# TP 1 : Ecriture, compilation et exécution de programmes C simples

Objectifs :

- Découvrir la syntaxe et la sémantique du langage C.
- Etre capable d'écrire en C un algorithme simple.
- Maîtriser l'écriture de fonctions.
- Etre capable d'écrire en C un programme structuré en plusieurs fonctions.



STEPHANIE CHOLLET, Ioannis PARISSIS 2018/2019

## 1. Informations pratiques

*Ces travaux pratiques sont à effectuer sous Linux (système UNIX sur PC : voir la documentation qui vous est fournie à ce sujet).*.
*Pour créer/éditer des programmes, utiliser un éditeur tel que gedit*[1].
*Pour lancer une compilation ou exécuter votre programme, utiliser un terminal*[2].
*On se réfère au polycopié « Introduction au langage C » de B. Cassagne.*

## 2. Mon premier programme C

Créer un répertoire CS351 puis un sous répertoire TP1. Dans TP1, créer un fichier `prog1.c` et y ajouter le texte suivant :

```
#include <stdio.h>
int main() {
     printf("Bonjour\n"); /* \n signifie " passage à la ligne " */
     return (0);
}
```

Ensuite exécuter la commande suivante[3] :

```
gcc prog1.c -Wall -ansi -pedantic -o prog1
```

Vous venez de compiler votre premier programme C. A l'issue de l'exécution de cette commande, le fichier `prog1` a été ajouté dans votre répertoire courant. Il s'agit du programme exécutable correspondant à `prog1.c`. Pour exécuter ce programme, taper simplement :

```
./prog1
```

## 3. Notions de base du langage C : types, constantes, variables, expressions, instructions conditionnelles

*Ce paragraphe est un résumé très compact du polycopié, chapitre 1, sections 1.1 à 1.13, dont la lecture est nécessaire.*

Les *identificateurs* du langage (noms de variables, fonctions etc.) sont des suites formées de lettres, chiffres (pas en première position) et du caractère '_' (p.ex. bonjour, b3jour, bon_jour, _bonjour).

Un *commentaire* est compris entre /* et */ (/* Ceci est un commentaire */ et ceci une faute de syntaxe */).

Les *types de base* du langage sont : caractère (char), entier (int, short int, long int), flottants (float, double, long double).

L'écriture de *valeurs constantes* se fait naturellement pour les cas simples : 2, 2.6, 'a', « toto ». De nombreuses écritures spécifiques sont disponibles (p.ex. 4L désigne l'entier 4 représenté comme un entier long ; '\n' correspond au caractère *newline*…) à découvrir dans le polycopié. Le moyen le plus simple de définir une *constante nommée* est d'utiliser la directive `#define` :

```
#define PI 3.14159 /* sans ';' à la fin */
```

[1] *Menu Accessoires>Editeur de texte>gedit*
[2] *Menu Accessoires>Terminal.*
[3] *Voir TP2 pour des explications sur la commande gcc.*

Cette ligne fait de sorte que toute occurrence de l'identificateur `PI` dans le programme est remplacée par `3.14159`.

L'affectation en C s'effectue à l'aide de l'opérateur = (égal).

```
x = 5 ; /* affectation à x de la valeur 5 */
```

Une affectation est une expression retournant une valeur :

```
int k = 1 ;
int i = (j = k) + 1 ; /* j prend la valeur de k, soit 1 ; i
prend la valeur 2, soit la valeur de (j = k) + 1 */
```

Les opérateurs de comparaison sont == (égalité), >, <, >=, <=, != (différence).

