

TP 1 : Notions de base en programmation C

1 Avant-Propos

Les TP et documents de TP du cours de programmation C ont plusieurs objectifs :

- Vous permettre de vous assurer par la pratique que vous maîtrisez certains concepts de la programmation en C. Le but n'est pas juste de comprendre mais de faire par vous même (pas par une IA).
- Vous fournir une documentation de base sur le langage C.
- Vous permettre de découvrir un environnement de programmation minimal dans lequel rien n'est implicite. N'utilisez pas d'IDE dans ce cours, car le fait que ces environnements de programmation soient utiles est précisément ce qui les rend inadaptés à un des objectifs pédagogiques de ce cours qui est de vous montrer les liens directs entre les aspects machine et logiciel.
- Vous donner une expérience de génie logiciel là encore de bas niveau : les projets et programmes sont simples. C'est plus facile pour prendre de bonnes habitudes quant à la forme de votre code (indentation, choix de nom de variables etc.).
- Ces TP ne présupposent aucune connaissance préalable de la programmation en C.

Ces TP ne sont pas notés. Il est par contre attendu que vous appreniez et compreniez tout ce qu'ils contiennent.

Si des points ne sont pas clairs il ne faut pas faire l'impasse ou rester bloqué dessus. Les enseignants sont là pour vous aider, mais pas pour faire le TP à votre place.

Ce ne sont pas des TP d'algorithmiques. La difficulté des questions n'est pas dans la logique des problèmes à résoudre (qui sont simples) mais dans la manière de le faire en utilisant les spécificités du langage C.

2 Objectifs

Ce premier TP abordera les éléments fondamentaux suivants de la programmation C :

- Commandes de compilation et d'exécution de programmes C.
- Exemple de programme C "Hello World!".
- Types primitifs de données.
- Tableaux, chaînes de caractères.
- Structures de contrôle : if, for, while, switch-case, default, break, continue.
- Fonctions.
- Questions subsidiaires

3 Rappel des principales commandes

3.1 Commande de compilation

Pour les TP de C vous devez avoir accès à une fenêtre de terminal de commandes de type Linux/Unix et un éditeur de texte basique, pas d'IDE !

Ci-dessous un exemple de commande pour compiler un programme `prog1.c` dans le terminal :

```
gcc -o prog prog1.c
```

Le fichier `prog1.c` contient le programmes en C. Cette commande permet de le compiler, c'est à dire de produire un programme exécutable (binaire) à partir du texte écrit en C. L'exécutable sera enregistré dans le fichier `prog`.

3.2 Commande d'exécution

Ci-dessous un exemple de commande d'exécution d'un programme contenu dans un fichier `prog` à donner au terminal :

```
./prog
```

4 Un exemple minimal complet : "Hello World !"

Le programme ci-dessous donne un exemple de programme C affichant le message "Hello World !".

```
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

Un programme C utilise une bibliothèque qui permet d'exécuter certaines fonctions nécessaires au programme. En l'occurrence, la fonction élémentaire `printf`, qui permet d'afficher un message à l'écran, est fournie par la bibliothèque `stdio`. Cette bibliothèque est donc incluse de la manière suivante dans le programme :

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

La deuxième partie du programme est le code exécuté par le programme. Ce code est contenu dans la fonction principale nommée `main`. Ce nom est un mot-clé du langage, vous ne pouvez pas choisir un autre nom.

```
int main() {
    ... code de la fonction principale du programme
}
```

Le mot-clé `int` indique que la fonction `main` retourne un nombre entier en guise de résultat. Le nombre retourné par la fonction indique si l'exécution du programme s'est déroulée correctement. Classiquement, lorsque le programme s'exécute correctement, il doit retourner le nombre 0. Si l'exécution du programme échoue, le résultat retourné doit être différent de 0, et peut indiquer différentes valeurs pour différents types d'échec de programme. Cela peut être utile quand plusieurs programmes sont enchaînés, le code d'échecs peut être réutilisé par ces autres programmes (vous verrez cela dans le cours de système d'exploitation notamment).

Dans cet exemple de programme, le programme retourne 0 pour indiquer qu'il s'est correctement exécuté :

