Exercices de TD

1. Analyse d'énoncés

Règle 1. Toujours commencer par lire entièrement l'énoncé.
Règle 2. Identifier données et résultats.
Règle 3. Identifier les actions qui permettent de passer des données aux résultats.

Exercice 1.1
Considérons la formule « $V = \frac{4}{3}\pi R^3$ ».

1) Que signifie cette formule ?

2) Précisez les opérateurs figurant dans cette formule.

3) Précisez les constantes et les variables apparaissant dans cette formule. Pour chaque opérande, précisez sa nature et sa valeur si elle est définie.

Exercice 1.2
Considérons l'équation « $x^2 + 2ax - 3a^2 = 0$ ».

1) Calculez les solutions de cette équation.

2) Précisez les opérateurs figurant dans cette expression.

3) Précisez les constantes et les variables apparaissant dans l'expression. Pour chaque opérande, précisez sa nature et sa valeur si elle est définie.

2. Variables, types, expressions, instruction d'affectation

Exercice 2.1
Rappelez :

1) la règle de formation des identificateurs ;

2) la règle de formation des constantes littérales ;

3) l'ordre de priorité des opérateurs.

Exercice 2.2
Considérez les suites de caractères ci-après. Quelles sont celles qui sont valides en C ? Précisez la nature de celles qui sont valides : caractère, entier, réel, chaine...

```
115.2    65    'f'        'f"    "f"    0    '0'      0.0
0.       .0    3.14159    -88    88-    '2   1.5e-1   "chaine␣valide"
```

Exercice 2.3
Quels sont les identificateurs de variables valides parmi les suites de caractères suivantes ?

```
var           k2            2var          j       int
ma␣variable   ma-variable   ma_variable   chaîne  l1i
```

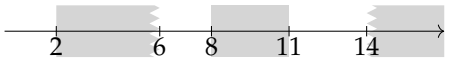
Exercice 2.4
Lorsqu'elles sont valides en C, quelle est la nature et la valeur des expressions suivantes ?

```
1.0          -1.0            1.
6 + 7 / 3    (6 + 7.0) / 3   6 + 7.0 / 3 - 3.14159
6.3 % 2      1 + 0.0         -4.5e1 + (10 + 7.2)
```

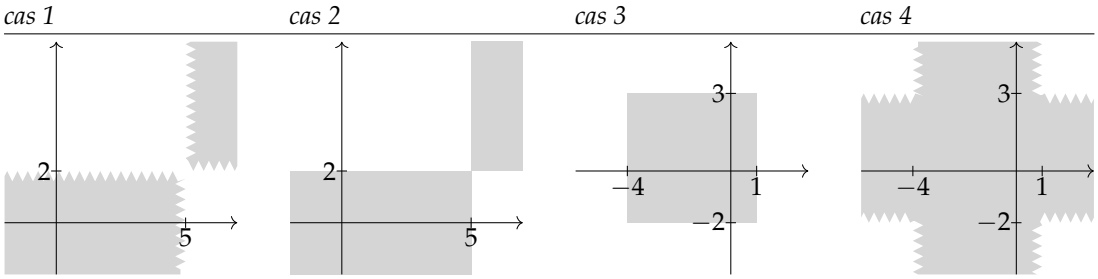
Exercice 2.5

Dans les figures qui suivent, des « régions » de la droite affine et du plan sont grisées. Si elles ne coïncident pas avec les bords des figures, les « frontières » de ces régions veulent signifier un *fermé* lorsqu'elles sont nettes et un *ouvert* lorsqu'elles sont en zigzag.

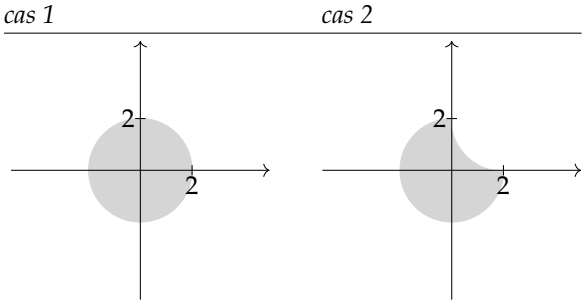
1) Donnez une expression booléenne C caractérisant l'appartenance d'un point d'abscisse x à la région grisée :



2) Même chose, mais dans le plan, pour un point d'abscisse x et d'ordonnée y, dans chacun des quatre cas :



3) Même chose pour les figures suivantes (dans la seconde figure, on a enlevé la partie du disque de centre O appartenant à un cercle de centre $(2, 2)$ et de rayon 2).



Exercice 2.6

Soit x, y et z trois variables. Exprimez sous la forme de deux expressions booléennes, la première en langage mathématique et la seconde en langage C, les propositions suivantes :

1) les valeurs de x, y et z sont dans l'ordre croissant ;

2) les valeurs de x, y et z sont toutes strictement supérieures à zéro ;

3) l'une au moins des valeurs de x, y et z est supérieure ou égale à zéro ;

4) les valeurs de x, y et z sont toutes les trois différentes.

Exercice 2.7

Les sensations « j'ai faim », « j'ai soif » et « j'ai sommeil » sont les seules que je puisse éprouver. Traduisez chacun des évènements suivants sous forme d'une expression booléenne :

1) « je n'éprouve que la faim » ;

2) « j'ai faim et soif, mais pas sommeil » ;

3) « j'éprouve toutes les sensations possibles » ;

4) « j'éprouve au moins une sensation » ;

5) « j'éprouve au moins deux sensations » ;

6) « je n'éprouve qu'une seule sensation » ;

7) « j'éprouve exactement deux sensations » ;

8) « je n'éprouve aucune sensation » ;

9) « je n'éprouve pas plus que deux sensations ».

Pour faire court, vous noterez m, b, d, pour « manger », « boire », « dormir », les trois sensations.