L'instruction conditionnelle `if` :

```
if(condition) {
     instruction
}
/* noter la parenthèse obligatoire autour de la condition*/
if(condition) {
     instruction
} else {
     instruction
}
```

Une instruction peut prendre la forme d'un *bloc d'instructions* délimitées par des accolades (qui correspondent à « début » et « fin » en langage algorithmique).

```
if(x > 0) {
   x = x + 1 ;
   y = 0 ;
} else {
   x = y = 1;
}
```

L'instruction de choix `switch` (voir polycopié 3.15.1) :

```
/* x, y sont des int */
switch (x) {
     case 0: y = 1; break;
     case 1: y = 2; break;
     default: y = 0;
}
```

## 4. Entrées sorties élémentaires

L'affichage à l'écran ou dans un fichier, la lecture depuis le clavier ou depuis un fichier se font au moyen d'appels à des fonctions des bibliothèques de C (en d'autres termes, il n'existe pas d'instruction de base du langage permettant d'effectuer une action d'entrée ou sortie). Avant une étude plus approfondie de ces fonctions, on se contente ici d'utiliser deux d'entre elles, `printf` et `scanf`. Leur utilisation est rapidement expliquée dans les sections 1.16 et 3.14 du polycopié. L'exemple ci-dessous en illustre quelques utilisations simples :

```
#include <stdio.h>

#define INC 2


int main() {
   int i;
   printf("Donnez un entier : ");
   scanf ("%d", &i); /* &i : i par adresse (voir plus loin) */
   i = i + INC ;
   printf("valeur de i=%d et son successeur i+1=%d\n", i,
i+1);

     return (0);
}
```

### Exercice 1 : Appréciation

Écrire un programme qui lit une note exprimée sous la forme d'une lettre de A à E, puis affiche un message correspondant à la lettre saisie (« Très bien », « Bien », « Assez bien », « Passable » et « Insuffisant »). Faire trois versions de programme :

- Une version utilisant des instructions `if` imbriquées (if… else… if... etc);
- Une version utilisant des `if` en séquence (sans imbrication : if ..; ; if… ;);
- Une version utilisant l'instruction `switch`.

### Exercice 2 : Année bissextile

Une année bissextile comprend un jour de plus que les années normales. On dit couramment que les années bissextiles reviennent tous les quatre ans, ce qui n'est pas tout à fait exact. La définition complète est la suivante :
Une année bissextile est divisible par 4 ; mais si elle est également divisible par 100, alors elle doit aussi être divisible par 400.

1. Avant d'écrire le programme, établir un ensemble de tests (entiers correspondant à une année) qui permettront de vérifier sa correction. Ajouter, sous forme de commentaire, ces tests dans votre programme de la question suivante en expliquant leur choix.
2. Écrire un programme qui lit un entier correspondant à une année et affiche un message approprié en fonction (« L'année 1996 est bissextile », « L'année 2001 n'est pas bissextile »).

## 5. Itérations

Il existe trois instructions permettant de réaliser des itérations en langage C : `while`, `do`, `for` (lire les sections 2.2 et 2.3 du polycopié).

```
/* condition est une expression booléenne */
while(condition) {
     instruction
}
```

```
do {
     instruction
```

```
} while (condition) ;
```

```
for(expression1 ; expression2 ; expression3) {
     instruction
}
```

IMPORTANT : Nous rappelons qu'une boucle bien construite **ne doit pas** contenir de `return`, `break` ou `continue` dans son corps. La sortie de la boucle doit se faire **exclusivement** au moment de l'évaluation de la condition, quel que soit le type de boule utilisé.

### Exercice 3 : Somme des n premiers entiers

Ecrire un programme calculant la somme des n premiers entiers. La valeur de n est fournie par l'utilisateur. Réaliser une version de ce programme utilisant l'instruction `while` et une autre utilisant `do`.

### Exercice 4 : Opérateurs de post et pré incrémentation et décrémentation

Exécuter et commenter le programme suivant :

```
#include <stdio.h>

int main() {
     int i, j, k, l;

     i = j = k = l = 0;

     while(i < 9) {
          printf("i++ = %d, ++j = %d, k-- = %d, --l = %d\n",
                  i++, ++j, k--, --l);
     }

     printf ("i = %d, j = %d, k = %d, l = %d\n", i, j, k, l);

     return (0);
}
```

## 6. Fonctions et procédures : notions de base

Lire les sections 1.15.1 à 1.15.3 du polycopié.

