UPVD
Université de Perpignan Via Domitia

Université de Perpignan Via Domitia

Licence de Sciences et Technologies
Mention Sciences de l'Ingénieur

Programmation en C – Exercices

Enoncés
Sujets des partiels et examens
Exemples de corrections des exercices
Exemples de corrections des partiels et des examens

Ph. LANGLOIS

Version du 18 septembre 2013

Table des matières

Première partie

Enoncés des exercices

Chapitre 1

Description de l'environnement de programmation

Cette partie décrit la prise en main de l'environnement de programmation `gcc` utilisé en TD sous environnement Linux ou avec `cygmin` sous Windows [1].

Convention à utiliser pour les extensions des noms de fichiers :
`.c` : fichier source C
`.h` : fichier en-tête (header) C
`.o` : fichier objet
sans extension : exécutable

On distingue deux cas, selon que le programme est composé d'un seul fichier (cas simple) ou de plusieurs (cas général).

1.1 Du code source à l'exécution : cas simple

Trois étapes sont nécessaires pour créer un fichier exécutable à partir d'un fichier de code source.

1. Compiler : génération du fichier objet `.o` – nous verrons plus loin que cette étape comporte en fait des étapes intermédiaires.
2. Linker : édition des liens entre les fichiers objets et création du fichier exécutable.

Ces deux étapes peuvent être aussi regroupées en une seule commande, ce que nous apprendrons d'abord.

1.1.1 Le code source

Utiliser votre éditeur favori pour taper la version C suivante du classique "Hello World" . . .en français.

```
/*Vous l'avez reconnu : Bonjour le monde !*/
#include<stdio.h>

int main()
{
  printf("Bonjour le (centre du) monde !\n");
  return 0;
}
```

Sauvegarder ce programme dans le fichier `bonjourlemonde.c`.

1. Les principales spécificités de cette configutaion sont décrites à la section 1.4

1.1.2 Méthode 1 : génération directe de l'exécutable

Le cas simple du code contenu dans un fichier unique permet d'utiliser une seule commande à cet effet.

La version la plus courte est

```
$ gcc  bonjourlemonde.c
```

A partir du fichier source `bonjourlemonde.c`, le compilateur `gcc` génère un exécutable appelé `a.out` qui se trouvera au même endroit que le fichier source. Pour l'exécuter il suffit alors d'entrer la commande suivante (dans une fenêtre `terminal`).

```
$ ./a.out
```

ou plus simplement encore,

```
$ a.out
```

si votre variable d'environnement `PATH` permet d'examiner le répertoire courant.

Il est plus pratique de pouvoir choisir le nom de son exécutable en le précisant avec l'option `-o`.

```
$ gcc -o bonjourlemonde bonjourlemonde.c
```

Profitons-en pour introduire quelques **options de compilation obligatoires tout au long de cette année**. Vous utiliserez en effet la commande plus longue suivante.

```
$ gcc -Wall -pedantic -std=c99 -o bonjourlemonde bonjourlemonde.c
```

- `-std=c99` indique au compilateur de traiter le code selon le standard C99 (et donc de rejeter certaines extensions comme celles de GNU par exemple)
- `-pedantic` permet de signaler les avertissements, ou *warnings*, selon la norme ISO ; et
- `-Wall` permet de signaler un grand nombre d'autres *warnings* décrit dans le `man gcc`.

En effet à la différence d'Ada, la grande permissivité de C réduit l'aide du compilateur (sans option) pour traquer certaines erreurs et les mauvaises pratiques de programation.

1.1.3 Méthode 2 : compiler et lier avant d'exécuter

Dès que votre application comporte plusieurs fichiers, il est nécessaire de procéder en deux étapes :
i) compilation-génération (option `-c`) d'un fichier objet (extension `.o`)
ii) puis création de l'exécutable (sans extension) par éditions des liens (option `-o`) entre les différents fichiers objets.
Dans le cas simple, chaque étape (compilation, édition des liens) correspond à une des deux lignes de commandes suivantes.

```
$ gcc -Wall -pedantic -std=c99 -c bonjourlemonde.c
$ gcc -Wall -pedantic -std=c99 -o bonjourlemonde  bonjourlemonde.o
```

Il suffit maintenant d'exécuter `bonjourlemonde`.

1.1.4 Pour bien comprendre

- Recommencer les méthodes 1 et 2 à partir de fichiers sources modifiés (contenu, nom du fichier, autres extensions), ailleurs dans votre arborescence.
- Observer le contenu du répertoire de travail après chaque commande.
- Observer les droits sur les différents fichiers.

1.2 Du code source à l'exécution : cas général

Nous détaillons maintenant le cas habituel d'applications comportant plusieurs fichiers sources et l'utilisation de la commande `make`.

1.2.1 Editer-compiler-lier-exécuter

Entre la création du code source (édition) et l'exécution du binaire qui en découle, nous retrouvons les deux étapes de compilation et d'édition des liens déjà décrits.
Lorsque que l'application est constituée de plusieurs fichiers (en général des fichiers `.c` et les fichiers d'en-tête `.h` correspondants[2]), générer l'exécutable nécessite de créer, un à un, tous les fichiers objets (`.o`) associé aux fichiers `.c` (compilation et génération des fichiers objets, option `-c`) , puis de lier (édition des liens, option `-o`) tous ces fichiers objets (et si besoin les bibliothèques extérieures) en précision le nom de l'exécutable (sans extension en général).
Par exemple, notre application est constituée des fichiers `F1.c, F2.c, main.c`, ce dernier contenant la procédure principale `main`. Nous voulons l'exécuter par le biais de la commande (sans argument) `endavant`. On y arrive en effectuant les deux étapes compilation–éditions des liens suivantes.

```
$ gcc -Wall -pedantic -std=c99 -c F1.c
$ gcc -Wall -pedantic -std=c99 -c F2.c
$ gcc -Wall -pedantic -std=c99 -c main.c
$ gcc -Wall -pedantic -std=c99 -o endavant  F1.o F2.o main.o
```

Il est courant de devoir modifier un fichier parmi tous ceux qui consitue une application. Dans ce cas, il n'est pas nécessaire de recompiler des fichiers non modifiés pour créer un nouvel exécutable. Il suffit de mettre à jour le fichier objet concerné puis de créer une nouvelle versi de l'éxécutable.

1.2.2 Utiliser un `Makefile`

Cette partie sera décrite une fois la compilation séparée traitée en cours.

1.3 Pratique et conseils associés

1.3.1 Une session classique

1. Créer un fichier source sous l'éditeur,
2. le compiler,
3. corriger (sous l'éditeur) les erreurs signalées par le compilateur,
4. le recompiler puis revenir à l'étape 3 tant que la compilation ne s'achève pas avec succès,
5. exécuter le programme sur différents jeux de données bien choisis,
6. identifier ainsi les (premiers) *bugs* et revenir à l'étape 3 jusqu'à obtenir l'exécution attendue.

1.3.2 Conseils pour l'écriture d'un fichier source

La plupart des conseils suivants ont pour objectif d'améliorer la lisibilité des fichiers sources, la sûreté, la maintenabilité et la portabilité des développements.

– Sauvegarder régulièrement vos fichiers (ne pas écrire du code pendant 10 minutes sans sauvegarder !).
– Choisir des identifiants explicites (peuvent être long, utiliser le trait bas (_), mettre des commentaires (le point précédent répond aussi à cet impératif).
– Initialiser chaque variable lors de sa déclaration, séparer par une ligne vide les déclarations des instructions. Limiter la portée des variables au minimum raisonable (compromis entre la localitéun regroupement des déclarations et la distance "déclaration–première utilisation" .

1.4 Utiliser `cygwin` sous M* Windows.

`cygwin` permet d'utiliser le compilateur `gcc` dans l'environnement système M* Windows. Les commandes présentées s'appliquent donc dans la fenêtre `cygwin`. Nous présentons maintenant quelques spécificités de cette configuration.

2. Ceci sera décrit en cours (compilation séparée).

Installation ? Une fois le fichier de `setup` téléchargé, l'installation nécessite deux éxecutions successives de ce `setup`. En effet, `gcc` n'est pas installé à la première exécution du `setup` (suivre la procédure avec copie des fichiers en local). Une fois celle-ci effectuée, il faut relancer le `setup`, choisir le mode manuel pour sélectionner les paquets `gcc` et `automake` qui seront alors installés et directement utilisables.

Où travailler ? La fenêtre `cygwin` fournit une invite de commande (*prompt*) dans son répertoire d'installation et non pas dans votre *home directory*. Il faut donc utiliser les commandes Unix `cd, ..,` `~userd-id`, ... pour retrouver une configuration de travail classique (accessibilité directe aux fichiers sources). Un lien sur une page qui présente ces commandes est indiqué sous l'ENT (dans "ressources utiles").

1.5 Pour être sûr d'avoir bien compris

1. Coder, compiler et exécuter le programme présenté en cours.
2. Récupérer l'archive `compiler_fichiers.zip` sur l'ENT et créer l'exécutable. Observer le contenu du répertoire de travail après chaque commande. Modifier certains fichiers et créer de nouveaux exécutables.
3. Ecrire un programme qui échange deux valeurs entières, flottantes et caractères. Ces valeurs seront (version 1) d'abord fixées dans le code, puis (version 2) saisies au clavier.
4. Ecrire un programme qui lit trois valeurs flottantes au clavier et qui affiche leur somme. Proposer ensuite des versions qui utilisent des structures itératives différentes : `for, while, ...`.

Chapitre 2

Variables, valeurs, constantes, déclarations, types scalaires, opérateurs, expressions, entrées-sorties simples

Exercice 1. (`ma_saisie`) La fonction `saisir` de l'archive utilisée au TD-1 illustre comment utiliser les fonctions d'entrées-sorties clavier-écran `printf` et `scanf`. En s'inspirant de cet exemple, écrire un programme qui lit une valeur au clavier puis l'affiche à l'écran. Vous modifierez ce programme pour afficher n fois une valeur V d'un des types vus en cours, les valeurs de n et V seront "lues au clavier".

Exercice 2. (`swap3`) Ecrire un programme qui permute deux valeurs entières. Ces valeurs seront lues au clavier. Même question pour trois valeurs flottantes et caractères.

Exercice 3. (`operateurs`) Ecrire un programme qui affiche le résultat des opérateurs entiers pour deux valeurs arbitraires choisies au clavier. Vous interpréterez les résultats pour un choix de ces opérateurs laissé à votre initiative (des exercices à venir porteront en particulier sur les opérateurs logiques, bit-à-bit et composés).
- Opérateurs arithmétiques : `+, -, *, /, %`
- Opérateurs relationnels : `==, !=, <=, >=`
- Opérateurs logiques : `&&, ||, !`
- Opérateurs bit-à-bit : `&, ^ , ~ , «, »`
- Affectation composée : `+=, -=, *=, /=, %=, &=, ^=, |=, «=, »=`
- Incrément et décrément : `x++, ++x, x--, --x`

Exercice 4. (`int2bin`) Ecrire un programme `int2bin` qui calcule la décomposition binaire d'un entier (valeur arbitraire lue au clavier). On rappelle que cette décomposition s'obtient par les restes successifs des divisions entières par 2. Faire afficher cette décomposition dans l'ordre du calcul. Proposer une solution pour afficher cette écriture dans l'ordre de lecture classique (bits de poids décroissant de la gauche vers la droite).

Exercice 5. (`tabverite`) Ecrire un programme qui calcule et affiche les tables de vérités des opérateurs logiques `&&, ||, !`. Vous pourrez utiliser le type `Bool` de `<stdbool.h>` et aussi des valeurs de votre choix.

Exercice 6. (`formats`) Parmi les formats d'affichage utilisables avec `printf` et `scanf`, les plus simples sont de la forme `%` suivi d'un caractère, le caractère pouvant être `d, i, o, O, x, X, c, f`. Utiliser ces différents formats d'affichage et comparer les sorties obtenues en les appliquant par exemple aux valeurs entières 74 et 137. Commenter ce que vous observez.

Chapitre 3

Opérateurs entiers, bit à bit, logiques, tests

Exercice 7. (`op_int`). Calculer et afficher les différents résultats des opérateurs arithmétiques entiers `/,%` (quotient, reste). Faites varier les signes des opérandes ; par exemple, `(5, 3),(-5, 3),(5, -3),(-5, -3)`.

Exercice 8. (`int_vs_unsigned`). Observer l'effet de la conversion de type entre des valeurs `int` et `unsigned int`, en particulier dans le cas de valeurs négatives. Utiliser la fonction `sizeof` ainsi que les formats d'affichage `d, u, ld, lu, x, lx` (précédés du caractère %). Même question avec les types `long` et `unsigned long`.

Exercice 9. (`minmax`) Ecrire une fonction `minmax` qui affiche le minimum et le maximum de deux valeurs entières saisies au clavier.

Exercice 10. (`div`). Quelles sont les valeurs des expressions `(float)i/j`, `(float)(i/j)` et `(float)i/(float)j` pour `int i, j;` ? Donner une réponse puis vérifier pour `i=1` et `j=4`.

Exercice 11. (`op_bit`). Dans cet exercice les affichages seront systématiquement réalisés en décimal et en hexadécimal (resp. `%d`, `%x`) et en binaire grâce à la fonction `printbin` disponible sur l'ENT. Les positions des bits sont comptés à partir de la droite en commençant par 0.

1. Calculer et afficher les tables de vérité des opérateurs bit-à-bit `not, and, or, xor` et les comparer aux opérateurs booléens "correspondants".
2. Appliquer les opérateurs bit-à-bit à des valeurs entières entrées au clavier. Vérifier les résultats.

Exercice 12. (`op_decall`). Dans cet exercice les affichages seront systématiquement réalisés en décimal, en hexadécimal (resp. `%d`, `%x`) et en binaire grâce à la fonction `printbin` disponible sur l'ENT.
Pour $i = -2, \ldots, 5$, calculer et afficher des décallages de i positions à droite et à gauche de valeurs arbitraires entrées au clavier. Interpréter le sens de $x << 2$, ou plus généralement de $x << i$, pour $i \geq 0$. Commenter les résultats obtenus dans le cas des positions négatives ($i < 0$).

Exercice 13. (`masques`). Dans cet exercice les affichages seront systématiquement réalisés en décimal et en hexadécimal (resp. `%d`, `%x`) et en binaire grâce à la fonction `printbin` disponible sur l'ENT. Les positions des bits sont comptés à partir de la droite en commençant par 0.

1. Utiliser l'opérateur de décallage pour déclarer des constantes littérales $m\_i = 2^i$, pour $i = 0, \ldots, 5$.
2. Utiliser ces constantes m_i pour réaliser les traitements suivants sur des valeurs arbitrairement entrées au clavier.
   (a) déterminer la parité de la valeur entrée (proposer 2 solutions) ;
   (b) déterminer si le bit de position 3 est égal à 0 ;
   (c) déterminer si l'octet de poids faible est égal à 7 ;
   (d) mettre à 1 le bit de position 0 ;

(e) mettre à 1 les bits de position 1 et 4 ;
(f) mettre à 0 les bits de position 1 et 2 ;
(g) afficher l'octet de poids faible ;

Chapitre 4

Fonctions mathématiques, tableaux, boucles, constantes symboliques

A propos des fonctions mathématiques. Les fonctions (dites) élémentaires de calcul numérique (fonctions trigonométriques, exponentielles, logarithmiques, inverses) sont fournies dans la bibliothèque mathématique d'en-tête `math.h`. Il peut être nécessaire de compléter l'édition de liens en spécifiant d'inclure cette bibliothèque ; ceci s'effectue grâce à l'option `-lm` (sous Linux).
Exemple : `gcc -Wall -pedantic -std=c99 -lm -o somme_p somme_p.c`

Exercice 14. (`somme, somme-p`) Ecrire un programme qui calcule $s_n = \sum_{k=1}^{n} k = 1 + 2 + \cdots n$, où n est arbitrairement saisi au clavier. Faire de même pour `somme_p` qui calcule $s_{n,p} = \sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p$. On pourra dans ce cas introduire des `float` ou des `double`.
On utilisera la fonction `pow(x,y)` ($= x^y$) de la bibliothèque `math.h`.

Exercice 15. (`felem`) Écrire un programme qui calcule, stocke puis affiche les valeurs de $\log_{10} N, N, N \times \log_{10} N, N^2$, pour $N = 10^k$ et k entier tel que $k = 1, 2, \ldots, 12$.
On utilisera la fonction `log10(x)` de la bibliothèque `math.h`.

Exercice 16. (`fibo, fibo-inv`) La suite de Fibonnacci est définie par $u_0 = 0, u_1 = 1$, et $u_n = u_{n-1} + u_{n-2}$ pour $n \geq 2$. On s'intéresse à la valeur du n-ième terme de cette suite pour une valeur de n arbitrairement choisie par l'utilisateur et inférieure à 100.

1. Écrire un programme `fibo` qui demande à l'utilisateur une valeur pour n, puis affiche les valeurs k et u_k pour k variant de 0 à n. Observer les résultats obtenus selon que les valeurs calculées soient en `int, long, long long` et `float`.
2. Écrire un programme `fibo-inv` qui demande à l'utilisateur une valeur pour n, puis affiche les valeurs k et u_k pour k variant de n à 0, c'est-à-dire qui affiche les n premières valeurs de la suite de Fibonnacci **en ordre inverse de celui du calcul**.

Exercice 17. (`fois2fois2etc`) Cet exercice utilise la suite u_i des puissances de 2, $u_i = 2^i = 2 \times 2 \times \cdots \times 2 = 2 \times u_{i-1}$. Ecrire le programme qui réalise les traitements suivants.

1. Stocker dans un tableau les 32 premières puissances de 2 (de 2^0 à 2^{31}) en utilisant la relation $u_i = 2^i$.
2. Stocker dans un tableau les 32 premières puissances de 2 (de 2^0 à 2^{31}) en utilisant la relation $u_i = 2 \times u_{i-1}$.
3. Calculer la valeur p de la plus grande puissance de 2 inférieure ou égale à un entier $n > 0$ donné, soit $2^p \leq n < 2^{p+1}$ (la valeur de n est entrée au clavier).
4. Calculer le "reste" $r = n - 2^p$ où p est la valeur donnée par le calcul précédent.

Exercice 18. (`max-tab`) Ecrire un programme `max-tab` qui identifie le maximum d'un tableau d'entiers de longueur quelconque. La longueur du tableau sera définie par une constante symbolique (macro sans paramètre).

Exercice 19. (`ind-max-tab`) Ecrire un programme `ind-max-tab` qui identifie l'indice de la première occurrence du maximum d'un tableau d'entiers de longueur quelconque. La longueur du tableau sera définie par une constante symbolique (macro sans paramètre).

Exercice 20. (`last-ind-max-tab`) Modifier le programme précédent en `last-ind-max-tab` qui identifie l'indice de la dernière occurrence du maximum d'un tableau d'entiers de longueur quelconque. La longueur du tableau sera définie par une constante symbolique (macro sans paramètre).