```
return 0;
```

A noter que chaque instruction dans un programme C se termine par un point-virgule (;). Même si votre programme ne comporte qu'une seule ligne il faut la terminer par un point-virgule.

QUESTION 1 ►

Ecrire le programme "Hello World !" dans un fichier intitulé `hello.c`

Exécuter la ligne de commande de compilation de ce programme.

Puis exécuter la ligne de commande d'exécution de ce programme.

Modifier le programme pour qu'il affiche deux messages : "Hello World !" puis "Goodbye World !"

Compiler de nouveau et exécuter cette nouvelle version.

5 Types primitifs de données

5.1 Liste des types primitifs

Le langage C permet de manipuler différents types de données, parmi lesquels les types primitifs suivants :

- Les caractères : `char`
- Les nombres entiers signés : `int`, `short int` (ou `short`), `long int` (ou `long`), `long long int` (ou `long long`), et également `char`.

- Les nombres entiers positifs : `unsigned int`, `unsigned short`, `unsigned long`, `unsigned long long`, et `unsigned char`.
- Les nombres réels positifs ou négatifs : `float`, `double`, `long double`.

On notera que le langage C ne possède pas de type booléen primitif. La valeur entière 0 représente la valeur booléenne `false` et tout autre entier représente le `true`. Il est possible par exemple d'utiliser la notation suivante pour plus de lisibilité :

```
#define FALSE 0
#define TRUE 1
```

Une autre possibilité est d'utiliser la librairie `stdbool.h` qui, entre autres, définit les constantes `true` et `false` et permet de déclarer des variables de type `bool`.

5.2 Notations décimales, octales, hexadécimales de nombres entiers

C permet d'écrire les entiers de différentes manières. C'est à dire que pour le même entier `n` il est possible de l'écrire dans un programme C :

- en notation décimale. Le nombre est représenté par une suite de chiffres compris 0 et 9.
- en notation octale. Le nombre est préfixé par 0 (zéro) suivi d'une suite de chiffres compris entre 0 et 7.
- en notation hexadécimale. Le nombre est préfixé par 0x et suivi d'une suite de chiffres compris entre 0 et 9 ou de lettres choisies dans l'ensemble {A, B, C, D, E, F}.

QUESTION 2 ►

Donner la valeur décimale de chacun des nombres suivants.

Expliquer la méthode de calcul pour obtenir ces valeurs.

- 0777
- 0x9A8

QUESTION 3 ►

Donner la valeur hexadécimale de chacun des nombres suivants.

Expliquer la méthode de calcul pour obtenir ces valeurs.

- 100
- 06401

5.3 Affichage de valeurs

L'affichage de valeurs peut se faire avec la fonction `printf` incluse dans la bibliothèque `stdio.h`. Cette fonction comprend un premier argument qui est une chaîne de caractères à imprimer. Dans cette chaîne de caractères peuvent apparaître des constructions précédées de `%`. Ce sont des formats d'impression. Pour chaque `%` correspond une valeur à afficher qui sera donnée en argument à `printf`. Par exemple

```
printf("%d\t%d\n", farhenheit, celsius);
```

comprend deux constructions `%` qui sont ici des nombres entiers. Cette instruction va donc imprimer les deux entiers qui sont les valeurs des variables `farhenheit` et `celsius` séparés par une tabulation (`\t`) et va sauter une ligne (`\n`).

Il existe beaucoup de formats et de constructions spécifiques ; des exemples sont donnés dans le Tableau ???. Reportez vous à la documentation officielle que vous pouvez obtenir dans votre terminal en tapant la commande

```
man 3 printf
```

QUESTION 4 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `affichage.c`

Compiler puis exécuter le programme.

Qu'affiche-t-il ?

Expliquer.

Format	Signification	Exemple de valeur
%d	entier décimal signé	65
%u	entier décimal non signé	110
%o	entier octal signé	365
%x	entier hexadécimal non signé	4a6d
%f	nombre réel flottant	392.65
%e	nombre réel en notation scientifique	3.9265e+2
%g	utilise la représentation la plus courte entre %e et %f	65
%a	nombré réel hexadécimal	-0xc.80fep-2
%c	caractère	z
\n	saut de ligne	

TABLE 1 – Exemples de formats d’affichage de valeurs

Type	Taille mémoire (nombre d’octets) en fonction du système sous-jacent			
	LP32	ILP32	LP64	ILP64
char	1	1	1	1
unsigned char	1	1	1	1
short	2	2	2	2
unsigned short	2	2	2	2
int	2	4	4	4
unsigned int	2	4	4	4
long	4	4	4	8
unsigned long	4	4	4	8
long long	8	8	8	8
unsigned long long	8	8	8	8

TABLE 2 – Types entiers et leur taille

