Dr M. GUEDJ

TP de Programmation en C

TP 1 – Fonctions et appels de fonctions – Algorithmique

Dr M. Guedj

Implanter et tester les fonctions, de `float` $\longrightarrow$ `float`, ci-dessous.

1. $f(x) = x + 1$

```
------ tp1.c ------
#include<stdio.h>
#include <math.h>

float f(float x) {
    return x+1;
}

void main() {
    printf("%f \n", f(0));
}
```

2. $g(x) = 2x + 3$
3. $h(x) = x^2 + 1$
4. $i(x) = 5x^2 + 3x + 1$
5. $j(x) = 3x^3 + 5x^2 + 3x + 4$

```
------ tp1.c ------
(...)
float j(float x) {
    return 3*pow(x, 3) + 5*x*x + 3*x + 4;
}
(...)
```

6. $k(x) = x^{10} - x^3 + 3$
7. $l(x) = 3x^{10} + 4x^2 + 5$
8. $m(x) = 10x^{100} + 3x + 1$
9. $n(x) = a(x) + b(x)$ où $\begin{cases} a(x) = x + 3 \\ b(x) = x^2 - 2 \end{cases}$

10. $o(x) = c \circ d(x) = c(d(x))$ où $\begin{cases} c(x) = x + 1 \\ d(x) = x^3 \end{cases}$

11. $p(x) = \frac{3x^3+2}{20x^2+1}$

TP 2 – Conditions – Algorithmique

Dr M. Guedj

Implanter et tester les fonctions suivantes.

1. $f(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{sinon.} \end{cases}$

   où f : `float` $\longrightarrow$ `float`.

```
------ tp2.c ------
#include<stdio.h>
#include <math.h>

float f(float x) {
    if (x<0) return 0;
    else return 1;
}

void main() {
    printf("f(0)=%f \n", f(0));
}
```

2. $g(x) = \begin{cases} 'a' & \text{si} \quad x \leq 0 \\ 'b' & \text{sinon.} \end{cases}$

   où g : `float` $\longrightarrow$ `char`.

3. $h(c) = \begin{cases} 0 & \text{si} \quad c = 'a' \\ 1 & \text{si} \quad c = 'b' \\ 2 & \text{si} \quad c = 'c' \\ 3 & \text{sinon.} \end{cases}$

   où h : `char` $\longrightarrow$ `short`.

4. $i(x) = \begin{cases} 0 & \text{si} \quad x \leq 0 \\ 1 & \text{si} \quad 0 \leq x \leq 100 \\ 2 & \text{sinon.} \end{cases}$

   où i : `double` $\longrightarrow$ `short`.

5. $j(x) = \begin{cases} x^2 & \text{si} \quad -10 < x < 10 \\ x^3 & \text{si} \quad 50 < x < 60 \\ 1 & \text{sinon.} \end{cases}$

   où j : `double` $\longrightarrow$ `double`.

6. $k(n) = \begin{cases} \text{'}a\text{'} & \text{si} \quad n = 1 \\ \text{'}b\text{'} & \text{si} \quad n = 2 \\ \text{'}c\text{'} & \text{si} \quad n = 3 \\ \text{'\_'} & \text{sinon.} \end{cases}$

   où k : `char` $\longrightarrow$ `short`.

7. $l(x, y) = \begin{cases} 1 & \text{si} \quad x + y = 1 \\ 2 & \text{si} \quad x < y \\ 0 & \text{sinon.} \end{cases}$

   où l : `short` $\times$ `short` $\longrightarrow$ `short`.

8. $m(c) = \begin{cases} 1 & \text{si} \quad c = \text{'}h\text{'} \\ 2 & \text{si} \quad c = \text{'}b\text{'} \\ 3 & \text{si} \quad c = \text{'}d\text{'} \\ 4 & \text{si} \quad c = \text{'}g\text{'} \\ 5 & \text{sinon.} \end{cases}$

   où m : `char` $\longrightarrow$ `short`.

9. $n(x, y) = \begin{cases} 1 & \text{si} \quad x = y \\ 0 & \text{sinon.} \end{cases}$

   où n : `double` $\times$ `double` $\longrightarrow$ `short`.

10. $o(a, b) = \begin{cases} Vrai & \text{si} \quad a = b \\ Faux & \text{sinon.} \end{cases}$

    où o : `char` $\times$ `char` $\longrightarrow$ `short`.

11. $p(x, y, z) = \begin{cases} Vrai & \text{si} \quad x = y = z \\ Faux & \text{sinon.} \end{cases}$

    où p : `float` $\times$ `float` $\times$ `float` $\longrightarrow$ `short`.

12. $q(x,y,z) = \begin{cases} 1 & \text{si} \quad x > y \quad \text{et} \quad x > z \\ 2 & \text{si} \quad y > x \quad \text{et} \quad y > z \\ 3 & \text{si} \quad z > x \quad \text{et} \quad z > y \\ 0 & \text{sinon.} \end{cases}$

où $q : \texttt{int} \times \texttt{int} \times \texttt{int} \longrightarrow \texttt{short}$.

13. $r(x,y,z) = \begin{cases} \frac{y+z}{x} & \text{si} \quad x \neq 0 \\ \frac{x+z}{y} & \text{si} \quad y \neq 0 \\ \frac{x+y}{z} & \text{si} \quad z \neq 0 \\ 0 & \text{sinon.} \end{cases}$

où $r : \texttt{int} \times \texttt{int} \times \texttt{int} \longrightarrow \texttt{float}$.

14. $s(x,y,z) = \begin{cases} x^4 + \frac{x^3+z^5}{x} & \text{si} \quad x \neq 0 \\ y^4 + \frac{x^3+z^5}{y} & \text{si} \quad y \neq 0 \\ z^4 + \frac{x^3+z^5}{z} & \text{si} \quad z \neq 0 \\ 0 & \text{sinon.} \end{cases}$

où $s : \texttt{int} \times \texttt{int} \times \texttt{int} \longrightarrow \texttt{float}$.