Chapitre 5

Structures de contrôle : répétition, choix

Exercice 21. `(double-indice)` Quelle est la suite d'indices décrite par la boucle suivante ? Répondre puis vérifier en codant pour différentes valeurs de `MIN` et `MAX`.

```
#define MIN 2
#define MAX 18
...
for (i=MIN, j=MAX; i<j; i++, j /= 2) ...
```

Exercice 22. `(des-boucles-for)` Ecrire les programmes qui produisent les affichages suivants.

1. Avec une boucle `for`.

   (a)
   ```
   1 2 3 4 5
   ```

   (b)
   ```
   1
   2
   3
   4
   5
   ```

2. Avec **deux** boucles `for`.

   (a)
   ```
   1
   1 2 3
   2
   1 2 3
   3
   1 2 3
   4
   1 2 3
   ```

   (b)
   ```
   1
   1
   2
   1 2
   3
   1 2 3
   4
   1 2 3 4
   ```

   (c)
   ```
   1
   1
   2
   2 4
   3
   3 6 9
   4
   4 8 12 16
   ```

3. Après avoir déclaré et initialisé une chaîne de caractères à la valeur `abcde`, écrire les programmes qui parcourent la chaîne de caractères pour donner les affichages suivants. Le cas présenté correspond à l'affichage de 4 caractères maximum. Une seconde version de ces programmes permettra de lire au clavier le nombre de caractères maximal à afficher.

   (a)
   ```
   1
   a
   2
   ab
   3
   abc
   4
   abcd
   ```

   (b)
   ```
   1
   e
   2
   ed
   3
   edc
   4
   edcb
   ```

   (c)
   ```
   1
   e
   2
   de
   3
   cde
   4
   bcde
   ```

Exercice 23. `(des-boucles-while)` Reprendre l'exercice précédent en utilisant des boucles `while` à la place des boucles `for`.

Exercice 24. (un-deux-etc) Ecrire un programme qui pour une valeur de i choisie au clavier, affiche i fois la valeur i sur la i-ème ligne. Pour $i = 5$, on a par exemple l'affichage suivant.

```
1
22
333
4444
55555
```

Exercice 25. (suites) Les suites $u_n = n - 5$ et $v_n = -2n + 10$ varient de façon opposée. Calculer à partir de quelle valeur de $n > 0$, $u_n \geq v_n$. Faire de même avec $u_n = 10n$ et $v_n = n + 100$, puis $u_n = 2^n$ et $v_n = 100n$ (qui elles, ne varient pas de façon opposée).

Exercice 26. (equa-prem-deg) Ecrire un programme qui résoud l'équation du premier degré $ax + b = 0$ pour a, b deux `float` arbitraires saisis au clavier (format de lecture %f). On affichera l'ensemble (éventuellement vide) des solutions calculées. Même question avec l'équation du second degré $ax^2 + bx + c = 0$, pour a, b, c des `float`.

Exercice 27. (aleas) La fonction `rand()` renvoit un entier (`int`) pseudo-aléatoire selon une distribution uniforme dans l'intervalle $[0, \texttt{RAND\_MAX}]$. Ecrire les traitements suivants en faisant varier à votre convenance le nombre de valeurs générées N en utilisant un `#define` ($N = 100, 1000, 10000$). Commenter les résultats observés.

1. Observer des ensembles de valeurs générées.
2. Calculer la répartition (en pourcentage) des valeurs inférieures et supérieures à `RAND_MAX/2` (moitié inférieure / moitié supérieure).
3. Calculer la répartition (en pourcentage) des valeurs sur les intervalles séparés par `RAND_MAX/4`, `RAND_MAX/2` et `3*RAND_MAX/4`.
4. Combien de tirages sont nécessaires pour générer 5 valeurs de plus dans la moitié inférieure que dans la moitié supérieure.
5. Généraliser le traitement précédent pour expérimenter combien de tirages sont nécessaires pour générer i valeurs de plus dans la moitié inférieure que dans la moitié supérieure avec $i = 1, 2, \ldots, 10$.
6. Il est classique de faire varier les valeurs générées à chaque exécution en initialisant la graine du générateur par l'heure du système. Ceci est obtenu grâce à l'appel `srand(time(NULL))` qui utilise la fonction `time` de la librairie `time.h` Observer les effets sur les résultats des questions précédentes

Exercice 28. (vect-alea, mat-alea) Ecrire un programme qui initialise de façon aléatoire puis affiche les objets suivants.

1. Un vecteur de `double` de taille définie comme constante symbolique.
2. Une matrice de `char` de taille (nombres de lignes et de colonnes) définie comme constante symbolique.

Chapitre 6

Fonctions : prototypes, définition, appels

Exercice 29. (`f-fibo`) Reprendre l'exercice `fibo` (Exercice 16) en regroupant le calcul dans une fonction et l'appel dans un `main`. Partager ensuite en 3 fichiers (un `.h` et deux `.c`) et obtenir un exécutable équivalent.

Exercice 30. (`f-masques`) Reprendre l'exercice `masques` (Exercice 13) en regroupant les traitements bit-à-bit dans des fonctions. Utiliser ces fonctions sur des valeurs arbitraires appellées depuis un `main`.

Exercice 31. (`f-fois2-var-glob`) Ecrire sous forme de fonctions la génération et l'affichage du tableau `fois2fois2etc` du chapitre 4. Ce tableau, de longueur fixée (32), sera une variable globale d'un programme `f-fois2-var-glob` qui génère puis affiche ce tableau (à l'aide des deux fonctions).

Exercice 32. (`f-fibo-tab`) On suppose maintenant que le tableau est un des paramètres passés aux fonctions demandées. Sa taille sera définie par une constante symbolique (macro sans paramètre).

1. Ecrire une fonction qui génère un tableau qui contient les n premières valeurs de la suite de Fibonacci.
2. Ecrire une fonction qui affiche ces n premières valeurs.
3. Effectuer ces traitements dans (`f-fibo-tab`).

Exercice 33. (`passage-param`) Coder et observer attentivement les effets du programme suivant qui illustre le passage de paramètre par valeur et le sens de l'identificateur d'un tableau.

```
#include <stdio.h>
#include <stdlib.h>

#define N 4

void modif_scal(int val)
{
  val = 1;
  printf("Dans modif_scal, val vaut %d \n", val);
}

void afficher(int tab[], int taille)
{
  for (int i=0; i<taille; i++) printf("%d \n", tab[i]);
}

void modif_tab(int tab[], int taille)
{
```

```
  for (int i=0; i<taille; i++) tab[i] = i;
  printf("Dans modif_tab, tab vaut \n");
  afficher(tab, taille);
}

int main(void)
{
  int un_i_global = 5;
  int un_tab[N] = {0}; // vaut 0 par défaut

  printf("Dans main, i vaut %d \n", un_i_global);
  modif_scal(un_i_global);
  printf("Après l'appel à modif_scal,  i vaut %d \n", un_i_global);
  printf("------------\n");

  printf("Dans main, tab vaut \n");
  afficher(un_tab, N);
  modif_tab(un_tab, N);
  printf("Après l'appel à modif_tab,  tab vaut \n");
  afficher(un_tab, N);


  return EXIT_SUCCESS;
}
```

Exercice 34. (`aff-vect-alea, aff-mat-alea`) Ecrire des fonctions qui affichent les objets suivants.

1. Un vecteur de `double` de taille définie comme constante symbolique.
2. Une matrice de `char` de taille (nombres de lignes et de colonnes) définie comme constante symbolique.

Les appliquer aux objets générés par `vect-alea` et `mat-alea` (Exercice 28).

Chapitre 7

Pointeurs

Exercice 35. (`ptr`) Ecrire un programme qui effectue les traitements suivants. Après chaque traitement et pour chaque pointeur ou valeur, afficher le contenu du pointeur, son adresse, la valeur pointée et l'adresse de la valeur pointée. Le format d'affichage des pointeurs est `%p`.
- déclarer un pointeur `p` sans l'initialiser et un pointeur `q` initialisé à `NULL` ;
- le pointeur `p` pointe sur une valeur entière `v` initialisée à 10 ;
- à l'aide du pointeur, modifier cette valeur entière `v` "à partir du clavier" ;
- utiliser seulement le pointeur `p` pour initialiser une valeur entière `w` à la valeur de `v` ;
- utiliser seulement un pointeur `r` qui pointe vers `p` pour modifier la valeur entière `v` "à partir du clavier".

Exercice 36. (`arith-ptr`) Ecrire un programme qui compare la différence des adresses entre deux pointeurs `p` et `q=p+1` pour des types d'objets pointés différents : `char`, `int`, `double`. Comparer avec le résultat de la fonction `sizeof(type)` (qui retourne un `unsigned long`).

Exercice 37. (`tab-ptr`) Déclarer un tableau de `double` et un pointeur sur son premier élément. Parcourir les éléments du tableau à l'aide de ce pointeur. Afficher les valeurs du tableau et leurs adresses, en utilisant le tableau et le pointeur.

Exercice 38. (`affiche-vect`) Ces fonctions d'affichage de vecteurs pourront être utilisées par la suite.

1. Ecrire une fonction `affiche` qui affiche un vecteur de `double` de longueur `nbval`. Commencer par une version qui travaille sur un tableau déclaré comme une variable globale.
2. Modifier la version précédente pour que le tableau et sa dimension soient des paramètres de `affiche`.
3. Ecrire une fonction `affiche2` qui affiche l'indice et la valeur associée d'un vecteur de `double` de longueur `nbval`. Cette fonction aura les mêmes paramètres que `affiche2`.
4. Regrouper ces fonctions dans un fichier d'en-têtes (`affiche-vect.h`) adapté.

Exercice 39. (`affichons-mat`) Ces fonctions d'affichage de matrices pourront être utilisées par la suite.

1. Ecrire une fonction `affiche_mat` qui affiche une matrice de `float` de taille `NL` $\times$ `NC`. Les nombres de lignes (`NL`) et (`NC`) seront des paramètres de la fonction d'affichage.
2. Appliquer cet affichage pour des matrices de taille arbitrairement choisie dans `affichons-mat`.
3. Proposer d'autres versions équivalentes à `affiche_mat` en modifiant les passages de paramètres : nombre minimal de paramètres, matrice vue comme un vecteur.

Exercice 40. (`mystery-inc`)

1. Compléter `affiche-vect.h` pour pouvoir afficher des tableaux de chaînes de caractères.
2. Déclarer les 5 tableaux de chaînes de caractères qui permettent de représenter les "personnages" suivants (nom, sexe, trait caractère principal, partenaire éventuel).

```
Fred, H, Volontaire, Daphné
Daphné, F, Esthétique, Fred
```

```
Sammy, H, Goinfre, Scooby-Doo
Scooby-Doo, Chien, Sammy
Véra, F, Analytique
```

Les afficher.

3. Déclarer un tableau `MystInc` qui regroupe les cinq personnages précédents. L'afficher.

Exercice 41. (`tab-dyn`) Ecrire un programme qui créé dynamiquement un vecteur (de `double`) de taille arbitrairement choisi par l'utilisateur. L'initialiser à l'aide de la fonction `rand()`. Ne pas oublier la libération de la place mémoire en fin de traitement. Appliquer les procédures d'affichage de l'exercice 38 au vecteur précédent.

Exercice 42. (`mat-dyn`) Ecrire un programme qui créé dynamiquement une matrice (de `float`) de taille (nombre de lignes, nombre de colonnes) arbitrairement choisie par l'utilisateur. L'initialiser à l'aide de la fonction `rand()`. Ne pas oublier la libération de la place mémoire en fin de traitement. Effectuer un affichage de cette matrice (ne pas appeler `affiche_mat` de l'exercice 39).

Exercice 43. (`main-inc`) Ecrire une fonction `incremente` qui incrémente une variable entière passée en paramètre. Son prototype est comme suit.

```
void incremente(int * val);
```

L'utiliser dans un `main-inc` en l'appelant un nombre de fois arbitrairement choisi par l'utilisateur. Afficher les adresses de la valeur incrémentée et du nombre de fois (le format d'affichage des pointeurs est `%p`).

Exercice 44. (`doubler-tab`) Ecrire une fonction qui double les valeurs d'un tableau de `double` passé en paramètre. L'appliquer à un tableau aléatoire et afficher.

Exercice 45. (`combien`, `echo`, `echo-inverse`)

1. Ecrire un programme-commande `combien` qui compte le nombre de mots d'une phrase entrée au clavier.
2. Ecrire un programme-commande `echo` qui affiche les valeurs de ses arguments entrés au clavier.
3. Ecrire un programme-commande `echo-inverse` qui affiche, dans l'ordre inversé, les valeurs de ses arguments entrés au clavier (la commande s'affiche en premier).

Exercice 46. (`max`) On va écrire un programme-commande `max` qui retourne la valeur maximale d'un nombre arbitraire de `double` passés en paramètres.

1. Utiliser la construction `argc,*argv[]` pour que l'appel à `max` donne le traitement suivant. On rappelle que la fonction `atof(val)` ou `strtod(val, NULL)` de `stdlib.h` convertit une chaîne de caractères ASCII en `double`.

```
$./max 34 55
55.000000

$./max 34 -11 88 55
88.000000
```

2. Définir une fonction interne `usage` qui permet de signaler l'erreur d'un appel à `max` sans aucun paramètre. `usage` provoquera un message d'erreur sur la sortie standard (pour l'instant).

Exercice 47. (`main-tx-var`) Ecrire une fonction `tx-var` qui calcule le taux de variation d'une fonction f entre a et b. La fonction f et les paramètres flottants a et b sont passés en paramètre de `tx-var`. Le taux de variation de f entre a et b vaut $(f(a) - f(b))/(a - b)$.
L'utiliser dans un `main-tx-var` en l'appelant pour des paramètres a et b arbitrairement choisis par l'utilisateur et les fonctions $f(x) = 2x, 3x$ et $-x$.

Chapitre 8

Types, `makefile`

Exercice 48.(`type-vect_mat`, `manip-type-vect-mat`, `makefile`) Les types `vecteurs` et `matrices` sont des tableaux (resp. 1D et 2D) d'entiers de taille arbitrairement fixée par une macro `#define`.

1. Définir ces types dans `type-vect-mat.h/c` ainsi que des fonctions d'affichage associés.
2. Tester ces fonctions à partir d'appels dans `manip-type-vect-mat`.
3. Ecrire un `makefile` pour ces codes.
4. Compléter `type-vect-mat.h/c` par une fonction qui génère le double d'un vecteur et une fonction qui initialise une matrice de façon aléatoire.
5. Tester ces fonctions à partir d'appels dans `manip-type-vect-mat`.

Deuxième partie

Sujets des partiels et examens.

Chapitre 9

Sujets

9.1 Partiel de novembre 2010

Durée : 1h15. Epreuve individuelle sur machine.
Tout document sous forme numérique enregistré sur un "support USB" autorisé.

Travail demandé. Vous traiterez l'exercice obligatoire **et deux** exercices que vous choisirez parmi les exercices 1, 2 et 3. La qualité de la programmation est prise en compte dans la notation.

Consigne pour la fin d'épreuve. *Vous rassemblez vos fichiers sources (`.c et .h`) dans une archive `votre_nom.zip` que vous déposez dans la zone "Travaux" de l'ENT. Vous vérifiez auprès de l'enseignant que votre dépôt est effectif avant de quitter la salle. L'absence de dépôt en fin d'épreuve est considérée comme "copie blanche".*

Exercice obligatoire.
Ecrire un programme `echauffement` qui réalise le réalise le traitement suivant sur des valeurs entière arbitrairement entrées au clavier. Dans cet exercice les affichages seront systématiquement réalisés en décimal et en hexadécimal (resp. `%d`, `%x`). Les positions des bits sont comptés à partir de la droite en commençant par 0.

1. Afficher la valeur entrée dans les formats demandés ;
2. tester si les bits de position 0 et 1 valent 0 ;
3. mettre à 0 les bits de position 0 et 1 ;
4. tester si les bits de position 0 et 1 valent 0.

Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrer un entier a : 14
Vous avez entré la valeur a suivante.
  a = 14 = 0xe
Affichage des bits de position 0 et 1 :
 res = 2= 0x2
Les bits de position 0 et 1 de 14 valent-ils O ?
 Faux.
On met à zéro les bits de position 0 et 1.
 res = 12 = Oxc
Les bits de position 0 et 1 de 12 valent-ils O ?
 Vrai.
```

Exercice 1.

1. Ecrire un programme `tri2` qui trie par ordre croissant deux valeurs `float` lues au clavier. Une exécution de ce programme donnera à l'écran ce qui suit.

   ```
   Entrez deux valeurs flottantes : 4.4 3.3
   Vous avez entré les valeurs 4.400000, 3.300000
   qui triées par ordre croissant sont 3.300000, 4.400000
   ```

2. Ecrire un programme `tri3` qui trie par ordre croissant trois valeurs `double` lues au clavier. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez trois valeurs flottantes : 4.4 1.1 3.3
Vous avez entré les valeurs 4.400000, 1.100000, 3.300000
qui triées par ordre croissant sont 1.100000, 3.300000, 4.400000
```

Exercice 2.

1. Ecrire un programme `expression` qui calcule $n_1 + 2n_2 - n_3^3$, pour trois valeurs entières positives $n1, n2, n3$ saisies au clavier. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez trois entiers positifs : 3 1 5
Pour n1=3, n2=1, n3=5, on calcule -120.
```

2. A partir du programme précédent, écrire le programme `des_expressions` qui répète le calcul précédent autant de fois que l'utilisateur le désire. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez trois entiers positifs : 1 2 3
Pour n1=1, n2=2, n3=3, on calcule -22.
Voulez-vous recommencer ?
Si oui, entrez 0 (zéro).
0
Entrez trois entiers positifs : 2 2 2
Pour n1=2, n2=2, n3=2, on calcule -2.
Voulez-vous recommencer ?
Si oui, entrez 0 (zéro).
0
Entrez trois entiers positifs : 1 0 0
Pour n1=1, n2=0, n3=0, on calcule 1.
Voulez-vous recommencer ?
Si oui, entrez 0 (zéro).
1
```

Exercice 3.
Ecrire le programme `exp_bol` qui calcule et affiche la table de vérité de l'expression booléenne,

`(b1 or b2) and not b3,`

où `b1, b2, b3` sont des valeurs booléennes. L'exécution de ce programme donnera à l'écran ce qui suit.

```
b1 b2 b3 :: (b1 or b2) and not b3
0  0  0  :: 0
0  0  1  :: 0
0  1  0  :: 1
0  1  1  :: 0
1  0  0  :: 1
1  0  1  :: 0
1  1  0  :: 1
1  1  1  :: 0
```

9.2 Examen de décembre 2010

Durée : 2h00. Epreuve individuelle sur machine.
Tout document sous forme numérique enregistré sur un "support USB" autorisé.

Travail demandé. Dans chaque fichier source de vos réponses vous indiquerez, sous la forme d'un commentaire en tout début de programme, la(les) commandes complète(s) pour obtenir l'exécutable correspondant. La qualité de la programmation est prise en compte dans la notation.