```
#include <stdio.h>

int main() {
    printf ("0. %c\n", 'a');
    printf ("1. %c\n", 65);
    printf ("2. %d\n", 100);
    printf ("3. %x\n", 100);
    printf ("4. %o\n", 100);
    printf ("5. %#x\n", 100);
    printf ("6. %#o\n", 100);
    printf ("7. %6.2f \n", 3.1416);
    printf ("8. %6.2f \n", 31.416);
    printf ("9. %E \n", 3.1416);
    printf ("10. %*d \n", 5, 10);
    return 0;
}
```

5.4 L’opérateur sizeof

L’opérateur `sizeof` prend en paramètre un type ou une variable, et retourne le nombre d’octets utilisés pour l’encodage du type ou de la variable dans votre environnement. Par exemple, `sizeof(char) = 1`. La taille mémoire nécessaire à l’encodage d’un type peut varier d’une architecture matérielle à une autre et d’un compilateur à un autre. Le Tableau ?? indique la taille mémoire nécessaire à l’encodage des types entiers dans différentes architectures. Il est donnée à titre indicatif.

5.5 Valeurs minimales et maximales

Comme nous venons de le voir les entiers ne sont pas tous codés sur le même nombre de bits en mémoire. Cela signifie que le même programme, suivant l’environnement dans lequel il est compilé et exécuté n’aura pas les mêmes limites. Pour cela il existe des constantes qui permettent de connaître ces valeurs limites.

Ces constantes peuvent être utilisées pour éviter des erreurs liées aux dépassement de capacité (cf. vol 501 d'Ariane 5).

- Les valeurs minimales et maximales des types caractère et entiers sont données par des constantes contenues dans le fichier `limits.h` : `INT_MIN`, `INT_MAX`, `LONG_MIN`, etc.
- Les valeurs minimales et maximales des types réels sont données par des constantes contenues dans le fichier `float.h` : `FLT_MIN`, `FLT_MAX`, `DBL_MIN`, etc.
- Sur N bits, il est possible de représenter tout nombre entier positif compris entre 0 et $2^N - 1$. Par exemple, sur 8 bits, il est possible de représenter l'ensemble des nombres entiers positifs allant de 0, 1, 2, ..., 254, 255.
- Sur N bits, il est possible de représenter tout nombre entier signé compris entre $-2^{(N-1)}$ et $2^{(N-1)} - 1$, grâce à la représentation en complément à deux pour encoder les nombres entiers signés (vous verrez ça en détail en cours d'architecture). Ainsi, sur 8 bits, il est possible de représenter l'ensemble des nombres entiers signés allant de -128, -127, ..., 0, 1, 2, ..., 126, 127.

QUESTION 5 ►

Écrire un programme C qui affiche pour chaque type primitif entier (char, short int, long, long long), signé et non signé : la taille mémoire (nombre d'octets) de représentation du type, la valeur minimale et la valeur maximale. Pour ce faire, utiliser l'opérateur `sizeof` ainsi que les constantes contenues dans le fichier `limits.h` introduites précédemment. Il est conseillé de consulter le fichier `limits.h`, disponible en ligne. Vous pouvez trouver ce fichier en tapant la commande :