TP 3 – Procédures – Algorithmique

Dr M. Guedj

Exercice 1.
Ecrire une procédure `afficher_arguments()` prenant trois arguments : un entier, un caractère et un flottant, et les affichant sur la console.

Correction.

```
------ tp3.c ------
#include <stdio.h>
#include <math.h>

void afficher_arguments(int a, char b, float c) {
    printf("%d %c %f \n", a, b, c);
}

void main() {
    afficher_arguments(1, 'a', 3.2);
}
```

Exercice 2.
Ecrire une procédure `egaux()` qui demande à l'utilisateur d'entrer deux chaines de caractères ch_1 et ch_2 ; et qui affiche "OK" si $ch_1 = ch_2$ et "KO" sinon.

Correction.

```
------ tp3.c ------
#include <stdio.h>
#include <math.h>
#include <string.h>

void afficher_arguments(int a, char b, float c) {
    printf("%d %c %f \n", a, b, c);
}

void egaux() {
    char ch1[30], ch2[30];
    printf("ch1 : ");
    scanf("%s", &ch1);
```

```
    printf("ch2 : ");
    scanf("%s", &ch2);
    if (strcmp(ch1, ch2) == 0) // ch1=ch2
        printf("OK");
    else printf("KO");
}

void main() {
    afficher_arguments(1, 'a', 3.2);
    egaux();
}
```

Exercice 3.
Ecrire une procédure `etes_vous_toto()` qui demande à l'utilisateur d'entrer son nom et qui affiche :

- "Vous êtes Toto" si l'utilisateur entre "Toto";
- "Vous n'êtes pas Toto" sinon.

Exercice 4.
Ecrire une procédure `etes_vous_toto2()` qui demande à l'utilisateur d'entrer son nom et qui affiche :

- "Vous êtes Toto" si l'utilisateur entre "Toto" ou "toto" ;
- "Vous n'êtes pas Toto" sinon.

Exercice 5.
Ecrire une procédure `etes_vous_untel()` qui prend en argument une chaine de caratères, par exemple "Tata" ; qui demande à l'utilisateur d'entrer son nom et qui affiche :

- "Vous êtes Tata" si l'utilisateur entre "Tata" ;
- "Vous n'êtes pas Tata" sinon.

Correction.

```
------ tp3.c ------
#include <stdio.h>
#include <math.h>
#include <string.h>

(...)

void etes_vous_untel(char *untel) {
    char ch[30];
```

```
    printf("ch : ");
    scanf("%s", &ch);
    if (strcmp(ch, untel) == 0) // ch=untel
        printf("Vous etes %s.\n", untel);
    else printf("Vous n'etes pas %s.\n", untel);
}

void main() {
    (...)
    etes_vous_untel("Tata");
}
```

Exercice 6.
Ecrire une procédure `qui_etes_vous()` qui demande à l'utilisateur d'entrer son nom, par exemple "Bond" ; puis d'entrer son prénom, par exemple "James" ; et enfin son âge, par exemple "40". Le programme affiche alors :

```
Vous etes James Bond et vous avez 40 ans.
```

Exercice 7.
Ecrire une procédure `qui_est_plus_grand(int, int)` qui se comporte ainsi (ici en Python) :

```
>>> qui_est_plus_grand(3,5)
5 est plus grand que 3
>>> qui_est_plus_grand(5,3)
5 est plus grand que 3
>>> qui_est_plus_grand(3,3)
3 et 3 sont egaux !
```

Exercice 8.
Ecrire une procédure `quelle_heure()` qui demande à l'utilisateur d'entrer une heure h (exprimée ici par un entier) et qui affiche :

- "Je vous demande pardon ?" si $h < 0$ ou $h \geq 24$;
- "Il faut dormir !" si $0 \leq h < 5$;
- "Bon matin !" si $5 \leq h < 12$;
- "Bon appétit !" si $12 \leq h < 14$;
- "Bon après-midi !" si $14 \leq h < 18$;
- "Bon soir !" si $18 \leq h < 21$;
- "Bonne nuit !" si $21 \leq h < 24$.

Exercice 9.
Ecrirre une procédure `calculette()` qui demande à l'utilisateur d'entrer un nombre x, un opérateur $<op>$, puis un nombre y ; et qui affiche le résultat du calcul $x<op>y$.

Exemple (ici en Python) :

```
>>> calculette()
Nombre : 5
Operateur : *
Nombre : 2
=
10
>>> calculette()
Nombre : 6
Operateur : /
Nombre : 3
=2
>>> calculette()
Nombre : 5
Operateur : +
Nombre : 6
=
11
>>> calculette()
Nombre : 5
Operateur : -
Nombre : 6
=
-1
```

TP 4 – Boucle *Pour* – Algorithmique

Dr M. Guedj

Exercice 1.
Implantez la procédure `dix()` qui affiche les 10 premiers entiers.

Correction.

```
------ tp4.c ------
#include <stdio.h>

void dix()
{
    int i;
    for (i=0; i<10; i++)
        printf("%d \n", i);
}

void main()
{
    dix();
}
```

Exercice 2.
Implantez la procédure `n_premiers_entiers()`, qui prend en argument un entier n, et affiche les n premiers entiers.

Exercice 3.
Implantez la procédure `n_premiers_carres()`, qui prend en argument un entier n, et affiche les n premiers entiers au carré.

Exercice 4.
Implantez la procédure `n_premiers_pairs()`, qui prend en argument un entier n, et affiche les entiers pairs compris entre 0 et $n-1$.

Exercice 5.
Implantez la procédure `n_premiers_impairs()`, qui prend en argument un entier n, et affiche les entiers impairs compris entre 0 et $n-1$.

Exercice 6.
Implantez la procédure `dix_fois_coucou()` qui affche 10 fois "Coucou !".

Exercice 7.
Implantez la procédur `n_fois_coucou()`, qui prend en argument un entier n, et affiche n fois "Coucou !".