Consigne pour la fin d'épreuve. *Vous rassemblez vos fichiers sources (`.c et .h`) dans une archive `votre_nom.zip` que vous déposez dans la zone "Travaux" de l'ENT. Vous vérifiez auprès de l'enseignant que votre dépôt est effectif avant de quitter la salle. L'absence de dépôt en fin d'épreuve est considérée comme "copie blanche".*

Exercice 1. (`paire`) Ecrire un programme `paire` qui détermine le signe et la parité d'un entier saisi au clavier. Ce calcul sera répété autant de fois que l'utilisateur le désire. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez une valeur entière : -1
Vous avez entré la valeur -1 qui est négative et impaire.
Voulez-vous recommencer ? Si oui, entrez 0 (zéro).
0
Entrez une valeur entière : 0
Vous avez entré la valeur 0 qui est positive et paire.
Voulez-vous recommencer ? Si oui, entrez 0 (zéro).
0
Entrez une valeur entière : 4
Vous avez entré la valeur 4 qui est positive et paire.
Voulez-vous recommencer ? Si oui, entrez 0 (zéro).
1
```

Exercice 2. (`code-ascii`) On a vu que les caractères sont représentés au moyen d'une valeur entière, leur code ASCII. Ecrire un programme `code-ascii` qui

1. affiche le code ASCII d'un caractère entré au clavier,
2. ainsi que les caractères qui le précéde et le suive,
3. puis qui affiche le caractère correspondant à un code entré au clavier
4. ainsi que les caractères qui le précéde et le suive.

Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez un caractère : g
Vous avez entré le caractère g  qui correspond au code 103;
Son prédécesseur est f et son successeur h.
Entrez une valeur entière : 55
Vous avez entré la valeur 55  qui correspond au caractère 7;
Son prédécesseur est 6 et son successeur 8.
```

Exercice 3. (`max-prod-mat`) Soient les vecteurs $x = [-1, -3, -5, 1, 3]$ et $y = [10, 20, 30]$ dont les valeurs sont des `float`. Le produit xy^T donne la matrice présentée un peu plus loin.
Ecrire un programme (`max-prod-mat`) qui

1. définit et initialise les vecteurs x et y,
2. calcule la matrice produit xy^T,
3. l'affiche,
4. identifie et affiche les indices de la plus grande composante en valeur absolue.

Une exécution de ce programme donnera à l'écran ce qui suit.

```
-10.00 -20.00 -30.00
-30.00 -60.00 -90.00
-50.00 -100.00 -150.00
10.00 20.00 30.00
30.00 60.00 90.00

La valeur max est A(3, 3) = 150.00
```

La fonction `fabs(float x)` de la bibliothèque `math.h` vous permet de calculer la valeur absolue d'un `float`.

Exercice 4. (`mois` et `f-mois, mois2`)

1. Ecrire le programme (`mois`) qui affiche le nombre de jours d'un mois entré au clavier comme un entier compris entre 1 et 12. Une entrée incohérente doit relancer la demande de saisie. On suppose que l'année n'est pas bissextile.

   ```
   Entrer le numéro d'un mois (compris entre 1 et 12) : 21
   Entrer le numéro d'un mois (compris entre 1 et 12) : 12
   Il y a 31 jours le mois 12.
   ```

2. Transformer ce programme en `mois2` de façon à ce que le calcul du nombre de jours soit effectué par un appel à une fonction `f-mois`, les entrées-sorties étant regroupées au sein du `main`.

9.3 Partiel de mars 2011

Durée : 1h10. Epreuve individuelle sur machine.
Tout document sous forme numérique enregistré sur un "support USB" autorisé.

Travail demandé. Vous traiterez les exercices 1, 2 et 3. La qualité de la programmation est prise en compte dans la notation.

Consigne pour la fin d'épreuve. *Vous rassemblez vos fichiers sources (`.c`) et exécutables dans une archive `votre_nom.zip` que vous déposez dans la zone "Travaux" de l'ENT. Vous vérifiez auprès de l'enseignant que votre dépôt est effectif avant de quitter la salle. L'absence de dépôt en fin d'épreuve est considérée comme "copie blanche".*

Exercice 1. Ecrire un programme `tables-mult` qui calcule et affiche les tables de multiplication de 1 à 10. Une exécution de ce programme donnera à l'écran ce qui suit.

```
 x ! 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
 3 ! 3  6  9 12 15 18 21 24 27 30
 4 ! 4  8 12 16 20 24 28 32 36 40
 5 ! 5 10 15 20 25 30 35 40 45 50
 6 ! 6 12 18 24 30 36 42 48 54 60
 7 ! 7 14 21 28 35 42 49 56 63 70
 8 ! 8 16 24 32 40 48 56 64 72 80
 9 ! 9 18 27 36 45 54 63 72 81 90
10 !10 20 30 40 50 60 70 80 90 100
---------------------------------
```

Exercice 2. Ecrire un programme `xoxo` qui produit à l'écran la figure suivante. Le nombre de `X` ou de `O` est entré au clavier par l'utilisateur. La valeur 0 sert à arrêter l'exécution.

```
Entrer une valeur entiere (0 si arret): 6
X
X X
X X X
X X X X
X X X X X
X X X X X X
O O O O O O
  O O O O O
    O O O O
      O O O
        O O
          O
Entrer une valeur entiere (0 si arret): 3
X
X X
X X X
O O O
  O O
    O
Entrer une valeur entiere (0 si arret): 0
```

Exercice 3. Ecrire le programme `compter` qui calcule le nombre d'occurences (le nombre d'apparitions) d'un caractère arbitrairement entré au clavier, dans le mot `anticonstitutionnellement`. L'exécution de ce programme donnera à l'écran ce qui suit. Pour éviter les répétitions indésirables des demandes, utiliser le format "%c%*c" pour la lecture du caractère au clavier.

```
anticonstitutionnellement
Entrer un caractère en minuscule (X si arret) : a
Le mot anticonstitutionnellement comporte 1 fois la lettre a.

Entrer un caractère en minuscule (X si arret) : t
Le mot anticonstitutionnellement comporte 5 fois la lettre t.

Entrer un caractère en minuscule (X si arret) : z
Le mot anticonstitutionnellement comporte 0 fois la lettre z.

Entrer un caractère en minuscule (X si arret) : X
```

9.4 Examen de mai 2011

Durée : 2h00. Epreuve individuelle sur machine.
Tout document sous forme numérique enregistré sur un "support USB" autorisé.

Travail demandé. Vous traiterez les exercices 1 et 2. La qualité de la programmation est prise en compte.

Exercice 1. Dans cet exercice, `%c` est le seul format autorisé pour l'affichage par `printf` (affichage d'un caractère). Le format `%s` ne sera pas utilisé.

1. Ecrire un programme `rev1` qui réalise les actions suivantes.
   (a) Déclare et définit une chaîne de caractères initialisée à `Universite de Perpignan`,
   (b) l'affiche de gauche à droite (sens de la lecture),
   (c) affiche les dix premiers caractères dans le sens de la lecture,
   (d) l'affiche entièrement de droite à gauche (sens inverse de la lecture),
   (e) affiche les dix premiers caractères dans le sens inverse de la lecture.

   Une exécution de ce programme donnera à l'écran ce qui suit.

   ```
   Universite de Perpignan
   Universite
   nangipreP ed etisrevinU
   etisrevinU
   ```

2. Ecrire un programme `rev2` qui réalise les actions suivantes.
   (a) Définit une fonction `aff` qui affiche une chaîne de caractères (dans le sens de la lecture) de longueur arbitraire. Cette longueur sera un paramètre de `aff`.
   (b) Obtenir les quatre affichages de la question précédente en appliquant `aff`.

3. On introduit les macro-constantes suivantes.

   ```
   #define LONGUEURMOT 23
   #define UP "Universite de Perpignan"
   ```

   (a) Ecrire un programme `rev3` qui réalise les quatre affichages de la question précédente en utilisant (au maximum) ces macro-constantes.
   (b) Modifier `rev3` en `rev4` qui réalise les actions précédentes sur `Universite de Perpignan Via Domitia`.

4. Les programme `rev5` réalisera les actions précédentes sur **des mots** entrés au clavier lors de l'appel à `rev5`. Cet ensemble de mots est supposé ne pas excéder 180 caractères. Il faut donc commencer par écrire quelques primitives de manipulation de chaînes de caractères (dont le traitement sera vérifié au fur et à mesure).
   (a) Ecrire une fonction `longueur` qui retourne le nombre de lettres d'une chaîne de caractères passée en argument.
   (b) Ecrire une fonction `concatene` qui retourne la concaténation de deux chaînes de caractères.
   (c) Modifier `rev4` en `rev5` de façon à obtenir le traitement suivant.

   ```
   $ ./rev5 C est enfin fini
   C est enfin fini
   C est enfi
   inif nifne tse C
   inif nifne
   ```

Exercice 2. Il s'agit de calculer, stocker dans un vecteur, puis afficher les carrés des premiers entiers. Les différentes versions de ce traitement varient selon le stockage du vecteur des valeurs.

1. On se limite aux 10 premiers carrés. Ecrire un programme `aff-tab-carres-1` qui effectue ce traitement (calcul, stockage) et qui affiche le résultat à l'écran. Une exécution de ce programme donnera à l'écran ce qui suit.

```
  1
  4
  9
 16
 25
 36
 49
 64
 81
100
```

2. On se limite aux 10 premiers carrés. Ecrire une fonction `tab-carres-glob` qui réalise le calcul et le stockage dans un vecteur déclaré comme une variable globale du `main`. Modifier le programme précédent en `aff-tab-carres-2` qui appelle `tab-carres-glob` et réalise l'affichage du vecteur résultat.
3. On se limite aux 10 premiers carrés. Ecrire une fonction `tab-carres` qui réalise le calcul et le stockage dans un vecteur passé comme paramètre. Modifier le programme précédent en `aff-tab-carres-3` qui appelle `tab-carres` et réalise l'affichage du vecteur résultat.
4. Modifier le programme précédent en `aff-tab-carres-4` qui effectue le même traitement (calcul, stockage, affichage) pour des entiers variant entre 1 et une `TAILLE` définie comme macro-constante.
5. Modifier le programme précédent en `aff-tab-carres-5` qui effectue le même traitement (calcul, stockage, affichage) pour des entiers variant entre 1 et une taille dynamiquement définie par l'utilisateur. On veillera à coder un traitement dynamique le plus fiable possible. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Calcul des carrés des N premiers nombres entiers.
Entrer N :
7
  1
  4
  9
 16
 25
 36
 49
```

6. Modifier le programme précédent en `aff-tab-carres` qui effectue le même traitement dynamique pour une valeur entière définie par l'utilisateur comme un paramètre de l'appel à `aff-tab-carres`. Trois appels de ce programme donnent à l'écran ce qui suit.

```
$ ./aff-tab-carres
La commande ./aff-tab-carres demande un seul argument entier > 0.

$ ./aff-tab-carres 5
  1
  4
  9
 16
 25

$ ./aff-tab-carres 7
  1
  4
  9
 16
 25
 36
 49
```

9.5 Examen de juin 2011 (session 2)

Durée : 2h00. Epreuve individuelle sur machine.
Tout document sous forme numérique enregistré sur un "support USB" autorisé.
La qualité de la programmation est prise en compte dans la notation.

Exercice 1.

1. Ecrire un programme `exp1` qui calcule $(x+y)(x-y)$, pour x, y, deux valeurs `float` saisies au clavier. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez deux flottants : 1 2
Pour x=1.000000, y=2.000000, on calcule -3.000000.
```

2. A partir du programme précédent, écrire le programme `exp2` qui répète le calcul précédent autant de fois que l'utilisateur le désire. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez deux flottants : 1 1
Pour x=1.000000, y=1.000000, on calcule 0.000000.
Voulez-vous recommencer ? Si oui, entrez 1.
1
Entrez deux flottants : 5 1
Pour x=5.000000, y=1.000000, on calcule 24.000000.
Voulez-vous recommencer ? Si oui, entrez 1.
0
```

3. Ecrire le programme `exp3` qui répète le calcul de l'expression $\sqrt{x}-1$, autant de fois que l'utilisateur le désire pour des valeurs `float` x entrées au clavier. Vous indiquerez sur votre copie les commandes de compilation utilisées. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrez un flottant : 4
Pour x=4.000000, on calcule 1.000000.
Voulez-vous recommencer ? Si oui, entrez 1.
1
Entrez un flottant : 9
Pour x=9.000000, on calcule 2.000000.
Voulez-vous recommencer ? Si oui, entrez 1.
0
```

Exercice 2.

1. Ecrire un programme `un` qui teste si le bit de poids faible d'un entier est égal à 1. L'appliquer aux entiers compris entre 0 et 10. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Le bit de poids faible de 1 vaut 1.
Le bit de poids faible de 3 vaut 1.
Le bit de poids faible de 5 vaut 1.
Le bit de poids faible de 7 vaut 1.
Le bit de poids faible de 9 vaut 1.
```

2. A partir du programme précédent et en utilisant les opérateurs de décallage, écrire `combiendeun` qui compte le nombre de bits égaux à 1 d'un entier. L'appliquer aux entiers compris entre 0 et 20. Une exécution de ce programme donnera à l'écran ce qui suit.

```
0 a 0 bits égaux à 1.    1 a 1 bits égaux à 1.
2 a 1 bits égaux à 1.    3 a 2 bits égaux à 1.
4 a 1 bits égaux à 1.    5 a 2 bits égaux à 1.
6 a 2 bits égaux à 1.    7 a 3 bits égaux à 1.
8 a 1 bits égaux à 1.    9 a 2 bits égaux à 1.
10 a 2 bits égaux à 1.    11 a 3 bits égaux à 1.
12 a 2 bits égaux à 1.    13 a 3 bits égaux à 1.
14 a 3 bits égaux à 1.    15 a 4 bits égaux à 1.
16 a 1 bits égaux à 1.    17 a 2 bits égaux à 1.
18 a 2 bits égaux à 1.    19 a 3 bits égaux à 1.
```

Exercice 3.

1. Ecrire un programme `suite` qui calcule et affiche les termes *d'indices pairs* de la suite entière $u_0 = 1, u_1 = 2$, et $u_n = 2u_{n-1} - u_{n-2}$. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Entrer le nbre de termes de la suite à calculer : 10

 i, u[i]
  0 1
  1 2
  2 3
  4 5
  6 7
  8 9
 10 11
```

2. A partir du programme précédent, écrire `suite2` qui stocke dans un tableau les valeurs calculées de la suite u_n de façon à pouvoir afficher l'indice, si il existe, d'une valeur arbitrairement entrée au clavier. Une exécution de ce programme donnera à l'écran ce qui suit.

```
Calcul des 100 valeurs de la suite.
On vérifie avec les 10 premiers :
 i, u[i]
  0 1
  1 2
  2 3
  3 4
  4 5
  5 6
  6 7
  7 8
  8 9
  9 10
Entrez une valeur entiere : 5
La valeur u[4] = 5
Voulez-vous recommencer ? Si oui, entrez 1.
1
Entrez une valeur entiere : -5
La valeur -5 n'apparait pas dans la suite
Voulez-vous recommencer ? Si oui, entrez 1.
1
Entrez une valeur entiere : 77
La valeur u[76] = 77
Voulez-vous recommencer ? Si oui, entrez 1.
0
```

9.6 Examen de juillet 2011 (session 2)

Durée : 2h00. Epreuve individuelle sur machine.
Tout document sous forme numérique enregistré sur un "support USB" autorisé.
La qualité de la programmation est prise en compte dans la notation.

Exercice 1. Cet exercice s'interesse au traitement d'une ou plusieurs séries de notes stockées dans un tableau.

1. On considère une série de `NbNotes` notes sur 20. Ici `NbNotes` = 10. Ecrire un programme `Moy0` qui effectue les traitements suivants.
   (a) Déclare et initialise le tableau `Notes` avec les valeurs : 13, 8, 10, 4, 2, 1, 7, 10, 8, 14. Faire un affichage de contrôle.
   (b) Calcule et affiche m la moyenne empirique (moyenne de l'échantillon) et l'écart type empirique s de cette série de N notes $(n_i)_i$ — $s^2 = \sum_{i=i}^{N} (n_i - m)^2/(N-1)$.
2. Ecrire un programme `Moy1` qui effectue les traitements précédents dans des fonctions indépendantes et qui sont appelées depuis un `main`. Ces fonctions effectueront donc les traitements suivants.
   (a) Afficher un tableau de longueur passée en paramètre ;
   (b) Calculer la moyenne (des valeurs) d'un tableau passé en paramètre ;
   (c) Calculer l'écart type empirique (des valeurs) d'un tableau passé en paramètre.
3. On considère maintenant `NbSeries` séries de `NbNotes` notes sur 20. Ici, `NbSeries` = 5 et `NbNotes` = 10. Ces 50 notes sont données dans le fichier `notes.txt` (clé usb en circulation). Ecrire un programme `Moy2` qui effectue les traitements suivants.
   (a) Déclare et initialise le tableau `Notes` avec les valeurs du fichier `notes.txt`. Faire un affichage de contrôle.
   (b) Calcule les `NbSeries` moyennes empiriques m_k (les moyennes empiriques de chaque série de notes). Ces moyennes m_k seront stockées dans un tableau ad-hoc que l'on affichera.
   (c) Calcule et affiche la moyenne empirique m de l'ensemble des `NbSeries`×`NbNotes` notes.
4. A la manière de la question 2, écrire un programme `Moy3` qui effectue les traitements précédents dans des fonctions indépendantes et qui sont appelées depuis un `main`.

Exercice 2. Ecrire un programme `ConcatTab` en utilisant tous les pointeurs nécessaires aux traitement suivants.

1. Lire au clavier les dimensions `d1` et `d2` de deux tableaux d'entiers `T1` et `T2`,
2. Initialise `T1` avec la suite des `d1` premiers entiers impairs et `T2` avec la suite des `d2` premiers entiers pairs,
3. Concatène `T1` et `T2` dans un tableau `T`,
4. Affiche `T`.

