GEii
Génie Electrique
et Informatique Industrielle

UVSQ
université PARIS-SACLAY
IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY

Ressource Info1

Informatique

Travaux Pratiques

GEII 1ère année

Année 2022-2023

Travaux Pratiques d'Informatique

Table des matières

Généralités

- pour tous les TP d'informatique, vous téléchargerez le fichier correspondant au TP sur Moodle → cours GE1R081 → TP → débuts de programme, dans le dossier « Téléchargements » de votre ordinateur. Vous décompresserez ensuite ce fichier dans un sous-dossier du dossier actuel et **vous travaillerez exclusivement dans ce sous-dossier**. A la fin de la séance, **vous mettrez ce dossier à la corbeille** et vous viderez cette dernière avant d'éteindre l'ordinateur.

- le logiciel utilisé durant les TP est Code::Blocks. Il contient le compilateur gcc, un certain nombre de bibliothèques standards ainsi qu'un environnement de programmation (IDE ou *Integrated Development Environment*). Il s'agit d'un logiciel libre que vous pouvez télécharger gratuitement sur le site : http://www.codeblocks.org/downloads/binaries (pour les utilisateurs sous Windows, veillez à bien **prendre le fichier avec mingw** qui contient le compilateur gcc en plus de l'IDE proprement dit).

- Une documentation en ligne sur toutes les fonctions définies dans les bibliothèques standards du C est disponible sur : http://www.delorie.com/djgpp/doc/libc-2.02/ (il est *fortement* conseillé d'ajouter cette page à vos signets internet).

TP 1 - INTRODUCTION AU LANGAGE C

Pour ce premier TP, vous pourrez vous référer aux chapitres 1 à 3 du Manuel de Survie.

1. Génération, compilation et exécution d'un programme niv 1

Pour obtenir un code exécutable sur votre ordinateur, il est nécessaire de procéder aux étapes suivantes :

- édition du programme écrit en langage C et mémorisation dans un fichier ayant l'extension .c (par exemple tp1_ex1.c pour indiquer l'exercice 1 du TP1). On parle de fichier source. Cette étape se fait à l'aide d'un éditeur de texte ou directement dans un environnement de programmation.
- compilation du programme source ; cette étape permet de vérifier les erreurs de syntaxes du fichier considéré (mais ne vérifie pas les fonctions appelées par le programme) et génère un fichier objet (tp1_ex1.o); puis édition de liens, cette étape fait le lien entre les différents codes objets nécessaire à l'exécution de l'application : les codes des fonctions des bibliothèques sont intégrés pour générer un fichier exécutable final (tp1_ex1.exe).
- exécution du fichier

Voici le programme écrit pour afficher " bonjour " à l'écran :

```
#include <stdio.h>

int main(void)
{
  printf ("bonjour !\n") ;
return 0 ;
}
```

Application :

- Lancez l'IDE Code::Blocks (vous pouvez copier l'exécutable dans la barre de lancement rapide de Windows)
- une fenêtre "*Compilers auto-detection*" s'ouvre : cliquez sur ok ;
- la fenêtre principale de Code::Blocks s'ouvre alors. Cette fenêtre comprend en haut des barres de menus et d'outils ainsi que 3 volets : volet des fichiers (à gauche), volet de l'éditeur (à droite) et volet du compilateur (en bas).
- Ouvrez un nouveau fichier (menu *file> New > Empty File*), une fenêtre d'édition vous permettant de rentrer votre code s'ouvre
- Sauvegardez le fichier, en sélectionnant bien le répertoire de travail et en changeant le nom du fichier. Pensez à bien préciser l'extension « .c » à la fin du nom qui correspond au fichiers sources en langage C.
- Saisissez le code source ci-dessus dans le volet d'édition en respectant bien la syntaxe indiquée, remarquez que l'indentation se fait automatiquement et que la coloration syntaxique du langage se met en place.
- Générez le code exécutable : ouvrir l'onglet *Build* puis *Build* de nouveau ;
- Regardez les éventuels messages d'erreurs et/ou d'avertissement dans le volet de compilation : si le code source contient des erreurs, le compilateur renvoie un message d'erreur (la compilation n'a pas abouti) ou un avertissement (la compilation a réussi mais le code contient une anomalie). Pour être sûr que tous les avertissements seront affichés, il faut modifier les options lors de la compilation : ouvrir l'onglet *Settings >Compiler Options*. **Cocher la case : *Enable all compiler warnings [-Wall]***
- Si aucune erreur ni aucun avertissement n'ont été signalés, vous pouvez l'exécuter : onglet *Build>Run*
- Vérifiez que votre programme s'exécute correctement.

2. Structure générale d'un programme écrit en langage C niv 1

Dans un programme C, les mots clefs du langage sont écrits en minuscules et les commentaires sont écrits entre /* et */
Un programme écrit en langage C a la forme générale suivante :

```
/* inclusion des différents fichiers nécessaires à l'utilisation de fonctions se
trouvant en bibliothèques */
#include <stdio.h>    /* directive d'inclusion du fichier stdio.h */

/* fonction principale */
int main(void)
{
      /* declarations des variables */

      /* corps du programme */
}
```

Explications complémentaires :
`#include <stdio.h>`
Dans les fichiers d'entête *.h se trouve les prototypes et déclarations nécessaires aux fonctions se trouvant en bibliothèques. Ici " stdio.h » signifie standard input-output et contient toutes les déclarations des fonctions réalisant les opérations d'entrée-sortie comme scanf pour saisir des valeurs au clavier et printf pour afficher des résultats sur l'écran. La directive #include signifie que ce fichier sera inclus dans votre fichier lors de la phase préparatoire à la compilation.
Pour chaque fonction existante en bibliothèque, vous devrez inclure le fichier .h qui lui correspond. De plus, lors de la compilation, il faudra indiquer au compilateur qu'il doit faire le lien avec la bibliothèque en question.

```
void main(void)
{
…
}
```

Le mot clef **main** est la porte d'entrée du programme : main en anglais signifie principal ; il s'agit donc ici de la fonction principale (nous reviendrons sur les fonctions ultérieurement).
Le code développé doit se trouver entièrement entre les accolades qui signifient début-code et fin-code.