Exercice 8.
Implantez la fonction `somme_n_premiers_entiers()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers ente 0 et $n-1$.

Exercice 9.
Implantez la fonction `somme_n_premiers_carres()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers au carré compris ente 0 et $n-1$.

Exercice 10.
Implantez la fonction `somme_n_premiers_pairs()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers pairs compris ente 0 et $n-1$.

Exercice 11.
Implantez la fonction `somme_n_premiers_impairs()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers impairs compris ente 0 et $n-1$.

Exercice 12.
Implantez la fonction `somme_n_premiers_pairs_carres()`, qui prend en argument un entier n, et retourne la somme des entiers pairs au carré, compris entre 0 et $n-1$.

Exercice 13.
Implantez l'algorithme suivant :

```
fonction toto(n)
    res = 0
    pour i=0,..,n-1 alors
        si i est divisible par 3 alors
            res = res + i
        fin si
    fin pour
    retourner res
```

Exercice 14.
Implantez l'algorithme suivant :

```
fonction toto2(n, k)
    res = 0
    pour i=0,..,n-1 faire
        si i est divisible par k alors
            res = res + i
        fin si
    fin pour
    retourner res
```

Donnez une nouvelle implantation de `toto()` à partir de `toto2()`.

TP 5 – Boucle *Tant que* – Algorithmique

Dr M. Guedj

Ce TP est en partie une redite du TP précédent ; mais vous utiliserez cette fois-ci les boucles *Tant que*.

Exercice 1.
Implantez la procédure `dix()` qui affiche les 10 premiers entiers.

Correction.

```
------ tp5.c ------
#include <stdio.h>

void dix()
{
    int i=0;
    while (i<10)
    {
        printf("%d \n", i);
        i++;
    }
}

void main()
{
    dix();
}
```

Exercice 2.
Implantez la procédure `n_premiers_entiers()`, qui prend en argument un entier n, et affiche les n premiers entiers.

Exercice 3.
Implantez la procédure `n_premiers_carres()`, qui prend en argument un entier n, et affiche les n premiers entiers au carré.

Exercice 4.
Implantez la procédure `n_premiers_pairs()`, qui prend en argument un entier n, et affiche les entiers pairs compris entre 0 et $n-1$.

Exercice 5.
Implantez la procédure `n_premiers_impairs()`, qui prend en argument un entier n, et affiche les entiers impairs compris entre 0 et $n-1$.

Exercice 6.
Implantez la procédure `dix_fois_coucou()` qui affche 10 fois "coucou".

Exercice 7.
Implantez la procédur `n_fois_coucou()`, qui prend en argument un entier n, et affiche n fois "coucou".

Exercice 8.
Implantez la fonction `somme_n_premiers_entiers()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers ente 0 et $n-1$.

Exercice 9.
Implantez la fonction `somme_n_premiers_carres()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers au carré compris ente 0 et $n-1$.

Exercice 10.
Implantez la fonction `somme_n_premiers_pairs()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers pairs compris ente 0 et $n-1$.

Exercice 11.
Implantez la fonction `somme_n_premiers_impairs()`, qui prend en argument un entier n, et retourne la somme des n premiers entiers impairs compris ente 0 et $n-1$.

Exercice 12.
Implantez la fonction `somme_n_premiers_pairs_carres()`, qui prend en argument un entier n, et retourne la somme des entiers pairs au carré, compris entre 0 et $n-1$.

Exercice 13.
Implantez l'algorithme suivant :

```
fonction toto(n)
    res = 0
    i = 0
    tant que i<n faire
        si i est divisible par 3 alors
            res = res + i
        fin si
        i = i+1
    fin tant que
    retourner res
```

Exercice 14.
Implantez l'algorithme suivant :

```
fonction toto2(n, k)
    res = 0
    i = 0
    tant que i<n faire
        si i est divisible par k alors
            res = res + i
        fin si
        i = i+1
    fin tant que
retourner res
```

Donnez une nouvelle implantation de `toto()` à partir de `toto2()`.

Exercice 15.
Implantez l'algorithme suivant :

```
fonction toto3()
    res = 0
    i = 1
    tant que i<=100:
        res = res + i
        i = i+1
    fin tant que
    retourner res
```

Exercice 16.
Implantez l'algorithme suivant :

```
fonction toto4(a,b)
    res = 0
    i = a
    tant que i<=b:
        res = res + i
        i = i+1
    fin tant que
    retourner res
```

Donnez une nouvelle implantation de `toto3()` à partir de `toto4()`.

Exercice 17.
Ecrire un programme qui boucle tant que 'q' n'est pas entré au clavier. A chaque tour de boucle, on affichera ce qui est entré au clavier.

Exercice 18.
Ecrire un programme qui modélise le jeu ci-dessous.

Ce jeu se joue à deux joueurs.
– *Le premier joueur choisit un entier compris entre 1 et 100.*
– *Le second joueur cherche cet entier.*
– *Si l'entier qu'il propose est trop grand, le joueur 1 dit "trop grand !".*
– *Si l'entier qu'il propose est trop petit, le joueur 1 dit "trop petit !".*
– *Et si le joueur 2 trouve l'entier, alors le joueur 1 dit "trouvé !".*

Note : dans le cas de cet exercice, le joueur 1 correspond à la machine et le joueur 2 est humain.

TP 6 – Tableaux de caractères – Algorithmique

Dr M. Guedj

Exercice 1.
Ecrire une procédure `afficher()`, qui prend en argument un tableau de caractères, ainsi que sa taille, et qui affiche ses éléments.

Correction.

```
------ tp6.c ------
#include <stdio.h>

void afficher(char *t, int n)
{
    int i=0;
    for (i=0; i<n; i++)
        printf("%c ", t[i]);
}

void main()
{
    char t[5] = {'a', 'b', 'c', 'd', 'e'};
    afficher(t, 5);
}
```

Exercice 2.
Ecrire une procédure `afficher_a()`, qui prend en argument un tableau de caractères *tab*, sa taille, ainsi qu'un entier a ; et affiche $tab[a]$ (si a est correctement défini).