Troisième partie

Exemples de corrections des exercices

Chapitre 10

Description de l'environnement de programmation

10.1 bonjourlemonde

```
/*Vous l'avez reconnu : Bonjour le monde !*/
#include<stdio.h>

int main()
{
  printf("Bonjour le (centre du) monde !\n");
  return 0;
}
```

10.2 swap1

```
/* Permute deux valeurs arbitraires de type int, float et char */
/* version 1 : fonction puis main dauns un seul fichier */
#include <stdio.h>

void swap_int(int * a, int * b)
/* Les parametres sont les adresses des valeurs a interchanger */
{
  int tmp;
  /*Bien comprendre qu'on echange les valeurs des objets pointes par a et b */
  tmp = *b; /* *b est la valeur pointee par b */
  *b = *a;
  *a = tmp;
}

int main()
{
  int aa=0, bb =0;

  printf("Entrer un premier entier : \n");
  scanf("%d", &aa);
  printf("Entrer un second entier : \n ");
  scanf("%d", &bb);
  swap_int(&aa, &bb); /* L'appel passe les adresses des valeurs a interchanger */
  printf("Les valeurs entrées sont, en ordre inverse, %d et %d \n", aa, bb);
  return 0;
}
```

10.3 swap2

```
/* Permute deux valeurs arbitraires de type int, float et char */
/* version 2 : main puis fonction  dans un seul fichier */
```

```c
#include <stdio.h>

void swap_int(int * a, int * b);

int main()
{
  int aa=0, bb =0;

  printf("Entrer un premier entier : \n");
  scanf("%d", &aa);
  printf("Entrer un second entier : \n ");
  scanf("%d", &bb);
  swap_int(&aa, &bb); /* L'appel passe les adresses des valeurs a interchanger */
  printf("Les valeurs entrées sont, en ordre inverse, %d et %d \n", aa, bb);
  return 0;
}

void swap_int(int * a, int * b)
/* Les parametres sont les adresses des valeurs a interchanger */
{
  int tmp;
  /*Bien comprendre qu'on echange les valeurs des objets pointes par a et b */
  tmp = *b; /* *b est la valeur pointee par b */
  *b = *a;
  *a = tmp;
}
```

10.4 swaps

```c
#ifndef _SWAPS_H
#define _SWAPS_H

void swap_int(int * a, int * b);

#endif
```

```c
#include "swaps.h"

void swap_int(int * a, int * b)
/* Les parametres sont les adresses des valeurs a interchanger */
{
  int tmp;
  /*Bien comprendre qu'on echange les valeurs des objets pointes par a et b */
  tmp = *b; /* *b est la valeur pointee par b */
  *b = *a;
  *a = tmp;
}
```

10.5 swap3

```c
/* Permute deux valeurs arbitraires de type int, float et char */
/* version 3 : main et fonctions dans des fichiers differents */
#include <stdio.h>
#include "swaps.h"

int main()
{
  int aa=0, bb =0;

  printf("Entrer un premier entier : \n");
  scanf("%d", &aa);
  printf("Entrer un second entier : \n ");
  scanf("%d", &bb);

  swap_int(&aa, &bb); /* L'appel passe les adresses des valeurs a interchanger */

  printf("Les valeurs entrées sont, en ordre inverse, %d et %d \n", aa, bb);
```

```
  return 0;
}
```

Chapitre 11

Variables, valeurs, constantes, déclarations, types scalaires, opérateurs, entrées-sorties simples

11.1 operateurs

```
#include <stdio.h>
#include <stdlib.h>


void printbin(int val){

  if (val > 0){
    printbin(val/2);
    printf("%d", (val % 2));
  }

int main (void){
  int a, b, res;

  printf("Entrer un entier a : \n");
  scanf("%d", &a);
  printf("Entrer un entier b : \n");
  scanf("%d", &b);

  printf("Vous avez entré les valeurs a et b suivantes. \n ");
  printf("%#x",a);
  printbin(a);
  print("\n");
  printf("%#x \n",b);
  printbin(b);
  print("\n");

  res = a >> 2;
  printf("a >> 2 = %d \n", res);
  res = b >> 4;
  printf("b >> 4 = %d \n", res);

  res = a << 1;
  printf("a << 1 = %d \n", res);
  res = b << 4;
  printf("b << 4 = %d \n", res);


  return EXIT_SUCCESS;
}
```

11.2 int2bin

```c
#include <stdio.h>
#include <stdlib.h>

void usage(char * argv[]){
  printf("usage: %s val. \n", argv[0]);
}

void convert(int val){

  if (val > 0){
    convert(val/2);
    printf("%d", (val % 2));
  }
  /* if (val == 0) printf("0");
     if (val == 1) printf("1");*/
}

void convert_str(int val, char[] *s){


  if (val > 0){
    convert(val/2);
    printf("%d", (val % 2));
  }
  /* if (val == 0) printf("0");
     if (val == 1) printf("1");*/
}


int main(int argc, char * argv[]){
  int val;

  // printf("%d", argc);
  if ((argc <= 1) || (argc > 2)){
    usage(argv);
  }
  else {
    val = atoi(argv[1]);
    if (val < 0){
      printf("Valeur négative interdite. \n");
        }
    else {
      printf("L'ecriture binaire de %s vaut ", argv[1]);
      convert(val);
      printf("\n");
    }
  }
  return 0;
}
```

11.3 tabverite

```c
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
  bool v;
  bool b[2] = {true, false} ;

  /* Soyons clairs : avec le type bool */
  printf("La valeur booleene vrai vaut %i \n", true);
  printf("La valeur booleene faux vaut %i \n", false);
  printf("\n");
  /* et logique */
  for (int i=0; i<2; i++)
    {
      for (int j=0; j<2; j++)
        {
          v = b[i] && b[j];
```

```
          printf("%i && %i = %i \n", b[i], b[j], v);
        }
    }
  printf("\n");

  /* ou logique */
  for (int i=0; i<2; i++)
    {
      for (int j=0; j<2; j++)
        {
          v = b[i] || b[j];
          printf("%i || %i = %i \n", b[i], b[j], v);
        }
    }
  printf("\n");

  /* négation */
  for (int i=0; i<2; i++)
    {
      v = ! b[i];
      printf("! %i = %i \n", b[i], v);
    }
  printf("\n");
/* ---------------------------------------------------------------- */
  /* Soyons clairs : valeurs arbitraires*/
  int bb[2] = {0, 2};
  printf("La valeur booleene vrai vaut maintenant %i \n", bb[1]);
  printf("La valeur booleene faux vaut %i (on est obligé) \n", bb[0]);
  printf("ce qui pose des problèmes de cohérence : \n");
  /* et logique */
  for (int i=0; i<2; i++)
    {
      for (int j=0; j<2; j++)
        {
          v = bb[i] && bb[j];
          printf("%i && %i = %i \n", bb[i], bb[j], v);
        }
    }
  printf("\n");
/* ---------------------------------------------------------------- */

  return 0;
}
```

11.4 formats

```
#include <stdio.h>
#include <stdlib.h>

/* Exemples des différences de formats de printf entier */
/* Regarder sorties dans formats-sorties.pdf */
/* TODO: trouver solution portablité */

int main (void)
{
  int i = 74, j=137;

  printf("i: %i %i \n", i, j);
  printf("d: %d %d \n", i, j);
  printf("o: %o %o \n", i, j);
  printf("x: %x %x \n", i, j);
  printf("X: %X %X \n", i, j);
  printf("c: %c %c \n", i, j);
  //  printf("s: %s %s \n", i, j);
  printf("f: %f %f \n", i, j);

  return EXIT_SUCCESS;
}
```

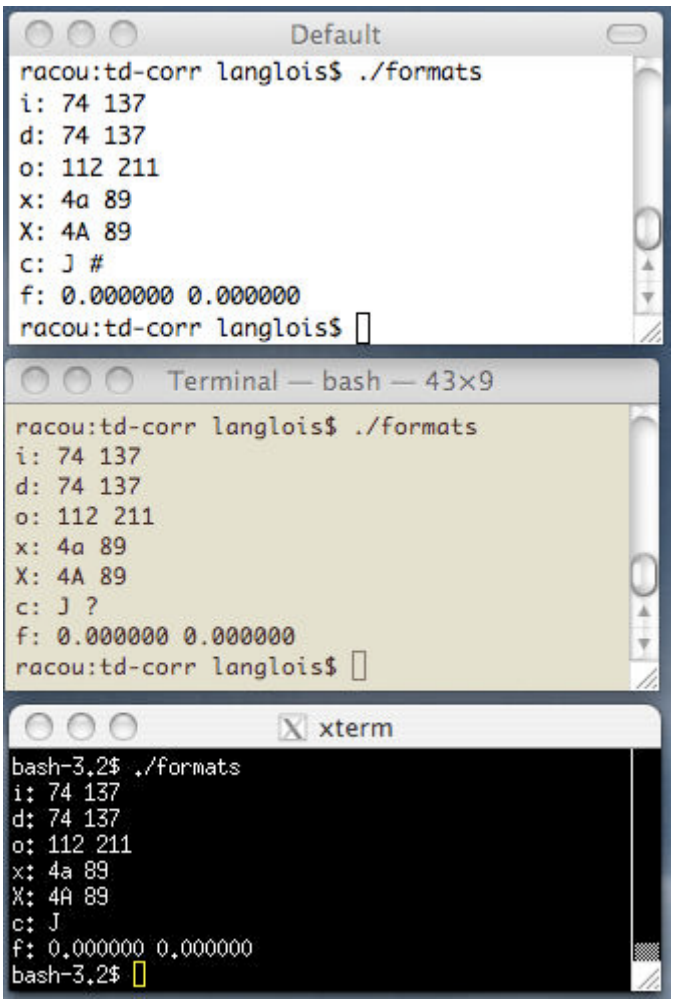

FIGURE 11.1 – Les affichages d'un même exécutable peuvent aussi dépendre de l'environnement d'exécution. Ici les sorties de trois exécutions sur la même machine+système (mac-intel+macosx) et des environnements graphiques différents (aqua+iTerm, aqua+Terminal, x11+xterm).

Chapitre 12

Opérateurs entiers, bit à bit, logiques, tests

12.1 vallimit

```c
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int main (void){
  printf("SHRT_MAX = %hd = %hx \n", SHRT_MAX,SHRT_MAX);
  printf("USHRT_MAX = %hu = %x \n", USHRT_MAX,USHRT_MAX);

  printf("INT_MAX = %d = %x \n", INT_MAX, INT_MAX);
  printf("UINT_MAX = %u = %x \n", UINT_MAX, UINT_MAX);

  printf("LONG_MAX = %ld = %lx \n", LONG_MAX, LONG_MAX);
  printf("ULONG_MAX = %lu = %lx \n", ULONG_MAX, ULONG_MAX);

  printf("LLONG_MAX = %lld = %llx \n", LLONG_MAX, LLONG_MAX);
  printf("ULLONG_MAX = %llu = %llx \n", ULLONG_MAX, ULLONG_MAX);

  printf("Caractéristiques gcc sur mac osx et racou \n");
  printf("- sur 16 bits : short \n");
  printf("- sur 32 bits : int long \n");
  printf("- sur 64 bits : long \n");

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

12.2 op-int

```c
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>

int main (void){
  int a=5, b=3;

  printf("a  b  a/b a%%b \n");
  printf("%2d %2d %2d %2d \n", a, b, a/b, a%b);
  printf("%2d %2d %2d %2d \n", -a, b, -a/b, -a%b);
  printf("%2d %2d %2d %2d \n", a, -b, a/(-b), a%(-b));
  printf("%2d %2d %2d %2d \n", -a, -b, (-a)/(-b), (-a)%(-b));
  printf("La troncature du quotient se fait vers 0.\n");
  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

12.3 int-vs-unsigned

```c
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include "printbin.h"

int main (void){
  int a;
  unsigned int uns_a;

  printf("size of int = %lu \n", sizeof(int));
  printf("size of unsigned int = %lu \n", sizeof(unsigned int));

  for (int i=1; i<=2; i++){
    printf("Entrer un entier a ");
    if (i == 1)
      printf("positif : \n");
    else
      printf("négatif : \n");
    scanf("%d", &a);
    uns_a = (unsigned)a;

    printf("Vous avez entré la valeur ");
    printf("a = %d = %#x = ", a, a);
    printbin(a);
    printf("\n et sa valeur unsigned est ");
    printf("uns_a = %u = %#x = ", uns_a, uns_a);
    printbin(uns_a);
    printf("\n");
  }
  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

12.4 div

```c
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{

  int i=1, j=4 ;
  //  float x, y;

  printf("i=%d, j=%d \n (float)(i/j)=%f , (float)i/j=%f, (float)i/(float)j=%f \n", i, j, (float)(i/j),
(float)i/j, (float)i/(float)j);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

12.5 op-bit

```c
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include "printbin.h"

/* ---------------------------------------------------------------- */
int main (void){
  printf("bit à bit vs. logique :\n");
  printf("i ~i !i \n");
  for (short i = 0; i < 3; i++){
    printf("%d %d %d \n", i, ~i, !i);
  }
```

```
  printf("Résultats différents car opérateurs bit à bit /= logique \n\n");

  printf("Opérateurs bit a bit : & (and), | (or), ^ (xor) \n");
  printf("i j i&j i&&j \n");
  for (int i = 0; i < 2; i++){
    for (int j = 0; j < 2; j++){
      printf("%d %d  %d %d \n", i, j, i&j, i&&j);
    }
  }
  printf("\n");

  printf("i j i|j i||j \n");
  for (int i = 0; i < 2; i++){
    for (int j = 0; j < 2; j++){
      printf("%d %d  %d %d \n", i, j, i|j, i||j);
    }
  }
  printf("\n");

  printf("i j i^j (pas de correspondant logique) \n");
  for (int i = 0; i < 2; i++){
    for (int j = 0; j < 2; j++){
      printf("%d %d  %d \n", i, j, i^j);
    }
  }
  printf("\n");
 printf("-------------\n");
/* ------------------------------------------------------------ */
  int a, b;

  printf("\n Entrer deux entiers a et b: \n");
  scanf("%d", &a);
  scanf("%d", &b);

  printf("Vous avez entré la valeur a suivante. \n ");
  printf("a = %d = %x \n", a, a);
  printf("Vous avez entré la valeur b suivante. \n ");
  printf("a = %d = %x \n", b, b);

  printf("a&b  a|b a^b \n");
  printf("%x  %x  %x \n",  a&b, a|b, a^b);


  return EXIT_SUCCESS;
}
/* ------------------------------------------------------------ */
```

12.6 op-decall

```
/* ------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include "printbin.h"

int main (void){
  int a, res;
  unsigned int uns_a, uns_res;

  printf("Entrer un entier a : \n");
  scanf("%d", &a);
  uns_a = (unsigned)a;

  printf("Vous avez entré la valeur ");
  printf("a = %d = %#x = ", a, a);
  printbin(a);
  printf("\n et sa valeur unsigned est ");
  printf("uns_a = %u = %#x = ", uns_a, uns_a);
  printbin(uns_a);
  printf("\n");

  for (int i = -2; i < 4; i++){
```

```
    res = a << i;
    printf("a << %d = %d = %#x = ", i, res, res);
    printbin(res);
    printf("\n");
  }
  printf("x<<i = x * 2**i pour i>=0 \n");
  printf("On complète à droite par des zéros.\n Les résultats sont indéterminés pour les positions négatives.

  for (int i = -2; i < 4; i++){
    res = a >> i;
    printf("    a >> %d = %d = %#x = ", i, res, res);
    printbin(res);
    printf("\n");
    uns_res = uns_a >> i;
    printf("uns_a >> %d = %d = %#x = ", i, uns_res, uns_res);
    printbin(uns_res);
    printf("\n");

  }


  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

12.7 masques

```
/* ---------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include "printbin.h"

#define  m0  0x01
#define  m1  m0 << 1
#define  m2  m0 << 2
#define  m3  m0 << 3
#define  m4  m0 << 4
/* ---------------------------------------------------------------- */
int main (void){

  printf("m0 =  %d = %x = ", m0, m0); printbin(m0); printf("\n");
  printf("m1 =  %d = %x = ", m1, m1); printbin(m1); printf("\n");
  printf("m2 =  %d = %x = ", m2, m2); printbin(m2); printf("\n");
  printf("m3 =  %d = %x = ", m3, m3); printbin(m3); printf("\n");
  printf("m4 =  %d = %x = ", m4, m4); printbin(m4); printf("\n");

/* ---------------------------------------------------------------- */
  int a, res;

  printf("\n Entrer un entier a : \n");
  scanf("%d", &a);
  printf("Vous avez entré la valeur a suivante. \n ");
  printf("a = %d = %x = ", a, a); printbin(a); printf("\n");

  // parité : solution 1 (reste division euclidienne)
  a % 2 == 0 ? printf("%d est pair ", a) : printf("%d est impair", a);
  printf("\n");

  // parité : solution 2 (masque avec 1)
  (a & m0) == m0 ? printf("%d est impair", a) :  printf("%d est pair", a);
  printf("\n");

  // test bit de position 3 == 0
  printf("Le bit de position 3 de %d vaut ", a);
  (a & m3) == m3 ?  printf("1 \n") :  printf("0 \n");

  // octet de poids faible == 7
  int sept = m2|m1|m0;
  printf("L'octet de poids faible de %d vaut 7 ?", a);
  (a & sept) == sept ?  printf(" Oui \n") :  printf(" Non\n");
```

```
  // bit de pos 1 et 4 mis a 1
  res = a | m1 | m4;
  printf("Les bits de position 1 et 4 mis à 1 (res = a ou a+2 ou a+16 ou a+18) \n");
  printf("res=%d=%x=", res, res); printbin(res); printf("\n");

  // bit de pos 1 et 2 mis a 0
  res = a & ~(m1 | m2);
  printf("Les bits de position 1 et 2 mis à 0 \n");
  printf("res=%d=%x=", res, res); printbin(res); printf("\n");


  // afficahge de l'octet de poids faible
  res = a & 0xff;
  printf("L'octet de poids faible de a vaut : \n");
  printf("res=%d=%x=", res, res); printbin(res); printf("\n");
  printf("\n");


  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

Chapitre 13

Fonctions mathématiques, tableaux, boucles, constantes symboliques

13.1 somme

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  int n, s=0, verif;
  printf("Somme des n premiers entiers : entrer n : \n");
  scanf("%d", &n);

  for (int i=0 ; i<n+1; i++) s += i;
  verif = n*(n+1)/2;

  printf("somme des %d premiers entiers = %d (= %d)\n", n, s, verif);

  return EXIT_SUCCESS;
}
```

13.2 somme-p

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void){
  int n, p;
  int s=0;
  printf("Somme des n premieres entiers à la puissance p : entrer n et p. \n");
  scanf("%d", &n);
  scanf("%d", &p);

  for (int i=0 ; i<n+1; i++) s += pow(i, p);

  printf("n = %d, p = %d, somme = %d \n", n, p, s);

/* ------------------------------------------------------------ */
  float fl_s=0.0;
  printf("Somme des n premieres entiers à la puissance p : entrer n et p. \n");
  scanf("%d", &n);
  scanf("%d", &p);

  for (int i=0 ; i<n+1; i++) fl_s += pow(i, p);

  printf("n = %d, p = %d, somme = %f \n", n, p, fl_s);

  return EXIT_SUCCESS;
}
```

13.3 felem

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define K 12

int main(void){
  float logN[K], N[K], NlogN[K], N2[K];
  float X;

/* Calcul */
  for (int i=0 ; i<K; i++){
    X = pow(10.0, i+1);
    logN[i] = log10(X);
    N[i] = X;
    NlogN[i] = X*log10(X);
    N2[i] = X*X;
  };