Exercice 2.8

Pour chacun des deux blocs d'instructions proposez des déclarations adaptées et donnez leur trace d'exécution.

bloc 1

```
x = 5;
n = 3;
--x;
x = -2;
```

bloc 2

```
x = 4.5;
z = 3.0 / 2;
y = -x;
++y;
```

Exercice 2.9

Soient `x` et `y` deux variables de type `int`. Considérons la suite d'instructions :

```
1 x = x + y;
2 y = x - y;
3 x = x - y;
```

1) On suppose que les valeurs des variables x et y avant l'exécution de la suite d'instructions sont respectivement 5 et 12. Donnez la trace d'exécution de cette suite. Vous en déduirez les valeurs de x et y en fin d'exécution.

2) Pouvez-vous généraliser ?

Exercice 2.10

On considère la suite de Fibonacci donnée par la récurrence

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \text{ si } n \geqslant 2 \end{cases}$$

Supposons que l'on ait deux variables entières : `U` contenant la valeur de F_n (pour un n donné) et `V` contenant la valeur de F_{n-1}. Complétez les instructions d'affectations pour que les valeurs des variables `U` et `V` évoluent comme dans la trace d'éxecution suivante (les valeurs des variables sont encadrées) :

$\boxed{F_n}$	$\boxed{F_{n-1}}$
U	V
U=?	
$\boxed{F_{n+1}}$	$\boxed{F_{n-1}}$
U	V
V=?	
$\boxed{F_{n+1}}$	$\boxed{F_n}$
U	V

Exercice 2.11

Donnez une suite d'instructions qui permet d'échanger de manière sûre et certaine les valeurs de deux variables de même type.

3. Entrée et sortie

Exercice 3.1

1) Donnez une instruction permettant d'affecter une valeur lue sur l'entrée :

- **a)** à la variable `c` de type `char` ;
- **b)** à la variable `n` de type `int` ;
- **c)** à la variable `x` de type `double`.

2) Donnez une instruction permettant d'afficher sur la sortie la valeur :

- **a)** de la variable `c` de type `char` ;
- **b)** de la variable `n` de type `int` ;
- **c)** de la variable `x` de type `double` au format virgule fixe avec quatre chiffres après la virgule ;
- **d)** de la variable `x` de type `double` au format virgule flottante.

Exercice 3.2

Le programme `aff_entrees.c` qui figure ci-après reçoit en entrée la suite de caractères :

```
0.3141593e1, -131
```

Qu'affiche-t-il ?

aff_entrees.c

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

int main(void) {
  double f;
  assert(scanf("%lf", &f) == 1);

  char c;
  int n;
  assert(scanf("%c%d", &c, &n) == 2);

  printf("%s : %8.4lf%c%d\n", "Vous avez entré", f, c, n);

  return EXIT_SUCCESS;
}
```

aff_entrees.c

Exercice 3.3

Écrivez un programme qui lit trois valeurs flottantes sur l'entrée et qui affiche leur moyenne sur la sortie.

Exercice 3.4

Écrivez un programme qui affiche sur la sortie la valeur de la fonction polynôme « $p(x) = 3x^2 - 5x + 1$ » pour toute valeur flottante x lue sur l'entrée.

4. Fonctions

Prototypes et déclarations des fonctions

Exercice 4.1

1) Expliquez les prototypes des fonctions suivantes (on ne demande pas de décrire ce que font les fonctions) :

- `double fmod(double x, double y);`
- `void srand(unsigned int seed);`
- `int rand(void);`

Parmi ces fonctions, lesquelles nécessitent l'utilisation du mot clef `return` dans l'écriture du corps ?

2) Écrire le prototype des fonctions correspondant aux énoncés suivants (on ne demande pas d'écrire le corps des fonctions) :
- une fonction calculant la valeur absolue d'un entier long ;
- une fonction qui renvoie une valeur de type `clock_t` correspondant à la durée écoulée depuis le début de l'exécution du programme ;
- une fonction qui indique si l'entier passé en paramètre est pair ou impair ;
- une fonction renvoyant la moyenne de deux réels passés en paramètre.

Exercice 4.2

1) Expliquez les prototypes des fonctions suivantes (on ne demande pas de décrire ce que font les fonctions) :
- `int concat(char *dest, char *src);`
- `void move(char *dest,  char *src, size_t n);`
- `char *int2str(int n);`
- `long int str2l(char *nptr, char **endptr, int base);`

Quelle est la signification et l'utilité des caractères `*` placés après les types dans les paramètres des fonctions ? En particulier, expliquez `char **endptr`.

2) Écrire les prototype des fonctions correspondants aux énoncés suivants (on ne demande pas d'écrire le corps des fonctions) :
- Une fonction permettant de convertir une durée en seconde en heure, minute, seconde.
- Une fonction qui incrémente le contenu d'une variable de type `int`.
- Une fonction qui échange le contenu de deux variables de type `double`.
- Une fonction renvoyant le quotient de deux entiers passés en paramètre et stockant à l'adresse passée en troisième paramètre le reste de la division des deux entiers.

Exercice 4.3
Écrire les déclarations des fonctions de la liste ci-dessous sous la forme suivante :

```
/* nom_de_la_fonction : description littérale
 * Entrées: liste des paramètres avec leurs types
 * Sortie: type de la fonction
 * AE: assertion d'entrée
 * AS: assertion de sortie
 */
prototype de la fonction
```

1) La fonction `isdigit` qui reçoit en paramètre un caractère et renvoie une valeur de type `bool` valant `true` si le caractère désigne un chiffre et `false` sinon.

2) La fonction `makenumber` qui reçoit en paramètre deux chiffres (de type `int`) et renvoie l'entier de type `int` obtenu avec ces deux chiffres.
Par exemple, si les chiffres sont `2` et `3`, alors on retourne le nombre `23`.

3) La fonction `time2hms` prenant en paramètre un entier `t` de type `int` ainsi que trois pointeurs vers des `int`. Le premier paramètre de cette fonction contient une durée en seconde. La fonction convertit la durée contenue dans `t` en heure, minute, seconde et stocke ces trois valeurs aux adresses des trois paramètres de type pointeur. Elle renvoie la valeur `-1` si la durée est supérieure à un jour (plus de 86400 secondes) et 0 sinon.

Traces des fonctions

Exercice 4.4
Donner la trace d'exécution (instructions exécutées, valeurs des expressions et valeurs contenues dans les variables) du programme suivant :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 /* carre:
```

```
 * calcule et renvoie le carré d'un entier passé en paramètre
 * Entrees: un entier x de type int.
 * Sortie: int
 * AE: Aucune
 * AS: carre == x * x
 */
int carre(int x);

int main(void) {
  int a = 2;
  printf("%d^2 = %d\n", a, carre(a));
  ++a;
  printf("%d^2 = %d\n", a, carre(a));

  return EXIT_SUCCESS;
}

int carre(int x) {
  return x * x;
}
```