Exercice 3.
Ecrire une procédure `afficher_a_b()`, qui prend en argument un tableau de caractères *tab*, sa taille, ainsi que deux entiers a et b ; et qui affiche les éléments de *tab* d'indices compris entre a et b (si a et b sont correctement définis).

Exemple (en Python) : Si *tab* contient les entiers $1, 2, 3, 4, 5, 6$; `afficher_a_b(tab, 6, 2, 5)` affichera : `3,4,5 et 6`.

Exercice 4.
Ecrire une fonction `concatener()`, qui prend en argument un tableau de caractères *tab*, ainsi que sa taille, et affiche ses éléments en les concaténant à la manière de l'exemple (en Python) ci-dessous :

```
>>> concatener(['a', 'b', 'c'], 3)
abc
```

Exercice 5.
Ecrire une fonction `appartient()`, qui prend en argument un tableau de caractères *tab*, sa taille, ainsi qu'un élément x ; et qui retourne :

- `Vrai` si x est dans *tab* ;
- `Faux` sinon.

Exercice 6.
Ecrire une fonction `non_appartient()`, qui prend en argument un tableau de caractères *tab*, sa taille, ainsi qu'un élément x ; et qui retourne :

- `Vrai` si x n'est pas dans *tab* ;
- `Faux` sinon.

Exercice 7.
Ecrire une fonction `appartient2()`, qui prend en argument deux tableaux de caractères *tab*1 et *tab*2, leurs tailles supposées identiques, ainsi qu'un élément x ; et qui retourne :

- `Vrai` si x est à la fois dans *tab*1 et dans *tab*2 ;
- `Faux` sinon.

Exercice 8.
Ecrire une fonction `appartient3()`, qui prend en argument deux tableaux de caractères *tab*1 et *tab*2, leurs tailles supposées identiques, ainsi qu'un élément x ; et qui retourne :

- `Vrai` si x est dans *tab*1 et n'est pas dans *tab*2 ;
- `Faux` sinon.

Exercice 9.
Ecrire une fonction `appartient4()`, qui prend en argument trois tableaux de caractères *tab*1, *tabt*2 et *tab*3, leurs tailles supposées identiques, ainsi qu'un élément x ; et qui retourne :

- `Vrai` si x est dans *tab*1 et également est dans (*tab*2 ou *tab*3) ;
- `Faux` sinon.

TP 7 – Tableau d'entiers – Algorithmique

Dr M. Guedj

Remarque :
On passera la taille du tableau en argument de chaque fonction/procédure demandée.

Exercice 1.
Implantez la procédure `afficher()` qui affiche les éléments du tableau d'entiers passé en argument.

Correction.

```
------ tp7.c ------
#include <stdio.h>

int afficher(int *t, int n)
{
    int i;
    for (i=0; i<n; i++)
        printf("%d ", t[i]);
}

void main()
{
    int t1[5] = {1,2,3,4,4};

    afficher(t1, 5);
}
```

Exercice 2.
Implantez la fonction `maximum()` qui retourne le maximum du tableau d'entiers passé en argument.

Exercice 3.
Implantez la fonction `somme()` qui retourne la somme des éléments du tableau d'entiers passé en argument.

Exercice 4.
Implantez la fonction `moyenne()` qui retourne la moyenne des éléments du tableau d'entiers passé en argument.

Exercice 5.
Implantez la fonction `tous_egaux()` qui retourne `Vrai` si tous les éléments du tableau passé en argument sont égaux ; et `Faux` sinon.

Exercice 6.
Implantez la fonction `produit_scalaire()` qui retourne le produit scalaire des deux tableaux d'entiers (de même taille) passés en argument. Formellement, le produit scalaire est défini par :

$$(x_1, ..., x_n).(y_1, ..., y_n) = x_1y_1 + ... + x_ny_n.$$

Exercice 7.
Implantez la fonction `egaux()` qui retourne `Vrai` si les deux tableaux d'entiers (de même taille) passés en argument contiennent les mêmes éléments (dans le même ordre) ; et `Faux` sinon. Si $tab1 = (x_1, ..., x_n)$ et $tab2 = (y_1, ..., y_n)$, `egaux(`$tab1, tab2$`, n)` retournera `Vrai` si : $x_i = y_i$ pour chaque i.

Exercice 8.
Implantez la fonction `est_trie()`, qui prend en argument un tableau d'entiers ; et qui retourne `Vrai` si le tableau est trié ; et `Faux` sinon.

TP 8 – Chaîne de caractères – Algorithmique

Dr M. Guedj

Exercice 1.
Implantez la procédure `afficher()`, qui prend en argument une chaîne de caractères ; et qui l'affiche sur la console.

Correction.

```
------ tp8.c ------
#include <stdio.h>
#include <stdlib.h>

void afficher1(char *s)
{
    int i=0;
    while (s[i] != '\0')
    {
        printf("%c", s[i]);
        i++;
    }
    fflush(stdout);
    printf("\n");
}

void afficher2(char *s)
{
    printf("%s\n", s);
}

void main()
{
    char ch1[] = "abcd";
    char ch2[] = "";

    afficher1(ch1);
    afficher1(ch2);
    afficher2(ch1);
```

```
    afficher2(ch2);
}
```

Exercice 2.
Implantez la fonction `taille()`, qui prend en argument une chaîne de caractères ; et qui retourne sa taille.

Exercice 3.
Implantez la fonction `char * concatener(char *, int)`, qui prend en argument un tableau de caractères ainsi que sa taille ; et qui retourne la concatenation des éléments de ce tableau sous la forme d'une chaîne de caractères.

Exercice 4.
Implantez une fonction, qui prend en argument une chaîne de caractères *ch* ainsi qu'un caractère *c*, et qui retourne `Vrai` si *c* est présent dans *ch* et `Faux` sinon.