/* Affichage */
 printf(" k |  Log_10(X)  |  X = 10**k  |  X*Log_10(X)|     X*X    |\n");
 for (int i=0 ; i<K; i++){
   printf("%2d | %6.5e | %6.5e | %6.5e | %6.5e |\n", i+1, logN[i], N[i], NlogN[i], N2[i]);
 };

  return EXIT_SUCCESS;
}
```

13.4 fibo

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

int main(void){
  int n;
  /*  float u0 = 0.0, u1 = 1.0;
      float u, v, w;*/
  long long u0 = 0, u1 = 1;
  long long u, v, w;

/* Entrées */
  printf("Entrer le nbre de termes de la suite de Fibonacci à calculer : ");
  scanf("%d", &n);

  printf("\n i, u[i] \n");
  // affichage systématique u0
  //    printf("%3d %f \n", i, u);
  printf("%3d %lld \n", 0, u0);

  // affichage u1
  if (n>0)
    {
      //    printf("%3d %f \n", i, u);
      printf("%3d %lld \n", 1, u1);
    }

  //
  /* Calcul : u = v + w */
  v = u0;
  w = u1;
  for (int i=2 ; i<n+1; i++){
    u = v + w;
    v = w;
    w = u;
    // autres affichage
    //    printf("%3d %f \n", i, u);
```

```
    printf("%3d %lld \n", i, u);
  };


  return EXIT_SUCCESS;
}
```

13.5 fibo-inv

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

int main(void){
  int n;
  //  float u0 = 0.0, u1 = 1.0;
  float u[MAX] = {0.0, 1.0};
  /*  long long u0 = 0, u1 = 1;
      long long u, v, w; */

/* Entrées */
  printf("Entrer le nbre de termes de la suite de Fibonacci à calculer : ");
  scanf("%d", &n);

/* Calcul : u = v + w */
  for (int i=2 ; i<n+1; i++){
    u[i] = u[i-1] + u[i-2];
  };

  printf("\n i, u[i]\n");
  for (int i=n; i>=0; i--){
    printf("%3d %f \n", i, u[i]);
  };

  return EXIT_SUCCESS;
}
```

13.6 fois2fois2etc

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX 32
int main(void){
  long long Puiss[MAX][2];

    /* on remplit */
    // question 1
  Puiss[0][0] = 1;
  for (int i=0 ; i<MAX; i++){
      Puiss[i][0] = 1;
      for (int j=i; j>0; j--){
        Puiss[i][0] *= 2;
      }
  };

  // question 2
  Puiss[0][1] = 1;
  for (int i=1 ; i<MAX; i++){
    Puiss[i][1] = 2*Puiss[i-1][1];
  };

  /* On affiche */
  printf("Les 31 premieres puissances de 2. \n");
  for (int i=0 ; i<MAX; i++){
    printf("%2d | %16lld | %16lld |\n", i, Puiss[i][0], Puiss[i][1]);
```

```
  }

  // question 3
  long long val;
  int i=0;
  printf("Entrer une valeur entiere : \n");
  scanf("%lld", &val);

  // on localise
  while (Puiss[i][0] <= val) i++;

  // on affiche
  printf("On a : 2**%d <= %lld < 2**%d; \n", i-1, val, i);

  // question 4
  int reste = val % Puiss[i-1][0];
  // on affiche
  printf("et   : %lld = 2**%d + %d = %lld + %d.  \n", val, i-1, reste, Puiss[i-1][0], reste);


  return EXIT_SUCCESS;
}
```

13.7 max-tab

```
#include <stdio.h>
#define TAILLE 5
/* Identifie le max d'un tableau passé en paramètre.
La taille du tableau est définie comme macro */

int max(int tab[], int l){
  int i;
  int max = tab[0];

  for (i=1; i<l; i++){
    max = tab[i] > max ? tab[i] : max;
  }
  return max;
}

int main(){
  int T[TAILLE] = {0, 5, 11, 2, 7};

  printf("Valeur max de T = %i \n", max(T, TAILLE));

}
```

13.8 ind-max-tab

```
#include <stdio.h>

#define TAILLE 5
/* Identifie le premier indice du max d'un tableau passé en paramètre.
La taille du tableau est définie comme macro */

int max(int tab[], int l){
  int i;
  int max = tab[0];

  for (i=1; i<l; i++){
    max = tab[i] > max ? tab[i] : max;
  }
  return max;
}

int ind_max(int tab[], int l){
  int i = 0;
  int M = max(tab, l);
```

```
  while (tab[i] != M){
    i++;
  }
  return i;
}

int main(){
  int T[TAILLE] = {0, 5, 11, 2, 7};

  printf("Valeur max de T = %i \n", max(T, TAILLE));
  printf("Premier indice du max de T = %i \n", ind_max(T, TAILLE));

}
```

13.9 last-ind-max-tab

```
#include <stdio.h>

#define TAILLE 5
/* Identifie le premier indice du max d'un tableau passé en paramètre.
La taille du tableau est définie comme macro */

int max(int tab[], int l){
  int i;
  int max = tab[0];

  for (i=1; i<l; i++){
    max = tab[i] > max ? tab[i] : max;
  }
  return max;
}

int ind_max(int tab[], int l){
  int i = 0;
  int M = max(tab, l);

  while (tab[i] != M){
    i++;
  }
  return i;
}

int last_ind_max(int tab[], int l){
  int i = 0;
  int last_i;
  int M = max(tab, l);

  for (i =0; i < l; i++){
    last_i = tab[i] = M ? i : last_i;
    i++;
  }
  return i;
}

int main(){
  int T[TAILLE] = {0, 5, 11, 2, 11};

  printf("Valeur max de T = %i \n", max(T, TAILLE));
  printf("Premier indice du max de T = %i \n", ind_max(T, TAILLE));
  printf("Derneir indice du max de T = %i \n", ind_max(T, TAILLE));

}
```

Chapitre 14

Structures de contrôle : répétition, choix

14.1 double-indice

```c
#include <stdio.h>
#include <stdlib.h>
#define MIN 2
#define MAX 18

int main(void)
{
  int i, j;

  for (i=MIN, j=MAX; i<j; i++, j /= 2) {
    printf("i = %d, j= %d \n", i, j);
  }
  return EXIT_SUCCESS;
}
```

```
i = 2, j= 20
i = 3, j= 10
i = 4, j= 5
```

14.2 des-boucles-for

```c
/* Bientot disponible */
```

14.3 des-boucles-while

```c
/* Bientot disponible */
```

14.4 un-deux-etc

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int i, j, nb_lignes;

  printf("Entrer un nombre de lignes : ");
  scanf("%d", &nb_lignes);
  for (i=1; i<nb_lignes+1; i++){
    for (j=0; j<i; j++){
      printf("%d", i);
```

```
      //printf("*");
    }
    printf("\n");
  }

  return EXIT_SUCCESS;
}
```

14.5 suites

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main (void)
{
  int n = 0;

  // version minimaliste
  do {
    n++;
  }
  while (n-5 < -2*n+10);
  printf("n = %d \n", n);

  // plus lisible
  int u, v;
  n = 0;
  do {
    n++;
    u = n-5;
    v = -2*n+10;
  }
  while (u < v);
  printf("n = %d \n", n);
/* ---------------------------------------------------------- */
  printf("2.............\n");
  n = 0;
  do {
    n++;
    v = n+100;
    u = 10*n;
    printf("%d %d \n", u, v);
  }
  while (u < v);
  printf("n = %d \n", n);

/* ---------------------------------------------------------- */
  printf("3.............\n");
  n = 0;
  do {
    n++;
    v = 100*n;
    u = pow(2,n);
    printf("%d %d \n", u, v);
  }
  while (u < v);
  printf("n = %d \n", n);

  return EXIT_SUCCESS;
}
```

14.6 equa-prem-deg

```
#include <stdio.h>
#include <stdlib.h>

void resol_equa_prem_deg(float a, float b){
```

```
  if (a == 0.){
    if (b ==0.) printf("tout l'ensemble R \n");
    else printf("vide \n");
  }
  else printf("l'unique x = %f \n", -b/a);
}

int main (){
  float a, b;
  // Saisie des coefficients de a x + b = 0
  printf("Entrer les coefficients de l'équation a*x + b = 0 \n");
  printf("a ="); scanf("%f", &a);
  printf("b ="); scanf("%f", &b);

  // Resolution
  printf("L'ensemble des solutions de l'equation %g*x + %g = 0 est ", a, b);
  resol_equa_prem_deg(a, b);

  return EXIT_SUCCESS;
}
```

14.7 aleas

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000

int main(void)
{

/* ---------------------------------------------------------------- */
  printf("1).............\n");
  printf("RAND_MAX = %d \n ++++++++++\n" , RAND_MAX);
  //  for (int i=0; i<N; i++) printf("%d \n", rand());

/* ---------------------------------------------------------------- */
  printf("2).............\n");
  int mid_value = RAND_MAX/2;
  int nb_inf = 0, nb_sup = 0;
  float p_inf, p_sup;
  for (int i=0; i<N; i++) rand() < mid_value ? nb_inf++ : nb_sup++;
  p_inf = (float) nb_inf*100.0 / (float) N;
  p_sup = (float) nb_sup*100.0 / (float) N;
  printf("premiere moitie : %4.2f %%, seconde moitie : %4.2f %% \n", p_inf, p_sup);

/* ---------------------------------------------------------------- */
  printf("3).............\n");
  int nb_int1 = 0, nb_int2 = 0, nb_int3 = 0, nb_int4 = 0;
  float val, p1, p2, p3, p4;

  for (int i=0; i<N; i++){
    val = (float) 4*rand()/RAND_MAX;
    //    printf("---%f= \n",val);
    if (val < 1)
      nb_int1++;
    else
      if (val < 2)
        nb_int2++;
      else
        if (val < 3)
          nb_int3++;
        else
          nb_int4++;
  }
  //  printf("%d %d %d %d  \n", nb_int1, nb_int2, nb_int3, nb_int4);
  p1 = (float) nb_int1*100 / (float) N;
  p2 = (float) nb_int2*100 / (float) N;
  p3 = (float) nb_int3*100 / (float) N;
```

```
  p4 = (float) nb_int4*100 / (float) N;
  printf("Quarts : 1er=%4.2f%%, 2eme=%4.2f%%, 3eme=%4.2f%%, 4eme=%4.2f%% \n", p1, p2, p3, p4);

/* ---------------------------------------------------------- */
  printf("4).............\n");
  int diff = 5; // nb valeurs de difference entre moitie 1 et 2
  int nb_val = 0;

  nb_inf = 0, nb_sup = 0;
  do
    {
      nb_val++;
      rand() < mid_value ? nb_inf++ : nb_sup++;
    }
  while (nb_inf - nb_sup < diff);

  printf("Il faut %d valeurs pour avoir %d valeurs de plus dans la moitie 1 que 2. \n", nb_val, diff);

/* ---------------------------------------------------------- */
  printf("5).............\n");
  for (diff=1; diff<11; diff++){
    nb_val = 0;
    nb_inf = 0, nb_sup = 0;
    do
      {
        nb_val++;
        rand() < mid_value ? nb_inf++ : nb_sup++;
      }
    while (nb_inf - nb_sup < diff);
    printf("Il faut %d valeurs pour avoir %d valeurs de plus dans la moitie 1 que 2. \n", nb_val, diff);
  }

/* ---------------------------------------------------------- */
  printf("6).............\n");
  srand(time(NULL));
  for (diff=1; diff<11; diff++){
    nb_val = 0;
    nb_inf = 0, nb_sup = 0;
    do
      {
        nb_val++;
        rand() < mid_value ? nb_inf++ : nb_sup++;
      }
    while (nb_inf - nb_sup < diff);
    printf("Il faut %d valeurs pour avoir %d valeurs de plus dans la moitie 1 que 2. \n", nb_val, diff);
  }

  return EXIT_SUCCESS;
}
```

Chapitre 15

Fonctions : prototypes, définition, appels

15.1 f-fibo

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

// prototype de fibo
float fibo(int n);

/* ---------------------------------------------------------------- */
// main
int main(void){
  int n;
  float val;

/* Entrées */
  printf("Entrer le nbre de termes de la suite de Fibonacci à calculer : ");
  scanf("%d", &n);

  printf("\n i, fibo[i] \n");

  // plein d'appels a fibo
  for (int i=0 ; i<n+1; i++){
    val = fibo(i);
    printf("%3d %f \n", i, val);
  }
  return EXIT_SUCCESS;
}

/* ---------------------------------------------------------------- */
// corps de fibo
float fibo(int n)
{
  float u0 = 0.0, u1 = 1.0;
  float u_i, u_im1, u_im2;

  switch (n) {
  case 0 : return u0; break;
  case 1 : return u1; break;
  default :
    /* Calcul : u = v + w */
    u_im2 = u0;
    u_im1 = u1;
    for (int i=2 ; i<n+1; i++){
      u_i = u_im1 + u_im2;
      u_im2 = u_im1;
      u_im1 = u_i;
    }
    return u_i;
  }
```

```
}
```

15.2 fibo

```
/*  prototype de fibo
    calcule le n-ieme terme de la suite de Fibonnacci
*/
float fibo(int n);
```

```
#include <stdio.h>
#include <stdlib.h>
#include "fibo.h"

/* ------------------------------------------------------------ */
// corps de fibo
float fibo(int n)
{
  float u0 = 0.0, u1 = 1.0;
  float u_i, u_im1, u_im2;

  switch (n) {
  case 0 : return u0; break;
  case 1 : return u1; break;
  default :
    u_im2 = u0;
    u_im1 = u1;
    for (int i=2 ; i<n+1; i++){
      u_i = u_im1 + u_im2;
      u_im2 = u_im1;
      u_im1 = u_i;
    }
    return u_i;
  }
}
```

15.3 f-fibo-main

```
#include <stdio.h>
#include <stdlib.h>
#include "fibo.h"

#define MAX 100

// main
int main(void){
  int n;
  float val;

/* Entrées */
  printf("Entrer le nbre de termes de la suite de Fibonacci à calculer : ");
  scanf("%d", &n);

  printf("\n i, fibo[i] \n");

  // plein d'appels a fibo
  for (int i=0 ; i<n+1; i++){
    val = fibo(i);
    printf("%3d %f \n", i, val);
  }
  return EXIT_SUCCESS;
}
```

15.4 f-masques

```
/* ------------------------------------------------------------ */
#define  m0  0x01
#define  m1  m0 << 1
#define  m2  m0 << 2
#define  m3  m0 << 3
#define  m4  m0 << 4
/* ------------------------------------------------------------ */
int parite(int n);
int bit(int n, int pos);
int changetozero(int n, int pos);
int changetoone(int n, int pos);
int octetfaible(int n);
/* ------------------------------------------------------------ */
```

```
#include "f-masques.h"

int m[5] = {m0, m1, m2, m3, m4};
/* ------------------------------------------------------------ */
int parite(int n)
{
  return n % 2;
}
/* ------------------------------------------------------------ */
int bit(int n, int pos)
{
  return n & m[pos];
}
/* ------------------------------------------------------------ */
int changetozero(int n, int pos)
{
  return n & ~m[pos];
}
/* ------------------------------------------------------------ */
int changetoone(int n, int pos)
{
  return n | m[pos];
}
/* ------------------------------------------------------------ */
int octetfaible(int n)
{
  return n & 0xff;
}
/* ------------------------------------------------------------ */
```

15.5 f-masques-main

```
#include <stdio.h>
#include <stdlib.h>
#include "printbin.h"
#include "f-masques.h"

int main(void)
{
  printf("m0 =  %d = %x = ", m0, m0); printbin(m0); printf("\n");
  printf("m1 =  %d = %x = ", m1, m1); printbin(m1); printf("\n");
  printf("m2 =  %d = %x = ", m2, m2); printbin(m2); printf("\n");
  printf("m3 =  %d = %x = ", m3, m3); printbin(m3); printf("\n");
  printf("m4 =  %d = %x = ", m4, m4); printbin(m4); printf("\n");

/* ------------------------------------------------------------ */
  int a, res;

  printf("\n Entrer un entier a : \n");
  scanf("%d", &a);
  printf("Vous avez entré la valeur a suivante. \n ");
  printf("a = %d = %x = ", a, a); printbin(a); printf("\n");

  // parité : solution 1 (reste division euclidienne)
  parite(a) == 0 ? printf("%d est pair ", a) : printf("%d est impair", a);
  printf("\n");
```

```
  // test bit de position 3 == 0
  printf("Le bit de position 3 de %d vaut ", a);
  bit(a, 3) == m3 ?  printf("1 \n") :  printf("0 \n");

  // octet de poids faible == 7
  int sept = m2|m1|m0;
  printf("L'octet de poids faible de %d vaut 7 ?", a);
  octetfaible(a) == sept ?  printf(" Oui \n") :  printf(" Non\n");

  // bit de pos 1 et 4 mis a 1
  res = changetoone(a, 1) ;
  res = changetoone(res, 4);
  printf("Les bits de position 1 et 4 mis à 1 (res = a ou a+2 ou a+16 ou a+18) \n");
  printf("res=%d=%x=", res, res); printbin(res); printf("\n");

  // bit de pos 1 et 2 mis a 0
  res = changetozero(a, 1);
  res = changetozero(res, 2);
  printf("Les bits de position 1 et 2 mis à 0 \n");
  printf("res=%d=%x=", res, res); printbin(res); printf("\n");


  // afficahge de l'octet de poids faible
  res = octetfaible(a);
  printf("L'octet de poids faible de a vaut : \n");
  printf("res=%d=%x=", res, res); printbin(res); printf("\n");
  printf("\n");


  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

```
m0 =  1 = 1 = 1
m1 =  2 = 2 = 10
m2 =  4 = 4 = 100
m3 =  8 = 8 = 1000
m4 =  16 = 10 = 10000

 Entrer un entier a :
1031
Vous avez entré la valeur a suivante.
 a = 1031 = 407 = 10000000111
1031 est impair
Le bit de position 3 de 1031 vaut 0
L'octet de poids faible de 1031 vaut 7 ? Oui
Les bits de position 1 et 4 mis à 1 (res = a ou a+2 ou a+16 ou a+18)
res=1047=417=10000010111
Les bits de position 1 et 2 mis à 0
res=1025=401=10000000001
L'octet de poids faible de a vaut :
res=7=7=111
```

Chapitre 16

Pointeurs

16.1 ptr

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int val = 10;
  int * p;
  int * q = NULL;

  printf("p pointe vers RIEN. \n");
  printf("q pointe vers NULL. \n");
  printf("val: %d, adr_val: %p \n", val, &val);
  printf("p: %p, adr_p: %p \n", p, &p);
  printf("q: %p, adr_q: %p \n\n", q, &q);

  p = &val; //p pointe vers val
  printf("p pointe vers val. \n");
  printf("val: %d, adr_val: %p \n", val, &val);
  printf("p: %p, adr_p: %p \n\n", p, &p);

  int w = *p; // w vaut la valeur pointée  par p
  printf("w vaut la valeur pointée  par p. \n");
  printf("val: %d, adr_val: %p \n", val, &val);
  printf("p: %p, adr_p: %p \n\n", p, &p);
  printf("w: %d, adr_w: %p \n", w, &w);


  int entree;
  printf("Entrer une valeur: ");
  scanf("%d", &entree);
  *p = entree; //val vaut entree
  printf("nouvelle valeur pour val. \n");
  printf("val: %d, adr_val: %p, adr_entree: %p \n", val, &val, &entree);
  printf("p: %p, adr_p: %p \n\n", p, &p);
  printf("w: %d, adr_w: %p \n", val, &val);


  int * *r;
  r = &p; // q pointe vers p

  printf("Entrer une valeur: ");
  scanf("%d", &entree);
  **r = entree; // on modifie val  sans toucher a p
  printf("r pointe vers p. \n");
  printf("p pointe vers val. \n");
  printf("val modifié. \n");
  printf("val: %d, adr_val: %p \n", val, &val);
  printf("p: %p, adr_p: %p \n", p, &p);
  printf("r: %p, adr_r: %p \n\n", r, &r);


  return EXIT_SUCCESS;
```

```
}
```