Préalablement à l'écriture d'instructions C, il est nécessaire de procéder à la déclaration de toutes les variables utilisées dans le programme.

De plus, il est **indispensable de bien commenter** un programme :
- permet une meilleure compréhension (par les autres et même par soi-même)
- augmente la durée de vie d'un programme (maintenance)
- facilite l'utilisation du programme par d'autres personnes
- ne consiste surtout pas à paraphraser le langage mais doit permettre de comprendre les fonctionnalités du programme, sa structuration et éventuellement de souligner un point particulier.

Soit le programme suivant :

```
#include <stdio.h>

int main(void)
{
      float Pi = 3.14 , Surface, Perimetre ;
      int Rayon  ;

      printf ("Calcul du périmètre et de la surface d'un cercle \n ") ;
      printf ("Saisissez le rayon : ") ;
      scanf("%f", &Rayon) ;
      Perimetre = 2* Pi * Rayon ;
      Surface = Pi * Rayon * Rayon ;
      printf ("Perimetre = %d,  Surface = %f \n", Perimetre, Surface) ;
      return 0;
}
```

a) Saisissez ce programme et examinez son exécution.
b) Modifiez les instructions de saisie et d'affichage pour corriger les erreurs. Que constatez-vous ?
c) En vous aidant du manuel, expliquez les lignes du programme : variables utilisées, types de ces variables, initialisations, calculs réalisés, utilisation des fonctions scanf et printf …

3. Types de variables — niv 1

Ouvrez le fichier tp1exo3etu.c

a) Complétez la ligne contenant l'instruction `scanf` et tester le programme
b) Modifiez ensuite ce programme pour que toutes les valeurs soient du type float
c) Modifiez ensuite le programme pour que toute les valeurs soient du type double

4. Ecriture d'un programme à partir d'un algorigramme — niv 1

Reprendre l'algorigramme permettant de calculer le quotient familial fait en TD (TD 0 - exercice 2) et écrire le programme correspondant en langage C.

5. Permutation de deux variables — niv 2

Ecrire un programme qui prend en entrée la valeur de deux variables entières *a* et *b*, qui permute leurs valeurs respectives (la valeur de *a* passe dans *b* et la valeur de *b* passe dans *a*) et affiche les nouvelles valeurs de *a* et *b*.

6. Codage des entiers — niv 2

Réalisez un programme permettant de déterminer le carré d'un nombre. En déclarant les variables comme des entiers courts (sur 16 bits), déterminer la valeur maximum du nombre pour lequel le résultat sera valide. Cette valeur change-t-elle si les variables sont déclarées comme des entiers courts non signés ?

7. Formats d'affichage niv 2

Tester le programme suivant et expliquez les différents formats d'affichage de la variable c (vous ferez directement apparaitre ces explications dans votre fichier source comme commentaires du code).

```
#include <stdio.h>
int main(void)
{
  char c ;
  c = 'H' ;

  printf ("%d \n", c) ;
  printf ("%o \n", c) ;
  printf ("%x \n", c) ;
  printf ("%c \n", c) ;
  return 0;
}
```

Compléter ensuite ce programme pour que l'utilisateur puisse rentrer un nombre réel, qui sera stocké dans une variable de type `float`. Le programme devra ensuite afficher ce nombre mutliplié par 1 million, en n'affichant que 2, puis 1 puis 0 chiffre après la virgule. Le nombre rentré par l'utilisateur devra ensuite être divisé par un million, puis affiché d'abord "classiquement" puis en utilisant la notation scientifique.

Pour cet exercice, il est fortement conseillé de lire au préalable l'annexe G du manuel de survie.

8. Conversion Heures-Minutes niv 2

Ecrire un programme dans lequel l'utilisateur entrera une durée en minutes. Le programme devra alors calculer et afficher la valeur de cette durée en heures et minutes

Exemple :

```
Saisissez une duree en minutes : 1254
1254 minutes correspond a 20h 54 mn
```

Indication : pensez à l'opérateur modulo

9. Conversion caractère → entier niv 3

Ecrire un programme qui convertit 5 chiffres indépendants en un nombre entier représenté par ces chiffres dans l'ordre inverse. Les 5 chiffres seront entrés par l'utilisateur, un par ligne. Dans un premier temps ils seront lus comme des entiers ("%d"), vous modifierez ensuite votre programme pour les lire comme des caractères ("%c"). Votre programme devra contenir 5 lectures mais un seul affichage.

10. Inversion d'un nombre à 3 chiffres niv 3

Ecrire un programme qui échange l'ordre des chiffres d'un nombre à 3 chiffres. Votre programme ne devra contenir qu'une seule lecture et un seul affichage

TP 2 - PROGRAMMATION MODULAIRE

Rappels généraux :

- Un programme non commenté et dont l'indentation n'est pas correcte ne sera pas lu.
- Cocher l'option –Wall pour afficher tous les avertissements du compilateur.
- Pour ce TP, vous pourrez vous référer au chapitre VII du Manuel de Survie.
- Récupérer le fichier compressé contenant les débuts de programme sur Moodle. Décompressez ce fichier dans un nouveau dossier pour pouvoir ensuite modifier les programmes. A la fin de la séance, vous penserez à placer ce dossier ainsi que le fichier compressé téléchargé dans la corbeille, puis à la vider.

1. PGCD selon Euclide — niv 1

Ouvrir le fichier tp2exo1.c. Compléter ce programme en y ajoutant la déclaration du prototype de la fonction `pgcd` puis le programme principal dans lequel on demande à l'utilisateur de rentrer un nombre entier, on calcule et affiche le pgcd de ce nombre et 12, on demande ensuite un 2nd nombre entier à l'utilisateur, on calcule et affiche alors le pgcd du 1er et du 2nd puis de 3 fois le 1er et 12 fois le 2nd.