Exercice 5.
Implantez une fonction, qui prend en argument une chaîne de caractères *ch*, ainsi qu'un caractère *c*, et qui retourne le nombre d'occurrences de *c* dans *ch* ; c'est-à-dire le nombre de fois où *c* apparaît dans *ch*.

Exercice 6.
Implantez une fonction, qui prend en argument deux chaînes de caractères ch_1 et ch_2, et qui retourne `Vrai` si $ch_1 = ch_2$ (c.-à-d. si ch_1 et ch_2 sont de même taille et ont les mêmes éléments de manière ordonnée) ; et `Faux` sinon.

Exercice 7.
Implantez une fonction, qui prend en argument un tableau de caractères *tab*, sa taille, ainsi qu'une chaînes de caractère *ch* ; et qui retourne `Vrai` si la concaténation des éléments de *tab* est égale au mot *ch* ; et `Faux` sinon.

Exercice 8.
Implantez une fonction qui prend en argument une chaîne de caractères *ch*, ainsi que deux caractères *x* et *y* ; et qui retourne une chaîne de caractères ch' qui est constituée des éléments de la chaîne de caractères *ch*, dans laquelle toute occurrence de *x* est remplacée par *y* . Par exemple, si *ch* vaut `"aaa bb aa bbb"`, *x* vaut `'a'` et *y* vaut `'c'`; alors ch' vaudra `"ccc bb cc bbb"`.

Exercice 9.
Implantez une fonction qui prend en argument une chaîne de caractères *ch* ; et qui retourne une chaîne de caractères ch' qui est constituée des éléments de *ch* hormis le caractère espace. Par exemple, si *ch* vaut `"aaa bb aa bbb"`, ch' vaudra `"aaabbaabbb"`.

Exercice 10.
Implantez une fonction, qui prend en argument un tableau de chaînes caractères *tab*, sa taille, ainsi qu'un caractère *c* ; et qui retourne `Vrai` si *c* est présent dans un élément de *tab*.

TP 9 – Matrices – Algorithmique

Dr M. Guedj

Exercice 1.
Implantez une procédure d'affichage de matrice d'entiers. La procédure prendra en argument la matrice ainsi que les nombres de ses lignes et de ses colonnes.

Correction.

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

// alloue une matrice de taille n x m
int **allocate_matrix(int n, int m)
{
    int **mat  = (int **) malloc(n * sizeof(int *));
    int i;
    for(i=0; i<n; i++)
        mat[i] = (int *) malloc(m * sizeof(int));
    return mat;
}

// desalloue une matrice de taille n x m
void deallocate_matrix(int** mat, int n, int m)
{
    int i;
    for (i=0 ; i<n; i++)
        free(mat[i]);
    free(mat);
}

// Affichage d'une matrice n x m
void affichage(int **mat, int n, int m)
{
    int i, j;
    for (i=0; i<n; i++) {
        for (j=0; j<m; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
```

```
    }
}

int main()
{
    int i, j, cpt;
    int **mat = allocate_matrix(3, 3);

    cpt=0;
    for (i=0; i<3 ; i++)
        for (j=0; j<3; j++)
            mat[i][j] = cpt++;

    affichage(mat, 3, 3);

    deallocate_matrix(mat, 3, 3);
    return EXIT_SUCCESS;
}
```

Exercice 2.
Implantez une fonction qui retourne `Vrai` si la matrice d'entiers passée en argument est nulle (c-à-d ne contient que des 0) et qui retourne `Faux` sinon.

Exercice 3.
Implantez une fonction qui prend en argument une matrice d'entiers mat, un entier x, et qui retourne `Vrai` si mat ne contient que des x, et qui retourne `Faux` sinon.

Exercice 4.
Implantez la multiplication d'une matrice d'entiers par un scalaire (ici un entier). Exemple :

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \times 2 = \begin{pmatrix} 1 \times 2 & 2 \times 2 \\ 2 \times 2 & 1 \times 2 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 4 & 2 \end{pmatrix}.$$

Exercice 5.
Implantez l'addition de deux matrices d'entiers (les deux matrices seront considérées comme ayant les mêmes dimensions). Exemple :

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} + \begin{pmatrix} 10 & 8 \\ 12 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 10 \\ 14 & 2 \end{pmatrix}.$$

Exercice 6.
Implantez une fonction qui retourne `Vrai` si la matrice d'entiers carrée (de taille $n \times n$) d'entiers passée en argument est diagonale, et `Faux` sinon. Une matrice diagonale mat vérifie :

$$i \neq j \implies mat_{i,j} = 0.$$

$$\begin{pmatrix} mat_{1,1} & 0 & 0 & ... & 0 & 0 \\ 0 & mat_{2,2} & 0 & ... & 0 & 0 \\ 0 & 0 & mat_{3,3} & ... & 0 & 0 \\ 0 & 0 & 0 & ... & mat_{(n-1),(n-1)} & 0 \\ 0 & 0 & 0 & ... & 0 & mat_{n,n} \end{pmatrix}.$$

Exercice 7.
Implantez une fonction qui retourne `Vrai` si la matrice carrée (de taille $n \times n$) d'entiers passée en argument est symétrique, et `Faux` sinon. Une matrice symétrique mat vérifie :

$$mat_{i,j} = mat_{j,i}.$$

Exemple de matrice symétrique :

$$\begin{pmatrix} 2 & 4 & 6 \\ 4 & 0 & 10 \\ 6 & 10 & 12 \end{pmatrix}.$$

Exercice 8.
Implantez une fonction qui prend en argument une matrice d'entiers mat, ainsi qu'un entier x ; et qui retourne le nombre d'occurrences de x dans mat. Par exemple, si

$$mat = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 3 \\ 7 & 3 & 9 \end{pmatrix}$$

et $x = 3$; la fonction demandée retournera 4.

TP 10 – Récursivité – Algorithmique

Dr M. Guedj

Exercice 1 (Suite arithmétique).
Implantez la suite artihmétique $(u)_n$ définie comme suit :

$$u_n = \begin{cases} 6 & \text{si } n = 0 \\ u_{n-1} + 3 & \text{sinon} \end{cases}$$

Correction.

