Séance 1 : Introduction au langage C

L2 – Université Côte d'Azur

Exercice 1 — Compiler avec `gcc`

1. Créez un répertoire `ProgrammationC`, puis, dans celui-ci, créer un repertoire `TP1`. Créez alors un fichier `exercice1.c` contenant le code ci-dessous. Vous pouvez utiliser l'éditeur de votre choix (gedit, emacs, autre).

```c
#include<stdio.h>
int c = 3;
void main(void) {
    int u,d,c;
    u = 1;
    d = 2;
    int n = 100*c + 10*d + u; /* c vaut 3, d vaut 2 et u vaut 1 */
    printf("321 = %i\n",n); /* affiche « 321 = 321 » */
}
```

2. Dans un terminal, compilez le fichier avec la commande : « `gcc exercice1.c` ».
3. Quel est le nom de l'exécutable produit ? Lancer-le. Le résultat est-il celui attendu ?
4. Pour éviter ce genre de mauvaise surprise, durant cette UE, nous allons systématiquement compiler le code avec les options `gcc -Wall -ansi -pedantic exercice1.c`. Corrigez le code jusqu'à ce que le compilateur n'affiche aucun *warning* avec ces options et que le code s'exécute comme souhaité.
5. Créer un fichier `Makefile` pour que la commande « `make` » crée un fichier `exercice1`.

Exercice 2 — Quelques manipulations simples

1. Créer un fichier `exercice2.c`. Modifiez le fichier `Makefile` pour qu'il compile aussi l'exercice 2. Prenez l'habitude de bien séparer le code en différentes fonctions (en régle générale, une par question).

```c
void question1(void) {
  /* À Compléter */
}

int main(void) {
   question1();
   …
   question6();
   return 0 ;
}
```

2. Écrivez un code C qui affiche les entiers de 65 à 90.
3. Écrivez un code C qui affiche les caractères dont le code ASCII va de 65 à 90.
4. Créer deux variables, une chaîne `s="Pi vaut"` et un nombre `x=3.1415926` puis affichez les avec `printf`.
5. Copiez et exécutez le code suivant :

```c
char c = 'a';
int i = 65;
printf("valeur de c en caractère est %c, et son équivalent décimal est %d\n", c, c);
printf("valeur de i en décimal est %d, et son équivalent caractère est %c\n", i, i);
```

6. Écrivez le code qui affiche les caractères de `'a'` à `'z'` en sachant que `'b'` a le code de `'a'`+1

Exercice 3 — Pas facile de compter juste…

Dans un fichier `exercice3.c`, recopiez puis exécutez le programme suivant. Expliquez les résultats.

```c
float fi = 10.1f;
float fj = 10.2f;
printf("%.16f %.16f %.16f %d %d\n", fi, fj, fi + fj, (int)fi, fi);
printf("%.32f %.32f %.32f %d %d\n", fi, fj, fi + fj, (int)fi, fi);
printf("%.8f %.8f %.8f %d %d\n", fi, fj, fi + fj, (int)fi, fi);
printf("%.2f %.2f %.2f %.2f\n", 5/2, 5.0/2, 5/2.0, 5.0/2.0);
```

Exercice 4 — Boucles imbriquées

Écrivez un programme qui affiche les nombre de 1 à 100 à raison de 10 par lignes. Les nombres devront être alignés correctement.

Exercice 5 — Lire des valeurs avec `scanf`

1. Écrivez un programme qui lit sur l'entrée standard un nombre entier compris entre deux bornes données.
2. Que se passe-t-il si vous entrez autre choses que des nombres ? Tapez autre chose que des caractères numériques et regardez le comportement…
   Une solution pour vider le buffer est d'appliquer la fonction `vider_buffer` vue en cours.
3. Écrivez un programme qui calcule la moyenne de notes (type entier) comprises entre 0 et 20 lues sur l'entrée standard (la dernière note saisie sera 99).

