

## R1.01 INITIATION AU DÉVELOPPEMENT

### *Sujets de TD et TP – 1<sup>ère</sup> partie*

BUT Informatique  
1<sup>ère</sup> année – 1<sup>re</sup> semestre

Denis Poitrenaud  
`denis.poitrenaud@u-paris.fr`

<b>1</b>	<b>Semaine n°1</b>	<b>2</b>
1.1	Travaux dirigés . . . . .	2
1.2	Travaux pratiques . . . . .	3
<b>2</b>	<b>Semaine n°2</b>	<b>6</b>
2.1	Travaux dirigés . . . . .	6
2.2	Travaux pratiques . . . . .	7
<b>3</b>	<b>Semaine n°3</b>	<b>9</b>
3.1	Travaux dirigés . . . . .	9
3.2	Travaux pratiques . . . . .	10
<b>4</b>	<b>Semaine n°4</b>	<b>12</b>
4.1	Travaux dirigés . . . . .	12
4.2	Travaux pratiques . . . . .	14
<b>5</b>	<b>Semaine n°5</b>	<b>18</b>
5.1	Travaux dirigés . . . . .	18
5.2	Travaux pratiques . . . . .	20
<b>6</b>	<b>Semaine n°6</b>	<b>22</b>
6.1	Travaux dirigés . . . . .	22
<b>7</b>	<b>Semaine n°7</b>	<b>24</b>
7.1	Travaux dirigés . . . . .	24

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°1

## Thèmes

- Évaluation d'expressions
- Expressions arithmétiques, relationnelles et logiques

Les exercices de 5 à 8 sont à coder en C.

1. En suivant le principe d'évaluation des expressions décrit dans le cours, donnez la valeur de l'expression suivante :

$$2 * 5 + 20 \% 7 / 3 - 12$$

*Rappel* : l'opérateur `/` réalise une division entière lorsque les opérandes sont des entiers (`7 / 2` vaut 3) et l'opérateur `%` renvoie le reste de la division entière (`7 % 2` vaut 1).

2. Sachant que les variables `a` et `b` de type `int` valent respectivement 2 et 16, évaluez l'expression booléenne suivante :

$$(a == 0) \&\& (a < b \&\& a > 0) \mid\mid (a > b \&\& a == 0)$$

3. Soient `x`, `y` et `z` trois variables entières de type `int`. Exécutez en séquence (l'une après l'autre) les 3 instructions suivantes :

```
1 x = 1;  
2 y = x++ + 5 * 3;  
3 z = ++x * 2;
```

Vous devez réécrire les instructions en précisant l'ordre d'évaluation par un parenthésage explicite. Combien valent `x`, `y` et `z` après chaque instruction. Schématisez l'état mémoire par un tableau indiquant la valeur de chaque variable après chaque instruction.

4. Soit `a` une année du calendrier grégorien (donc postérieure à 1582). Écrivez une expression booléenne qui indique s'il s'agit d'une année bissextile. Une année est bissextile si c'est un multiple de 4 sauf si c'est un multiple de 100. Toutefois, les années multiples de 400 sont tout de même bissextiles.

**Rappel** : Un nombre `n` est un multiple de `x` (avec `x`  $\neq$  0) si le reste de la division entière de `n` par `x` est nul.

5. Donnez la suite d'instructions permettant de convertir 100 dollars américains en euros. Le taux de conversion à appliquer est de 1.17 dollars pour 1 euro. Vos instructions doivent afficher le résultat sous la forme "`X dollars = Y euros`", les montants en dollars et en euros étant sur 2 décimales.
6. Reprenez l'exercice précédent et faites en sorte que le montant en dollars et le taux de conversion soient saisis par l'utilisateur.
7. Écrivez un programme demandant la saisie de 2 entiers à l'utilisateur puis affichant la plus grande des deux valeurs. Pour déterminer celle-ci, vous devez employer l'opérateur `?:` d'expression conditionnelle.
8. Adaptez le programme précédent à 3 entiers.

# IUT de Paris – Rives de Seine

## R1.01 – Initiation au développement

### Travaux Pratiques – Sujet n°1

## Thèmes

- Prise en main de Visual Studio
- Expressions, types de données, algorithmes simples

Chaque semaine, vous avez deux séances de travaux pratiques (TP). Lors de première séance de TP, vous traiterez dans l'ordre les exercices du sujet de la semaine. Lors de la seconde séance, et suivant votre avancement, vous devez continuer à passer sur machine les exercices de TP qui ne l'ont pas été en première séance.

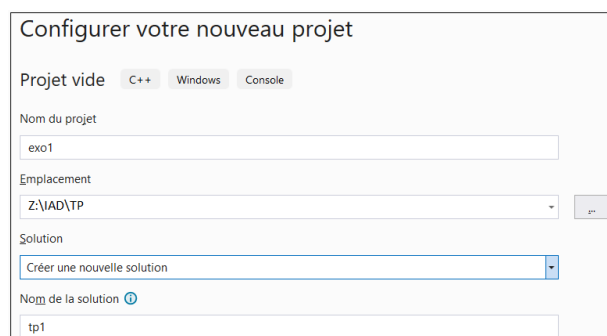
Une bonne pratique est aussi de tester sur machine les exercices de travaux dirigés (TD). Cette semaine, vous pourrez passer sur machine et tester le code en langage C correspondant aux exercices 4 à 8 du TD.

1. Lancez *Visual Studio* en tapant "visual" dans le menu **démarrer** de *Windows*, et en cliquant sur **Visual Studio 2022**. Lancer la création de votre premier projet en cliquant sur **Créer un projet**.

*Visual Studio* supporte beaucoup de langages différents. Vous devez créer un **Projet vide** de type **C++**, **Windows**, **Console**. Vous trouverez ce type de projet dans la liste qui se présente à vous. Après avoir cliquer sur **Suivant**, une boîte de dialogue vous invite à choisir le nom de votre projet, son emplacement et le nom de la solution.

Si vous voulez pouvoir récupérer votre travail d'une séance à l'autre vous devez impérativement sauvegarder votre travail sur votre disque personnel (Z:). Nous vous conseillons d'y créer un dossier spécifique (nommé **IAD** par exemple) et de sélectionner ce dossier en tant qu'emplacement de vos projets.

Dans *Visual Studio*, une *solution* permet de regrouper plusieurs projets dans un même ensemble. Nous vous recommandons de nommer votre premier projet **exo1** et de nommer la solution **tp1**. Vous pourrez ensuite ajouter de nouveaux projets (**exo2**, **exo3**, ...) à cette solution. Si vous avez suivi ces recommandation, la boîte de dialogue doit fortement ressembler à ceci :



Configurer votre nouveau projet

Projet vide C++ Windows Console

Nom du projet

exo1

Emplacement

Z:\IAD\TP


Solution

Créer une nouvelle solution

Nom de la solution

tp1

Corrigez ce qui doit l'être le cas échéant puis cliquez sur **Créer** pour finaliser la création de votre premier projet.