16.2 arith-ptr

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int val = 10;
  int  *p, *q ;
  char *a, *b;
  char aa = 'a';

  p = &val; //p pointe vers val
  q = p + 1;
  printf("p pointe vers val de type int, q=p+1. \n");
  printf("p: %p, q: %p, sizeof(int): %lu \n", p, q, sizeof(int));
  printf("adr(q) - adr(p): %lu, q-p: %lu \n\n", (unsigned long)q - (unsigned long)p, q - p);

  a = &aa; //p pointe vers val
  b = a + 1;
  printf("b pointe vers aa de type char, b=a+1. \n");
  printf("p: %p, q: %p, sizeof(char): %lu \n", a, b, sizeof(char));
  printf("adr(q) - adr(p): %lu, q-p: %lu \n\n", (unsigned long)b - (unsigned long)a, b - a);

  return EXIT_SUCCESS;
}
```

16.3 affiche-vect

```
/* Procedure d'affichage de tableaux */
#ifndef _AFFICHE_VECT_H_
#define _AFFICHE_VECT_H_

//affiche les valeurs de tab
void affiche_vect(double tab[], int nb_val);
void affiche_char(char * tab[], int nb_val);

//affiche les indices et les valeurs de tab
void affiche2(double tab[], int nb_val);
void affiche2_char(char * tab[], int nb_val);

#endif // _AFFICHE_VECT_H_
```

```
#include <stdio.h>
#include "affiche-vect.h"
/* ------------------------------------------------------------ */
void affiche(double tab[], int nb_val)
{
  for (int k=0; k<nb_val; k++) printf("%f \n", tab[k]);
}
/* ------------------------------------------------------------ */
void affiche_char(char * tab[], int nb_val)
{
  for (int k=0; k<nb_val; k++) printf("%s \n", tab[k]);
}
/* ------------------------------------------------------------ */
void affiche2(double tab[], int nb_val)
{
  for (int k=0; k<nb_val; k++) printf("%i \t %f \n", k, tab[k]);
}
/* ------------------------------------------------------------ */
void affiche2_char(char * tab[], int nb_val)
{
  for (int k=0; k<nb_val; k++) printf("%i \t %s \n", k, tab[k]);
}
```

```
/* ---------------------------------------------------------------- */
```

16.4 affichons-mat

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NBLIGS 5
#define NBCOLS  3
/* ---------------------------------------------------------------- */
void affiche_mat(int nb_lig, int nb_col, float tab[nb_lig][nb_col]);
void affiche2_mat(int nb_lig, int nb_col, float tab[][nb_col]);
void affiche3_mat(float *tab, int nb_lig, int nb_col);
/* ---------------------------------------------------------------- */
int main(void)
{
  srand(time(NULL)); // init graine rand

  //init
  float id2[2][2] = {{1., 0.}, {0., 1.0}};
  float id3[3][3] = {{1., 0., 0.},
                     {0., 1.0, 0.},
                     {0., 0., 1.0}};

  float mat[NBLIGS][NBCOLS];
  for (int i=0; i<NBLIGS; i++)
    for (int j=0; j<NBCOLS; j++)
      mat[i][j] = (float) rand();

  //
  printf("\nAffichage matrices : déclaration avec les deux paramètres.\n");
  affiche_mat(2, 2, id2);
  printf("\n");

  affiche_mat(3, 3, id3);
  printf("\n");

  affiche_mat(NBLIGS, NBCOLS, mat);

  //les versions 1 et 2 sont equivalentes
  printf("\nVersion sans premier paramètre.\n");
  affiche2_mat(2, 2, id2);
  printf("\n");

  affiche2_mat(3, 3, id3);
  printf("\n");

  affiche2_mat(NBLIGS, NBCOLS, mat);

  //version 3 : la matrice vue comme un vecteur
  printf("\nVersion matrice vue comme un vecteur.\n");
  // appel avec l'adresse du premier element
  affiche3_mat(&id2[0][0], 2, 2);
  printf("\n");

  //autre appel equivalent :
  //chgmmt de type : id3 est vue comme un vecteur de float
  affiche3_mat((float *)id3, 3, 3);
  printf("\n");

  affiche3_mat(&mat[0][0], NBLIGS, NBCOLS);

 return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
void affiche_mat(int nb_lig, int nb_col, float tab[nb_lig][nb_col])
{
  for (int k=0; k<nb_lig; k++)
    {
      for (int l=0; l<nb_col; l++)
```

```
        printf("%3.1f \t", tab[k][l]);
      printf("\n");
    }
}
/* ---------------------------------------------------------- */
void affiche2_mat(int nb_lig, int nb_col, float tab[][nb_col])
{
  for (int k=0; k<nb_lig; k++)
    {
      for (int l=0; l<nb_col; l++)
        printf("%3.1f \t", tab[k][l]);
      printf("\n");
    }
}
/* ---------------------------------------------------------- */
void affiche3_mat(float *tab, int nb_lig, int nb_col)
//la matrice est vue comme un vecteur
// Noter l'indice de tab dans la boucle interne
// la double boucle est necessaire pour afficher les sauts de lignes
{
  for (int k=0; k<nb_lig; k++)
    {
      for (int l=k*nb_col; l<(k+1)*nb_col; l++)
        printf("%3.1f \t", tab[l]);
      printf("\n");
    }
}
```

16.5 mystery-inc

```
#include <stdlib.h>
#include <stdio.h>
#include "affiche-tab.h"

int main(void)
{
  char * Fred[] = {"Fred", "H", "Volontaire", "Daphné"};
  char * Daph[] = {"Daphné", "F", "Esthétique", "Fred"};
  char *  Sam[] = {"Sammy", "H", "Goinfre", "Scoub" };
  char *Scoob[] = {"Scooby-Doo", "Chien", "Sammy" };
  char * Vera[] = {"Véra", "F", "Analytique"};
  /*
  affiche_char(Fred, 4);
  affiche_char(Daph, 4);
  affiche_char(Sam, 4);
  affiche_char(Scoob, 3);
  affiche_char(Vera, 3);
*/
  char ** MystInc[] = {Fred, Daph, Sam, Scoob, Vera};
  int taille = 4;
  for (int i=0; i<5; i++)
    {
      if ((MystInc[i] == Scoob) || (MystInc[i] == Scoob)) taille = 3;
      affiche_char(MystInc[i], taille);
    }

  return EXIT_SUCCESS;
}
```

16.6 tab-ptr

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TAILLE 10

int main(void)
```

```
{
  double tab[TAILLE];
  double  *p = NULL;

  srand(time(NULL)); // init graine rand

  for (int i=0; i<TAILLE; i++) tab[i] = (double)rand();

  p = &tab[0]; // equivaut a p = tab
  printf("\n p: %p, sizeof(double): %lu \n", p, sizeof(double));

  printf("tab       \t *(p+i)   \t p+i     \n");
  for (int i=0; i<TAILLE; i++)
    {
      printf("%8.7e \t %8.7e \t %p \n", tab[i], *(p+i), p+i);
    }

  return EXIT_SUCCESS;
}
```

16.7 main-inc

```
#include <stdio.h>
#include <stdlib.h>

void incremente(int * val);

int main(void)
{
  int nbfois, val_init;

  printf("Valeur initiale: ? \n");
  scanf("%d", &val_init);
  printf("Nb fois : ? \n");
  scanf("%d", &nbfois);

  for (int i=0; i<nbfois; i++)
    {
      incremente(&val_init);
    }
  printf("Valeur finale : %d. \n\n", val_init);

  printf("nbfois  : %p \n",&nbfois);
  printf("val_int : %p \n",&val_init);

  return EXIT_SUCCESS;
}

void incremente(int * val)
{
  *val = (*val)++;
}
```

16.8 doubler-tab

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TAILLE 10

/* ---------------------------------------------------------- */
double * deux_fois(double tab[], int nbval)
{
  for (int i=0; i< nbval; i++) tab[i] *= 2;
  return tab;
}
/* ---------------------------------------------------------- */
```

```c
void affiche(double tab[], int nb_val)
{
  for (int k=0; k<nb_val; k++) printf("%f \n", tab[k]);
  printf("\n");
}
/* ---------------------------------------------------------- */

int main(void)
{
  double tab[TAILLE];
  double  *p = NULL;
  double  *q = NULL;

  srand(time(NULL)); // init graine rand
  for (int i=0; i<TAILLE; i++) tab[i] = (double)rand();

  p = tab; // equivaut a p = tab[0]
  affiche(p, TAILLE);

  q = deux_fois(p, TAILLE);

  affiche(q, TAILLE);
  affiche(p, TAILLE); // p et q pointent vers le mm tableau

  return EXIT_SUCCESS;
}
```

16.9 tab-dyn

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "affiche-vect.h"

/* ---------------------------------------------------------- */
int main(void)
{
  int nb;
  printf("Entrer nb : \n");
  scanf("%d", &nb);

  double * p = NULL;
  p = (double *) malloc(nb*sizeof(double));
  // p est un tableau de nb double

  if (p == NULL) return EXIT_FAILURE;


  srand(time(NULL)); // init graine rand

  for (int k=0; k<nb; k++) p[k] =  (double)rand();

  affiche2(p, nb);

  free(p); // on libère la memoire ou p est stockée
  // mais il arrive qu'elle reste non modifiée
  printf("\nLa mémoire a été libérée mais les valeurs sont encore dispo : \n");
  affiche2(p, nb);  //... danger

  return EXIT_SUCCESS;
}
```

16.10 mat-dyn

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```c
/* ---------------------------------------------------------- */
void affiche_mat(int nb_lig, int nb_col, float tab[nb_lig][nb_col]);
void affiche3_mat(float *tab, int nb_lig, int nb_col);
/* ---------------------------------------------------------- */
int main(void)
{
  srand(time(NULL)); // init graine rand

  int nbl, nbc;
  printf("Entrer nb lignes, nb colonnes : \n");
  scanf("%d", &nbl);
  scanf("%d", &nbc);

  float ** mat = NULL;
  // l'allocation d'un tableau a 2 dimension s'effectue en 2 temps
  // étape 1 : allocation de nbl lignes
  // bien noter la taille du sizeof
  mat = (float *) malloc(nbl*sizeof(float *));
  // étape 2 : allocation de nbc cols
  for (int i=0; i<nbl; i++)
    mat[i] = (float *) malloc(nbc*sizeof(float));

  // check alloc
  if (mat == NULL) return EXIT_FAILURE;

  srand(time(NULL)); // init graine rand

  for (int i=0; i<nbl; i++){
    for (int j=0; j<nbc ; j++){
      mat[i][j] =  (double)rand();
    }
  }

  // affichage direct (sans appel à une fonction)
  for (int i=0; i<nbl; i++){
    for (int j=0; j<nbc ; j++){
      printf("%f3.1 ", mat[i][j]);
    }
    printf("\n");
  }


  //TODO
  //impossible d'appeler fction affichage avec matrice dynamique
  //  affiche_mat(nbl, nbc, &mat[0][0]);
  //  affiche3_mat(*mat, nbl, nbc);

  // liberation ... en 2 temps aussi
  for (int i=0; i<nbl; i++)
    free(mat[i]); // liberation de chaque ligne
  free(mat); // liberation ptr sur mat

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------- */
void affiche_mat(int nb_lig, int nb_col, float tab[nb_lig][nb_col])
{
  for (int k=0; k<nb_lig; k++)
    {
      for (int l=0; l<nb_col; l++)
        printf("%3.1f \t", tab[k][l]);
      printf("\n");
    }
}
/* ---------------------------------------------------------- */
void affiche3_mat(float *tab, int nb_lig, int nb_col)
//la matrice est vue comme un vecteur
// Noter l'indice de tab dans la boucle interne
// la double boucle est necessaire pour afficher les sauts de lignes
{
  for (int k=0; k<nb_lig; k++)
    {
      for (int l=k*nb_col; l<(k+1)*nb_col; l++)
        printf("%3.1f \t", tab[l]);
```

```
      printf("\n");
    }
}
/* ---------------------------------------------------------------- */
```

16.11 echo

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char * argv[])
{
  for (int i=0; i<argc; i++)
    {
      printf("%s ", argv[i]);
    }
  printf("\n");

  return EXIT_SUCCESS;
}
```

16.12 echo-inverse

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char * argv[])
{

  printf("%s ", argv[0]);
  for (int i=argc; i>1; i--)
    {
      printf("%s ", argv[i-1]);
    }
  printf("\n");

  return EXIT_SUCCESS;
}
```

16.13 combien

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
  printf("%d \n", argc);

  return EXIT_SUCCESS;
}
```

16.14 max

```
#include <stdio.h>
#include <stdlib.h>

/* ---------------------------------------------------------------- */
void usage(char * argv0)
{
  printf("\n La commande %s demande au moins 1 argument.\n", argv0);
  exit(EXIT_FAILURE);
```

```
}
/* ------------------------------------------------------------ */
int main(int argc, char * argv[])
{
  if (argc < 2) usage(argv[0]);

  double val, m = strtod(argv[1], NULL);

  for (int i=2; i < argc; i++)
    {
      val = strtod(argv[i], NULL);
      if (m < val) m = val;
    }

  printf("%lf \n", m);

  return EXIT_SUCCESS;
}
```

16.15 main-tx-var

```
#include <stdio.h>
#include <stdlib.h>

double deux_x(double x)
{
  return 2.0*x;
}

double trois_x(double x)
{
  return 3.0*x;
}

double moins_x(double x)
{
  return -x;
}

double tx_var(double (*f)(double x), double a, double b)
{
  return ((*f)(a) - (*f)(b))/(a-b);
}

int main(void)
{
  double val_a, val_b, res, res2, res3;

  printf("Valeurs a et b : ? \n");
  scanf("%lf %lf", &val_a, &val_b);

  res2 = tx_var(&deux_x, val_a, val_b);
  res3 = tx_var(&trois_x, val_a, val_b);
  res  = tx_var(&moins_x, val_a, val_b);


  printf("taux de variation de 2x = %2.1f (=2.0) \n", res2);
  printf("taux de variation de 3x = %2.1f (=3.0) \n", res3);
  printf("taux de variation de -x = %2.1f (=-1.0) \n", res);

  printf("val_a : %p, val_b : %p \n", &val_a, &val_b);
  printf("res2 : %p, res3 : %p \n ", &res2, &res3);
  printf("2x : %p, 3x : %p, tx-var : %p \n",&deux_x, &trois_x, &tx_var);

  return EXIT_SUCCESS;
}
```

Chapitre 17

Types, `makefile`

17.1 type-vect-mat

```
// ifndef à finir
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 5
#define M 3

typedef int vecteurs[N];

void deux_fois(vecteurs v, vecteurs deux_v);
void affiche(vecteurs v);


typedef int matrices[N][M];
void rand_mat(matrices m);
void affiche_mat(matrices m);
```

```
#include "type-vect-mat.h"
/* ---------------------------------------------------------------- */
void deux_fois(vecteurs v, vecteurs deux_v)
{
  for (int i=0; i< N; i++) deux_v[i] = 2*(v[i]);
  //  return v;
}
/* ---------------------------------------------------------------- */
void affiche(vecteurs v)
{
  for (int k=0; k<N; k++) printf("%d ", v[k]);
  printf("\n");
}
/* ---------------------------------------------------------------- */
void rand_mat(matrices m)
{
  srand(time(NULL)); // init graine rand */
  for (int i=0; i<N; i++)
    for (int j=0; j<M; j++)
      m[i][j] = rand();
}
/* ---------------------------------------------------------------- */
void affiche_mat(matrices m)
{
  for (int i=0; i<N; i++){
    for (int j=0; j<M; j++)
      printf("%d ", m[i][j]);
    printf("\n");
  }
}
/* ---------------------------------------------------------------- */
```

17.2 manip-type-vect-mat

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "type-vect-mat.h"

int main(void)
{
  vecteurs v = {1, 2, 3, 4, 5};
  vecteurs u;
  vecteurs *p = &v;
  vecteurs *q = NULL;

/*   srand(time(NULL)); // init graine rand */
/*   for (int i=0; i<TAILLE; i++) tab[i] = (double)rand(); */

  // on commence
  printf("v ="); affiche(v);

  printf("u = 2*v :\n");
  deux_fois(v, u);
  printf("v ="); affiche(v);
  printf("u ="); affiche(u);

  printf("manip ptr: \n");
  printf("p (= v) = "); affiche(*p);
  p = &u;
  printf("p (= 2v) = "); affiche(*p);


  matrices damier;
  printf("matrices: \n");
  rand_mat(damier);
  affiche_mat(damier);

/*   p = tab; // equivaut a p = tab[0]  */
/*   affiche(p, TAILLE); */

/*   q = deux_fois(p, TAILLE); */

/*   affiche(q, TAILLE); */
/*   affiche(p, TAILLE); // p et q pointent vers le mm tableau */

  return EXIT_SUCCESS;
}
```

17.3 makefile

```
# Pour différencier les appels make nommés des fichiers
.PHONY: clean

# Config compilo et linker
CC = gcc
CFLAGS = -Wall -pedantic -std=c99
LDFLAGS =

# Tous les objets et l'exe à construire
OBJ = type-vect-mat.o manip-type-vect-mat.o
EXEC = manip-type-vect-mat

# Pour obtenir l'exe à partir de tous les objets
$(EXEC): $(OBJ)
        $(CC) $(LDFLAGS) -o $@ $+

# (Optionnel) Pour obtenir les objets à partir des sources
%.o: %.c
        $(CC) $(CFLAGS) -c $+
```

```
# Pour assurer la dépendance des .o à la mise à jours des .h
manip-type-vect-mat.o: type-vect-mat.h
type-vect-mat.o: type-vect-mat.h

#
clean:
        rm $(OBJ) $(EXEC)
```

Quatrième partie

Exemples de correction des partiels et examens.