Exercice 4.5

Donner la trace d'exécution du programme suivant :

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

/* mille:
 * fonction mystere
 * Entrees: un entier de type int
 * Sortie: void
 * AE: Aucune
 * AS: aucune
 */
void mille(int a);

int main(void) {
  int a = 2;
  printf("%d\n", a);
  mille(a);
  printf("%d\n", a);
  return EXIT_SUCCESS;
}

void mille(int a) {
  printf("%d\n", a);
  a = 1000;
  printf("%d\n", a);
}
```

Exercice 4.6

Exécuter le programme suivant et donner la valeur de chaque variable manipulée après chaque instruction :

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

```

```c
/* incremente:
 * fonction mystere
 * Entrees: un entier de type int et un pointeur vers un int
 * Sortie: void
 * AE: Aucune
 * AS: mystere
 */
void incremente(int x, int *p);

int main(void) {
  int a = 0;
  int b = 0;
  incremente(a, &b);
  printf("%d␣%d\n", a, b);
  return EXIT_SUCCESS;
}

void incremente(int x, int *p) {
  x = x + 1;
  *p = *p + 1;
  printf("%d␣%d\n", x, *p);
}
```

Exercice 4.7

Exécuter le programme suivant et donner la valeur de chaque variable manipulée après chaque instruction :

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

/* modifier:
 * fonction mystere
 * Entrees: un entier de type int et deux pointeur vers des int
 * Sortie: void
 * AE: Aucune
 * AS: mystere
 */
void modifier(int x, int *y, int *z);

int main(void) {
  int a = 3;
  int b = -8;
  int c = 12;
  printf("%d␣%d␣%d\n", a, b, c);
  modifier(a, &b, &c);
  printf("%d␣%d␣%d\n", a, b, c);
  return EXIT_SUCCESS;
}

void modifier(int x, int *y, int *z) {
  int t = x;
  int *v = z;
  x = *y;
  *y = *v;
  *z = t;
}
```

Écriture d'une fonction

Exercice 4.8

Écrire un programme comprenant :

1) une fonction `moyenne` prenant en paramètre deux réels et renvoyant leur moyenne,

2) une fonction `main` qui demande à l'utilisateur deux valeurs réelles, puis calcule et affiche leur moyenne.

Exercice 4.9

Écrivez une fonction qui ajoute 3 à un entier donné en paramètre et qui ne renvoie rien.

Exercice 4.10

Écrivez une fonction permet d'échanger le contenu de deux variables de type `element` (on ne précise pas ce qu'est le type `element`).

Exercice 4.11

Le but de cet exercice est d'écrire une fonction qui prend en paramètre une somme en euros et qui renvoie le nombre minimal de pièces de monnaie necessaire pour payer cette somme.

1) On suppose que le prototype de la fonction est

```
int nb_pieces_mini(double prix);
```

Pourquoi avoir choisi un paramètre qui n'est pas d'un type entier ? Quelles doivent être les assertions d'entrées de la fonction ?

2) Pour réaliser l'opération, on convertit d'abord la somme en centimes. Donnez une expression dont l'évaluation est une valeur de type `int` qui soit la somme contenue dans `prix` en centimes.

3) La première étape consistera donc à stocker la somme en centime dans une variable `s` de type `int`. Est-ce que cette stratégie modifie les assertions d'entrée ?

4) En utilisant la fonction `assert`, écrire une instruction ou une séquence d'instructions permettant de quitter le programme si les assertions d'entrée ne sont pas satisfaites.

5) Donnez une expression permettant de calculer le nombre maximal de pièces de deux euros nécessaires pour approcher la somme contenue dans `s` sans la dépasser.

6) Si on suppose que l'on a utilisé le maximum possible de pièces de deux euros, donnez une expression permettant de calculer le nombre maximal de pièces de un euro nécessaires pour approcher la somme restante (après déduction des pièces de deux euros déjà utilisées) sans la dépasser.

7) La stratégie pour obtenir le nombre minimal de pièces consiste à rendre le maximum possible de pièces de plus grande valeur. Donnez une séquence d'instructions permettant d'obtenir le nombre de pièces minimal et de le stocker dans une variable `n` de type `int`.

8) En déduire un programme complet comprenant la fonction de conversion et une fonction `main` permettant de la tester.

5. Conditionnelles et alternatives

Exercice 5.1

1) Soient `x` et `y` deux variables du type `int`. Dessinez l'arbre des tests du code suivant