2. Fonction Altitude — niv 1

1. Ouvrir le fichier tp2exo2.c. Compléter ce programme en y ajoutant la définition de la fonction `altitude`, qui renvoie l'altitude d'un objet en fonction du temps t et de sa vitesse initiale v0 selon la loi :

$$z(t) = -\frac{g}{2}t^2 + v_0 t$$

2. Modifier ensuite le programme de façon à ce que toutes les variables soient déclarées comme réelles

3. Conversions de température — niv 1

La température Celsius(centigrade) peut être convertie en Fahrenheit par la formule suivante :
$F = 9/5\ C + 32$

1. Ecrire le code de la fonction `Cels2Fahr` qui convertit une température Celsius en Fahrenheit.
2. Ecrire le code de la fonction `Fahr2Cels` qui effectue l'opération inverse.
3. Ecrire le programme principal dans lequel on demande à l'utilisateur de saisir une température en degrés Celsius, on la convertit et on l'affiche en degrés Fahrenheit puis on la reconvertit en degrés Celsius.

4. Fonctions Carré et puissance4 — niv 2

Ecrire :

1. Une fonction `carre` qui calcule le carré d'un nombre entier donné.
2. Une fonction `puissance4` qui élève un nombre entier passé en argument à la puissance 4. Votre fonction ne devra contenir qu'une seule fois le signe '*' (voire aucune fois).
3. Le programme principal qui demande un nombre entier à l'utilisateur et affiche son carré et ce nombre à la puissance 4.

Votre programme ne devra faire appel à aucune fonction de la bibliothèque mathématique.

5. Fonctions aléatoires niv 2

1. Ecrire un programme principal qui affiche la valeur de la constante RAND_MAX (cette valeur est définie dans la bibliothèque stdlib, il est donc illicite de la redéfinir).
2. Ecrire une fonction `dice6` qui renvoie un nombre entier au hasard entre 1 et 6. La tester dans le programme principal.
3. Ecrire une fonction `edice` qui renvoie un nombre entier au hasard entre 1 et une valeur maximale donnée en paramètre de la fonction. La tester dans le programme principal.
4. Ecrire une fonction `aleat` qui renvoie un entier au hasard entre 2 valeurs passées comme argument (on admettra que les 2 nombres en entrée sont rangés dans l'ordre croissant). La tester dans le programme principal.
5. Ecrire une fonction qui renvoie une lettre minuscule au hasard. La tester dans le programme principal.

6. Temps de vol niv 3

1. Ecrire une fonction qui calcule la durée de vol en minutes connaissant l'heure de départ et l'heure d'arrivée (sous la forme heure et minutes).
2. Ecrire une fonction qui affiche le temps de vol en heures et minutes connaissant le temps de vol en minutes
3. Ecrire un programme qui demande les heures de départ et d'arrivée à l'utilisateur et affiche le temps de vol.

4. Reprendre la 1ère fonction en supposant maintenant que la durée de vol est inférieure à 24 heures mais que l'arrivée peut avoir lieu le lendemain.

Annexe : nombres aléatoires

Pour générer aléatoirement une suite X_i de nombres aléatoires, on utilisera la fonction `rand()` définie dans la bibliothèque mathématique qui renvoie un nombre entier au hasard entre 0 et RAND_MAX (défini dans stdlib.h). Il est également nécessaire d'appeler `srand(time(0))` au moins une fois avant l'utilisation de `rand()` (la fonction time est définie dans la bibliothèque time.h).

Idées :

- si *n* est un entier tiré au hasard entre 0 et RAND_MAX, alors n%p est un entier tiré au hasard entre 0 et (p-1) (en supposant p << RAND_MAX)
- si *n* est un entier tiré au hasard entre 0 et RAND_MAX, alors (float) n / (RAND_MAX + 1.0) est un nombre flottant compris dans l'intervalle [0 , 1[

TP 3 - STRUCTURES CONDITIONNELLES

Rappels généraux :

- Un programme non commenté et dont l'indentation n'est pas correcte ne sera pas lu.
- Cocher l'option –Wall pour afficher tous les avertissements du compilateur.
- Les exercices suivants font appel au chapitre IV-I du manuel de survie (rn plus du cours et des TD).
- Récupérer le fichier compressé contenant les débuts de programme sur Moodle. Décompressez ce fichier dans un nouveau dossier pour pouvoir ensuite modifier les programmes. A la fin de la séance, vous penserez à placer ce dossier ainsi que le fichier compressé téléchargé dans la corbeille, puis à la vider

1. Multiple de 7 — niv 1

Ouvrir le fichier tp3_exo1.c. Le compiler puis le tester. Modifier ensuite la ligne du test en utilisant l'opérateur « modulo » à la place de l'opérateur « quotient de la division euclidienne ».

2. Maximum de deux nombres — niv 1

Écrire un programme qui, à partir de 2 valeurs entières saisies au clavier, détermine la plus grande
Exemple :

```
Saisissez 2 nombres :
15
45
45 est plus grand que 15
```

3. Lettre, chiffre, pair ou impair — niv 1

Ecrire un programme qui teste si un caractère saisi au clavier est une lettre, un chiffre ou un autre caractère. Le programme devra également indiquer si le code ASCII est pair ou impair.

4. Salaire hebdomadaire — niv 1

Reprendre l'algorigramme permettant de calculer le salaire net hebdomadaire fait en TD (TD 0 - exercice 4) et écrire le programme correspondant en langage C.

5. Maximum de trois nombres — niv 2

Écrire :

- une fonction, nommée `max3`, qui possède 3 arguments entiers et qui renvoie le plus grand des 3.
- Le programme principal, dans lequel on demande à l'utilisateur de rentrer 3 valeurs entières, puis on affiche la plus grande des 3.

Exemple 1:

```
Saisissez 3 nombres :
15
45
-6
La plus grande des 3 valeurs est 45
```

Exemple 2 :

```
Saisissez 3 nombres :
15
15
-6
La plus grande des 3 valeurs est 15
```

6. Equation du second degré — niv 2

Compléter le programme tp3exo6.c qui permet de résoudre dans R une équation du type :

$$a \,.\, x^2 + b \,.\, x + c = 0$$

Remarque : vous ne prendrez pas en compte le cas a = 0.