```
locate limits.h
```

Compiler puis exécuter le programme.

Qu'affiche-t-il ?

QUESTION 6 ►

Écrire un programme C qui affiche pour chaque type primitif réel (float, double, long double), signé et non signé : la taille mémoire (nombre d'octets) de représentation du type, la valeur minimale et la valeur maximale. Pour ce faire, utiliser la fonction `sizeof`, ainsi que les constantes cette fois contenues dans le fichier `float.h`. Il est conseillé de consulter le fichier `float.h`, disponible en ligne.

Compiler puis exécuter le programme.

Qu'affiche-t-il ?

5.6 Conversion de types

Il est possible de convertir une variable d'un type vers un autre type ; cette opération de conversion s'appelle `cast`.

QUESTION 7 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `conversion1.c`

Compiler et exécuter le programme.

Qu'affiche-t-il ?

Expliquer. Si besoin, consulter la table des codes de caractères ASCII ici <http://www.asciitable.com>

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

int main(){
    int i = 4;
    float f = (float) i;
    char c = 100;
    int dc = (int) c;
    printf("i = %d \n", i);
    printf("f = %f \n", f);
    printf("c = %c \n", c);
    printf("dc = %d \n", dc);
```

```
    return 0;
}
```

QUESTION 8 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `conversion2.c`
Compiler et exécuter le programme.

Qu'affiche-t-il ?

Expliquer.

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

int main(){
    printf("4 / 5 = %f \n", (float)(4/5));
    printf("(float)4 / (float)5 = %f\n", (float)4 / (float)5);
    return 0;
}
```

5.7 Les opérations bit à bit

Un des aspects spécifiques du langage C est le fait que c'est un langage qui permet d'accéder directement à la mémoire. Si les aspects concernant les pointeurs seront vus dans le prochain TP, il y a une manière qu'à cette caractéristique de s'exprimer : la gestion bit à bit.

La taille minimale d'un objet en C est l'octet qui représente 8 bits. Par exemple le type `unsigned char` représente 8 bits et est censé représenter un caractère. On considère le programme suivant :

```
#include <stdio.h>

// On definit des masques de bits (chaque puissance de 2 est un bit unique)
#define LUMIERE_SALON  0x01 // 00000001
#define CUISINE        0x02 // 00000010
#define CHAMBRE        0x04 // 00000100
#define ALARME         0x80 // 10000000

int main() {
    unsigned char etat_maison = 0; // Tout est eteint (00000000)

    // 1. Allumer le salon (Operateur OR '|')
    etat_maison |= LUMIERE_SALON;

    // 2. Allumer la cuisine et l'alarme
    etat_maison |= (CUISINE | ALARME);

    // 3. Verifier si l'alarme est active (Operateur AND '&')
    if (etat_maison & ALARME) {
        printf("Attention : L'alarme est active !\n");
    }

    // 4. Inverser l'etat de la cuisine (Operateur XOR '^')
    etat_maison ^= CUISINE;

    // 5. Eteindre le salon (Operateur AND avec le complement NOT)
    etat_maison &= ~LUMIERE_SALON;

    printf("Valeur finale de l'octet : %d\n", etat_maison);

    return 0;
}
```

En plus de ces opérateurs, existent les décalages à gauche, `<<` et à droite `>>`. C'est à dire que `x<<2` revient à décaler vers la gauche `x` deux fois, c'est à dire à que si `x` était (10011101) alors `x<<2` représente (01110100). Ces opérations de bas niveau sont optimisées et très rapides. Elles permettent aussi de représenter l'information de manière minimale. Elles demandent par contre de bien maîtriser ce qui est fait.

6 Tableaux, chaînes de caractères

6.1 Tableaux

Ci-dessous un exemple de déclaration de variable représentant un tableau de nombres entiers constitué de 10 éléments.

```
/* Defines an array of integers containing 10 elements */  
int numbers[10];
```

La première ligne de code est un commentaire, et la seconde ligne de code représente l'instruction de déclaration d'une variable tableau.

L'exemple de code ci-dessous présente, quant à lui, l'initialisation des valeurs des 7 premiers éléments du tableau déclaré précédemment.

```
numbers[0] = 10;  
numbers[1] = 20;  
numbers[2] = 30;  
numbers[3] = 40;  
numbers[4] = 50;  
numbers[5] = 60;  
numbers[6] = 70;
```

QUESTION 9 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `tableau.c`

Compiler et exécuter de ce programme.

Qu'effectue-t-il ?

Expliquer ce qui ne fonctionne pas et corriger.