Chapitre 18

Correction du partiel de novembre 2010

18.1 echauffement

```c
/* ------------------------------------------------------------ */
/* echauffement */
/* ------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>

#define  m0  0x01
#define  m1  m0 << 1
/* ------------------------------------------------------------ */
int main (void){
  int a, res, masque;

  printf("Entrer un entier a : ");
  scanf("%d", &a);
  printf("Vous avez entré la valeur a suivante. \n ");
  printf(" a = %d = 0x%x \n", a, a);

  // affichage des bits 0 et 1
  masque = 0x3;
  res = a & masque;
  printf("Affichage des bits de position 0 et 1 :\n");
  printf(" res = %d= 0x%x \n", res, res);

  // test bit de position 0 et 1 == 0
  printf("Les bits de position 0 et 1 de %d valent-ils O ?", a);
  (a & masque) == !masque ?  printf("\n Vrai. \n") :  printf("\n Faux. \n");

  // bit de pos 0 et 1 mis a 0
  res = a & ~(m0 | m1);
  printf("On met à zéro les bits de position 0 et 1. \n");
  printf(" res = %d = Ox%x \n", res, res);

  a = res;
  // test bit de position 0 et 1 == 0
  printf("Les bits de position 0 et 1 de %d valent-ils O ?", a);
  (a & masque) == !masque ?  printf("\n Vrai. \n") :  printf("\n Faux. \n");


  return EXIT_SUCCESS;
}
```

18.2 tri2

```c
/* ------------------------------------------------------------ */
/* tri2 */
/* ------------------------------------------------------------ */
#include "stdio.h"
#include "stdlib.h"
```

```c
int main(void)
{
  float a, b;

  printf("Entrez deux valeurs flottantes : ");
  scanf("%f", &a);
  scanf("%f", &b);
  printf("Vous avez entré les valeurs %f, %f \n",a, b);

  // tri
  float t;

  if (a > b)
    {
      t = a, a = b, b = t; // on permute a et b
    }
  // maintenant on a bien a < b
  printf("qui triées par ordre croissant sont %f, %f \n", a, b);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

18.3 tri3

```c
/* ---------------------------------------------------------------- */
/* tri3 */
/* ---------------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
  double a, b, c;

  printf("Entrez trois valeurs flottantes : ");
  scanf("%lf", &a);
  scanf("%lf", &b);
  scanf("%lf", &c);
  printf("Vous avez entré les valeurs %lf, %lf, %lf \n",a, b, c);

  // tri
  double t;
  // on veut : a b c
  if (a > b)
    {
      t = a, a = b, b = t; // on permute a et b
    }
  // maintenant on a bien a < b
  // rappel : on veut : a b c
  if (c < b)
    {
      if (c < a) // on a : c a b
        {
          t = c, c = b, b = a, a = t;
        }
      else // on a : a c b
        {
          t = c, c = b, b = t;
        }
      // maintenant on a bien : a b c
      }
  printf("qui triées par ordre croissant sont %lf, %lf, %lf \n", a, b, c);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

18.4 expression

```c
/* ---------------------------------------------------------- */
/* expression */
/* ---------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
  unsigned n1, n2, n3;
  int res;

  printf("Entrez trois entiers positifs : ");
  scanf("%ud", &n1);
  scanf("%ud", &n2);
  scanf("%ud", &n3);

  res = n1 +2*n2 -n3*n3*n3;

  printf("Pour n1=%u, n2=%u, n3=%u, on calcule %d. \n", n1, n2, n3, res);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------- */
```

18.5 des-expressions

```c
/* ---------------------------------------------------------- */
/* des_expressions */
/* ---------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
  unsigned n1, n2, n3;
  int res;
  unsigned rep = 1;

  do
    {
      printf("Entrez trois entiers positifs : ");
      scanf("%ud", &n1);
      scanf("%ud", &n2);
      scanf("%ud", &n3);

      res = n1 +2*n2 -n3*n3*n3;

      printf("Pour n1=%u, n2=%u, n3=%u, on calcule %d. \n", n1, n2, n3, res);

      printf("Voulez-vous recommencer ? Si oui, entrez 0 (zéro).  \n");
      scanf("%u", &rep);
    }
  while (rep == 0);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------- */
```

18.6 exp-bool

```c
/* ---------------------------------------------------------- */
/* exp_bool */
/* ---------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"
```

```c
#include "stdbool.h"

int main(void)
{
  bool e;
  bool b[2] = {false, true};

  printf("b1 b2 b3 :: (b1 or b2) and not b3 \n");

  for (int i=0; i<2; i++)
    {
      for (int j=0; j<2 ; j++)
        {
          for (int k=0; k<2; k++)
            {
              e = (b[i] || b[j]) && (!b[k]);
              printf("%i  %i  %i  :: %i\n", b[i], b[j], b[k], e);
            }
        }
    }

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

Chapitre 19

Correction de l'examen de décembre 2010

19.1 paire

```
/* ------------------------------------------------------------ */
/* paire */
/* ------------------------------------------------------------ */
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
  int val;
  unsigned rep = 1;
  do
    {
      //entrees
      printf("Entrez une valeur entière : ");
      scanf("%i", &val);
      printf("Vous avez entré la valeur %d ",val);

      // test signe
      if (val >= 0 )
          printf("qui est positive ");
      else
        printf("qui est négative ");

      // test parite
      if (val % 2 == 0)
        printf("et paire. \n");
      else
        printf("et impaire. \n");

      // next value
      printf("Voulez-vous recommencer ? Si oui, entrez 0 (zéro).  \n");
      scanf("%u", &rep);
    }
  while (rep == 0);


  //sorties
  return EXIT_SUCCESS;
}
/* ------------------------------------------------------------ */
```

19.2 code-asci

```
/* ------------------------------------------------------------ */
/* code-ascii */
/* ------------------------------------------------------------ */
```

```c
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
  char val;
  unsigned code;

  printf("Entrez un caractère : ");
  scanf("%c", &val);
  printf("Vous avez entré le caractère %c ",val);

  // code ascii
  printf(" qui correspond au code %u; \n", val);
  printf("Son prédécesseur est %c et son successeur %c. \n", val-1, val+1);

  // next value
  printf("Entrez une valeur entière : ");
  scanf("%u", &code);
  printf("Vous avez entré la valeur %u ",code);

  // code ascii
  printf(" qui correspond au caractère %c;  \n", code);
  printf("Son prédécesseur est %c et son successeur %c. \n", code-1, code+1);


  //sorties
  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

19.3 max-prod-mat

```c
/* ---------------------------------------------------------------- */
/* max-prod-mat */
/* ---------------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"
#include "math.h"

#define TX 5
#define TY 3


int main(void)
{
  float x[TX] = {-1, -3, -5, 1, 3};
  float y[TY] = {10, 20, 30};
  float A[TX][TY];

  unsigned imax=0, jmax=0;
  float max = 0.0;

  // calcul A = x * yT
  for (int i=0 ; i<TX; i++){
    for (int j=0; j<TY; j++){
      A[i][j] = x[i]*y[j];
    };
  };

  // affichage
  for (int i=0 ; i<TX; i++){
    for (int j=0; j<TY; j++){
      printf("%5.2f ",A[i][j]);
    };
    printf("\n");
  }

  // identification indice max val abs
  for (int i=0 ; i<TX; i++){
    for (int j=0; j<TY; j++){
```

```
      if ( fabs(A[i][j]) > max )
           {
             max = fabs(A[i][j]);
             imax = i;
             jmax = j;
           };
    };
  };

  // affichage resultat
  printf("La valeur max est A(%d, %d) = %5.2f \n",imax+1, jmax+1, max);

  //sorties
  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

19.4 moiss

```
/* ---------------------------------------------------------------- */
/* mois */
/* ---------------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"



int main(void)
{
  int num_mois, nb_jours;

  // entree
  do {
    printf("Entrer le numéro d'un mois (compris entre 1 et 12) : ");
    scanf("%d",&num_mois);
  } while ((num_mois <= 0) || (num_mois > 12));

  switch (num_mois) {
  case 1:
    nb_jours = 31;
    break;
  case 2:
    nb_jours = 28;
    break;
  case 3:
    nb_jours = 31;
    break;
  case 4:
    nb_jours = 30;
    break;
  case 5:
    nb_jours = 31;
    break;
  case 6:
    nb_jours = 30;
    break;
  case 7:
    nb_jours = 31;
    break;
  case 8:
    nb_jours = 31;
    break;
  case 9:
    nb_jours = 30;
    break;
  case 10:
    nb_jours = 31;
    break;
  case 11:
    nb_jours = 30;
    break;
```

```
  case 12:
    nb_jours = 31;
    break;
  }

  // affichage resultat
  printf("Il y a %d jours le mois %d. \n", nb_jours, num_mois);

  //sorties
  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

19.5 f-mois

```
/* ---------------------------------------------------------------- */
/* f_mois */
/* ---------------------------------------------------------------- */
int f_mois(int num_mois);
```

```
/* ---------------------------------------------------------------- */
/* f_mois */
/* ---------------------------------------------------------------- */
#include "stdio.h"
#include "stdlib.h"

int f_mois(int num_mois)
{
  int nb_jours;

  switch (num_mois) {
  case 1:
    nb_jours = 31;
    break;
  case 2:
    nb_jours = 28;
    break;
  case 3:
    nb_jours = 31;
    break;
  case 4:
    nb_jours = 30;
    break;
  case 5:
    nb_jours = 31;
    break;
  case 6:
    nb_jours = 30;
    break;
  case 7:
    nb_jours = 31;
    break;
  case 8:
    nb_jours = 31;
    break;
  case 9:
    nb_jours = 30;
    break;
  case 10:
    nb_jours = 31;
    break;
  case 11:
    nb_jours = 30;
    break;
  case 12:
    nb_jours = 31;
    break;
  }
  return nb_jours;
}
```

```
/* ------------------------------------------------------------ */
```

19.6 mois2

```
/* ------------------------------------------------------------ */
/* mois2
/* gcc -Wall -pedantic -std=c99 -o mois2 f_mois.c mois2.c */
/* ------------------------------------------------------------ */
#include "stdio.h"
#include "stdlib.h"
#include "f_mois.h"

int main(void)
{
  int num_mois;

  // entree
  do {
    printf("Entrer le numéro d'un mois (compris entre 1 et 12) : ");
    scanf("%d",&num_mois);
  } while ((num_mois <= 0) || (num_mois > 12));


  // affichage resultat
  printf("Il y a %d jours le mois %d. \n", f_mois(num_mois), num_mois);

  //sorties
  return EXIT_SUCCESS;
}
/* ------------------------------------------------------------ */
```

Chapitre 20

Correction du partiel de mars 2011

20.1 tables-mult

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  printf(" x !");
  for (int i=1; i<11; i++){
    printf("%2i ", i);
  }
  printf("\n");
  printf("%s", "----------------------------------");
  printf("\n");

  for (int i=1; i<11; i++){
    printf("%2i !", i);
    for (int j=1; j<11; j++){
      printf("%2i ", i*j);
    }
    printf("\n");
  }
  printf("%s \n", "----------------------------------");

  return EXIT_SUCCESS;
}
```

20.2 xoxo

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  unsigned int n;

  printf("Entrer une valeur entiere (0 si arret): ");
  scanf("%ui",&n);

  while (n != 0)
    {
      for (int i=1; i<=n; i++){
        for (int j=1; j<=i; j++){
          printf("X ");
        }
        printf("\n");
      }
      for (int i=0; i<n; i++){
        for (int j=0; j<i; j++){
          printf("  ");
        }
        for (int j=i; j<n; j++){
```

```
          printf("O ");
        }
        printf("\n");
      }

      printf("Entrer une valeur entiere (0 si arret): ");
      scanf("%ui",&n);
    }

  return EXIT_SUCCESS;
}
```

20.3 compter

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  char m[26]="anticonstitutionnellement";
  char c;
  int nb=0;

  printf("%s \n",m);
  printf("Entrer un caractère en minuscule (X si arret) : ");
  scanf("%c%*c", &c);
  //  printf("\n");

  while (c != 'X'){
    nb = 0;
    for (int i=0; i<=26; i++){
      if (m[i] == c) nb += 1;
    }
    printf("Le mot %s comporte %i fois la lettre %c.", m, nb, c);
    printf("\n \n");
    printf("Entrer un caractère en minuscule (X si arret) : ");
    scanf("%c%*c", &c);
    //    printf("\n");

}

  return EXIT_SUCCESS;
}
```

Chapitre 21

Correction de l'examen de mai 2011

21.1 rev1

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
  char upvd[23] = "Universite de Perpignan";

  // affichage
  for (int i=0; i<23; i++)
    printf("%c", upvd[i]);
  printf("\n");

  // affichage 10 premieres lettres
  for (int i=0; i<10; i++)
    printf("%c", upvd[i]);
  printf("\n");

  // affichage inverse
  for (int i=22; i>=0; i--)
    printf("%c", upvd[i]);
  printf("\n");

  // affichage inverse des 10 premieres lettres
  char tmp[10];
  for (int i=0; i<10; i++)
    tmp[i] = upvd[9-i];
  // tmp contient les 10 premieres lettres dans l'ordre inverse
  for (int i=0; i<10; i++)
    printf("%c", tmp[i]);
  printf("\n");

  return EXIT_SUCCESS;
}
```

21.2 rev2

```c
#include <stdlib.h>
#include <stdio.h>

void aff(char * val, int nblet);
/* ------------------------------------------------------------ */
int main(void)
{
  char upvd[23] = "Universite de Perpignan";

  // affichage
  aff(upvd, 23);
  printf("\n");
```

```
  // affichage 10 premieres lettres
  aff(upvd, 10);
  printf("\n");

  // tmp contient upvd dans l'ordre inverse
  char tmp[23];
  for (int i=0; i<23; i++)
    tmp[i] = upvd[22-i];

  // affichage inverse
  aff(tmp, 23);
  printf("\n");

  // affichage inverse des 10 premieres lettres
  aff(tmp, 10);
  printf("\n");

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
void aff(char * val, int nblet)
{
  for (int i=0; i<nblet; i++)
    printf("%c", val[i]);
}
```

21.3 rev3

```
#include <stdlib.h>
#include <stdio.h>

#define LONGUEURMOT 23
#define UP "Universite de Perpignan"

void aff(char * val, int nblet);
/* ---------------------------------------------------------------- */
int main(void)
{
  char mot[LONGUEURMOT] = UP;

  // affichage
  aff(mot, LONGUEURMOT);
  printf("\n");

  // affichage 10 premieres lettres
  aff(mot, 10);
  printf("\n");

  // tmp contient upvd dans l'ordre inverse
  char tmp[LONGUEURMOT];
  for (int i=0; i<LONGUEURMOT; i++)
    tmp[i] = mot[LONGUEURMOT-1-i];

  // affichage inverse
  aff(tmp, LONGUEURMOT);
  printf("\n");

  // affichage inverse des 10 premieres lettres
  aff(tmp, 10);
  printf("\n");

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
void aff(char * val, int nblet)
{
  for (int i=0; i<nblet; i++)
    printf("%c", val[i]);
}
```

21.4 rev4

```c
#include <stdlib.h>
#include <stdio.h>

#define LONGUEURMOT 35
#define UP "Universite de Perpignan Via Domitia"

void aff(char * val, int nblet);
/* ---------------------------------------------------------------- */
int main(void)
{
  char mot[LONGUEURMOT] = UP;

  // affichage
  aff(mot, LONGUEURMOT);
  printf("\n");

  // affichage 10 premieres lettres
  aff(mot, 10);
  printf("\n");

  // tmp contient upvd dans l'ordre inverse
  char tmp[LONGUEURMOT];
  for (int i=0; i<LONGUEURMOT; i++)
    tmp[i] = mot[LONGUEURMOT-1-i];

  // affichage inverse
  aff(tmp, LONGUEURMOT);
  printf("\n");

  // affichage inverse des 10 premieres lettres
  aff(tmp, 10);
  printf("\n");

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
void aff(char * val, int nblet)
{
  for (int i=0; i<nblet; i++)
    printf("%c", val[i]);
}
```

21.5 rev5

```c
#include <stdlib.h>
#include <stdio.h>

void usage(char * argv0);
void aff(char * val, int nblet);
int longueur(char * val);
void concatene(char * res, char * ch1, int l1, char * ch2, int l2);
/* ---------------------------------------------------------------- */
int main(int argc, char * argv[])
{
    // check appel complet
  if (argc < 2) usage(argv[0]);

  int LongChaine = 0;
  char LaChaine[180];

  for (int nummot = 1; nummot < argc; nummot++)
    {
      //on recupere la longueur de l'argv
      int LongMot = longueur(argv[nummot]);
      //      printf("\nlongueur = %i\n", LongMot);


      //on concatene en inserant un blanc entre chaque mot
```

```
      concatene(LaChaine, LaChaine, LongChaine, argv[nummot], LongMot);
      LongChaine += LongMot;
      concatene(LaChaine, LaChaine, LongChaine, " ", 1);
      LongChaine += 1;
    }
  //un blanc de trop a été rajout en fin
  LongChaine -= 1;


  // affichage
  aff(LaChaine, LongChaine);
  printf("\n");


  // affichage 10 premieres lettres
  aff(LaChaine, 10);
  printf("\n");

  // tmp contient upvd dans l'ordre inverse
  char tmp[LongChaine];
  for (int i=0; i<LongChaine; i++)
    tmp[i] = LaChaine[LongChaine-1-i];

  // affichage inverse
  aff(tmp, LongChaine);
  printf("\n");

  // affichage inverse des 10 premieres lettres
  aff(tmp, 10);
  printf("\n");


  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
void aff(char * val, int nblet)
{
  for (int i=0; i<nblet; i++)
    printf("%c", val[i]);
}
/* ---------------------------------------------------------------- */
void usage(char * argv0)
{
  printf("\nLa commande %s demande au moins un argument. \n", argv0);
  exit(EXIT_FAILURE);
}
/* ---------------------------------------------------------------- */
int longueur(char val[])
{
  int res = 0;
  while ( val[res] != '\0')
    res++;

  return res;
}
/* ---------------------------------------------------------------- */
void concatene(char * res, char * ch1, int l1, char * ch2, int l2)
{
  for (int i=0; i<l1; i++)
    res[i] = ch1[i];
  for (int i=0; i<l2; i++)
    res[l1+i] = ch2[i];
}
```

21.6 aff-tab-carres-1

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
  int carre[10];
```

```
  for (int i=1; i<11; i++){
    carre[i-1] = i*i;
  }

  for (int i=1; i<11; i++){
    printf("%4i \n", carre[i-1]);
  }

  return EXIT_SUCCESS;
}
```

21.7 aff-tab-carres-2

```
#include <stdio.h>
#include <stdlib.h>


int carre[10];

void tab_carre_glob(void){
  for (int i=1; i<11; i++){
    carre[i-1] = i*i;
  }
}

int main(void){

  printf("Calcul et stockage : début ... ");
  tab_carre_glob();
  printf("... fin \n");

  for (int i=1; i<11; i++){
    printf("%4i \n", carre[i-1]);
  }

  return EXIT_SUCCESS;
}
```

21.8 aff-tab-carres-3

```
#include <stdio.h>
#include <stdlib.h>

void tab_carre(int * tab){
  for (int i=1; i<11; i++){
    tab[i-1] = i*i;
  }
}

void aff_tab(int * tab){
  for (int i=1; i<11; i++){
    printf("%4i \n", tab[i-1]);
  }
}

int main(void){
  int carre[10];

  printf("Calcul et stockage : début ... ");
  tab_carre(carre);
  printf("... fin \n");

  // affichage
  aff_tab(carre);

  return EXIT_SUCCESS;
}
```

21.9 aff-tab-carres-4

```c
#include <stdio.h>
#include <stdlib.h>

#define TAILLE 17

void tab_carre(int * tab, int nbval){
  for (int i=1; i<nbval+1; i++){
    tab[i-1] = i*i;
  }
}

void aff_tab(int * tab, int nbval){
  for (int i=1; i<nbval+1; i++){
    printf("%4i \n", tab[i-1]);
  }
}

int main(void){
  int carre[TAILLE];

  printf("Calcul et stockage : début ... ");
  tab_carre(carre, TAILLE);
  printf("... fin \n");