7. Année Bissextile niv 2

Une année est bissextile si son millésime est multiple de 4, sauf les années de fin de siècle qui ne sont bissextiles que si leur millésime est divisible par 400. Ouvrir le fichier tp3_exo7.c qui contient le programme principal puis écrire :

- une fonction, nommée isbissextile, qui indique si l'année passée en argument est bissextile. Vous emploierez dans un premier temps une succession de tests simples imbriqués (en utilisant éventuellement un drapeau initialement levé)
- Une fonction isbissextile2 qui fait la même chose que la fonction isbissextile mais avec une seule structure conditionnelle ; vous modifierez le programme principal de façon à tester également cette fonction.

8. Impôts Bordures niv 3

Dans le royaume de Bordurie, le calcul des impôts est régi par le texte suivant :

- Le nombre de parts dans un foyer est de 1 pour une personne, 2 pour 2 personnes, plus 0,5 pour chaque personne supplémentaire ;
- on divise le revenu total (en couronnes borduriennes) par le nombre de parts pour obtenir le revenu par parts (RPP)
- Les 2000 premières couronnes du RPP sont exonérées d'impôts
- Les 1000 couronnes suivantes sont imposées à 10%
- Les 2000 couronnes suivantes sont imposées à 20%
- Les couronnes suivantes sont imposées à 50%
- L'impôt à payer est alors remultiplié par le nombre de parts
- Si il y a une personne âgée dans le foyer, l'impôt est diminué de 5%, si il y a une personne faisant partie de la famille royale, la diminution est de 15% mais ces deux décotes ne sont pas cumulatives : seule la plus intéressante est appliquée.

Ouvrir le fichier tp3exo8.c qui contient le programme principal puis écrire (prototype et définition) la fonction `Impots_Bordures()` qui renvoie le montant de l'impôt à payer.

9. Lendemain niv 3

Ecrire un programme qui, à partir d'une date fournie par l'utilisateur, détermine la date du lendemain. Vous pourrez réutiliser les fonctions définies dans l'exercice 7.

Indication : vous pourrez commencer par écrire une fonction qui, pour un mois et une année donnés, renvoie le nombre de jours de ce mois.

10. Lendemain – Date valide niv 3

Complétez le programme de l'exercice 9 de façon à vérifier, avant de chercher la date du lendemain, que la date entrée est valide ; vous écrirez pour cela une fonction IsDateValide qui indique si une date donnée est valide.

11. Equation du second degré en complexe niv 3

Compléter le programme de l'exercice 6 de façon à résoudre l'équation en incluant les racines complexes. Il faudra également prendre en compte les cas où $a = 0$.

TP 4 - STRUCTURES REPETITIVES

1. Code ASCII niv 1

Ecrire un programme qui attend une valeur entière et affiche le caractère correspondant. Les instructions seront réitérées tant que le caractère correspondant est une lettre minuscule.

2. Racine carrée niv 1

Ecrire un programme qui attend une valeur réelle et qui calcule la racine carrée de ce nombre. Le calcul sera réitéré jusqu'à ce qu'une valeur négative soit entrée. La fonction racine carrée du langage C (`sqrt`) est définie dans la bibliothèque mathématique.

3. Afficher 10 nombres niv 1

Ecrire un programme qui affiche les 10 entiers suivants un entier entré par l'utilisateur en utilisant une boucle `for`.

4. Carré niv 1

Compléter le programme tp4exo4.c, disponible sur Moodle, qui affiche un carré rempli d'étoiles dont le côté est entré par l'utilisateur.

Exemple d'exécution :

```
entrez la hauteur : 4
voici le carré :
****
****
****
****
```

5. Somme des n premiers entiers niv 2

- Récupérer sur Moodle le fichier tp4exo5.c
- Ecrire la définition de la fonction `somme` qui calcule la somme des *n* premiers entiers.
- Modifier le programme principal de façon à ce que le calcul soit réitéré tant que l'utilisateur rentre un nombre positif.

6. Table de multiplication niv 2

Ecrire un programme qui affiche la table de multiplication des nombres de 1 à 10, sous la forme suivante :

```
     |  1   2   3   4   5   6   7   8   9  10
-----------------------------------------------
  1  |  1   2   3   4   5   6   7   8   9  10
  2  |  2   4   6   8  10  12  14  16  18  20
  …
 10  | 10  20  30  40  50  60  70  80  90 100
```

Pour vous assurer du bon alignement des colonnes de la table, vous pourrez utiliser les options d'affichage de la fonction printf (par exemple %3d pour afficher le nombre sur au moins 3 caractères, voir manuel de survie).

7. Statistiques niv 2

On désire calculer la moyenne d'une série de nombres entiers saisis au clavier, ainsi que la valeur maximale et la valeur minimale. On saisira les nombres un par un, on calculera leur moyenne, le maximum et le minimum et on les affichera.

Vous écrirez le programme en utilisant une boucle `pour` et en demandant au préalable à l'utilisateur le nombre de valeurs qu'il va rentrer.

8. Nombres premiers niv 2

- Récupérer sur Moodle le fichier tp4exo8.c
- Ecrire la définition de la fonction `ispremier` permettant de déterminer si un nombre entier *n* donné est un nombre premier. Pour cela, on utilisera un drapeau qui sera initialement levé et on testera tous les nombres inférieurs à *n* (ou jusqu'à une certaine valeur à déterminer) afin de vérifier qu'aucun d'entre eux ne divise *n.*

  Rappel : un nombre premier est un nombre qui ne possède que 2 diviseurs distincts : 1 et lui-même (donc 1 n'est pas premier).