2. Ajoutez un nouveau fichier source à votre projet via le menu **Projet** » **Ajouter un nouvel élément...**. Dans la boîte de dialogue qui s'affiche, remplacez le nom proposé par **exo1.c**. Le nom du fichier est libre mais doit impérativement se terminer par ".c". Dans le cas contraire, *Visual Studio* ne pourra pas déterminer le bon compilateur à employer.  
Vous pouvez noter dans le bandeau à droite de l'écran que le nouveau fichier a été ajouté aux fichiers sources de votre projet.  
Vous êtes maintenant en position de taper votre premier programme dans la fenêtre principal. Écrivez un programme affichant "Bonjour" sur une première ligne et "Ceci est mon premier programme C avec Visual Studio" sur la ligne suivante.
3. Compilez votre projet en choisissant dans le menu **Générer** » **Générer la solution**. Si vous êtes un être humain normalement constitué, le compilateur doit reporter une liste d'erreurs dans la partie basse de l'écran. En double-cliquant sur la première erreur, *Visual Studio* met en surbrillance la ligne incriminée. Lisez attentivement le message d'erreur, étudiez la ligne correspondante, modifiez votre code en conséquence. Vous pouvez relancer la compilation après chaque correction. Il n'est pas rare qu'une correction permette de corriger plusieurs erreurs.  
Une fois l'ensemble des erreurs corrigées, le compilateur doit afficher, dans la partie basse de l'écran, le message "Build : 1 réussite(s), 0 échec(s), 0 à jour, 0 ignorée(s)". Vous êtes à présent en position pour exécuter votre programme. Pour ce faire, choisissez **Déboguer** » **Exécuter sans débogage** dans le menu. Une fenêtre d'exécution doit s'ouvrir. Vérifiez que les messages affichés sont ceux attendus. Le cas échéant, corrigez votre programme jusqu'à ce que le résultat soit le bon.  
Vous pouvez aller plus vite en cliquant directement sur la flèche verte ajourée  dans la barre d'outil en haut de votre écran ou en pressant la combinaison de touches **Ctrl** + **F5**. Si votre programme a été modifié, cliquer sur cette flèche ou taper cette combinaison va lancer la compilation et, en cas de succès, lancer l'exécution du programme.
4. Créer un second projet de même type (**Projet vide**, **C++**, **Windows**, **Console**), en lui donnant le nom **exo2** et sélectionnant **Ajouter à la solution** dans la boîte de dialogue de création de projet. Vous avez à présent deux projets au sein d'une même solution.  
**Attention**, avant de pouvoir lancer l'exécution de ce nouveau projet, vous devez sélectionner le nom du projet dans le bandeau à gauche de l'écran et choisir **Projet** » **Définir en tant que projet de démarrage** dans le menu. Si vous ne faites pas cela, c'est le premier projet qui sera exécuté. Le projet dont le nom est en gras dans la liste des projets est celui qui s'exécute.  
Ajoutez un nouveau fichier source à ce nouveau projet (**Projet** » **Ajouter un nouvel élément...**).  
Écrivez un programme demandant la saisie de deux nombres réels (un **double** et un **float**) et affichez leur moyenne. Indiquez **%f** pour saisir un float avec **scanf** et **%lf** pour saisir un double sous peine de récupérer une valeur totalement différente ! Par contre pour l'affichage avec **printf**, **%f** correspond aussi bien à un **double** qu'à un **float**.  
N'oubliez pas que *Visual Studio* (et plus précisément son compilateur) se méfie de **scanf**. Vous devez ajouter la directive suivante en préambule de votre programme pour pouvoir compiler sans erreur :  
**#pragma warning(disable : 4996)**  
Essayez de mettre **%f** au lieu de **%lf** dans le **scanf** lisant le **double** et voyez la différence. Vous pourrez remarquer que le compilateur rapporte un message d'avertissement et que les résultats sont éronnés si vous n'en tenez pas compte.
5. Au sein d'un nouveau projet, écrivez un programme déclarant un réel **x** de type **float** initialisé à **5.2f**, puis un réel **y** de type **double** initialisé à la même valeur et affichant le résultat du calcul **10000000 \* (x - y)**. Exécutez ce programme. Que remarquez-vous ?
6. Nous reprenons l'exercice 5 de la séance de travaux dirigés. Il s'agit ici de vérifier automatiquement que le résultat est correct en employant la fonction **assert** de la bibliothèque standard (fichier d'entête **assert.h**).  
Cette fonction prend en paramètre une expression logique. Si celle-ci est évaluée à faux, **assert** produit un message d'erreur et stoppe l'exécution du programme en indiquant le numéro de ligne où s'est produit l'erreur.

Employez cette fonction pour vérifier que le nombre d'euros calculé correspond bien à 100 dollars convertis au taux de 1.17 dollars pour 1 euro, c'est à dire 85.470085470085470085470085470085 euros.

Attention toutefois, à cause des problèmes d'arrondi, il n'est pas possible de vérifier le résultat avec l'opérateur d'égalité `==`. Vous devez vérifier la validité du résultat avec une marge d'erreur de plus ou moins 0.0001 euros.

7. Créez un dernier projet et réalisez un programme affichant pour chaque type de données élémentaire (voir le support de cours), la taille (en nombre d'octets) occupée en mémoire par une donnée de ce type (que vous pouvez obtenir via l'opérateur `sizeof`) ainsi que les valeurs minimale et maximale autorisées pour ce type. Ces 2 dernières informations doivent être affichée uniquement pour les types entiers. Pour afficher ces valeurs, vous devez employer les constantes définies au sein du fichier d'entête `limits.h` de la bibliothèque standard (<https://en.cppreference.com/w/c/types/limits>). Le format d'affichage doit être le suivant :

```
char (1 octets) valeurs entre -128 et 127
unsigned char (1 octets) valeurs entre 0 et 255
short (2 octets) valeurs entre -32768 et 32767
unsigned short (2 octets) valeurs entre 0 et 65535
int (4 octets) valeurs entre -2147483648 et 2147483647
unsigned int (4 octets) valeurs entre 0 et 4294967295
long (4 octets) valeurs entre -2147483648 et 2147483647
unsigned long (4 octets) valeurs entre 0 et 4294967295
long long (8 octets) valeurs entre -9223372036854775808 et 9223372036854775807
unsigned long long (8 octets) valeurs entre 0 et 18446744073709551615
float (4 octets)
double (8 octets)
long double (8 octets)
```

Vous pouvez vous limiter à quelques types parmi ceux ci-dessus.