Exercice 6 — Déboguer un code faux

1. Compilez et exécutez le programme suivant :

```c
/* ATTENTION: programme faux!!! */
#include <stdio.h>
int main(void) {
  nb = f();
  printf("Valeur du nombre lu = %d\n", nb);
  return 0; True
}

int f (void) {
  int i;
  printf("un nombre décimal? ");
  scanf("%d", &i);
  return i;
}
```

2. Que faut-il ajouter pour que le programme compile correctement ? Faites la modification et expliquez le message d'avertissement du compilateur. Exécutez le code malgré les *warnings* (on peut exécuter malgré des messages d'avertissement : mais il est préférable de toujours les vérifier et d'essayer de les supprimer ; ces messages ne sont pas si anodins que ça).
3. Modifiez le type de retour de la fonction `f` en `double`. Compilez, exécutez et expliquez le comportement. En clair, il faut toujours déclarer les variables qu'on utilise dans un programme avant de les utiliser. Évitez autant que possible les conversions unaires (*cast*) qui ne sont pas sûres.

Exercice 7 — Un exercice chaud patate

Écrivez un programme qui fournit trois fonctions :

1. une fonction qui transforme une température en degrés Celsius en une température en degrés Fahrenheit `celsius = 5 * (fahrenheit - 32) / 9`
2. la fonction inverse, c'est-à-dire celle qui calcule une température en degrés Fahrenheit en degrés Celsius
3. une fonction qui renvoie la moyenne en km/h d'un coureur, lorsqu'on lui donne la distance en mètres et le temps en minutes mis par ce coureur pour parcourir la distance.

Exercice 8 — ASCII art

Écrivez un programme `pyramide`, qui écrit sur la sortie standard une pyramide d'un nombre de lignes donné (par exemple, lu sur l'entrée standard). Vous pourrez écrire une procédure qui écrit sur la sortie standard un caractère donné un certain nombre de fois. Le caractère pour dessiner la pyramide peut être l'étoile.

Exercice 9 — Recodons les utilitaires Unix

Écrivez un programme `mywc` (qui est une version semblable au `wc` du Shell), qui compte les lignes, les mots et les caractères d'un fichier, et écrit les résultats sur la sortie standard. Nous nous limiterons à l'entrée standard et ne traiterons pas les options. Un mot est une suite de caractères quelconques, encadrée par une ou plusieurs espaces. Une `"espace"` compte pour un caractère.

Exercice 10 — Les nombres particuliers

1. Un nombre est dit premier s'il est différent de 1, et s'il n'admet comme diviseurs que 1 et lui-même.
   (a) Écrivez une fonction qui détermine si un nombre entier est premier.
   (b) Écrivez un programme C qui lit sur l'entrée standard un nombre entier k et affiche les k premiers nombres premiers.
2. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs autres que lui-même. 6 est parfait, il admet comme diviseurs 1, 2, 3 et 6. La somme de ses diviseurs autre que lui-même est $1 + 2 + 3 = 6$. 28 est un autre nombre parfait.
   (a) Écrivez une fonction qui détermine si un nombre entier est parfait.
   (b) Écrivez un programme C qui affiche tous les nombres parfaits compris entre 2 bornes lues sur l'entrée standard.
3. Un nombre est dit doublon si le produit de ses diviseurs est un multiple de la somme de ses diviseurs. 6 est doublon, il admet comme diviseurs 1, 2, 3 et 6. La somme de ses diviseurs est $1 + 2 + 3 + 6 = 12$, le produit de ses diviseurs est $1 \times 2 \times 3 \times 6 = 36$, et 36 est un multiple de 12.
   (a) Écrivez une fonction qui détermine si un nombre entier est doublon.
   (b) Écrivez un programme C qui affiche tous les doublons compris entre 2 bornes lues sur l'entrée standard.
4. Écrivez un programme qui permet de tester les 3 fonctions précédentes. Il propose à l'utilisateur de taper :
   0 pour arrêter le programme
   1 pour calculer et afficher les k premiers nombres premiers
   2 pour calculer et afficher tous les nombres parfaits compris entre 2 bornes
   3 pour calculer et afficher tous les doublons compris entre 2 bornes