```
#include <stdio.h>  
  
int main() {  
    int average;  
  
    grades[0] = 80;  
    grades[2] = 90;  
  
    average = (grades[0] + grades[1] + grades[2]) / 3;  
    printf("The average of the 3 grades is: %d", average);  
  
    return 0;  
}
```

QUESTION 10 ►

Modifier le programme ci-dessus pour que la valeur affichée soit 85.

6.2 Chaînes de caractères sous forme de tableaux

Il y a plusieurs façons de représenter une chaîne de caractères en C. Une façon simple de le faire est d'utiliser un tableau de caractères dont le dernier élément est un caractère spécial (0) indiquant la fin de la chaîne de caractère. Ce codage est utilisé car 0 représente le faux, on peut donc parcourir une chaîne jusqu'à ce que le caractère rencontré soit "faux" sans avoir à savoir la longueur effective de la chaîne.

Ci-dessous un exemple :

```
/* Defines an array of characters containing 11 elements */  
char name[11] = "John Smith";
```

6.3 Fonctions de manipulation de chaînes de caractères

Pour utiliser les fonctions sur les chaînes de caractères il faut inclure la librairie `string.h` en début de programme. La fonction `strlen` retourne longueur (nombre de caractères) d'une chaîne de caractères.

```
char name[11] = "John Smith";

printf("%d\n",strlen(name));
```

La fonction `strncmp` compare deux chaînes de caractères selon l'ordre lexicographique, en considérant au maximum un certain nombre de caractères. Elle retourne la valeur 0 si les deux chaînes de caractères sont égales. Si la première chaîne est plus petite que la seconde chaîne selon l'ordre lexicographique, la fonction de comparaison retourne une valeur strictement négative. Si la première chaîne est plus grande que la seconde chaîne, la fonction de comparaison retourne une valeur strictement positive.

```
char name[11] = "John Smith";

if (strncmp(name, "John", 4) == 0) {
    printf("Hello, John!\n");
} else {
    printf("You are not John.\n");
}
```

La fonction `strncat` permet de concaténer les `n` premiers caractères d'une chaîne de caractères à une autre chaîne de caractères.

QUESTION 11 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `concatenation.c`

Compiler et exécuter le programme.

Qu'affiche-t-il ?

Expliquer.

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

int main() {
    char dest[20]="Hello";
    char src[20]="World";
    strncat(dest,src,3);
    printf("%s\n",dest);
    strncat(dest,src,20);
    printf("%s\n",dest);
    return 0;
}
```

De même que pour `limits.h` vous pouvez regarder le fichier `string.h` pour voir les fonctions qui sont offertes par cette bibliothèque. Vous pouvez également vous référer à un site détaillant la librairie, par exemple wikipedia : <https://fr.wikipedia.org/wiki/String.h>

7 Structures de contrôle

7.1 Structure de contrôle conditionnelle `if`

Le programme ci-dessous donne un exemple de définition de condition avec `if`.

```
int foo = 100;
int bar = 200;

if (foo < bar) {
    printf("foo is smaller than bar.");
} else if (foo == bar) {
    printf("foo is equal to bar.");
} else {
    printf("foo is greater than bar.");
}
```

QUESTION 12 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `condition.c`

Compléter le programme pour utiliser `if` et tester les cas suivants :

- Si `val` est égale à 1000, afficher le message "Vous êtes dans le mille!".
- Si `val` est inférieure à 1000, afficher le message "Valeur trop petite".
- Si `val` est inférieure à 100, afficher le message "Valeur beaucoup trop petite".
- Si `val` est supérieure à 1000, afficher le message "Valeur trop grande".
- Et si `val` est supérieure à 10000, afficher le message "Valeur beaucoup trop grande".

Compiler et exécuter de ce programme.

Ré-exécuter le programme avec différentes valeurs de la variable `val`.

Qu'affiche le programme ?

```
#include <stdio.h>

int main() {
    int val = 1000;

    /* Completer le code */
}
```

7.2 Structure de contrôle conditionnelle `switch`

L'instruction `switch` permet d'effectuer plusieurs tests sur une même variable. Elle évite ainsi l'utilisation de plusieurs tests imbriqués avec des instructions `if`. La syntaxe de `switch` est indiquée ci-dessous. La valeur de la variable du `if` est testée successivement par chacun des `case`. Lorsque la valeur de la variable testée est égale à une valeur de `case`, la liste des instructions de ce `case` est exécutée. Le mot-clé `break` indique la sortie de la structure conditionnelle, donc la fin de ce `case` et la fin du `switch`. Le mot-clé `default` indique les instructions qui seront exécutées si la valeur de la variable testée n'est égale à aucune des valeurs de `case`.