9. Suites et séries niv 2

On considère la suite définie par récurrence suivante :

$$\begin{cases} u_0 = 1 \\ u_n = 5n - 2u_{n-1} \end{cases}$$

- Récupérer sur Moodle le fichier tp4exo9.c
- Ecrire la fonction `int suite(int n)` qui calcule le terme de rang *n* de cette suite.
- Ecrire la fonction `int Serie_bof(int n)` qui prend en entrée un entier *n* et calcule la somme de tous les termes de la suite définie précédemment jusqu'au rang *n* : définie par

  $S_n = \sum_{i=0}^{n} u_i = u_0 + u_1 + \cdots + u_{n-1} + u_n$

  Pour calculer chacun des termes de la somme, vous appellerez la fonction `suite`
- Compléter le programme principal de façon à calculer et afficher S_n

10. Statistiques encore niv 3

Reprendre l'exercice 7 en remplaçant la boucle pour par une boucle `tant que` ou `faire … tant que`. L'utilisateur ne précisera plus le nombre de valeurs à rentrer mais on sortira de la boucle dès qu'une valeur négative sera saisie.

11. Sapin niv 3

Ecrire un programme qui affiche un sapin sous la forme d'un triangle isocèle rempli d'étoiles, suivi d'une colonne verticale d'étoiles de même hauteur. La hauteur du triangle est entrée par l'utilisateur.

Exemple d'exécution :

```
entrez la hauteur : 3
voici le sapin :
  *
 ***
*****
  *
  *
  *
```

12. Suites et séries le retour niv 3

Reprendre l'exercice 9 :

- Ecrire une fonction `Serie_top` qui fait le même calcul que la fonction `Serie_bof` définie précédemment mais sans faire appel à 2 boucles imbriquées ;
- Ajouter l'appel de cette nouvelle fonction dans le programme principal
- Appeler cette fonction dans votre programme principal

La suite de Fibonacci est définie par :

$$\begin{cases} u_0 = 1 \\ u_1 = 1 \\ u_n = u_{n-1} + u_{n-2} \end{cases}$$

- Ecrire la fonction `void Fibonacci(int n)` qui prend en entrée un entier n et calcule et affiche tous les termes de la suite de Fibonacci jusqu'à celui de rang n.
- Appeler cette fonction dans votre programme principal

13. Calcul de π niv 3

L'objectif de ce programme est de calculer une valeur approchée de π de façon statistique : on se place pour cela dans le carré, centré sur le point (0,0) de côté RAND_MAX (RAND_MAX étant la constante définie dans la bibliothèque `stdlib`).

1. On appelle S1 l'aire de ce carré ; quelle est sa valeur en fonction de RAND_MAX ?
2. Quelle est l'aire S2 du cercle inscrit dans le carré ?
3. Si l'on tire au hasard un point du carré, quelle est la probabilité *P* qu'il soit également à l'intérieur du cercle (l'exprimer en fonction de S1 et S2) ? En déduire l'expression de π en fonction de *P*.
4. Si on génère aléatoirement un grand nombre de points à l'intérieur du carré, la loi des grands nombres indique que : $\lim_{N\to\infty} \frac{Nc}{N} = P$, où *N* est le nombre total de points et *Nc* le nombre de points à l'intérieur du cercle.

Écrire :

- Une fonction `int CoordAléat(void)` qui génère un nombre aléatoire entre –RAND_MAX/2 et +RAND_MAX/2 ;
- Une fonction `int InCercle(int x, int y)` qui détermine si un point donné (défini par ses coordonnées x et y) est à l'intérieur du cercle de rayon RAND_MAX/2
- Une fonction `double PiProba(int n)` qui détermine la valeur approchée de p selon la méthode définie précédemment, n étant le nombre total de points utilisés ;
- Le programme principal où on demande à l'utilisateur d'entrer le nombre de points, on affiche la valeur de π calculée à partir de la fonction précédente, puis sa valeur « théorique » donnée par la constante M_PI (définie dans la bibliothèque math.h), ainsi que la différence entre ces 2 valeurs.

TP 5 - TABLEAUX A UNE DIMENSION

Préambule : Déclaration de constantes symboliques en utilisant la directive #define du préprocesseur

Le préprocesseur C est l'ensemble formé par certaines instructions particulières, appelées directives et exécutées au début de la phase de compilation. Les instructions #include et #define en font partie.

Syntaxe `#define CONSTANTE valeur`

Exemples :
```
#define   NBR_CAR_MAX      15
#define   PI               3.14
#define   MESSAGE          "bonjour\n"
```

Cette directive placée en début de programme (avant `main()`) permet de déclarer des constantes symboliques de valeurs respectives 15 , 3.14 et "bonjour\n".
Lors de la compilation, le préprocesseur remplacera toutes les constantes symboliques par leur valeur : il s'agit effectivement d'un simple remplacement de caractères. Il est totalement inconcevable de réaliser des modifications sur ces constantes symboliques.

Utilisation :

`char nom[NBR_CAR_MAX] ;`	déclare un tableau nom de NBR_CAR_MAX =15 caractères
`for(i=0 ; i< NBR_CAR_MAX ; i++)`	parcourt nom de i=0 à i = NBR_CAR_MAX =15
`surface = PI * Rayon * Rayon ;`	calcule la surface d'un cercle de rayon donné
`printf (MESSAGE);`	affiche le message de bienvenue

Avantages :
- meilleure lisibilité (`NBR_CAR_MAX` aide mieux à la compréhension que 15)
- meilleure maintenabilité (si la constante symbolique a été correctement utilisée dans tout le programme, il suffit de changer la valeur de définition de la ligne `#define` pour réaliser une modification)

Convention : on écrit toujours les constantes symboliques en MAJUSCULES pour les différencier des variables.

Lors de l'écriture des programmes en C, vous essaierez de définir des constantes vous permettant facilement de changer les valeurs de la taille des tableaux.

---

1. Comparaison de 2 tableaux — niv 1

Récupérer sur Moodle le fichier tp5exo1.c. Compléter ce programme pour qu'il affiche les deux tableaux ainsi que le nombre d'indices vérifiant `A[i]=B[i]`

Exemple d'exécution :
```
A : 0 1 2 3 4 5 6 7 8 9
B : 5 4 3 3 2 0 6 5 1 9
Il y a 3 cases identiques
```

---

2. Parcours d'un tableau — niv 1

Ecrire un programme qui permet de saisir 10 nombres dans un tableau à une dimension, qui affiche les éléments entrés dans l'ordre inverse, puis qui affiche le maximum de ces 10 nombres.