```
int u(int n)
{
    if (n==0) return 6;
    return u(n-1)+3;
}
```

Exercice 2 (Suite arithmétique).
Implantez la suite artihmétique $(v)_n$ définie comme suit :

$$v_n = \begin{cases} 20 & \text{si } n = 0 \\ v_{n-1} + 5 & \text{sinon} \end{cases}$$

Exercice 3 (Suite géométrique).
Implantez la suite géométrique $(w)_n$ définie comme suit :

$$w_n = \begin{cases} 5000 & \text{si } n = 0 \\ w_{n-1} \times 2 & \text{sinon} \end{cases}$$

Exercice 4 (Suite géométrique).
Implantez la suite géométrique $(x)_n$ définie comme suit :

$$x_n = \begin{cases} 2 & \text{si } n = 0 \\ x_{n-1} \times 0,3 & \text{sinon} \end{cases}$$

Exercice 5 (Factorielle).
Implantez, récursivement, la factorielle définie par :

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{sinon} \end{cases}$$

Exercice 6 (Suite de Fibonacci).
Implantez, récursivement, la suite de Fibonacci définie par :

$$F(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ F(n-1) + F(n-2) & \text{sinon} \end{cases}$$

Les premiers termes de la suite de Fibonacci sont : 0, 1, 1, 2, 3, 5, 8, 13 et 21.

Exercice 7 (Successeurs).
Implantez, récursivement, la suite $succ_{n\in\mathbb{N}}$ définies formellement par :

$$succ(n) = \begin{cases} 1 & \text{si } n = 0 \\ succ(n-1) + 1 & \text{sinon} \end{cases}$$

Exercice 8 (Puissance).
Implantez, récursivement, la puissance n-ième ($n \in \mathbb{N}$) du nombre $a \in \mathbb{R}$, en utilisant la relation : $a^n = a.a^{n-1}$.

Exercice 9 (Somme).
Implantez, récursivement, la somme des n premiers entiers (somme(5) = 0+1+2+3+4+5).

TP 11 – Nombres complexes – Algorithmique

Dr M. Guedj

Un nombre complexe z se présente en général sous forme algébrique comme une somme $a + ib$, où a et b sont des nombres réels quelconques et où i (l'unité imaginaire) est un nombre particulier tel que $i^2 = -1$.

Le réel a est appelé partie réelle de z et se note $Re(z)$, le réel b est sa partie imaginaire et se note $Im(z)$.

Deux nombres complexes sont égaux si et seulement si ils ont la même partie réelle et la même partie imaginaire ; *i.e* :

$$\forall z, z' \in \mathbb{C}, z = z' \iff \big(Re(z) = Re(z') \text{ et } Im(z) = Im(z')\big)$$

Un nombre complexe z est dit imaginaire pursi sa partie réelle est nulle, dans ce cas il s'écrit sous la forme $z = ib$. Un nombre complexe dont la partie imaginaire est nulle est dit réel.

Soit le type complexe définit comme suit :

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

qui nous permettra de manipuler les nombres complexes[1].

Exercice 1.
Implanter une procédure qui affiche le nombre complexe passé en argument.

Exercice 2.
Implanter une fonction `somme()` qui retourne la somme des deux nombres complexes passés en paramètre.

Exercice 3.
Implanter une fonction `oppose()` qui retourne l'opposé du nombre complexe passé en paramètre.

Note : l'opposé de $a + bi$ est $-a - bi$.

Exercice 4.
Implanter la fonction `p_imaginaire_pur()` qui prend en argument un nombre complexe z et retourne 1 si z est un imaginaire pur ($i.e$: $z = 0 + bi$) et 0 sinon.

[1] Remarque : ISO C99 introduit les nombres complexes en langage C.

Exercice 5.
Implanter une fonction `conjugue()` qui retourne le conjgué du nombre complexe passé en paramètre.

Note : le conjugué de $a + bi$ est $a - bi$.

Exercice 6.
Implantez une fonction `module()` qui retourne le module du nombre complexe passé en paramètre.

Note : le module de $z = a + bi$ (noté $|z|$) est défini par : $|z| = \sqrt{a^2 + b^2}$.

Exercice 7.
Implantez la multiplication de deux nombres complexes.

Note : $(a + i.b) \times (a' + i.b') = (aa'–bb') + i(ab' + ba')$.

Remarque : $(a + i.b) \times (a–i.b) = a^2 + b^2$.

TP 12 – Complément à deux

La représentation la plus courante pour les entiers dans l'ordinateur est la représentation dite *en complément à 2*, qui présente deux avantages :

- D'une part on peut additionner des nombres en complément à deux avec les règles usuelles de l'addition, sans se soucier de leurs signes : le résultat de l'addition aura le bon signe ;
- D'autre part, chaque nombre qu'on peut représenter ne possède qu'une représentation unique.

1. Notion de complément à deux

Le complément à deux est une représentation binaire des entiers relatifs qui permet d'effectuer les opérations arithmétiques usuelles naturellement.

Avec *n* bits on peut représenter des entiers entre : -2^{n-1} et $2^{n-1}-1$.

Un codage avec *n* bits de la forme :

x_{n-1}	x_{n-2}	...	x_1	x_0

Représente l'entier :

$$-x_{n-1}.2^{n-1}+\sum_{i=0}^{n-2} x_i.2^i$$

Noter que le bit de signe est placé en tête (bit de poids fort).

2. Exemples

Exemple 1

L'octet :

7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	1

Représente l'entier :

$$-2^7+2^4+2^3+2^0$$

C'est-à-dire :

$$-128+16+8+1=-103$$

Exemple 2

L'octet :

7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1

Représente l'entier :

$$2^5+2^4+2^3+2^0$$

C'est-à-dire :

$$32+16+8+1=57$$

3. Propriétés du complément à deux

Propriété 1

Avec *n* bits, on peut représenter des entiers entre -2^{n-1} et $2^{n-1}-1$.

En particulier, avec 8 bits, on peut représenter des entiers entre -2^7 et 2^7-1 ; soit entre -128 et 127.

Propriété 2

Soit un entier *n* dont la représentation en complément à deux est :

x_{n-1}	x_{n-2}	...	x_1	x_0

Alors *n* est strictement négatif si et seulement si $x_{n-1}=1$ [1].

1 Car : $2^{k+1}=(\sum_{i=0}^{k}2^i)+1$.

Propriété 3

Soit un entier *n* dont la représentation en complément à deux est :

x_{n-1}	x_{n-2}	...	x_1	x_0

Alors *n* est pair si et seulement si $x_0 = 0$.

4. Questions

Dans ce qui suit nous considérons un type `Octet` défini comme suit :