Attention cette structure de contrôle est très sommaire. IL faut que chaque `case` soit une valeur et non une expression booléenne.

```
switch (Variable) {
    case Valeur1:
        /* Liste d'instructions */
        break;

    case Valeur2:
        /* Liste d'instructions */
        break;

    case Valeurs...:
        /* Liste d'instructions */
        break;

    default:
        /* Liste d'instructions */
}
}
```

QUESTION 13 ►

Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `conditions.c`

Compiler et exécuter le programme.

Qu'affiche-t-il ? Expliquer

Modifier le programme pour initialiser à la variable `val` avec la valeur 3.

Compiler et exécuter le programme.

Qu'affiche-t-il ? Expliquer

```
#include <stdio.h>

int main() {
    int val = 2;
```

```

switch(val) {
    case 1:
        printf("Hello 1!\n");
        break;
    case 2:
        printf("Hello 2!\n");
        break;
    case 3:
        printf("Hello 3!\n");
        break;
    default:
        printf("Other Hello!\n");
}
return 0;
}

```

QUESTION 14 ►

Considérer le programme C ci-dessous qui est différent du programme précédent en ce que des `break` ont été retirés pour les conditions 1 et 3 ; le copier dans un fichier intitulé `conditions_prime.c`

Compiler et exécuter le programme.

Qu'affiche-t-il ? Expliquer

Modifier le programme pour initialiser à la variable `val` avec la valeur 3.

Compiler et exécuter le programme.

Qu'affiche-t-il ?

```

#include <stdio.h>

int main() {
    int val = 2;

    switch(val) {
        case 1:
            printf("Hello 2!\n");
        case 2:
            printf("Hello 2!\n");
            break;
        case 3:
            printf("Hello 3!\n");
        default:
            printf("Other Hello!\n");
    }
    return 0;
}

```

La structure de contrôle `switch` est en fait une construction de type `goto`. Les valeurs des `case` sont les étiquettes et le `switch` correspond à un `goto` de la valeur représentée par l'expression. Les instructions `break` sont aussi des `goto`. Le compilateur rajoute une étiquette `break` : juste à la sortie de l'accolade fermante, et le `break` est en fait un `goto` à cette étiquette. Donc quand il n'y a pas de `break` les instructions continuent de s'enchaîner comme dans la version `condition_prime.c`.

7.3 Structure de contrôle itérative `for`

Ci-dessous un exemple de programme avec une boucle `for`. Elle permet d'exécuter de manière itérative une suite d'instructions, en se basant sur un itérateur (la variable entière `i` dans cet exemple), la valeur initiale de l'itérateur (0 ici), la mise à jour de la valeur de l'itérateur (par un incrément de 1 ici), et le test de valeur de fin de l'itérateur (10 ici).

```

int i;
for (i = 0; i < 10; i++) {
    /* suite d'instructions */
}

```

QUESTION 15 ►

La factorielle d'un nombre se calcule comme suit : $N! = 1 * 2 * \dots * N$. Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `facto.c`
Compléter le programme pour utiliser une boucle `for` pour calculer la factorielle de 10.
Compiler et exécuter le programme.

```
#include <stdio.h>

int main() {
    int facto,i;
    /* utiliser une boucle for pour calculer la factorielle de 10 */

    printf("10! = ...");
    return 0;
}
```

QUESTION 16 ► On veut remplir un tableau qui contient les 10 premiers carrés. On utilisera le fait que le nième carré est la somme des n premiers nombres impairs.
Considérer le programme C ci-dessous ; le copier dans un fichier intitulé `facto.c`
Ecrire un programme dans un fichier intitulé `carrés.c` qui utilise une boucle `for` qui remplit un tableau de dix cases avec les dix premiers carrés et afficher ses carrés.
Compiler et exécuter le programme.