---

3. Chaîne de caractères — niv 1

Récupérer sur Moodle le fichier tp5exo3.c. Compléter ce programme pour qu'il calcule sans utiliser la fonction strlen le nombre de caractères et le nombre de lettres minuscules de la chaîne.

4. Opérations sur un tableau niv 2

Récupérer sur Moodle le fichier tp5exo4.c. Compléter ce programme en écrivant les définitions des fonctions appelées par le programme principal. **Vous ne modifierez pas le programme principal.**

5. Conjugaison d'un verbe niv 2

Ecrire un programme qui lit un verbe régulier du premier groupe en "er" au clavier et qui en affiche la conjugaison au présent de l'indicatif de ce verbe. Contrôlez s'il s'agit bien d'un verbe en "er" avant de conjuguer. Vous pourrez utiliser les fonctions `gets, puts, strcat et strlen`

Exemples :

```
Verbe : travailler
     je travaille
     tu travailles
     il travaille
     nous travaillons
     vous travaillez
     ils travaillent
```

```
Verbe : dormir
Ce n'est pas un verbe du premier groupe !
```

6. Dé électronique non pipé niv 2

L'objectif de l'exercice est de vérifier que la fonction `int edice(int n)` définie dans le TP2 renvoie bien un nombre au hasard entre 1 et *n* de façon équiprobable (on supposera $n<100$) : pour cela on « lancera » *p* fois le dé (où *p* sera un nombre entré par l'utilisateur), et on comptera le nombre d'occurrences de chaque valeur entre 1 et *n*. Pour cela on définit un tableau de *n+1* cases où l'élément d'indice *i* indiquera le nombre d'occurrences de la valeur *i* sur les *p* lancers (la case d'indice 0 ne sera donc pas utilisée). On affichera alors, pour chaque valeur entre 1 et *n*, le nombre d'occurrences ainsi que le pourcentage sur le nombre total de lancers. Une amorce de programme est disponible sur Moodle (tp5exo6.c).

7. Tri d'un tableau niv 3

Ecrire un programme qui demande à l'utilisateur de saisir un entier *n*, au maximum égal à 20, puis deux autres valeurs entières : *min* et *max*. Le programme devra ensuite, en faisant appel à des fonctions :

1. Remplir un tableau de taille *n* avec des valeurs aléatoires comprises entre *min* et *max*,
2. afficher ce tableau,
3. trier ce tableau,
4. afficher le tableau résultant

Algorithme de tri par extraction simple : pour trier un tableau initialement non ordonné, de nombreux algorithmes sont disponible, plus ou moins simples et plus ou moins efficaces. En voici un simple mais relativement peu efficace (il n'est donc pas conseillé pour trier des tableaux de grande taille) :

- ✓ on recherche le plus petit des n éléments du tableau,
- ✓ on échange cet élément avec le 1er élément du tableau,
- ✓ le plus petit élément se trouve alors en première position, on peut alors appliquer les deux opérations précédentes aux n-1 éléments restants, puis aux n-2, … et cela jusqu'à ce qu'il ne reste qu'un seul élément, qui sera alors le plus grand

8. Opérations complémentaires sur un tableau niv3

Complétez le programme de l'exercice 4 en ajoutant les fonctions suivantes :

- Une fonction `int SuppValTab(int [] ,int, int)` qui supprime toutes les occurrences d'une valeur donnée dans un tableau d'entiers et tasse les éléments restants. Les arguments de la fonction seront le tableau à modifier, sa taille et la valeur à supprimer ; la valeur renvoyée par la fonction sera la nouvelle taille du tableau, une fois les cases supprimées

- Une fonction `? MedTab( ?)` qui calcule et renvoie la médiane d'un tableau d'entiers. On rappelle que la médiane d'une série de nombres est la valeur qui coupe en 2 parties égales cette série, une fois celle-ci ordonnée (autrement dit, si *med* est la médiane, il y a autant de valeurs supérieures à *med* que de valeurs inférieures à *med*). Vous pourrez procéder de la façon suivante :
  - ✓ on met dans *med* la valeur minimum du tableau,
  - ✓ on compte le nombre de valeurs du tableau inférieurs ou égales à *med*,
  - ✓ si ce nombre est inférieur au nombre total de valeur divisé par 2, on incrémente *med* de 1 et on recommence,
  - ✓ la valeur de la médiane est la 1ère valeur de *med* pour laquelle la condition ci-dessus n'est plus vérifiée.

- Modifiez le programme principal de façon à appeler afficher la médiane du tableau puis demander à l'utilisateur la valeur à supprimer et afficher le tableau une fois cette valeur supprimée

9. Lecture d'un nombre niv 3

Ecrire un programme permettant d'entrer une chaîne de caractères qui doit former un nombre ayant au plus 12 chiffres, éventuellement une virgule ou un point, avec un signe + ou - pouvant éventuellement précéder le nombre. Tant que ces contraintes ne sont pas respectées, l'utilisateur du programme doit être invité à recommencer la saisie du nombre. Sinon, le programme doit transformer cette suite de caractère en nombre flottant et l'afficher comme tel.

TP 6 - REVISIONS ET COMPLEMENTS

1. Conjecture de Syracuse niv 1

Prenez un entier positif ; s'il est pair, divisez-le par 2 ; s'il est impair, multipliez-le par 3 et ajoutez lui 1… Réitérez ce processus sur plusieurs exemples : que semble-t-il se passer ?

Partons par exemple de l'entier 7, et regardons la suite alors construite : 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1…. Cette suite devient cyclique, puisque l'obtention de la valeur 1 fait « boucler » indéfiniment l'algorithme.