```
1 if (x <= 2 && x >= -2) {
2   y = y + 3;
3 } else {
4   if (x == 10) {
5     y = 100;
6   } else {
7     y = y + x;
8   }
9 }
```

2) Quelles sont les valeurs des variables après l'exécution de ce code lorsque les valeurs initiales sont

a) `x` $= 2$ et `y` $= 12$.

b) `x` $= 123$ et `y` $= -16$?

c) `x` $= 10$ et `y` $= 200$?

Exercice 5.2

1) Écrivez une fonction qui prend en paramètres deux entiers et qui renvoie le maximum de ces deux entiers.

2) Écrivez ensuite un programme qui utilise la fonction précédement écrite pour calculer le maximum de trois entiers donnés en entrée.

Exercice 5.3

Le but de cet exercice est d'écrire un programme qui, pour une séquence de trois caractères donnés en entrée, calcule puis affiche le nombre de caractères distincts dans cette séquence.

1) Dessinez un arbre de test permettant de résoudre ce problème.

2) Si l'arbre de test possède plusieurs fois la même feuille, simplifiez le pour qu'aucune feuille n'apparaisse plusieurs fois.

3) Écrire le programme correspondant à l'une des solutions. Contrainte : vous devrez écrire une fonction qui réalisera le calcul dans laquelle vous ne devrez jamais utiliser le mot clef `else`.

Exercice 5.4

Une bibliothèque est ouverte de 9 h à 13 h et de 14 h à 18 h, sauf le samedi matin et le dimanche toute la journée.

1) Nous supposerons que les jours sont codés par des entiers : Lundi$\Rightarrow$ 1, Mardi$\Rightarrow$ 2, . . . , Dimanche$\Rightarrow$ 7. On supposera que le numéro du jour est stocké dans une variable `j` et que l'heure est stockée dans une variable `h`. Ces deux variables sont de type entier. Donnez une expression booléenne s'évaluant en `true` si la bibliothèque est ouverte et en `false` sinon.

2) En utilisant un opérateur de condition `...?...:...`, écrire une instruction affichant `"Ouverte"` si les variables `j` et `h` correspondent à un des horaires d'ouverture et `"Fermée"` sinon. Écrivez un programme qui affiche si la bibliothèque est ouverte ou fermée au jour et à l'heure donnés par l'utilisateur.

Exercice 5.5

Dans cet exercice, nous nous intéressons aux ordinaux abrégés en anglais, où le nombre est écrit en chiffres. Les premiers sont 1st, 2nd, 3rd, 4th, etc. (abréviations de firtst, second, third et fourth...). Pour déterminer le suffixe, on regarde le dernier chiffre du nombre : si c'est 1 on ajoute le suffixe -st, si c'est 2 le suffixe -nd, si c'est 3 le suffixe est -rd, sinon le suffixe est -th. Il y a une exception : si l'avant dernier chiffre du nombre est 1, le suffixe est toujours -th.

1) On supposera que le nombre est positif et stocké dans une variable `n` de type `int`. Donnez deux expressions, l'une donnant le chiffre des unités et l'autre le chiffre des dizaines de l'entier stocké dans `n`.

2) Dessinez l'arbre de tests correspondant au problème.

3) Écrire le code correspondant à cet arbre.

Exercice 5.6

Le but de cet exercice est d'écrire un programme demandant trois entiers à l'utilisateur et affichant ces entiers triés dans l'ordre croissant. On supposera que ces entiers sont stockés dans les variables `a`, `b` et `c` de type int :

1) Combien il y a-t-il de possibilités ? Dessinez l'arbre des tests de ce problème.

2) Afin d'éviter tous ces tests en cascade, nous allons procéder d'une façon un peu différente :
- Si la valeur de `a` est plus grande que celle de `b` alors on échange les valeurs de `a` et de `b`.
- Si la valeur de `b` est plus grande que celle de `c` alors on échange les valeurs de `b` et de `c`.
- Si la valeur de `a` est plus grande que celle de `b` alors on échange les valeurs de `a` et de `b`.

Vérifiez que dans tous les cas, cette méthode trie les trois valeurs dans l'ordre croissant.

3) Écrire une fonction de prototype

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

qui échange les valeurs situées aux adresses `pa` et `pb` si celle située en `pa` est plus grande que celle située en `pb`

4) Utilisez cette fonction pour produire un programme qui demande à l'utilisateur de saisir trois valeurs entières et qui les affiche dans l'ordre croissant. On fera bien attention à vérifier que la saisie s'est bien passée avant de poursuivre et à quitter le programme si celle-ci s'est mal passée.

Exercice 5.7

Analyse d'un énoncé

On considère un programme constitué de deux fonctions (une auxiliaire et une principale).
La fonction auxiliaire prend en paramètre deux entiers `a` et `b`, un caractère `c` et deux pointeurs vers des entiers `presult` et `paux`.
Si `c` vaut `'+'`, `'-'` ou `'*'` alors la fonction calcule le résultat de l'opération correspondante et le stocke à l'adresse `presult`.
Si `c` vaut `'/'` alors il y a deux cas de figure. Soit `b` est non nul et dans ce cas la fonction stocke le quotient de `a` par `b` à l'adresse `presult` et le reste de la division de `a` par `b` à l'adresse `paux`. Soit `b` est nul et dans ce cas, la variable globale `errno` (déclarée dans le fichier d'en-tête `errno.h`) prend la valeur `EINVAL`.
Si `c` a une autre valeur que celles évoquées ci-dessus, alors la variable `errno` prend la valeur `EDOM`.
Dans tous les cas la fonction renvoie le nombre de valeurs effectivement modifiées parmi `*presult` et `*paux`.

La fonction principale initialise la variable globale `errno` à `0`. Elle demande à l'utilisateur d'entrer deux valeurs entières et une opération sous le format $v_1\ op\ v_2$, où v_1 et v_2 désignent les valeurs entières et op désigne l'opération choisie. Il affiche alors les valeurs des variables modifiées par l'appel à la fonction auxiliaire ou un éventuel message d'erreur si la variable `errno` a été modifiée.

1) Écrire le squelette du programme avec les deux fonctions (sans les corps). Donnez un nom soigneusement choisi à la fonction auxiliaire. Le programme doit être compilable.

2) Écrire le corps de la fonction auxiliaire en respectant les spécifications.

3) Écrire le corps de la fonction principale en respectant les spécifications. Discutez éventuellement des ambiguités de l'énoncé.

4) Donner un exemple d'utilisation avec redirection des entrées/sorties vers des fichiers.

6. Itératives

Exercice 6.1

Que font chacune des boucles suivantes ?