7.4 Structure de contrôle itérative while

Une autre structure de contrôle itérative est le `while`, dont la syntaxe générale est donnée ci-dessous.

```
while (condition) {
    /* do something */
}
```

7.5 Directives break et continue des structures de contrôle itératives

Les deux directives `break` et `continue` peuvent être utilisées avec toute structure de contrôle itérative, par exemple avec des boucles `while` ou `for`.

La directive `break` permet de quitter, même prématurément, une structure de contrôle itérative. Dans l'exemple ci-dessous, la directive `break` permet de quitter la boucle `while` même dans le cas d'une boucle infinie.

```
int n = 0;
while (1) { /* boucle infinie */
    n++;
    if (n == 10) {
        break;
    }
}
```

La directive `continue` permet de ne pas exécuter la suite d'instructions d'une itération, et de passer directement à l'itération suivante. Ci-dessous un exemple.

```
int n = 0;
while (n<15) {
    n++;
    if (n == 10) {
        continue;
    }
    printf("n = %d\n", n);
}
```

QUESTION 17 ►

Considérer le programme ci-dessous ; le copier dans un fichier intitulé `iter.c`
Compléter le programme (sans modifier l'instruction `printf`) de telle sorte que : si la i-ème valeur du tableau est inférieure à 5 il n'y a pas d'affichage, et si la i-ème valeur du tableau est supérieure à 10 il n'y a

pas d'affichage et la boucle s'arrête.
Compiler et exécuter le programme.
Qu'affiche-t-il ?

```
#include <stdio.h>

int main() {
    int tab[] = {1, 7, 4, 5, 9, 3, 5, 11, 6, 3, 4};
    int i = 0;

    while (i < 10) {
        /* suite du code */

        printf("%d\n", tab[i]);
        i++;
    }

    return 0;
}
```

8 Fonctions

Une fonction peut avoir 0, un ou plusieurs paramètres. Elle peut soit retourner au plus un résultat, ou aucun résultat. Elle est constituée d'une suite d'instructions exécutées à chaque appel de la fonction, et elle peut elle-même appeler une autre fonction. Des exemples sont donnés ci-dessous.

```
/* fonction sans parametres et sans resultat le type de "rien" est void*/
void f1() {
    /* suite d'instructions */
}

/* fonction sans parametres et retournant un resultat de type entier */
int f2() {
    int res = ...;
    /* suite d'instructions */
    return res;
}

/* fonction avec un parametre de type caractere, un parametre de type entier,
et retournant un resultat de type reel */
float f3(char c, int i) {
    float res = ...;
    /* suite d'instructions */
    return res;
}
```

QUESTION 18 ►

Considérer le programme ci-dessous ; le copier dans un fichier intitulé `func.c`
Compléter le programme (sans modifier la fonction `main`) pour que la fonction `f` affiche le message "`n` est grand" à chaque fois que le paramètre de la fonction est supérieur à 10.
Compiler et exécuter le programme. Qu'affiche-t-il ?

```
#include <stdio.h>

void f(int n) {
    /* code de la fonction */
}

int main() {
    int tab[] = { 1, 11, 2, 22, 3, 33 };
    int i;
    for (i = 0; i < 6; i++) {
        f(tab[i]);
    }
    return 0;
}
```

8.1 Fonctions récursives

Une fonction récursive est une fonction qui s'appelle elle-même, en variant les valeurs des paramètres passés lors de l'appel récursif. La récursion est une façon élégante d'écrire des programmes.

QUESTION 19 ►

Considérer le programme ci-dessous ; le copier dans un fichier intitulé `factorielle.c`

Compléter le programme (sans modifier la fonction `main`) pour écrire une fonction récursive `facto` qui calcule la factorielle d'un entier passé en paramètre.