On conjecture que l'on finit toujours par trouver la valeur « 1 » au fil des calculs quel que soit l'entier de départ : c'est la « **conjecture de Syracuse** » (encore appelée « problème *3n+1* ») qui attend toujours une preuve ! Beaucoup d'encre a coulé, et coulera sans doute encore, autour de ce problème, si bien qu'un vocabulaire métaphorique s'est construit comme souvent en mathématiques.

Reprenons l'exemple initial de l'entier 7. On appellera la suite (7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1) la **trajectoire** ou le **vol** de 7. Chaque entier de cette suite est une **étape** du vol, 52 est **l'altitude maximale** de la trajectoire. La **durée** d'un vol (16, ici) est le nombre d'étapes nécessaires avant l'apparition du premier '1' (s'il apparaît bien sûr !). Le **facteur d'expansion** est le rapport entre l'altitude maximale et la valeur initiale.

Concevez un programme permettant d'expérimenter la conjecture de Syracuse. Dans cette première version de l'algorithme, on ne calculera et affichera que la trajectoire de l'entier considéré.

2. Nombres amicaux niv 2

En mathématiques, deux nombres entiers *n* et *m* sont dits **amicaux** ou **aimables** si la somme des diviseurs de *n* (*n* **exclus**) vaut *m* et la somme des diviseurs de *m* (*m* **exclus**) vaut *n*.

Par exemple, 220 et 284 forment une paire de nombres aimables :
Somme des diviseurs de 220 : 1+2+4+5+10+11+20+22+44+55+110 = 284
Somme des diviseurs de 284 : 1+2+4+71+142 = 220

- ✓ Ecrire une fonction `int somdiv(int a)` qui renvoie la somme des diviseurs d'un entier *a* (*a* exclus).
- ✓ Ecrire une fonction `? isamical(?)` qui teste si le nombre entier passé en argument est un nombre aimable : dans ce cas, la valeur renvoyée sera la 2nde valeur de la paire de nombre amicaux, sinon le sous-programme renverra la valeur 0.
- ✓ Ecrire le programme principal dans lequel on demande à l'utilisateur de rentrer un nombre entier, puis on indique si le nombre est aimable et on affiche, le cas échéant, le second nombre de la paire de nombres amicaux.

3. Encore des opérations sur un tableau niv 2

On désire écrire un programme dans lequel on demande à l'utilisateur de rentrer la taille *n* d'un tableau d'entiers (max : NT = 50) puis le programme doit :

- ✓ Remplir le tableau avec des valeurs aléatoires entre 10 et 30
- ✓ Afficher le tableau en écrivant les valeurs les unes à la suite des autres sur la même ligne
- ✓ Demander une nouvelle valeur à l'utilisateur
- ✓ Calculer et renvoyer le nombre de multiples de ce nombre contenus dans le tableau
  Par exemple, si on donne en entrée de la fonction la valeur 3 et le tableau suivant, de taille 8 :
  { 7, -3, 22, 9, 12, -13, 11, 27} on affichera la valeur 4
- ✓ Permuter le tableau (*ie*. permuter la 1ère et la dernière case, la 2ème et l'avant-dernière, …) puis le réafficher.

Récupérer sur Moodle le fichier tp6exo3.c. Compléter ce programme en écrivant les définitions des fonctions appelées par le programme principal. **Vous ne modifierez pas le programme principal.**

4. Ecriture et lecture dans des fichiers (en mode texte) niv 2

Il est souvent utile de pouvoir mémoriser des valeurs de façon pérenne une fois un programme terminé. Pour cela, ces informations seront stockées dans un fichier, il est pour cela nécessaire de savoir manipuler un fichier pour **écrire** dedans. De même, une fois ces informations stockées, il faut être capable de les récupérer pour pouvoir les réutiliser, il est alors nécessaire de savoir **lire** un fichier.

La gestion des fichiers en langage C est expliquée au chapitre 12 du Manuel de Survie. Pour pouvoir manipuler un fichier, il faut en premier l'ouvrir et définir une variable de type « pointeur sur fichier » pointant sur celui-ci. Les données peuvent être écrites (et lues) dans un fichier en mode texte (fonctions fprintf et fscanf) ou en mode binaire (fwrite et fread). Chaque mode possède des avantages et des inconvénients (taille du fichier, sécurité, facilité d'accès …). Dans le cadre de ce TP, nous ne verrons que les fichiers ouverts en mode texte.

La manipulation d'un fichier se fera ainsi en plusieurs étapes :

- ✓ Déclaration d'une variable de type « pointeur sur fichier » (soit `FILE*` en langage C) qui nous permettra de faire référence à ce fichier
- ✓ Appel de la fonction fopen en mode texte pour ouvrir le fichier que l'on désire manipuler. On précisera à ce moment si l'on souhaite écrire (mode « write ») ou lire (mode « read ») dans le fichier.
- ✓ Transfert des informations entre le programme et le fichier : on utilisera la fonction fprintf pour écrire dans le fichier ou fscanf pour lire depuis celui-ci
- ✓ Appel de la fonction fclose pour fermer l'accès au fichier. Cette opération est nécessaire, en particulier si le fichier est ouvert en mode « write », l'écriture des données dans le fichier ne sera définitive que une fois celui-ci fermé.

a) Entrainement en mode écriture

Récupérez sur Moodle le fichier tp6exo4a.c. Complétez ce programme pour que les valeurs manipulées (entier a, réel x, tableau T et chaine de caractère chaine) soient copiées dans un fichier nommé « tp6exo4.txt ». Pour vérifier que les opérations d'écriture se sont bien déroulées, vous pourrez ouvrir le fichier avec un éditeur de texte pour visualiser ou même modifier directement son contenu.

b) Entrainement en mode lecture

Ecrire un second programme nommée tp6exo4b.c permettant de lire ces valeurs depuis le fichier et de les afficher à l'écran. Votre programme devra également afficher le produit de l'entier lu par le réel lu ainsi que la somme des éléments du tableau.

5. Syracuse, le retour niv 2

Enrichissez le programme de l'exercice 1 pour calculer également l'altitude maximale, la durée de vol et finalement le facteur d'expansion.