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°2

## Thèmes

- Structures de contrôle alternatives
- Structures de contrôle itératives

1. Soient  $m$  un entier prenant une valeur de 1 à 12 et représentant un numéro de mois et  $a$  une année du calendrier grégorien, calculez le nombre de jours du mois  $m$  de l'année  $a$ .
2. Étant donnée une date  $j/m/a$  du calendrier grégorien, écrivez le programme modifiant les 3 variables pour les positionner au lendemain. Vous ferez l'hypothèse qu'une variable  $nbj$  est initialisée avec le nombre de jours du mois  $m$  dans l'année  $a$ .
3. En mathématiques, la factorielle d'un entier naturel  $n$  est notée  $!n$  et est définie comme suit. Si  $n$  est nul alors  $!n = 1$ , sinon  $!n = 1 \times 2 \times \dots \times n$ . Écrivez en pseudo-code l'algorithme calculant  $!n$  pour une valeur quelconque de  $n$ . Vous devez employer une boucle **tant que**.
4. Traduisez en C l'algorithme de l'exercice précédent. Vous devez employer une boucle **for**. Indiquez pour quelles valeurs de  $n$ , il est nécessaire de tester ce programme.
5. Écrivez un programme affichant tous les diviseurs d'un entier naturel non nul. Nous rappelons qu'un entier  $d$  est diviseur d'un entier  $n$  si le reste de la division de  $n$  par  $d$  est nul. Les diviseurs doivent être affichés sur une même ligne séparés par un espace.
6. Le plus grand commun diviseur ( $pgcd$ ) de deux entiers naturels  $a$  et  $b$  peut être calculé comme suit.

Si  $b \neq 0$  alors  $pgcd(a, b) = pgcd(b, a \% b)$ , sinon  $pgcd(a, b) = a$

Écrivez un programme affichant le  $pgcd$  de deux nombres saisis au clavier.

7. Un entier naturel est premier s'il n'accepte que deux diviseurs distincts : 1 et lui-même. Par exemple, 1 n'est pas un nombre premier ainsi que 4 alors que 2, 3 et 5 le sont.  
Écrivez un programme indiquant si un nombre saisi au clavier est premier ou s'il ne l'est pas.

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Pratiques – Sujet n°2

## Thèmes

- Structures de contrôle alternatives
- Structures de contrôle itératives
- Algorithmes

1. Écrivez un programme qui après la saisie de deux entiers affiche un des messages suivant selon leur valeur :
  - Le premier est plus grand que le second
  - Les deux se valent
  - Le second est plus grand que le premier
2. Un médecin de campagne peut selon le jour de la semaine être : **présent** (du lundi à vendredi), **en congé** (le dimanche) ou **d'astreinte** (le samedi). Selon le numéro du jour dans la semaine, affichez le statut du médecin ou un message d'erreur si le numéro du jour est incorrect. Le lundi est le jour n°1. Codez trois versions du programme en utilisant :
  1. des instructions `if/else`
  2. l'instruction `switch`
  3. l'opérateur conditionnel ternaire ?:
3. Soit  $N$  un entier tel que  $N \geq 1$ . Écrivez un programme affichant tous les entiers compris entre 1 et  $N$  en employant une boucle `while`, puis une boucle `do` et enfin une boucle `for`.
4. Modifiez le programme précédent pour les entiers soient affichés de  $N$  à 1.
5. Écrivez un programme qui invite l'utilisateur à saisir la taille (en cm) de chaque étudiant de la promotion. La saisie se termine lorsque l'utilisateur saisit la taille 0 (taille qui n'est pas à prendre ne compte). La saisie terminée, le programme doit afficher : la plus petite taille, la taille moyenne et la plus grande.
6. Écrivez un programme qui calcule la note de bac d'un étudiant. L'utilisateur saisit, de manière répétitive, un coefficient (un entier naturel non nul) suivi d'une note (un réel compris entre 0 et 20). La saisie est considérée comme terminée si le dernier coefficient est nul. Le programme affiche alors la moyenne de l'étudiant. Notez bien que cette moyenne est pondérée par les coefficients :  $\sum (coef_i \times note_i) / \sum coef_i$ . Vous ferez l'hypothèse que l'utilisateur ne fait pas d'erreur de saisie.
7. Nous devons gérer un compte en banque. Écrivez un programme qui, de façon répétitive, demande la saisie du type d'opération voulue (retrait ou dépôt), demande la saisie d'un montant et affiche, si nécessaire un message d'erreur si l'opération est impossible (opération inconnue ou solde insuffisant), et de manière systématique, le nouveau solde du compte en banque.

Le solde initial du compte est nul. Par souci de simplification, le type d'opération est caractérisé par un entier (0 pour un retrait et '1' pour un dépôt). Le montant de l'opération est toujours saisi sous une forme positive. Un dépôt est toujours possible, un retrait ne l'est que si le solde est suffisant. Pour interrompre le programme, l'utilisateur devra taper `ctrl-C`.

8. Un nombre de Kaprekar est un entier naturel qui, lorsqu'il est élevé au carré, peut être séparé en une partie gauche et une partie droite (non nulle) telles que la somme donne le nombre initial. Par exemple, 703 est un nombre de Kaprekar car  $703^2 = 494\,209$  et  $494 + 209 = 703$ .  
Écrivez un programme qui indique si un nombre est de Kaprekar ou ne l'est pas.



IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°3

## Thèmes

- Types énumérés
- Types composés
- Tableaux

1. Nous voulons manipuler un jeu de 32 cartes. Définissez un type nommé **Couleur** ne contenant que les 4 couleurs que peut avoir une carte. Les 4 couleurs doivent être définies de manière à ce que Trèfle < Carreau < Cœur < Pique.  
Faites de même avec un type nommé **Valeur**. Les valeurs croissantes des cartes sont sept, huit, neuf, dix, Valet, Dame, Roi et As. Vous devez faire en sorte que la valeur sept soit associée à l'entier 7 et que chacune des cartes suivantes prenne la valeur immédiatement supérieure à la précédente. En conséquence, un as doit valoir 14.
2. Donnez la définition d'un type nommé **Carte** associant une couleur et une valeur.
3. Écrivez un programme déclarant un tableau de carte nommé **paquet** et le remplissant de toutes les cartes possibles par ordre croissant (*i.e.* du deux de Trèfle à l'as de Pique). Veillez à ce que le tableau ait exactement la taille requise mais aussi à ne pas introduire de nombres magiques dans votre code.
4. Compléter le programme de la question précédente de manière à mélanger le paquet de carte. Pour ce faire, vous devez mettre en œuvre l'algorithme vu lors du premier cours et présenté à nouveau ci-dessous (sous une forme légèrement adaptée).

```
début
  pour i variant de 0 à 30 par pas de 1 faire
    Choisir au hasard une position p comprise entre i et 31;
    Permuter la carte du tableau se trouvant à l'indice i avec celle étant à l'indice p;
  fin
fin
```

Pour choisir une position au hasard, vous devez employer la fonction **alea** qui retourne un entier choisi aléatoirement entre une borne minimale et une borne maximale passées en paramètre. Par exemple, l'appel **alea**(10, 15) retourne une valeur choisie entre 10 et 15 (inclus). Cette fonction ne fait pas partie de la bibliothèque standard et, lors d'une autre séance, nous verrons comment l'écrire soit même.