```
for (int k = 0 ; k < 10 ; ++k) {
  printf("%d\t", k);
}
```

```
for (int k = 10 ; k > 0 ; --k) {
  if (k % 2 == 0) {
    printf("%d\n", k);
  }
}
```

```
int k = 0;
while (k < 10) {
  printf("%d\n", 2 * k);
  ++k;
}
```

Exercices de TD

1. Analyse d'énoncés

Règle 1. Toujours commencer par lire entièrement l'énoncé.
Règle 2. Identifier données et résultats.
Règle 3. Identifier les actions qui permettent de passer des données aux résultats.

Exercice 1.1
Considérons la formule « $V = \frac{4}{3}\pi R^3$ ».

1) Que signifie cette formule ?

2) Précisez les opérateurs figurant dans cette formule.

3) Précisez les constantes et les variables apparaissant dans cette formule. Pour chaque opérande, précisez sa nature et sa valeur si elle est définie.

Exercice 1.2
Considérons l'équation « $x^2 + 2ax - 3a^2 = 0$ ».

1) Calculez les solutions de cette équation.

2) Précisez les opérateurs figurant dans cette expression.

3) Précisez les constantes et les variables apparaissant dans l'expression. Pour chaque opérande, précisez sa nature et sa valeur si elle est définie.

2. Variables, types, expressions, instruction d'affectation

Exercice 2.1
Rappelez :

1) la règle de formation des identificateurs ;

2) la règle de formation des constantes littérales ;

3) l'ordre de priorité des opérateurs.

Exercice 2.2
Considérez les suites de caractères ci-après. Quelles sont celles qui sont valides en C ? Précisez la nature de celles qui sont valides : caractère, entier, réel, chaine...

```
115.2    65    'f'      'f"    "f"    0    '0'      0.0
0.       .0    3.14159  -88    88-    '2   1.5e-1   "chaine␣valide"
```

Exercice 2.3
Quels sont les identificateurs de variables valides parmi les suites de caractères suivantes ?

```
var          k2           2var         j       int
ma␣variable  ma-variable  ma_variable  chaîne  l1i
```

Exercice 2.4
Lorsqu'elles sont valides en C, quelle est la nature et la valeur des expressions suivantes ?

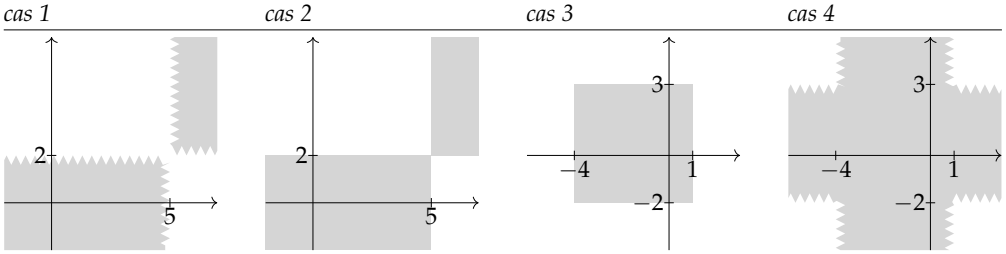
```
1.0          -1.0           1.
6 + 7 / 3    (6 + 7.0) / 3  6 + 7.0 / 3 - 3.14159
6.3 % 2      1 + 0.0        -4.5e1 + (10 + 7.2)
```

Exercice 2.5

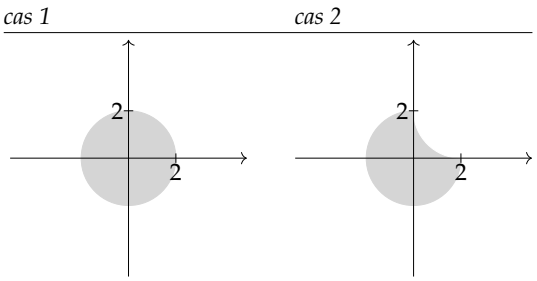
Dans les figures qui suivent, des « régions » de la droite affine et du plan sont grisées. Si elles ne coïncident pas avec les bords des figures, les « frontières » de ces régions veulent signifier un *fermé* lorsqu'elles sont nettes et un *ouvert* lorsqu'elles sont en zigzag.

1) Donnez une expression booléenne C caractérisant l'appartenance d'un point d'abscisse x à la région grisée :

2) Même chose, mais dans le plan, pour un point d'abscisse x et d'ordonnée y, dans chacun des quatre cas :



3) Même chose pour les figures suivantes (dans la seconde figure, on a enlevé la partie du disque de centre O appartenant à un cercle de centre $(2, 2)$ et de rayon 2).



Exercice 2.6

Soit x, y et z trois variables. Exprimez sous la forme de deux expressions booléennes, la première en langage mathématique et la seconde en langage C, les propositions suivantes :

1) les valeurs de x, y et z sont dans l'ordre croissant ;

2) les valeurs de x, y et z sont toutes strictement supérieures à zéro ;

3) l'une au moins des valeurs de x, y et z est supérieure ou égale à zéro ;

4) les valeurs de x, y et z sont toutes les trois différentes.

Exercice 2.7

Les sensations « j'ai faim », « j'ai soif » et « j'ai sommeil » sont les seules que je puisse éprouver. Traduisez chacun des évènements suivants sous forme d'une expression booléenne :

1) « je n'éprouve que la faim » ;

2) « j'ai faim et soif, mais pas sommeil » ;

3) « j'éprouve toutes les sensations possibles » ;

4) « j'éprouve au moins une sensation » ;

5) « j'éprouve au moins deux sensations » ;

6) « je n'éprouve qu'une seule sensation » ;

7) « j'éprouve exactement deux sensations » ;

8) « je n'éprouve aucune sensation » ;

9) « je n'éprouve pas plus que deux sensations ».

Pour faire court, vous noterez m, b, d, pour « manger », « boire », « dormir », les trois sensations.

Exercice 2.8

Pour chacun des deux blocs d'instructions proposez des déclarations adaptées et donnez leur trace d'exécution.

bloc 1

