TP 1 : Notions de base en programmation C

1 Objectifs

Ce TP abordera les principaux éléments suivants de la programmation C :

- Rappel des 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

2 Rappel des principales commandes

2.1 Commande de compilation

Ci-dessous un exemple de commande :

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

Les fichiers `prog1.c ...progn.c` contiennent des programmes en C. Cette commande permet de compiler et de lier les programmes, pour produire un programme exécutable (binaire) global dans le fichier `prog`

2.2 Commande d'exécution

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

```
./prog
```

2.3 Makefile

Un fichier `Makefile` sert à regrouper différentes commandes et options de compilation et d'exécution de programmes. Un exemple de fichier Makefile est fourni avec ce TP. S'en inspirer pour construire les fichiers Makefile de vos programmes C.

3 Exemple de programme "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 peut utiliser des bibliothèques qui permettent d'exécuter des fonctions nécessaires au programme. Par exemple, 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`.

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

```
  return 0;
```

A noter que chaque ligne de programme C se termine par un point-virgule (`;`) pour indiquer au compilateur qu'une nouvelle ligne de code commence.

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 !"

4 Types primitifs de données

4.1 Rappel des types primitifs

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

- Un caractère : `char`
- Les nombres entiers positifs ou négatifs : `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`.

Par ailleurs, le langage C ne possède pas de type booléen. Les valeurs entières `0` et `1` représentent respectivement les valeurs booléennes `faux` et `vrai`. Il est possible par exemple d'utiliser la notation suivante pour plus de facilité :

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

4.2 Notations décimales, octales, hexadécimales de nombres entiers

Un nombre entier en notation décimale est représenté par une suite de chiffres compris 0 et 9. Un nombre entier en notation octale est préfixé par 0 (zéro) suivi d'une suite de chiffres compris entre 0 et 7. Un nombre entier en notation hexadécimale est préfixé par 0x et suivi d'une suite de chiffres compris entre 0 et 9 et lettres (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.

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 0.1 – Exemples de formats d'affichage de 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

4.3 Affichage de valeurs

L'affichage de valeurs peut se faire avec la fonction `printf` incluse dans la bibliothèque `stdio.h` Différents formats de valeurs peuvent être affichés ; des exemples sont donnés dans le Tableau **??**.

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.

```
#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;
}
```

4.4 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. 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.
Les différents systèmes sous-jacents sont soit des systèmes à 32 bits :

— **LP32** :

Type	Taille mémoire (nombre d'octets) en fonction du système sous-jacent			
	LP32	**ILP32**	**LLP64**	**LP64**
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 0.2 – Types entiers et leur taille

  - Win16 API

- **ILP32** :
  - Win32 API
  - Unix and Unix-like systems (Linux, Mac OS X)

Soit des systèmes à 64 bits :

- **LP64** :
  - Win64 API
- **ILP64** :
  - Win64 API
  - Unix and Unix-like systems (Linux, Mac OS X)

4.5 Valeurs minimales et maximales

- 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 (positif ou négatif) 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. Dans la représentation en complément à 2, les bits des nombres négatifs sont tout d'abord inversés, puis on leur ajoute 1.

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 les fichiers `limits.h`, disponible en ligne. Compiler puis exécuter le programme.
Qu'affiche-t-il ?

QUESTION 6 ►
Ecrire 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 l'opérateur `sizeof`, ainsi que les constantes contenues dans le fichier `float.h` introduites précédemment. Il est conseillé de consulter le fichier `float.h`, disponible en ligne.
Compiler puis exécuter le programme.
Qu'affiche-t-il ?

QUESTION 7 ►
Ecrire un fichier Makefile incluant les commandes de compilation et d'exécution des programmes précédents.
Utiliser ce Makfefile pour compiler et exécuter les programmes.

4.6 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 8 ►
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 9 ►
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 Tableaux, chaînes de caractères

5.1 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 10 ►
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 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 11 ►
Modifier le programme ci-dessus pour que la valeur affichée soit 85.

5.2 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. Ci-dessous un exemple :

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

5.3 Fonctions de manipulation de chaînes de caractères

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 contacter les n premiers caractères d'une chaîne de caractères à une autre chaîne de caractères.

QUESTION 12 ►
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;
}
```

6 Structures de contrôle

6.1 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 13 ►
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 */
  }
}
```

6.2 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 .

```
  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 14 ▶
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 = 1;

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

6.3 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'intérateur (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 * ... * 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 tab[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

  /* utiliser une boucle for pour calculer la factorielle de 10 */

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

6.4 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 */
}
```

6.5 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` meêm dans le cas d'une boucle infinie.

```
int n = 0;
while (1) {
    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 (1) {
    n++;
    if (n == 10) {
        continue;
    }
    printf("n = %d\n", n);
}
```

QUESTION 16 ►
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 ?

```c
#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;
}
```

7 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.

```c
/* fonction sans parametres et sans resultat */
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 17 ▶
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 ?

```c
#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;
}
```

7.1 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 18 ►
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 Questions subsidiaires

QUESTION 19 ►
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 20 ►
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 21 ►
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 ?

```c
#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 22 ►

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 ?