```
typedef unsigned short Octet[8];
```

De plus, les booléens *Vrai* et *Faux* seront exprimés par 1 et 0 respectivement.

1. Réaliser une procédure `afficher()` prenant en argument un octet, et l'affichant sur la console.

2. Réaliser la fonction `zero()` prenant en argument un octet ; et retournant *Vrai* si l'octet passé en argument est nul (*i.e.* 00000000_2), et *Faux* sinon.

3. Réaliser la fonction `negatif()` prenant en argument un octet ; et retournant *Vrai* si l'entier, que représente en complément à deux cet octet, est négatif ; et *Faux* sinon.

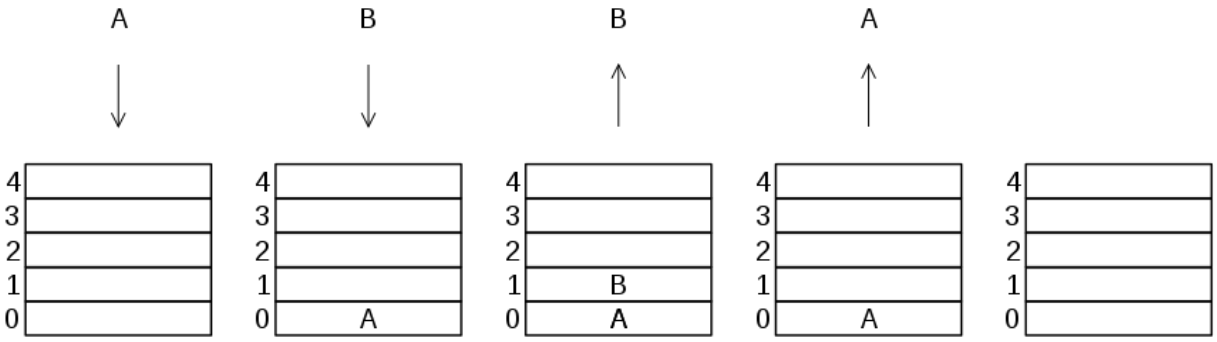
4. Réaliser la fonction `pair()` prenant en argument un octet ; et retournant *Vrai* si l'entier, que représente en complément à deux cet octet, est pair ; et *Faux* sinon.

5. Réaliser la fonction `egaux()` prenant en argument deux octets ; et retournant *Vrai* si les deux entiers, que représentent en complément à deux ces deux octets, sont égaux ; et *Faux* sinon.

6. Réaliser la fonction `en_entier()` prenant en argument un octet ; et retournant l'entier, que représente en complément à deux, cet octet.

TP 13 – Implémentation d'une pile par un tableau

Une **pile** (en anglais *stack*) est un conteneur fondé sur le principe « dernier arrivé, premier sorti » (ou LIFO pour *last in, first out*).



Source : https://commons.wikimedia.org/wiki/File:Stack_(data_structure)_LIFO.svg

On note ci-après les primitives usuelles pour le type abstrait *pile*.

- **est_vide :** retourne VRAI si la pile est vide et FAUX sinon ;
- **taille :** renvoie le nombre d'éléments dans la pile ;
- **empiler :** ajoute un élément sur la pile ;
- **dépiler :** enlève un élément de la pile et le retourne ;
- **sommet :** retourne l'élément de tête sans le dépiler.

On considère ici une pile de caractères, implémentée à l'aide d'un tableau. On donne ci-après un embryon de code.

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

#define MAX 30

typedef struct Pile
{
    int taille;
    char tab[MAX];
} Pile;

// initialisation d'une pile
void init(Pile *p)
{
    p->taille=0;
}

int main()
{
    Pile *p;
    char tmp;

    p = malloc(sizeof(Pile));
    init(p);

    free(p);

    return EXIT_SUCCESS;
}
```

- Implanter les primitives associées au type abstrait *pile*.
- Tester votre travail (utiliser une procédure d'affichage).

TP 14 – Programmation en C

Exercice 1

a) Compiler le code suivant :

- tp14.c -

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

// - 1 -
// afficher le fichier de nom "name"
void affichage1(char *name)
{
    FILE *f_in = fopen(name, "r");
    int c;

    if (f_in == NULL)
    {
        fprintf(stderr, "\nError : Impossible to read the file %s\n", name);
        return EXIT_FAILURE;
    }

    c = fgetc(f_in);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(f_in);
    }

    fclose(f_in);
}
```

```c
int main(int argc, char **argv)
{
    int i;

    printf("\n-- TP 14 --\n");

    // affichage des arguments du programme en cours d'execution
    printf("Arguments du programme %s : \n", argv[0]);