```
x = 5;
n = 3;
--x;
x = -2;
```

bloc 2

```
x = 4.5;
z = 3.0 / 2;
y = -x;
++y;
```

Exercice 2.9

Soient `x` et `y` deux variables de type `int`. Considérons la suite d'instructions :

```
1 x = x + y;
2 y = x - y;
3 x = x - y;
```

1) On suppose que les valeurs des variables x et y avant l'exécution de la suite d'instructions sont respectivement 5 et 12. Donnez la trace d'exécution de cette suite. Vous en déduirez les valeurs de x et y en fin d'exécution.

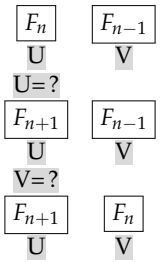
2) Pouvez-vous généraliser ?

Exercice 2.10

On considère la suite de Fibonacci donnée par la récurrence

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \text{ si } n \geqslant 2 \end{cases}$$

Supposons que l'on ait deux variables entières : `U` contenant la valeur de F_n (pour un n donné) et `V` contenant la valeur de F_{n-1}. Complétez les instructions d'affectations pour que les valeurs des variables `U` et `V` évoluent comme dans la trace d'éxecution suivante (les valeurs des variables sont encadrées) :



Exercice 2.11

Donnez une suite d'instructions qui permet d'échanger de manière sûre et certaine les valeurs de deux variables de même type.

3. Entrée et sortie

Exercice 3.1

1) Donnez une instruction permettant d'affecter une valeur lue sur l'entrée :
- **a)** à la variable `c` de type `char` ;
- **b)** à la variable `n` de type `int` ;
- **c)** à la variable `x` de type `double`.

2) Donnez une instruction permettant d'afficher sur la sortie la valeur :
- **a)** de la variable `c` de type `char` ;
- **b)** de la variable `n` de type `int` ;
- **c)** de la variable `x` de type `double` au format virgule fixe avec quatre chiffres après la virgule ;
- **d)** de la variable `x` de type `double` au format virgule flottante.

Exercice 3.2

Le programme `aff_entrees.c` qui figure ci-après reçoit en entrée la suite de caractères :

```
0.3141593e1, -131
```

Qu'affiche-t-il ?

aff_entrees.c

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

int main(void) {
  double f;
  assert(scanf("%lf", &f) == 1);

  char c;
  int n;
  assert(scanf("%c%d", &c, &n) == 2);

  printf("%s : %8.4lf%c%d\n", "Vous avez entré", f, c, n);

  return EXIT_SUCCESS;
}
```

aff_entrees.c

Exercice 3.3

Écrivez un programme qui lit trois valeurs flottantes sur l'entrée et qui affiche leur moyenne sur la sortie.

Exercice 3.4

Écrivez un programme qui affiche sur la sortie la valeur de la fonction polynôme « $p(x) = 3x^2 - 5x + 1$ » pour toute valeur flottante x lue sur l'entrée.

4. Fonctions

Prototypes et déclarations de sfonctions

Exercice 4.1

1) Expliquez les prototypes des fonctions suivantes (on ne demande pas de décrire ce que font les fonctions) :
- `double fmod(double x, double y);`
- `void srand(unsigned int seed);`
- `int rand(void);`

Parmi ces fonctions, lesquelles nécessitent l'utilisation du mot clef `return` dans l'écriture du corps ?

2) Écrire le prototype des fonctions correspondant aux énoncés suivants (on ne demande pas d'écrire le corps des fonctions) :
- une fonction calculant la valeur absolue d'un entier long ;
- une fonction qui renvoie une valeur de type `clock_t` correspondant à la durée écoulée depuis le début de l'exécution du programme ;
- une fonction qui affiche si l'entier passé en paramètre est pair ou impair ;
- une fonction renvoyant la moyenne de deux réels passés en paramètre.

Exercice 4.2

1) Expliquez les prototype des fonctions suivantes (on ne demande pas de décrire ce que font les fonctions) :
- `int concat(char *dest, char *src);`
- `void move(char *dest, char *src, size_t n);`
- `char *int2str(int n);`
- `long str2l(char *nptr, char **endptr, int base);`

Quelle est la signification et l'utilité des caractères `*` placés après les types dans les paramètres des fonctions ? En particulier, expliquez `char **endptr`.

2) Écrire les prototype des fonctions correspondants aux énoncés suivants (on ne demande pas d'écrire le corps des fonctions) :
- Une fonction permettant de convertir une durée en seconde en heure, minute, seconde.
- Une fonction qui incrémente le contenu d'une variable de type `int`.
- Une fonction qui échange le contenu de deux variables de type `float`.
- Une fonction renvoyant le quotient de deux entiers passés en paramètre et stockant à l'adresse passée en troisième paramètre le reste de la division des deux entiers.

Exercice 4.3
Écrire les déclarations des fonctions de la liste ci-dessous sous la forme suivante :

```
/* nom_de_la_fonction : description littérale
 * Entrées: liste des paramètres avec leurs types
 * Sortie: type de la fonction
 * AE: assertion d'entrée
 * AS: assertion de sortie
 */
prototype de la fonction
```

1) La fonction `isdigit` qui reçoit en paramètre un caractère et renvoie une valeur de type `bool` valant `true` si le caractère désigne un chiffre et `false` sinon.

2) La fonction `makenumber` qui reçoit en paramètre deux chiffres (de type `int`) et renvoie l'entier de type `int` obtenu avec ces deux chiffres.
Par exemple, si les chiffres sont `2` et `3`, alors on retourne le nombre `23`.

3) La fonction `time2hms` prenant en paramètre un entier `t` de type `int` ainsi que trois pointeurs vers des `int`. Le premier paramètre de cette fonction contient une durée en seconde. La fonction convertit la durée contenue dans `t` en heure, minute, seconde et stocke ces trois valeurs aux adresses des trois paramètres de type pointeur. Elle renvoie la valeur `-1` si la durée est supérieure à un jour (plus de 86400 secondes) et 0 sinon.