6. Manipulation de notes niv 2/3

On se propose d'écrire un programme de manipulation des notes d'un groupe d'élèves (maximum : 50 élèves). Chaque élève dispose de 4 notes. On définit pour cela un tableau à 1 dimension pour chaque matière.
Récupérer sur Moodle le fichier tp6exo6.c.

a) Compléter ce programme en écrivant la définition de la fonction `AffNotes` qui permet d'afficher les notes sous la forme d'un tableau à 2 dimensions : une ligne par élève et une colonne par matière.

b) Modifier le programme de façon à ce que le remplissage des 4 tableaux de notes se fasse dans une fonction `void RempNote(int [], int)`, le programme principal se contentant d'appeler cette fonction pour chaque matière.

c) Lorsque l'application commence à être conséquente, il devient pénible, voire dangereux de travailler avec un seul fichier source. De plus, le fait de découper l'application sur plusieurs fichiers permet à plusieurs personnes de travailler dessus sans avoir à modifier en parallèle le même fichier.

Vous allez donc l'éclater en plusieurs fichiers en regroupant les fonctions logiquement. A ce stade de l'application, vous isolerez les fonctions remplissant et affichant les tableaux dans un 1er fichier et les fonctions s'occupant des calculs de statistiques dans un 2ème fichier. Il est alors nécessaire de définir un fichier d'entête unique (header, suffixe .h) spécifique à votre application dans lequel vous mettrez toutes les inclusions de bibliothèques standard, définitions de constantes symboliques et prototypes de fonctions. Il faudra alors inclure ce fichier d'entête dans chacun des fichiers sources de la même façon que les fichiers d'entête standard (stdio.h, math.h, etc.). Il faut cependant remplacer les séparateurs '<' et '>' par des apostrophes ' " '.

L'obtention du programme exécutable final se fait en réalité en 2 étapes :
- Chaque fichier source est compilé séparément, ce qui génère un fichier objet (suffixe .o). Pour cela, il suffit de lancer le compilateur gcc avec l'option -c
- Les différents fichiers objet sont réunis lors de l'étape d'édition de liens, le fichier exécutable est alors créé. Pour cela, il suffit de compiler communément tous les fichiers objet précédemment créés.

Ceci permet de ne compiler que ce qui est nécessaire : lors de la modification d'un fichier source, il suffit de ne recompiler que ce fichier, puis de refaire l'édition de liens. De même, cela permet de déboguer les différents fichiers source un par un, sans avoir à attendre que la totalité du projet ait été écrite.

Cet ensemble de tâche peut être automatisé par l'intermédiaire de la commande make et d'un fichier Makefile qui regroupe les différentes instructions de compilation à effectuer. Cette commande tient compte des dates de modifications des différents fichiers et ne recompilera que les fichiers sources qui auront été modifiés depuis la dernière génération d'exécutable.

Dans Code::Blocks, cette étape est effectuée automatiquement : il faut pour cela définir un projet (menu File > New > Project > Empty Project) dans lequel vous inclurez tous les fichiers utilisés (source (*.c) et entête (*.h))

A ce stade, le répertoire contentant les fichiers sources de votre application devra contenir 4 fichiers :
- Notes.h : fichier d'entête qui contient les inclusions de bibliothèques, les constantes symboliques et les prototypes de fonctions ;
- Notes_main.c : fichier qui contient le programme principal ;
- Notes_inout.c : fichier qui contient les fonctions `AffNotes et RempNotes` ;
- Notes_Stats.c : fichier qui contient les fonctions `MoyEleves` et `MaxMatiere`

d) Compléter maintenant les fonctions `MoyEleves` et `MaxMatieres` pour implémenter le traitement qu'elles sont censées réaliser.

e) L'objectif est maintenant de pouvoir mémoriser les notes des élèves dans un fichier. Pour cela, vous allez écrire une fonction `SaveNotes` qui stockera les 4notes de chaque élève dans un fichier, sur le même modèle que l'affichage qui a été réalisé : une ligne par matière et une colonne par note. Vous sauvegarderez au préalable, en 1ère information dans le fichier, le nombre d'élèves. Cette fonction sera écrite dans un nouveau fichier, Notes_files.c, qui sera intégré dans votre projet. La fonction sera appelée dans le programme principal une fois le reste des opérations effectuées.

f) Il est maintenant possible de récupérer les notes et le nombre d'élèves depuis le fichier. Pour cela, vous écrirez la fonction `ReadNotes` (dans le fichier source Notes_files.c) qui remplira les 4 tableaux de notes et renverra le nombre d'élèves. Dans le programme principal, l'utilisateur devra avoir le choix entre saisir le nombre d'élèves et générer les notes aléatoirement comme précédemment ou récupérer ces informations depuis le fichier.

g) Compléter le programme de façon à afficher un histogramme des moyennes de la classe. Ceci sera réalisé par une fonction `void HistoMoy(float Moy[], int n)` qui sera écrite dans le fichier Notes_stats.c.

Pour cela, vous pourrez établir un tableau de taille 5 composé de la façon suivante :

```
l'élément d'indice 0 contient le nombre de notes de 0 à 4
l'élément d'indice 1 contient le nombre de notes de 4 à 8
l'élément d'indice 2 contient le nombre de notes de 8 à 12
l'élément d'indice 3 contient le nombre de notes de 12 à 16
l'élément d'indice 4 contient le nombre de notes de 16 à 20
```

Il faut alors établir un graphique de barreaux représentant le tableau. Utilisez les symboles `#######` pour la représentation des barreaux et affichez le domaine des notes en dessous du graphique.

Idée: Déterminer la valeur maximale maxn dans le tableau et afficher autant de lignes sur l'écran. (Dans l'exemple ci-dessous, maxn = 6).

Exemple:

```
6  >                      #######
5  >                ####### #######
4  >         ####### ####### #######
3  >         ####### ####### ####### #######
2  > ####### ####### ####### ####### #######
1  > ####### ####### ####### ####### #######
    +-------+-------+-------+-------+-------+
    I  0-4  I  4-8  I  8-12 I 12-16 I 16-20 I
```