  // affichage
  aff_tab(carre, TAILLE);

  return EXIT_SUCCESS;
}
```

21.10 aff-tab-carres-5

```c
#include <stdio.h>
#include <stdlib.h>


void tab_carre(int * tab, int nbval){
  for (int i=1; i<nbval+1; i++){
    tab[i-1] = i*i;
  }
}

void aff_tab(int * tab, int nbval){
  for (int i=1; i<nbval+1; i++){
    printf("%4i \n", tab[i-1]);
  }
}

int main(void){
  int n;

  // recuperation taille n
  do {
    printf("Calcul des carrés des N nombres entiers. \nEntrer N : \n");
    scanf("%i",&n);
  } while (n < 1);

  // allocation dynamique d'un vecteur de int de taille n
  int * carres = NULL;
  carres = (int *) malloc(n*sizeof(int));

  if (carres == NULL) return EXIT_FAILURE;

  // calcul et stockage
  printf("Calcul et stockage : début ... ");
  tab_carre(carres, n);
  printf("... fin \n");
```

```
  // affichage
  aff_tab(carres, n);

  // recuperation memoire dynamique
  free(carres);

  return EXIT_SUCCESS;
}
```

21.11 aff-tab-carres

```
#include <stdio.h>
#include <stdlib.h>

/* ---------------------------------------------------------------- */
void tab_carre(int * tab, int nbval){
  for (int i=1; i<nbval+1; i++){
    tab[i-1] = i*i;
  }
}
/* ---------------------------------------------------------------- */
void aff_tab(int * tab, int nbval){
  for (int i=1; i<nbval+1; i++){
    printf("%4i \n", tab[i-1]);
  }
  printf("\n");
}
/* ---------------------------------------------------------------- */
void usage(char * argv0)
{
  printf("\nLa commande %s demande un seul argument entier > 1. \n", argv0);
  exit(EXIT_FAILURE);
}
/* ---------------------------------------------------------------- */
int main(int argc, char * argv[])
{
  // check appel complet
  if (argc != 2) usage(argv[0]);

  // recuperation taille n
  int n = atoi(argv[1]);
  if (n < 1) usage(argv[0]);

  // allocation dynamique d'un vecteur de int de taille n
  int * carres = NULL;
  carres = (int *) malloc(n*sizeof(int));

  if (carres == NULL) return EXIT_FAILURE;

  // calcul et stockage
  tab_carre(carres, n);

  // affichage
  aff_tab(carres, n);

  // recuperation memoire dynamique
  free(carres);

  return EXIT_SUCCESS;
}
```

Chapitre 22

Correction de l'examen de juin 2011 (session 2)

22.1 exp2

```c
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
  float x, y;
  float res;
  unsigned rep = 1;

  do
    {
      printf("Entrez deux flottants : ");
      scanf("%f", &x);
      scanf("%f", &y);

      res = (x-y)*(x+y);

      printf("Pour x=%f, y=%f, on calcule %f. \n", x, y, res);

      printf("Voulez-vous recommencer ? Si oui, entrez 0 (zéro).  \n");
      scanf("%u", &rep);
    }
  while (rep == 0);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

22.2 exp3

```c
//
// gcc -Wall -pedantic -std=c99 -lm -o exp3 exp3.c
//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
  float x, res;
  unsigned rep = 1;

  do
    {
      printf("Entrez un flottant : ");
      scanf("%f", &x);
```

```
      res = sqrt(x) - 1;

      printf("Pour x=%f, on calcule %f. \n", x, res);

      printf("Voulez-vous recommencer ? Si oui, entrez 0 (zéro).  \n");
      scanf("%u", &rep);
    }
  while (rep == 0);

  return EXIT_SUCCESS;
}
/* ---------------------------------------------------------------- */
```

22.3 un

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 20

int main(void){
  int res = 0;

  for (int i=0; i<MAX; i++){
    res = i & 0x1;
    if (res == 1)
      printf("Le bit de poids faible de %i vaut 1.\n", i);
  }

  return EXIT_SUCCESS;
}
```

22.4 combiendeun

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 20

int main(void){
  int save_val, val = 0;
  int nb_bits = 0;

  for (int i=0; i<MAX; i++)
    {
      val = i;
      save_val = i;
      while (true)
        {
          if (val == 0) // plus rien a decaller
            break;

          // test bit poids faible = 1
          if ((val & 0x1) == 1)
            nb_bits++;

          // on passe au bit suivant
          val = val >>1;
        }
      printf("%i a %i bits égaux à 1.    ", save_val, nb_bits);
      nb_bits = 0;
      // saut de ligne une fois sur 2
      if ((i%2) == 1)
        printf("\n");
    }

  return EXIT_SUCCESS;
```

```
}
```

22.5 suite

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

int main(void){
  int n;
  int u0 = 1, u1 = 2;
  int u, v, w;

/* Entrées */
  printf("Entrer le nbre de termes de la suite à calculer : ");
  scanf("%d", &n);

  printf("\n i, u[i] \n");
  // affichage systématique u0, u1
  printf("%3d %d \n", 0, u0);
  printf("%3d %d \n", 1, u1);


  /* Calcul : u = v + w */
  v = u0;
  w = u1;
  for (int i=2 ; i<n+1; i++){
    u = 2*w - v;
    v = w;
    w = u;

    // affichage des termes d'indices pairs uniquement
    if (i%2 == 0) printf("%3d %d \n", i, u);
  };


  return EXIT_SUCCESS;
}
```

22.6 suite2

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

int main(void){
  int  val, rep = 0;
  int u[MAX] = {1, 2}; // initialisation u0 et u1

  //calcul
  printf("Calcul des %d valeurs de la suite.\n", MAX);

  for (int i=2 ; i<MAX; i++){
    u[i] = 2*u[i-1] - u[i-2];
  }

  // affichage
  printf("On vérifie avec les 10 premiers : \n \n i, u[i] \n");
  // affichage des termes d'indices pairs uniquement
  for (int i=0 ; i<10; i++)
    printf("%3d %d \n", i, u[i]);

  //
  do
    {
      printf("Entrez une valeur entiere : ");
```

```
      scanf("%d", &val);

      // version 1 : parcours complet
      int trouve = 0;
      for (int i=0; i<MAX; i++){
        if (u[i] == val){
          printf("La valeur u[%d] = %d \n", i, val);
          trouve = 1;
        }
      }
      if (trouve == 0)
        printf("La valeur %d n'apparait pas dans la suite\n", val);

      printf("Voulez-vous recommencer ? Si oui, entrez 1.  \n");
      scanf("%d", &rep);
    }
  while (rep == 1);

  return EXIT_SUCCESS;
}
```

Chapitre 23

Correction de l'examen de juillet 2011 (session 2)

23.1 moy1

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NBNOTES 10

double Moyenne(int *val, int n)
{
  double moy = 0.0;
  for (int i=0; i<n; i++)
    moy += val[i];
  return (moy/n);
}
/* ---------------------------------------------------------- */
double EcartType(int *val, int n)
{
  double Moy = Moyenne(val, n);
  double EcTy = 0.0;
  for (int i=0; i<n; i++)
    EcTy += (val[i] - Moy)*(val[i] - Moy);
  return (sqrt(EcTy/(n-1)));
}
/* ---------------------------------------------------------- */
void Affichage(int *val, int n)
  {
    for (int i=0; i<n; i++)
      printf("%d \n", val[i]);
  }
/* ---------------------------------------------------------- */
int main(void){
  int Notes[NBNOTES] = {13, 8, 10, 4, 2, 1, 7, 10, 8, 14};
  double Moy = 0.0;
  double EcTy = 0.0;

  Affichage(Notes, NBNOTES);
  Moy = Moyenne(Notes, NBNOTES);
  EcTy = EcartType(Notes, NBNOTES);

  printf("Moyenne = %f, Ecart Type = %f\n", Moy, EcTy);

  return EXIT_SUCCESS;
}
```

23.2 moy2

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <math.h>

#define NBNOTES 10

double Moyenne(int *val, int n)
{
  for (int i=0; i<n; i++)
    moy += val[i];
  return (moy/n);
}

double EcartType(int *val, int n)
{
  double Moy = Moyenne(val, n);
  for (int i=0; i<n; i++)
    EcTy += (Notes[i] - Moy)*(Notes[i] - Moy);
  return (sqrt(EcTy/(n-1)));
}

void Affichage(int *val, int n)
  {
    for (int i=0; i<n; i++)
      printf("%d \n", Notes[i]);
  }

int main(void){
  int Notes[NBNOTES] = {13, 8, 10, 4, 2, 1, 7, 10, 8, 14};
  double Moy = 0.0;
  double EcTy = 0.0;

  Affichage(Notes, NBNOTES);
  Moy = Moyenne(Notes, NBNOTES);
  EcTy = EcartType(Notes, NBNOTES);

  printf("Moyenne = %f, Ecart Type = %f\n", Moy, EcTy);

  return EXIT_SUCCESS;
}
```

23.3 moy4

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NBNOTES 10
#define NBSERIES 5

#define FILE_NAME "./notes.txt"

double Moyenne(double *val, int n)
{
  double moy = 0.0;
  for (int i=0; i<n; i++)
    moy += val[i];
  return (moy/n);
}
/* ------------------------------------------------------------ */
double EcartType(double *val, int n)
{
  double Moy = Moyenne(val, n);
  double EcTy = 0.0;
  for (int i=0; i<n; i++)
    EcTy += (val[i] - Moy)*(val[i] - Moy);
  return (sqrt(EcTy/(n-1)));
}
/* ------------------------------------------------------------ */
void Affichage(double *val, int n)
  {
    for (int i=0; i<n; i++)
```

```
      printf("%6.2f \n", val[i]);
  }
/* ---------------------------------------------------------------- */
void Affichage2(int m, int n, double val[m][n])
  {
    for (int i=0; i<m; i++){
      for (int j=0; j<n; j++)
        printf("%6.2f ", val[i][j]);
      printf("\n");
    }
  }
/* ---------------------------------------------------------------- */
int main(void){
  double Notes[NBSERIES][NBNOTES];
  double Moy = 0.0;
  double MoySerie[NBSERIES];
  double EcTy = 0.0;
  double val;

  // Lire notes
  FILE * file_in = fopen(FILE_NAME, "r+"); //lecture-ecriture
    // check ouverture correcte
  if (file_in == NULL)
    {
      perror("Erreur: fopen()");
      return EXIT_FAILURE;
    }
  for (int s=0; s<NBSERIES; s++){
    for (int i=0; i<NBNOTES; i++){
      fscanf(file_in, "%lf", &val);
      Notes[s][i] = val;
    }
    fscanf(file_in, "\n");
  }

  fclose(file_in);

  // traitements
  Affichage2(NBSERIES, NBNOTES, Notes);

  // traitement par serie
  for (int s=0; s<NBSERIES; s++)
    MoySerie[s] = Moyenne(Notes[s], NBNOTES);

  printf("moyenne par serie \n");
  Affichage(MoySerie, NBSERIES);

  //traitement global
  printf("moyenne globale \n");
  Moy = Moyenne(MoySerie, NBSERIES);
  printf("%6.2f \n", Moy);

  return EXIT_SUCCESS;
}
```

23.4 concattab

```
#include <stdio.h>
#include <stdlib.h>

/* ---------------------------------------------------------------- */
void InitPaire(int dim, int val[dim]){
  for (int i=0; i<dim; i++)
    val[i] = 2*i;
}
/* ---------------------------------------------------------------- */
void InitImpaire(int dim, int val[dim]){
  for (int i=0; i<dim; i++)
    val[i] = 2*i+1;
}
/* ---------------------------------------------------------------- */
```

```c
void Affichage(int n, int *val)
  {
    for (int i=0; i<n; i++)
      printf("%3i \n", val[i]);
  }
/* ---------------------------------------------------------- */
int main(void){
  int d1, d2;
  int * t1 = NULL; // tab dynamique
  int * t2 = NULL;

  printf("Entrer d1:"); scanf("%i", &d1);
  printf("Entrer d2:"); scanf("%i", &d2);

  t1 = (int *) malloc(d1*sizeof(int)); // t1 est un tableau de d1 int
  t2 = (int *) malloc(d2*sizeof(int));

  if (t1 == NULL) return EXIT_FAILURE;
  if (t1 == NULL) return EXIT_FAILURE;

  InitPaire(d1, t1);
  InitImpaire(d2, t2);

  Affichage(d1, t1);
  printf("\n");
  Affichage(d2, t2);
  printf("\n");

  // concat t1, t2
  int * t = NULL;
  int d = d1+d2;
  t = (int *) malloc(d*sizeof(int));

  for (int i=0; i<d1; i++)
    t[i]  = t1[i];
  for (int i=d1; i<d; i++)
    t[i]  = t2[i-d1];

  Affichage(d, t);

  free(t1);
  free(t2);
  free(t);

  return EXIT_SUCCESS;
}
```

Cinquième partie

Annexes

Chapitre 24

C Reference Card (ANSI)

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$,...)	function declarations
type name	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$,...) {	function definition
declarations	local variable declarations
statements	
`return` *value*`;`	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(`*arg*`)`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int`, `float`,...	`*int, *float,...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

initialize variable	*type name*`=`*value*
initialize array	*type name*`[]={`*value*$_1$,...`}`
initialize char string	`char` *name*`[]="`*string*`"`

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a', '\`*ooo*`', '\x`*hh*`'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]`...

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	

create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single value, multiple type structure	`union`
bit field with *b* bits	*member* `:` *b*

Operators (grouped by precedence)

structure member operator	*name*`.`*member*
structure pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimeters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break`
next iteration of `while`, `do`, `for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.h>`	`<math.h>`	`<setjmp.h>`	`<signal.h>`	`<stdarg.h>`
`<stddef.h>`	`<stdio.h>`	`<stdlib.h>`	`<string.h>`	`<time.h>`

Character Class Tests `<ctype.h>`

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case?	`tolower(c)`
convert to upper case?	`toupper(c)`

String Operations `<string.h>`

`s`,`t` are strings, `cs`,`ct` are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
up to `n` chars	`strncpy(s,ct,n)`
concatenate `ct` after `s`	`strcat(s,ct)`
up to `n` chars	`strncat(s,ct,n)`
compare `cs` to `ct`	`strcmp(cs,ct)`
only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs`	`strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `cs`	`memset(s,c,n)`

Chapitre 25

Résumé des commandes de base UNIX

Les commandes suivantes sont suffisantes pour les séances d'exercices. Elles sont entrées en ligne de commande dans une fenêtre de type "terminal". Plétore de descriptions complètes se trouvent sur le web.

Edition de la ligne de commande :
Ctrl-a pour aller au début de la ligne ;
Ctrl-e pour aller à la fin de la ligne ;
Tab pour compléter automatiquement les noms de fichiers et de répertoire (interactivité si correspondances multiples).

Comment se déplacer dans l'arborescence ?
– pour aller dans le répertoire `chemin_d_acces`
  `cd chemin_d_acces`
– pour remonter dans l'arborescence :
  `cd ..`
– pour aller dans le répertoire de l'utilisateur `toto` :
  `cd ~toto`
– pour aller dans son propre répertoire (*home directory*)
  `cd`
  ou
  `cd ~`

Comment lister le contenu du répertoire courant ?
`ls`
ou
`ls -l` pour plus de renseignements
ou
`ls -al` pour encore plus de renseignements.

Comment lister le contenu d'un répertoire ?
`ls nom_du_repertoire`
ou
`ls -al nom_du_repertoire`

Comment lister les programmes C du répertoire courant ?
`ls *.c`
L'étoile remplace n'importe quelle suite de caractères.

Comment connaître le répertoire courant ?
`pwd`
pour Print Working Directory

Comment créer un repertoire ?

```
mkdir nom_du_nouveau_repertoire
```

Comment effacer un répertoire ?
`rmdir nom_du_répertoire`
après avoir vidé le répertoire de tout son contenu par la commande dangereuse
`rm * *.*`

Comment afficher le contenu d'un fichier ?
`cat nom_fichier`
ou
`more nom_fichier`
`more` permet des traitements plus élaborés (recherche de motif –: `/motif`), répétition (`:n`) ...– et reconnait les expressions régulières.

Comment copier un fichier ?
`cp nom_fichier_source nom_fichier_cible`
ou
`cp nom_fichier_source nom_du_repertoire_cible`

Comment déplacer un fichier ?
`mv nom_fichier_source nom_fichier_cible`
ou
`mv nom_fichier_source nom_du_repertoire_cible`

Comment déplacer plusieurs fichiers ?
`mv nom_fichier_source_1 nom_fichier_source_2 ...nom_du_repertoire_cible`
L'étoile * peut être utilisée pour déplacer tous les fichiers du répertoire source (répertoire courant ou autre) vers le répertoire cible.

Comment effacer un fichier ?
`rm nom_fichier`
Rappelons la commande dangereuse pour effacer tout le contenu du répertoire courant.
`rm * *.*`

Comment renommer un fichier ou un répertoire ?
`mv ancien_nom nouveau_nom`

Comment connaître les options d'une commande ? Comment obtenir de l'aide sur une commande ?
`man nom_commande`
et votre ami
`man gcc`

Chapitre 26

Description de l'enseignement

Licence de Sciences et Technologies
S3 et S4 – mentions Sciences de l'Ingénieur

Université de Perpignan Via Domitia

Programmation en C

Présentation de l'enseignement

1 Organisation

Année : 2010-2011

Organisation des cours : 10 séances de 1h30 par semestre

Organisation des travaux dirigés : 10 séances de 1h30. Des feuilles de travaux dirigés sont disponibles sur l'ENT.

Enseignant des cours : Ph. Langlois

Enseignant des travaux dirigés : Ph. Langlois

Supports de TD et examens des années précédentes, corrections associées : Disponibles sur l'ENT de l'UPVD

2 Pré-requis

Types et structures de données (vu au S2).

3 Contrôle de connaissances

Cet enseignement comprend des séances de TD sur ordinateur. La note finale comprend les évaluations d'un partiel et d'un examen final réalisés individuellement sur ordinateur (voir modalités de contrôle des connaissances).

4 Plan du semestre S3

Introduction. Fondements de tout langage de programmation impératif. Etat d'esprit de leur implémentation dans le langage C.

Types prédéfinis. Entiers, caractères, flottants, booléens.

Expressions. Variables, opérateurs, expressions.

Entrées-sorties ... de base.

Structures de contrôle. Répéter, choisir.

Tableaux. Déclaration, initialisation, parcours.

Sous-programmes. Prototype, corps, appel.

5 Bibliographie

Il existe beaucoup de polycopiés disponibles sur le ouaib ; certains sont très biens. Des ouvrages classiques sont disponibles à la BU ; nous mentionnons en particulier.

1. A. Braquelaire. *Méthodologie de la programmation en C*. 4ème édition. Dunod. 2005.
2. S. Varrette et N. Bernard. *Programmation avancée en C*. Hermes-Lavoisier. 2007.