5. Terminez le programme en affichant le paquet résultant. Vous devez afficher une carte par ligne en respectant le format suivant : "**Valet de Carreau**" pour une figure et "**10 de Cœur**" pour une carte ordinaire.

# IUT de Paris – Rives de Seine

## R1.01 – Initiation au développement

### Travaux Pratiques – Sujet n°3

## Thèmes

- Tableaux
- Structures de contrôle itératives
- Chaînes de caractères

1. Écrivez un programme qui déclare un tableau d'entiers dont la taille est fixée à l'aide d'une constante et dont le contenu est défini à la déclaration. Le programme doit afficher ce contenu.
2. Complétez votre programme pour qu'il copie le contenu du tableau dans un second tableau de même taille mais en inversant ses valeurs (la première doit se trouver en dernière position, la deuxième en avant-dernière position, ...). Enfin le programme doit afficher ce second tableau.
3. Complétez à nouveau votre programme. Il s'agit à présent d'inverser les valeurs du premier tableau sans se servir du second. Pour ce faire, vous devez permuter la valeur de la première case avec celle de la dernière, puis celle de la seconde avec celle de l'avant-dernière, ...  
Le programme doit vérifier (en employant `assert`) que les deux tableaux ont à présent les mêmes valeurs aux mêmes positions.
4. On dispose des définitions suivantes :

```
enum { LONGUEUR_MAX = 30, NB_ETUDIANTS_MAX = 20 };

typedef struct {
    char nom[LONGUEUR_MAX];
    unsigned int taille; // (en cm)
} Etudiant;
```

- Écrivez un programme qui déclare un tableau d'étudiants de taille `NB_ETUDIANTS_MAX` et qui le remplisse par les données (nom et taille) saisies par l'utilisateur. La saisie doit se terminer lorsque le nombre maximal d'étudiants est atteint ou que la taille entrée est nulle. Le programme doit alors afficher les noms et tailles du plus petit étudiant et du plus grand (le premier dans le tableau en cas d'égalité).
5. Compléter le programme de manière à afficher le nom et la taille de l'étudiant dont cette dernière est la plus proche de la moyenne. Si nécessaire, vous pouvez employer la fonction `fabs` de la bibliothèque standard (fichier d'entête `math.h`) qui retourne la valeur absolue d'un flottant.
  6. Écrivez un programme déclarant une chaîne de caractère de longueur maximale que vous choisirez. Faites en sorte que votre programme demande à l'utilisateur de saisir la chaîne, puis affiche la chaîne saisie encadré d'apostrophes (`'`), sa longueur (fonction `strlen`, fichier d'entête `string.h`) et son occupation mémoire (opérateur `sizeof` du C). Exécutez plusieurs fois votre programme et expliquez les différences.
  7. Vous devez réaliser un programme réalisant le cryptage de César d'une chaîne de caractère saisie au clavier. Ce cryptage est simple, il s'agit de décaler chaque lettre d'un nombre donné de positions. Si le décalage est de 2 alors les 'A' seront transformés en 'C', les 'B' en 'D', ..., les 'X' en 'Z', les 'Y' en 'A'

et les 'Z' en 'B'. Par exemple, la chaîne "TOTO" sera cryptée en "VQVQ". La valeur du décalage doit être stockée dans une constante (un entier naturel).

Préalablement au cryptage de la chaîne, votre programme doit la sauvegarder dans une autre variable. Vous devez ensuite compléter ce programme en décodant la chaîne obtenue et vérifier (`assert`) a posteriori que la chaîne a bien sa valeur initiale.

8. On veut calculer le chiffre magique du prénom d'une personne. Ce chiffre est obtenu en commençant par sommer la position dans l'alphabet de chaque lettre composant son prénom (1 pour 'A', 2 pour 'B', ...). Par exemple, pour "EVE", la somme donne  $5 + 22 + 5 = 32$ . Il s'agit ensuite de sommer les chiffres composant ce nombre en répétant cette opération jusqu'à obtenir un unique chiffre. Pour notre exemple, ce calcul s'arrête après une unique décomposition ( $32 \Rightarrow 3 + 2 = 5$ ). Le nombre magique de "EVE" est donc 5.

Écrivez un programme calculant le nombre magique d'un prénom saisi au clavier. Nous faisons l'hypothèse que la chaîne saisie ne contient que des caractères alphabétique (sans accent). Vous pouvez employer la fonction `toupper` (fichier d'entête `ctype.h`) qui retourne la majuscule d'une lettre.

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°4

## Thèmes

- Définition de fonction
- Invocation de fonction

1. Soit le programme suivant :

```
int main() { unGrosCalcul(); }
```

La fonction `unGrosCalcul` n'est pas précisée mais, comme son nom l'indique, elle réalise un traitement prenant un temps considérable. Il vous est demandé de transformer ce programme de manière à afficher le nombre de millisecondes pris par cet appel de fonction.

Pour ce faire, vous devez vous reposer sur la fonction `clock` de la bibliothèque standard (fichier d'entête `time.h`). Cette fonction a le prototype suivant :

```
// définition de type
typedef unsigned long clock_t;
// définition de constante
const clock_t CLOCKS_PER_SEC = /* dépend de l'implémentation */;
// déclaration de fonction
clock_t clock();
```

`clock` retourne le nombre de "ticks" consommés par l'application en cours d'exécution. Cela correspond à sa consommation CPU, et non pas à la durée écoulée depuis que l'application a démarré. Pour connaître le nombre de secondes correspondantes, il faut diviser la valeur retournée par `CLOCKS_PER_SEC`.

2. En vous inspirant du programme ci-dessous, définissez une fonction calculant la factorielle d'un entier naturel. Réécrivez le programme de manière à employer votre fonction.

```
#include <stdio.h>
#pragma warning(disable : 4996 6031)

int main() {
    unsigned int n;
    printf("entrez un entier naturel : ");
    scanf("%u", &n);
    unsigned int fact = 1;
    for (unsigned int i = 2; i <= n; ++i)
        fact *= i;
    printf("la factorielle de %u est %u\n", n, fact);
}
```

3. Programmez une fonction qui retourne le résultat de la division entière de deux entiers. Votre algorithme doit procéder en comptant le nombre de soustractions successives possibles. Si pour certaines valeurs

de paramètre, votre fonction ne peut pas calculer de résultat juste, elle doit prévenir l'appelant par un moyen ou un autre. Interrompre le programme (en invoquant `assert` de `assert.h` par exemple) est très raisonnable. Vous devez faire attention au signe des opérandes pour rendre un résultat correct.

Si nécessaire, vous pouvez employer la fonction `abs` de la bibliothèque standard (`stdlib.h`). Celle-ci retourne la valeur absolue de l'entier passé en paramètre.