Traces des fonctions

Exercice 4.4
Donner la trace d'exécution (instructions exécutées, valeurs des expressions et valeurs contenues dans les variables) du programme suivant :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 /* carre:
```

```
 * calcule et renvoie le carré d'un entier passé en paramètre
 * Entrees: un entier x de type int.
 * Sortie: int
 * AE: Aucune
 * AS: carre == x * x
 */
int carre(int x);

int main(void) {
  int a = 2;
  printf("%d^2 = %d\n", a, carre(a));
  ++a;
  printf("%d^2 = %d\n", a, carre(a));

  return EXIT_SUCCESS;
}

int carre(int x) {
  return x * x;
}
```

Exercice 4.5

Donner la trace d'exécution du programme suivant :

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

/* mille:
 * fonction mystere
 * Entrees: un entier de type int
 * Sortie: void
 * AE: Aucune
 * AS: aucune
 */
void mille(int a);

int main(void) {
  int a = 2;
  printf("%d\n", a);
  mille(a);
  printf("%d\n", a);
  return EXIT_SUCCESS;
}

void mille(int a) {
  printf("%d\n", a);
  a = 1000;
  printf("%d\n", a);
}
```

Exercice 4.6

Exécuter le programme suivant et donner la valeur de chaque variable manipulée après chaque instruction :

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

```

```c
/* incremente:
 * fonction mystere
 * Entrees: un entier de type int et un pointeur vers un int
 * Sortie: void
 * AE: Aucune
 * AS: mystere
 */
void incremente(int x, int *p);

int main(void) {
  int a = 0;
  int b = 0;
  incremente(a, &b);
  printf("%d %d\n", a, b);
  return EXIT_SUCCESS;
}

void incremente(int x, int *p) {
  x = x + 1;
  *p = *p + 1;
  printf("%d %d\n", x, *p);
}
```

Exercice 4.7

Exécuter le programme suivant et donner la valeur de chaque variable manipulée après chaque instruction :

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

/* modifier:
 * fonction mystere
 * Entrees: un entier de type int et deux pointeur vers des int
 * Sortie: void
 * AE: Aucune
 * AS: mystere
 */
void modifier(int x, int *y, int *z);

int main(void) {
  int a = 3;
  int b = -8;
  int c = 12;
  printf("%d %d %d\n", a, b, c);
  modifier(a, &b, &c);
  printf("%d %d %d\n", a, b, c);
  return EXIT_SUCCESS;
}

void modifier(int x, int *y, int *z) {
  int t = x;
  int *v = z;
  x = *y;
  *y = *v;
  *z = t;
}
```

Écriture d'une fonction

Exercice 4.8

Écrire un programme comprenant :

1) une fonction `moyenne` prenant en paramètre deux réels et renvoyant leur moyenne,

2) une fonction `main` qui demande à l'utilisateur deux valeurs réelles, puis calcule et affiche leur moyenne.

Exercice 4.9

Écrivez une fonction qui ajoute 3 à un entier donné en paramètre et qui ne renvoie rien.

Exercice 4.10

Écrivez une fonction permet d'échanger le contenu de deux variables de type `element` (on ne précise pas ce qu'est le type `element`).

Exercice 4.11

Le but de cet exercice est d'écrire une fonction qui prend en paramètre une somme en euros et qui renvoie le nombre minimal de pièces de monnaie necessaire pour payer cette somme.

1) On suppose que le prototype de la fonction est

```
int nb_pieces_mini(double prix);
```

Pourquoi avoir choisi un paramètre qui n'est pas d'un type entier ? Quelles doivent être les assertions d'entrées de la fonction ?

2) Pour réaliser l'opération, on convertit d'abord la somme en centimes. Donnez une expression dont l'évaluation est une valeur de type `int` qui soit la somme contenue dans `prix` en centimes.

3) La première étape consistera donc à stocker la somme en centime dans une variable `s` de type `int`. Est-ce que cette stratégie modifie les assertions d'entrée ?

4) En utilisant la fonction `assert`, écrire une instruction ou une séquence d'instructions permettant de quitter le programme si les assertions d'entrée ne sont pas satisfaites.

5) Donnez une expression permettant de calculer le nombre maximal de pièces de deux euros nécessaires pour approcher la somme contenue dans `s` sans la dépasser.

6) Si on suppose que l'on a utilisé le maximum possible de pièces de deux euros, donnez une expression permettant de calculer le nombre maximal de pièces de un euro nécessaires pour approcher la somme restante (après déduction des pièces de deux euros déjà utilisées) sans la dépasser.

7) La stratégie pour obtenir le nombre minimal de pièces consiste à rendre le maximum possible de pièces de plus grande valeur. Donnez une séquence d'instructions permettant d'obtenir le nombre de pièces minimal et de le stocker dans une variable `n` de type `int`.

8) En déduire un programme complet comprenant la fonction de conversion et une fonction `main` permettant de la tester.

5. Conditionnelles et alternatives

Exercice 5.1

1) Soient `x` et `y` deux variables du type `int`. Dessinez l'arbre des tests du code suivant

```
1 if (x <= 2 && x >= -2) {
2   y = y + 3;
3 } else {
4   if (x == 10) {
5     y = 100;
6   } else {
7     y = y + x;
8   }
9 }
```

2) Quelles sont les valeurs des variables après l'exécution de ce code lorsque les valeurs initiales sont

a) `x` $= 2$ et `y` $= 12$.

b) `x` $= 123$ et `y` $= -16$?

c) `x` $= 10$ et `y` $= 200$?

Exercice 5.2

1) Écrivez une fonction qui prend en paramètres deux entiers et qui renvoie le maximum de ces deux entiers.

2) Écrivez ensuite un programme qui utilise la fonction précédemment écrite pour calculer le maximum de trois entiers donnés en entrée.

Exercice 5.3

Le but de cet exercice est d'écrire un programme qui, pour une séquence de trois caractères donnés en entrée, calcule puis affiche le nombre de caractères distincts dans cette séquence.