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

    if (argc < 2)
    {
        fprintf(stderr, "\nUsage : An argument is required.\n");
        return EXIT_FAILURE;
    }

    affichage1(argv[1]);
    //sans_voyelles_sauv(argv[1]);

    return EXIT_SUCCESS;
}
```

- -

b) Ouvrir un terminal de commande (sous *Windows*, prendre par exemple « cmd »).

c) Placez-vous dans le dossier contenant votre exécutable ;

- sous *Windows*, utiliser les commandes : « cd » et « dir » ;
- sous *Unix-like*: « cd » et « ls » .

d) Télécharger le fichier « informatique.txt » disponible sur le site relatif au cours.

e) A partir du terminal de commande, exécuter le programme par la commande suivante :

```
c:\Users\toto\TP> tp.exe  informatique.txt
```

Pour les exercices 2 à 5, utiliser la procédure `affichage1()` comme modèle.

Exercice 2

Réaliser une procédure `affichage2()` qui affiche un fichier texte sur la console, en remplaçant chaque caractère '#' par un caractère '-'.

(Comme pour `affichage1()`, la procédure `affichage2()` prend en argument le nom du fichier à considérer).

Exercice 3

Réaliser une procédure `affichage2()` qui affiche un fichier texte sur la console, en remplaçant les caractères '#' par des caractères '-'.

Exercice 4

Réaliser une procédure `affichage3()` qui affiche un fichier texte sur la console en remplaçant les voyelles par des caractères '*'.

Exercice 5

Réaliser une procédure `affichage4()` qui affiche un fichier texte sur la console en éliminant toutes les voyelles.

Exercice 6

Tester le code suivant :

- -

```
void sans_voyelles_sauv(char *name)
{
    FILE *f_in = fopen(name, "r");
    FILE *f_out = fopen("new.txt", "w");

    int c;

    if (f_in == NULL)
    {
        fprintf(stderr, "\nError : Impossible to read the file %s\n", name);
        return EXIT_FAILURE;
    }

    c = fgetc(f_in);
```

```
    while (c != EOF)
    {
        if (c=='a' || c == 'e' || c == 'o' || c == 'u' || c == 'y')
        {
            fprintf(f_out, "");
        }
        else
        {
            fprintf(f_out, "%c", c);
        }
        c = fgetc(f_in);
    }

    fclose(f_in);
    fclose(f_out);
}
```

- -

TP 15 – Mini projets – Programmation en C

Dr M. Guedj

Exercice 1 (Pierre-Feuille-Ciseau)**.**
Implantez le jeu Pierre-Feuille-Ciseau. L'utilisateur joue contre la machine. Il utilise les touches 'p' pour "pierre", 'f' pour "feuille", 'c' pour "ciseau" et 'q' pour quitter le jeu ; qui boucle sinon.

Exemple de partie :

```
>>>
p
pierre contre feuille
joueur 2 gagne !
p
pierre contre pierre
Ex aequo !
p
pierre contre ciseau
joueur 1 gagne !
f
feuille contre ciseau
joueur 2 gagne !
f
feuille contre feuille
Ex aequo !
f
feuille contre ciseau
joueur 2 gagne !
q
>>>
```

Exercice 2 (Dessin de triangle)**.**
Ecrire un programme qui dessine, sur la console, un triangle comme suit :

```
#
##
###
####
#####
####
###
##
#
```

en demandant, prealablement, a l'utilisateur d'entrer la hauteur du triangle (ici la hauteur est de 5).

Exercice 3 (Jeu des allumettes)**.**
Realisez (en mode console) le jeu des allumettes ; dont la description, extraite (et modiée) du site maths.amatheurs.fr, est donnee ci-dessous.

Ce jeu se joue a deux. Les joueurs sont devant un certain nombre d'allumettes. A chaque tour, il faut en enlever 1, 2 ou 3. Celui qui prend la derniere perd.

Exercice 4 (Jeu de Toto)**.**
Realisez un programme, qui affiche sur la console un plateau, dans lequel Toto, symbolisé par la lettre 'T', se deplace. C'est un bon exercice de maîtrise des éléments de base de l'algorithmique (structures de donnés et de contrôle). Note: le plateau sera exprimé par une matrice de caractères.

1. Dans un premier temps, Toto se deplacera aleatoirement dans le plateau (dans les limites du plateau).

2. Dans un second temps, recherchez un deplacement plus fluide de Toto.

3. Enfin, realisez un deplacement "manuel" de Toto, i.e. via entrées de l'utilisateur au clavier.

Exercice 5 (Bulls and Cows)**.**
Realisez (en mode console) le jeu Bulls and Cows ; dont la description (extraite de Wikipedia) est donnée ci-dessous.

The numerical version of the game is usually played with 4 digits, but can also be played with 3 or any other number of digits. On a sheet of paper, the players each write a 4-digit secret number. The digits must be all different. Then, in turn, the players try to guess their opponent's number who gives the number of matches. If the matching digits are in their right positions, they are "bulls", if in different positions, they are "cows". Example:

- *Secret number: 4271*

– *Opponent's try: 1234*

– *Answer: 1 bull and 2 cows. (The bull is "2", the cows are "4" and "1".)*

The first one to reveal the other's secret number wins the game.

Dans notre cas, on s'interesse a un matche humain versus CPU. L'humain seul droit trouver le code du CPU.

Exercice 6 (Tic-tac-toe)**.**
Realisez (en mode console) le jeu Tic-tac-toe ; dont la description (tirée de Wikipedia) est donnée ci-dessous.

Le Tic-tac-toe se joue sur une grille carree de 3×3 *cases. Deux joueurs s'affrontent. Ils doivent remplir chacun à leur tour une case de la grille avec le symbole qui leur est attribué : O ou X. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale. Une partie gagnée par le joueur X : Une partie nulle :*

Figure 1: Source : `https://fr.wikipedia.org/wiki/Tic-tac-toe`

Figure 2: Source : `https://fr.wikipedia.org/wiki/Tic-tac-toe`

– Dans un premier temps, en mode deux joueurs "humains".

– Ensuite en mode joueur "humain" versus joueur CPU (jeu aleatoire pour le CPU).

– Enfin, interressez-vous a l'IA du CPU.