4. Dans le même projet et à la suite de la fonction précédente, écrivez un programme qui invoque cette fonction et vérifie (en employant `assert`) que le résultat fourni est correct. Choisissez judicieusement les valeurs à tester et, en particulier, toutes les combinaisons possibles de signes.
5. Une suite célèbre de nombres est ainsi définie :
  - on choisit un entier strictement positif  $u_0$ .
  - si  $u_0$  est pair, on pose  $u_1 = u_0/2$ . Sinon, on pose  $u_1 = 3u_0 + 1$ .
  - si  $u_1$  est pair, on pose  $u_2 = u_1/2$ . Sinon, on pose  $u_2 = 3u_1 + 1$ .
  - on procède de la même façon pour construire  $u_{n+1}$  à partir de  $u_n$ .

Par exemple, si on part de  $u_0 = 15$ , alors  $u_1 = 3 \times 15 + 1 = 46$ , puis  $u_2 = 46/2 = 23$ . Le tableau qui suit donne les premiers termes :

$n$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$u_n$	15	46	23	70	35	106	53	160	80	40	20	10	5	16	8	4	2	1	4	2	...

Écrivez une fonction qui pour une valeur (légale) de  $u_0$  et un indice (légal)  $n$  calcule la valeur de  $u_n$ .

6. La *conjecture de Syracuse* consiste à supposer que pour n'importe quelle valeur de  $u_0$ , la suite correspondante prendra la valeur 1 à un certain moment. La plus petite valeur de  $n$  pour laquelle  $u_n$  vaut 1 est appelé le *temps de vol* de  $u_0$ . Pour  $u_0 = 15$ , le temps de vol est 17 (comme on peut le voir dans le tableau ci-dessus).

Écrivez une fonction qui, pour une valeur (légale) de  $u_0$ , calcule son temps de vol.

# IUT de Paris – Rives de Seine

## R1.01 – Initiation au développement

### Travaux Pratiques – Sujet n°4

## Thèmes

- Fonctions
- Débogage

1. Recopiez le fichier `tp4.c`<sup>1</sup>. Créez un projet dans *Visual Studio* et ajoutez la copie de ce fichier à ce projet (en choisissant `Projet` » `Ajouter un élément existant` dans le menu). Lisez le fichier source, compilez le et exécutez le sans l'outil de mise au point (en cliquant sur `▶` dans la barre d'outil, en pressant la combinaison de touches `Ctrl`+`F5` ou encore en choisissant `Déboguer` » `Démarrer sans débogage` dans le menu). Assurez vous que ce qui est affiché correspond à ce que vous avez compris.

Ajoutez une valeur illégale (donc négative) dans le tableau `rayons` déclaré dans la fonction `main`. Compilez et exécutez le code à nouveau. Que remarquez-vous ?

2. Relancez l'exécution en employant l'outil de mise au point (en cliquant sur `▶ Débogueur Window local` dans la barre d'outil, en pressant la touche `F5` ou encore en choisissant `Déboguer` » `Démarrer le débogage` dans le menu). Cliquez sur le bouton `Recommencer` dans la fenêtre qui s'affiche.

L'exécution du programme est à présent sous le contrôle du débogueur. Cet outil va vous permettre d'observer le contenu des variables, d'identifier les appels de fonctions successifs qui ont conduit à l'erreur et d'observer le contexte d'exécution de chacun de ces appels.

L'instruction ayant provoqué l'erreur est signalée par une croix blanche sur fond rouge. C'est la ligne 13 de notre programme. Vous pouvez fermer la fenêtre explicative du débogueur qui indique qu'un point d'arrêt a été atteint.

Le cadre en bas à gauche montre les variables et leur valeur. En bas de ce cadre, vous pouvez choisir entre une sélection `Automatique` des variables ou l'ensemble des `Variables locales`. La vue `Automatique` nous montre que la variable `r` qui pose problème a bien la valeur négative que vous avez ajouté dans tableau `rayons` de la fonction `main`.

Le cadre en bas à droite montre la pile des appels de fonction. Si vous regardez en bas de la pile (en utilisant l'ascenseur de ce cadre), vous pouvez observer que la fonction `main` a invoqué la fonction `affiche` qui elle-même a invoqué la fonction `aire`. En double-cliquant sur l'un ou l'autre de ces appels, vous pouvez voir quelle ligne de la fonction a été atteinte par l'exécution en cours. De même, vous pouvez observer dans le cadre en bas à gauche la valeur des variables locales lors de l'appel. Ainsi, la valeur de la variable `i` de la fonction `main` nous précise quelle est la position dans le tableau `rayons` de la valeur qui pose problème.

Pour terminer l'exécution du programme, vous devez soit cliquer sur `■` dans la barre d'outil, soit presser la combinaison de touches `⇧`+`F5`, soit sélectionner `Déboguer` » `Arrêter le débogage` dans le menu.

Corrigez le programme en retirant la valeur incriminée.

---

1. Ce fichier est sur MOODLE. Il est aussi joint au présent document et est accessible dans les signets. Toutefois, tous les *viewers PDF* ne supportent pas l'extraction de fichiers attachés.


3. Le débogueur permet aussi d'exécuter pas à pas (*i.e.* instructions après instructions) vos programmes. Pour ce faire, vous devez commencer par choisir à partir de quelle instruction vous voulez contrôler l'exécution. Positionnez votre curseur sur la première instruction de la fonction `main` (ligne 28), faites un clic droit et sélectionnez `Exécuter jusqu'au curseur` dans le menu qui s'affiche (ou pressez les touches `Ctrl`+`F10`). L'exécution du programme est lancée jusqu'à ce que cette instruction soit atteinte. Le débogueur vous permet à présent de contrôler la suite de cette exécution. Notez que les cadres en bas à gauche et à droite sont mis à jour et qu'à tout moment vous pouvez observer le contenu des variables ainsi que naviguer dans la pile des appels. Les options qui s'offrent à vous sont réunies dans le menu `Déboguer` et ont leur équivalent dans la barre d'outils et par une combinaison de touches. Les principales options sont les suivantes :

- `Continuer` reprend l'exécution normale du programme.
- `Arrêter le débogage` stoppe l'exécution en cours.
- `↓ Pas à pas détaillé` passe à l'instruction élémentaire suivante. Si l'instruction courante comprend un appel de fonction, l'exécution de celle-ci sera elle aussi pas à pas.
- `↶ Pas à pas principal` passe à l'instruction suivante de la fonction courante. Si l'instruction courante comprend un appel de fonction, l'exécution de cet appel ne sera pas détaillée.
- `↑ Pas à pas sortant` termine l'exécution de la fonction courante et s'arrête là où elle a été invoquée.

Les trois dernières options vous permettent de choisir le niveau de détail que vous voulez obtenir. Par exemple, si l'instruction courante est un appel d'une fonction que vous savez correcte, vous pouvez l'exécuter en une fois en choisissant `↶ Pas à pas principal`. Si par contre, vous avez un doute, vous pouvez l'exécuter instruction par instruction en choisissant `↓ Pas à pas détaillé`.