1) Dessinez un arbre de test permettant de résoudre ce problème.

2) Si l'arbre de test possède plusieurs fois la même feuille, simplifiez le pour qu'aucune feuille n'apparaisse plusieurs fois.

3) Écrire le programme correspondant à l'une des solutions. Contrainte : vous devrez écrire une fonction qui réalisera le calcul dans laquelle vous ne devrez jamais utiliser le mot clef `else`.

Exercice 5.4

Une bibliothèque est ouverte de 9 h à 13 h et de 14 h à 18 h, sauf le samedi matin et le dimanche toute la journée.

1) Nous supposerons que les jours sont codés par des entiers : Lundi⇒ 1, Mardi⇒ 2, . . . , Dimanche⇒ 7. On supposera que le numéro du jour est stocké dans une variable `j` et que l'heure est stockée dans une variable `h`. Ces deux variables sont de type entier. Donnez une expression booléenne s'évaluant en `true` si la bibliothèque est ouverte et en `false` sinon.

2) En utilisant un opérateur de condition `...?...:...`, écrire une instruction affichant `"Ouverte"` si les variables `j` et `h` correspondent à un des horaires d'ouverture et `Fermée` sinon. Écrivez un programme qui affiche si la bibliothèque est ouverte ou fermée au jour et à l'heure donnés par l'utilisateur.

Exercice 5.5

Dans cet exercice, nous nous intéressons aux ordinaux abrégés en anglais, où le nombre est écrit en chiffres. Les premiers sont 1st, 2nd, 3rd, 4th, etc. (abréviations de firtst, second, third et fourth...). Pour déterminer le suffixe, on regarde le dernier chiffre du nombre : si c'est 1 on ajoute le suffixe -st, si c'est 2 le suffixe -nd, si c'est 3 le suffixe est -rd, sinon le suffixe est -th. Il y a une exception : si l'avant dernier chiffre du nombre est 1, le suffixe est toujours -th.

1) On supposera que le nombre est positif et stocké dans une variable `n` de type `int`. Donnez deux expressions, l'une donnant le chiffre des unités et l'autre le chiffre des dizaines de l'entier stocké dans `n`.

2) Dessinez l'arbre de tests correspondant au problème.

3) Écrire le code correspondant à cet arbre.

Exercice 5.6

Le but de cet exercice est d'écrire un programme demandant trois entiers à l'utilisateur et affichant ces entiers triés dans l'ordre croissant. On supposera que ces entiers sont stockés dans les variables `a`, `b` et `c` de type int :

1) Combien il y a-t-il de possibilités ? Dessinez l'arbre des tests de ce problème.

2) Afin d'éviter tous ces tests en cascade, nous allons procéder d'une façon un peu différente :
- Si la valeur de `a` est plus grande que celle de `b` alors on échange les valeurs de `a` et de `b`.
- Si la valeur de `b` est plus grande que celle de `c` alors on échange les valeurs de `b` et de `c`.
- Si la valeur de `a` est plus grande que celle de `b` alors on échange les valeurs de `a` et de `b`.

Vérifiez que dans tous les cas, cette méthode trie les trois valeurs dans l'ordre croissant.

3) Écrire une fonction de prototype

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

qui échange les valeurs situées aux adresses `pa` et `pb` si celle située en `pa` est plus grande que celle située en `pb`

4) Utilisez cette fonction pour produire un programme qui demande à l'utilisateur de saisir trois valeurs entières et qui les affiche dans l'ordre croissant. On fera bien attention à vérifier que la saisie s'est bien passée avant de poursuivre et à quitter le programme si celle-ci s'est mal passée.

Exercice 5.7

Analyse d'un énoncé

On considère un programme constitué de deux fonctions (une auxiliaire et une principale.
La fonction auxiliaire prend en paramètre deux entiers `a` et `b`, un caractère `c` et deux pointeurs vers des entiers `presult` et `paux`.
Si `c` vaut `'+'`, `'-'` ou `'*'` alors la fonction calcule le résultat de l'opération correspondante et le stocke à l'adresse `presult`.
Si `c` vaut `'/'` alors il y a deux cas de figure. Soit `b` est non nul et dans ce cas la fonction stocke le quotient de `a` par `b` à l'adresse `presult` et le reste de la division de `a` par `b` à l'adresse `paux`. Soit `b` est nul et dans ce cas, la variable globale `errno` (de la bibliothèque `errno.h`) prend la valeur `EINVAL`.
Si `c` a une autre valeur que celles évoquées ci-dessus, alors la variable `errno` prend la valeur `EDOM`.
Dans tous les cas la fonction renvoie le nombre de valeurs effectivement modifiées parmi `*presult` et `*paux`.

La fonction principale initialise la variable globale `errno` à `0`. Elle demande à l'utilisateur d'entrer deux valeurs entières et une opération sous le format $v_1\ op\ v_2$, où v_1 et v_2 désignent les valeurs entières et op désigne l'opération choisie. Il affiche alors les valeurs des variables modifiées par l'appel à la fonction auxiliaire ou un éventuel message d'erreur si la variable `errno` a été modifiée.

1) Écrire le squelette du programme avec les deux fonctions (sans les corps) et la variable globale. Donnez un nom soigneusement choisi à la fonction auxiliaire. Le programme doit être compilable.

2) Écrire le corps de la fonction auxiliaire en respectant les spécifications.

3) Écrire le corps de la fonction principale en respectant les spécifications. Discutez éventuellement des ambiguités de l'énoncé.

4) Donner un exemple d'utilisation avec redirection des entrées/sorties vers des fichiers.

6. Itératives

Exercice 6.1

Que font chacune des boucles suivantes ?

```
for (int k = 0 ; k < 10 ; ++k) {
  printf("%d\t", k);
}
```

```
for (int k = 10 ; k > 0 ; --k) {
  if (k % 2 == 0) {
    printf("%d\n", k);
  }
}
```

```
int k = 0;
while (k < 10) {
  printf("%d\n", 2 * k);
  ++k;
}
```