableau statique de taille 10. Le programme affiche la moyenne et l'écart type de ces nombres.

Exercice 2. Concevoir un programme en langage *C* contenant les fonctions suivantes :

- `double det(double A[][2])` : cette fonction prend un argument une matrice carrée réelle A de dimension 2 et renvoie son déterminant,
- `void resol(double A[][2] , double B[] , double X[])` : cette fonction prend en argument une matrice carrée réelle A de dimension 2 et de déterminant non nul, un vecteur réel B de dimension 2 et résout le système $AX = B$.

Pour tester votre programme, faire une fonction `main` demandant à l'utilisateur une matrice A et un vecteur B et résolvant le système $AX = B$ si le déterminant de A est non nul ; dans le cas contraire, il affiche un message dans la console.

Exercice 3. Même question que la précédente avec une matrice **complexe** de taille 2. Vous pouvez réutiliser la structure et les fonctions suivantes, que vous avez implémentées dans la liste précédente :

```
struct NbCompl{
        double re;
        double im;
};
typedef struct NbCompl NbCompl;

NbCompl somme(NbCompl z1, NbCompl z2); //renvoie la somme z1+z2
NbCompl produit(NbCompl z1, NbCompl z2);   //renvoie le produit z1*z2
NbCompl inverse(NbCompl z1); //renvoie l'inverse du complexe z1 en supposant
    qu'il est different de 0
```

Exercices supplémentaires

Exercice 4. Concevoir un programme qui demande à l'utilisateur d'entrer 10 nombres réels et détermine si la suite est croissante, strictement croissante, décroissante, strictement décroissante, constante ou aucune des propositions ci-dessus.

Exercice 5. Concevoir un programme qui demande à l'utilisateur d'entrer 10 nombres réels stockés dans un tableau, un entier $i \in \{1, \dots, 10\}$ et un réel r. Celui-ci doit ensuite modifier le tableau pour que le $i^{\text{ème}}$ élément soit r, décalant les autres valeurs vers la droite et supprimant la dernière.
Par exemple, si l'utilisateur entre $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, 7, 20$, le programme doit renvoyer le tableau $\{1, 2, 3, 4, 5, 6, 20, 7, 8, 9\}$.