Notez que vous pouvez modifier le contenu d'une variable durant l'exécution en double-cliquant sur sa valeur dans le cadre en bas à gauche. Cela peut être utile pour tester une correction sans la réaliser dans le code.

Expérimentez les différentes options de contrôle de l'exécution.

4. Nous nous intéressons à la fonction `aire` et souhaitons observer son comportement. Nous pouvons demander au débogueur de nous donner le contrôle de l'exécution chaque fois que celle-ci est invoquée. Pour ce faire, il suffit de définir un *point d'arrêt* sur l'instruction qui nous intéresse. Placez votre curseur sur la dernière instruction de cette fonction (ligne 16). Faites un clic droit et choisissez l'option `Point d'arrêt`  `Insérer un point d'arrêt` dans le menu qui s'affiche (ou en cliquant dans le bandeau grisé à gauche des numéros de ligne du code). Un point rouge à gauche du numéro de ligne symbolise la présence du point d'arrêt. Démarrer le débogage (`F5`). Le programme s'exécute jusqu'à atteindre le point d'arrêt et le débogueur vous donne le contrôle de l'exécution. Vous pouvez observer les valeurs de variables locales (`a` vaut 7.06859970) et la pile des appels. Si vous cliquez sur l'appel de la fonction `affiche`, vous pouvez remarquer que la variable `a` de cette dernière n'a rien à voir avec la variable précédente (elles n'ont pas la même valeur). Cela nous confirme que les variables locales des fonctions sont propres à chaque appel. Cliquez sur `Continuer`. L'exécution reprend normalement jusqu'à être stoppée lorsque le point d'arrêt est de nouveau atteint. Vous pouvez mettre autant de points d'arrêt que vous le souhaitez. Pour en supprimer un, il suffit de cliquer sur le point rouge qui le symbolise à gauche des numéros de ligne. Tous les points d'arrêt déjà définis sont listés dans l'onglet `Points d'arrêt` du cadre en bas à droite. À partir de ce même onglet, il est possible de tous les supprimer, de les configurer un à un, ...
- Expérimentez les différentes possibilités offertes par les points d'arrêt.

5. Vous devez programmer un jeu simple où interviennent un arbitre et un joueur. L'arbitre choisit un entier naturel dont la valeur est inférieure à une limite connue de tous. Le joueur propose successivement des valeurs et l'arbitre lui indique à chaque fois si la valeur à trouver est inférieure ou supérieure à la valeur annoncée. La partie cesse dès que le joueur propose la bonne valeur ou lorsque le nombre maximal de tentatives est atteint. Dans ce dernier cas, la partie est perdue.

Le rôle de l'arbitre est simple et peut être totalement automatisé. Il choisit une valeur aléatoire (voir plus loin), puis répond honnêtement aux propositions du joueur. Votre programme doit comprendre un arbitre automatique.

Pour le joueur, votre programme doit proposer (une ou) des stratégies parmi les suivantes :

- Le joueur est un humain. Les valeurs proposées sont saisies au clavier.

- Le joueur est automatique et joue des valeurs choisies aléatoirement.
- Le joueur est automatique et joue stratégiquement. Il analyse les réponses de l'arbitre et restreint coup après coup l'espace des solutions.

L'objectif de cet exercice est de décomposer le problème en sous-problèmes et que chacun d'entre eux soit transcrit par une fonction ayant une taille raisonnable. De plus, les différentes stratégies implémentées doivent bénéficier au plus de ces fonctions en les invoquant évitant ainsi du code qui serait redondant.

Pour obtenir des valeurs aléatoires, vous pouvez vous reposer sur les fonctions `srand` et `rand` (`stdlib.h`). `srand` initialise le générateur de nombres aléatoires et est généralement invoquée qu'une unique fois au début du programme. Si vous ne le faites pas, la série aléatoire sera systématiquement la même.

À chaque appel, `rand` retourne le prochain entier de la série, il est toujours compris entre 0 et `RAND_MAX` (une constante, elle aussi issue de `stdlib.h` et valant  $2^{15} - 1 = 32767$ ). Le programme suivant illustre l'utilisation de ces deux fonctions.

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

// Affiche 20 nbres aléatoires
// compris entre 0 et (MAX - 1)
void demo() {
    const int MAX = 10;
    for (int i = 0; i < 20; ++i) {
        int n = rand() % MAX;
        printf("%d ", n);
    }
    printf("\n");
}

// Essais avec plusieurs séries aléatoires
int main() {
    srand(1); // essai avec la série n°1
    demo();

    srand(2); // puis avec la série n°2
    demo();

    // puis avec un numéro de série aléatoire
    unsigned int num = (unsigned int)time(NULL);
    srand(num); // choix de la série n°num
    demo();
}
```

Le cas où les valeurs aléatoires sont distinctes à chaque jeu est assuré par le dernier essai ci-dessus. La fonction `time` (`time.h`) retourne une valeur dépendant de l'heure précise à laquelle est lancé le programme. C'est les conditions normales de jeu.

Par contre, pour déboguer, il est important de pouvoir rejouer une même partie où vous auriez vu une erreur. En observant le contenu de la variable `num` de la fonction `main`, vous pouvez identifier le numéro de la série et donc la rejouer (placer un point d'arrêt, modifier le contenu de `num` puis reprendre l'exécution tout en la contrôlant).

Une alternative consiste à mettre au point son programme en le testant avec des numéros de série connus. C'est ce qui est fait dans les deux premiers essais ci-dessus.



## Annexe

Le fichier des exercices 1 à 4.

```
1 #include <assert.h>
2 #include <stdio.h>
3
4 // Calcule la puissance carré de f
5 float carre(float f) {
6     float c = f * f;
7     return c;
8 }
9
10 // Calcule l'aire d'un cercle de rayon r
11 // Précondition : r est positif
12 float aire(float r) {
13     assert(r >= 0.);
14     const float PI = 3.1416f;
15     float a = PI * carre(r);
16     return a;
17 }
18
19 // Affiche le rayon et l'aire d'un cercle de rayon r
20 void affiche(float r) {
21     printf("un cercle ayant un rayon de %.2fcm", r);
22     float a = aire(r);
23     printf(" occupe une surface de %.2fcm^2\n", a);
24 }
25
26 // Teste les fonctions ci-dessus pour quelques valeurs
27 int main() {
28     float rayons[] = { 1.5, 3., 10. };
29     int nb = sizeof(rayons) / sizeof(float);
30     for (int i = 0; i < nb; ++i)
31         affiche(rayons[i]);
32 }
```

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°5

## Thèmes

- Pointeurs
- Mode des paramètres
- Tableaux et types composés

1. Soit la fonction suivante :