Exemple de définition d'une fonction :

```
/* la fonction somme retourne un int */
/* ses paramètres formels sont les int i et j */
int somme(int i, int j) {
     int res; /* variable locale */
     res = i + j;
     return(res); /* retour de la valeur de res */
}
```

`int somme(int i, int j)` est le *prototype* de la fonction `somme`.

Appel de fonction :

```
#include <stdio.h>

int main(){
     int r, a, b;

     a = 3;
     b = 2;
     r = somme(a, b);
     printf("a + b = %d\n", r);

     return (0);
}
```

En C, il n'existe pas de procédure, à proprement parler. En fait, une procédure est une fonction qui ne retourne pas de valeur (auquel cas, on spécifie `void` comme type de retour) :

```
#include <stdio.h>

void aff_somme(int i, int j) {
     int res = i + j;
     printf("somme = %d\n", res);
}

int main(){
     int r, a, b;

     a = 3;
     b = 2;
     aff_somme(a, b);

     return (0);
}
```

Pour chacun des exercices, on écrira, **dans un même fichier**, la ou les fonctions demandées ainsi que la fonction `main` les appelant.

## Exercice 1 : Fibonacci

On rappelle la suite de Fibonacci définie par :

$$u_0 = 0$$
$$u_1 = 1$$
$$u_n = u_{n-1} + u_{n-2} \text{ si } n > 1$$

- Ecrire une fonction `fibonacci` calculant le terme de rang n de la suite dont le prototype est : `int fibonnacci(int n);`
  La fonction doit utiliser une instruction `while`, `for` ou `do`.
- Ecrire ensuite une fonction `main` demandant la valeur de n à l'utilisateur et affichant le terme correspondant de la suite.

## Exercice 2 : PGCD

On rappelle que le pgcd est défini par les relations suivantes (a et b étant des entiers naturels) :

pgcd(a, 0) = a

pgcd(a, b) = pgcd(b, r) avec r = a mod b, si b ≠ 0 (mod est le reste de la division entière).

- Ecrire une fonction (**utilisant une itération**) `pgcd`, à deux paramètres entiers, retournant le pgcd de ses paramètres.
- Ecrire une fonction `main` demandant deux valeurs entières à l'utilisateur et affichant leur pgcd.

## Exercice 3 : Factorielles

- Ecrire une fonction `factorielle` qui calcule et retourne la valeur de n! (1 x 2 x 3 x…x n). Cette fonction doit être réalisée à l'aide d'une instruction `for` ou `while`.
- Ecrire une fonction `factorielleBis` à un paramètre entier m qui calcule et retourne la valeur du plus petit entier positif n tel que n! (factorielle de n) soit supérieur à m. Cette fonction ne doit pas faire appel à la fonction `factorielle`.
- Ecrire ensuite une fonction `main` demandant la valeur de n à l'utilisateur et affichant le résultat de chaque fonction ci-dessus.

## Exercice 4 : Jeu de multiplication

On veut écrire une procédure `jeuMulti` qui demande à l'utilisateur de réciter sa table de multiplication. L'utilisateur commence par entrer un nombre entre 2 et 9 (si le nombre est incorrect, le programme le redemande). Ensuite l'algorithme affiche une à une les lignes de la table de multiplication de ce nombre, en laissant le résultat vide et en attendant que l'utilisateur entre le résultat. Si celui-ci est correct, on passe à la ligne suivante, sinon on affiche un message d'erreur donnant la bonne valeur et on termine. Si toutes les réponses sont correctes, on affiche un message de félicitations. On représente ci-dessous une exécution possible (les entrées de l'utilisateur sont affichées en italiques) :

Valeur de n : *12*
Réessayez : la valeur doit être comprise entre 2 et 9
Valeur de n : *6*
1 x 6 = *6*
2 x 6 = *12*
3 x 6 = *21*
Erreur ! 3 x 6 = 18 et non 21

1. Ecrire la procédure `jeuMulti`.
2. Ecrire une nouvelle procédure `jeuMultiPoints` qui ne s'arrête pas quand une réponse fausse est donnée, mais affiche à la fin le nombre d'erreurs commises et un message éventuel de félicitations.
3. Testez ces deux procédures en justifiant vos tests.