Compiler et exécuter le programme. Qu'affiche-t-il ?

```
#include <stdio.h>

int facto(int n) {
    /* suite d'instruction */
}

int main() {
    printf("1! = %i\n", facto(1));
    printf("3! = %i\n", facto(3));
    printf("5! = %i\n", facto(5));
    return 0;
}
```

8.2 Fonctions mutuellement récursives

Il est possible que des fonctions soient mutuellement récursives. C'est-à-dire que la fonction `f` fasse appelle à la `g` qui elle même fasse appelle à la fonction `f`. Attention c'est délicat à manipuler. Cela permet de mettre en avant un autre aspect des fonctions : la déclaration d'une fonction peut être faite avant sa définition et ses utilisations. Considérons l'exemple suivant :

```
#include<stdio.h>

int f(int x);
int g(int x);

int main(){
    int i;

    for(i=0;i<=12;i++)
        printf("g[%d]=%d\n",i,f(i));

    return 0;
}

int f(int x){
    if (x<=1) return x; else return g(x-2);
}

int g(int x){
    if (x<=1) return x; else return f(x-2);
}
```

QUESTION 20 ► Considérer le programme ci-dessus ; le copier dans un fichier intitulé `mystere.c`
Compiler et exécuter le programme. Qu'affiche-t-il ? Quelle est la signification de ce programme ?

QUESTION 21 ► On suppose qu'on a un tableau d'entiers `trajet` de taille `TAILLE`. On déclarera `TAILLE` comme une constante grâce à la commande `#define TAILLE 6` par exemple qui la définit `TAILLE` comme valant 6.

On demande la construction d'un programme qui utilise deux fonctions mutuellement récursives `verifieimpair` et `verifiepair` qui permettent de vérifier que dans le tableau des alternent des valeurs paires et impaires.

9 Questions subsidiaires

QUESTION 22 ►

C'est Noël ! Ecrire un programme permettant de dessiner un sapin, étant données une hauteur de cône, une largeur et une hauteur de tronc.

Par exemple, pour un cône de hauteur 6, un tronc de hauteur 2 et de largeur 3, le sapin affiché sera comme suit.

```
  *
 ***
*****
*****
*****
*****
*****
  |||
  |||
```

QUESTION 23 ►

A présent, le programme considère également un étage et une place à cet étage, de telle sorte qu'une boule de Noël sera placée à cet endroit. Si l'étage et la place spécifiés sont, par exemple, 2 et 3, le sapin précédent s'affichera comme suit.

```
  *
 ***
*****
*****
**0*****
*****
  |||
  |||
```

QUESTION 24 ►

Nous souhaitons à présent afficher un sapin dont la taille est aléatoire, et sur lequel les boules sont placées aléatoirement.

Dans un premier temps, considérer l'exemple de programme C ci-dessous qui génère et utilise des nombres aléatoires. Copier ce code dans un fichier intitulé `aleatoire.c`, et le compiler.

Exécuter les programme à plusieurs reprises.

Qu'affiche-t-il ?

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

int main () {
    int i, n;
    n = 5;

    /* initialiser le generateur de nombres aleatoires */
    srand((unsigned) time(NULL));

    /* afficher 5 nombres entiers aleatoires compris 0 et 49 */
    for( i = 0 ; i < n ; i++ ) {
        printf("%d\n", rand() % 50);
    }

    return(0);
}
```

QUESTION 25 ►

Ecrire un programme C qui affiche un sapin dont la taille est aléatoire (hauteur de cône, largeur et hauteur de tronc), et sur lequel une boule est placée aléatoirement (étage et place).

Compiler le programme et l'exécuter à plusieurs reprises. Qu'affiche-t-il ?

QUESTION 26 ► On propose de réaliser un programme qui chiffre une chaîne de caractère en appliquant un XOR bit à bit entre le texte d'origine et une clé de la même taille que le texte à chiffrer. La propriété

intéressante est que $(T^C)^C = T$, c'est à dire que si on applique deux fois XOR avec la même clé on va retrouver le texte d'origine. Recopier le programme suivant et complétez le pour qu'il fonctionne selon cette spécification.

Testez votre programme sur différentes entrées

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

#define MAX_SIZE 256
int main(){
    char texte_clair[]="test a chiffrer exemple";
    char cle[]="cle_de_chiffre_representee_comme_string";
    char texte_code[MAX_SIZE],texte_decode[MAX_SIZE]

    // --- ETAPE 1 : CRYPTAGE ---
    ...
    printf("Message crypte : %s\n", texte_code);

    // --- ETAPE 2 : DECRYPTAGE ---
    ...
    printf("Message decrypte : %s\n", texte_decode);

    return 0;
}
```