```
1 void sort(int t[], int nb) {  
2     for ( ; nb > 1; --nb) {  
3         int m = 0;  
4         for (int i = 1; i < nb; ++i)  
5             if (t[i] > t[m])  
6                 m = i;  
7         int tmp = t[nb - 1];  
8         t[nb - 1] = t[m];  
9         t[m] = tmp;  
10    }  
11 }
```

Écrivez un programme qui déclare un tableau d'entiers, l'initialise avec des valeurs non-triées et invoque la fonction ci-dessus de manière à le trier par ordre croissant.

2. Écrivez une fonction qui vérifie qu'un tableau est trié par ordre croissant. Votre fonction doit se baser sur `assert` pour interrompre le programme si le tableau n'est pas trié. Modifiez le programme précédent pour vérifier que `sort` trie bien le tableau.
3. Les instructions des lignes 3 à 6 de la fonction `sort` initialisent la variable `m` avec l'indice de la case ayant la plus grande valeur parmi les `nb` premières cases de `t`. Extrayez ces instructions pour en faire une fonction et remplacez les par un appel à cette fonction.
4. Les instructions des lignes 7 à 9 de la fonction `sort` permute le contenu des deux cases de `t` situées aux indices `(nb - 1)` et `m`. Extrayez ces instructions pour en faire une fonction et remplacez les par un appel à cette fonction.

Celle-ci peut recevoir le tableau et les deux indices des cases à permuter, ou alternativement, uniquement les adresses de ces deux cases. Programmez la solution qui vous semble la meilleure et expliquez votre choix.

5. Nous disposons des définitions suivantes :

```
enum { MAX_NOM = 30, MAX_ETU = 175, MAX_EPR = 50 };  
  
typedef struct {  
    int nbEtudiants;  
    char etudiants[MAX_ETU][MAX_NOM];  
};
```

```
    int nbEpreuves;  
    float notes[MAX_ETU][MAX_EPR];  
} Promo;  
  
const float NOTE_MIN = 0.f, NOTE_MAX = 20.f;
```

Écrivez une fonction qui applique à certains étudiants d'une promotion un bonus (ou un malus) à une épreuve donnée. Les étudiants doivent être désignés par leur indice dans le tableau d'étudiants, l'épreuve par son indice dans les tableaux de notes et le bonus/malus par un flottant. L'opération est une bonification si ce dernier est positif (et un malus dans le cas contraire). Vous devez veiller à ce que les notes restent comprises entre `NOTE_MIN` et `NOTE_MAX`.

# IUT de Paris – Rives de Seine

## R1.01 – Initiation au développement

### Travaux Pratiques – Sujet n°5

## Thèmes

- Fonctions
- Algorithmes

À partir d'une position donnée dans un labyrinthe, un joueur doit se déplacer pour rejoindre une position d'arrivée. Le jeu est pimenté par le fait qu'il se déroule dans le noir et que le joueur peut de ce fait entrer en collision avec les murs.

Une première implémentation du jeu vous est proposée. Le labyrinthe est considéré comme étant carré. Le joueur indique successivement dans quelle direction il souhaite se déplacer et le programme lui indique si le déplacement a pu se faire ou pas. Le labyrinthe n'est jamais affiché (car le joueur est dans le noir). Le programme s'arrête dès que le joueur atteint la position d'arrivée.

1. Recopiez le fichier `tp5.c`<sup>1</sup>. Créez un projet dans *Visual Studio* et ajoutez la copie de ce fichier à ce projet (en choisissant `Projet` » `Ajouter un élément existant` dans le menu). Lisez le fichier source en portant une attention particulière à la définition du type `Cellule` et des constantes `TAILLE`, `TERRAIN`, `DIR` et `NB_DIR`. Corrigez le code de la fonction `estValide`. Cette fonction doit retourner la valeur 0 si les coordonnées passées en paramètre désignent une case en dehors du terrain ou une case contenant un mur. Dans le cas contraire, elle doit retourner une valeur différente de 0. Actuellement, elle retourne systématiquement la valeur 0 interdisant de fait tout déplacement.

Compilez votre projet et jouez une partie.

2. Le programme actuel juge le résultat du joueur en comparant le nombre de déplacements qu'il a réalisé au nombre minimal requis pour atteindre le point d'arrivée. Ce nombre minimal devrait être calculé par la fonction `distance` mais, pour l'instant, celle-ci retourne une valeur inscrite en dur.

Notre objectif est de faire en sorte qu'elle calcule automatiquement ce nombre minimal en fonction du labyrinthe et des points de départ et d'arrivée.

Le principe consiste à calculer pour chaque case accessible sa distance minimale (en nombre de déplacements) par rapport au point de départ.

Initialement, seule la distance du point de départ à lui-même est connue (0 déplacement). L'étape suivante consiste à affecter la distance 1 à toutes les cases voisines du point de départ. L'étape suivante consiste à affecter la distance 2 à toutes les cases voisines de celles identifiées à l'étape précédente et ainsi de suite. Bien entendu, la distance d'une case ne doit être affectée que si sa valeur n'a pas été définie au préalable.

Proposez une structure de données permettant de mémoriser les données mises en œuvre par cet algorithme. Définissez en C le (ou les) type(s) correspondant(s).

3. L'algorithme 1 est une formalisation de la description précédente. Implémentez cet algorithme dans la fonction `distance`. Vérifiez que la valeur calculée est la bonne (6). Changez le point de départ pour la coordonnée (0, `TAILLE` - 1) et vérifiez que la valeur calculée est toujours correcte (9).

---

1. Ce fichier est sur MOODLE. Il est aussi joint au présent document et est accessible dans les signets. Toutefois, tous les viewers PDF ne supportent pas l'extraction de fichiers attachés.

---

**Algorithme 1** : Calcul de la distance minimale de  
chaque case du terrain

---

```
début
  dep ← 0;
  répéter
    pour chaque case dont la distance est égale à dep
      faire
        pour chaque case voisine non encore visitée faire
          lui associer la distance  $dep + 1$ ;
          la marquer comme étant visitée;
        fin
      fin
    fin
    dep ← dep + 1;
  tant que une nouvelle case a été marquée;
  si la case d'arrivée a été visitée alors
    retourner sa distance;
  sinon
    retourner une valeur particulière;
  fin
fin
```

---

4. Transformez l'algorithme ci-dessus de manière à ce que le résultat soit retourné dès qu'il est connu.
5. Les étudiants ayant terminé les exercices précédents doivent travailler sur leur projet. Les prochaines séances de travaux pratiques seront consacrées à ce travail. Lors de la dernière séance, vous devrez présenter votre projet à votre chargé de TP et cette présentation sera évaluée.

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°6

## Thèmes

- Choix des structures de donnée
- Conception d’algorithmes
- Complexité des algorithmes

Nous faisons l’hypothèse que nous sommes dans un monde parfait dans lequel les devises ”les amis de mes amis sont mes amis” et ”mes amis sont mes amis pour la vie” ont été adoptées par chacun. De plus, seuls des immortels infertiles peuplent ce monde (presque) parfait. Le nombre d’habitants est connu et il n’évolue plus.

Lorsque deux habitants de ce monde se rencontrent, ils ont souvent à décider s’ils sont déjà amis ou pas. Pour répondre à cette question, chacun doit énumérer ses amis actuels afin de détecter s’ils ont des amis communs. Cette tâche (laborieuse et source d’erreur) est si fréquente qu’il a été décidé de proposer une solution informatique.

Le système que vous développerez pourra être interrogé par chacun pour savoir si deux personnes sont connues comme étant amies. Dans le cas contraire, il devra être possible d’enregistrer leur amitié nouvelle auprès du système.

Chaque habitant sera désigné dans le système par un numéro qui lui est propre (allant de 0 à  $n - 1$  avec  $n$  le nombre d’habitants). Le système devra être initialisé de façon à ce qu’aucun habitant n’ait d’ami.

Tous les algorithmes demandés devront être exprimés en pseudo-code.

### 1. Première analyse

Proposez une structuration de donnée permettant d’encoder la relation d’amitié liant les habitants. Indiquez dans la mesure du possible le coût (la complexité temporelle et spatiale) des deux fonctionnalités devant être assurées par le système. Pour ce faire vous devrez imaginer et exprimer (en pseudo-code) les algorithmes correspondants.

### 2. Une structuration alternative

Une structuration des données bien connue pour ce problème permet d’obtenir des algorithmes de très faible complexité (quasi constante).

Le principe général consiste à désigner au sein de chaque groupe d’amis, un représentant (unique) du groupe. Comme initialement personne n’a d’ami, chaque habitant est le représentant de son propre groupe.

La structure de données est constituée d’un simple tableau (nommé *amis*) contenant une case par habitant. L’indice d’une case correspond au numéro de l’habitant représenté par la case.

Dans chaque case du tableau est stocké l’indice d’une personne appartenant au même groupe d’amis (le représentant du groupe par exemple).

Initialement, chaque habitant est seul dans son groupe et en est le représentant. Donnez le tableau dans sa configuration initiale pour un monde composé de 4 habitants. Donnez le pseudo-code de cette initialisation pour une population de  $n$  habitants.

### 3. Premier algorithme

Un algorithme essentiel de cette structuration est savoir déterminer le représentant du groupe d'amis d'une personne donnée. Les représentants de groupe sont les seuls pour lesquels la valeur de la case est égale à son indice. Pour une personne donnée, trouver le représentant du groupe auquel elle appartient revient à suivre la suite d'indices formée par les cases jusqu'à trouver un représentant. Par exemple, considérons la situation suivante (avec une population de 6 habitants) :

	0	1	2	3	4	5
<i>amis</i>	0	0	1	3	0	3

Déterminer le représentant du groupe auquel appartient 2 revient à parcourir les cases d'indice 2, 1 puis 0.

De ce tableau, nous pouvons déduire que 0 est le représentant du groupe auquel appartiennent 1 et 4, que 2 a le même représentant que 1 (et donc que c'est 0 qui représente ce groupe) et que 3 est le représentant du groupe auquel appartient 5. On peut en conclure que  $\{0, 1, 2, 4\}$  ainsi que  $\{3, 5\}$  sont les deux groupes d'amis.

Nous verrons par la suite comment nous assurer que le tableau ne contienne pas de chaîne d'indices formant un circuit non élémentaire (*i.e.* ne passant jamais par un représentant de groupe).

Donnez le pseudo-code de l'algorithme déterminant le représentant d'un groupe auquel appartient une personne donnée.

### 4. Algorithme complémentaire

Deux personnes sont des amis si leurs groupes respectifs ont le même représentant. L'algorithme est immédiat en se basant sur la solution de l'exercice précédent.

Lorsque deux personnes sont déclarées comme étant liées d'amitié alors que ce n'est pas encore le cas, le principe consiste à indiquer que le représentant du second groupe est à présent le représentant du premier groupe (ceci nous assure que le tableau ne contiendra jamais de circuit d'indice). Pour ce faire, il est suffisant d'affecter une seule case du tableau. Déterminez laquelle et donnez le pseudo-code de l'algorithme correspondant.

### 5. Complexité

Imaginons un monde de 10 personnes. À partir de la situation initiale, les personnes 0 et 1, puis 1 et 2, ..., jusqu'à 8 et 9 sont successivement déclarées amis. Donnez la situation finale du tableau et déduisez en la complexité dans le pire des cas des deux précédents algorithmes (cf. exercices 3 et 4).

### 6. Première optimisation

L'idée est de profiter du calcul du représentant du groupe auquel appartient un habitant pour compresser le chemin d'indices parcouru. Cette compression permettra d'accélérer les futures recherches impliquant les habitants correspondants à ces indices.

Le principe consiste à mettre à jour chaque case visitée avec le contenu de la future case visitée. Par exemple, dans le tableau illustrant l'exercice 3, le calcul du représentant de 2 conduit à visiter les cases 2, 1 et 0. Lors de ce parcours, il est facile d'affecter à *amis*[2] la valeur de *amis*[1] puis à *amis*[1] celle de *amis*[0]. Ainsi les recherches devant visiter la case d'indice 2 sont à présent plus rapides.

Donnez la valeur du tableau de l'exercice 3 après la compression due au calcul du représentant de 2 ainsi que le pseudo-code de l'algorithme le réalisant.

### 7. Une optimisation plus radicale

Une façon plus coûteuse mais plus efficace en termes de compression consiste à calculer la valeur du représentant puis à affecter cette valeur à toutes les cases impliquées dans la recherche. Donnez le pseudo-code de l'algorithme intégrant cette nouvelle compression.

IUT de Paris – Rives de Seine  
R1.01 – Initiation au développement

Travaux Dirigés – Sujet n°7

## Thèmes

- Révisions
- Fonctions et paramètres
- Domaine de définition d'une fonction
- Préconditions et vérification d'utilisation d'une fonction

1. Quel est l'affichage résultant de l'extrait de code qui suit.

```
int i = 2, k = 3;
int* p1 = &i, *p2 = &k;
*p1 -= 4;
*p2 *= 3;
printf("%d %d\n", i, k);
```

2. Soit le prototype d'une fonction nommée `fiche` :

```
void fiche(float *x, float *y, int i, char z, char r ) ;
```

et les déclarations de variables suivantes :

```
float a, c ;
int j ;
char b, h ;
```

Supposons que toutes ces variables sont correctement initialisées. Donnez le numéro de tous les appels corrects de la fonction `fiche` :

1. `fiche (a, c; j ; b, h);`
  2. `fiche (&a, &b, c, j, h);`
  3. `fiche (&a, &c, 3, 'b', b);`
  4. `fiche (a; j; b; h);`
  5. `fiche (&a, &c, j, b, h);`
3. Documentez et déclarez une fonction `f` qui au nombre réel `x` donne pour résultat  $1 / (2 - x)$ . Donnez la ou les préconditions de `f`.
4. Donnez l'assertion qui permet de vérifier que l'appel de la fonction `f` pour 1.0 retourne bien 1.0.
5. Définissez la fonction `f`. Si besoin, vérifiez l'utilisation de `f`.
6. Testez la fonction `f` dans un programme principal.