TP1 – Les bases

Projet de programmation M1

22 Septembre 2015

Exercice 1. On peut utiliser la fonction `scanf` pour récupérer des informations saisies par l'utilisateur. Pour stocker un nombre saisi par l'utilisateur dans une variable `int n`, on pourra utiliser :

```
int n;
scanf("%d", &n);
```

1. Écrire un programme qui demande l'année courante et l'année de naissance de l'utilisateur et calcule son âge.
   Solution : Voir question suivante.

En C, il existe de nombreuses façons de tester des choses :

Commande	Résultat
`a == b`	Vrai si et seulement si `a` et `b` ont la même valeur.
`a != b`	Vrai si et seulement si `a` et `b` ont une valeur différente.
`a < b`	Vrai si et seulement si `a` est strictement inférieur à `b`.
`a <= b`	Vrai si et seulement si `a` est inférieur ou égal à `b`.
`a > b, a >= b`	idem
`condition1 && condition2`	Vrai si `condition1` et `condition2` sont vraies.
`condition1 || condition2`	Vrai si l'une des deux conditions est vraie.
`!(condition)`	Vrai si la `condition` est fausse.

2. Reprendre le programme précédent et afficher une erreur si l'année de naissance est postérieure à l'année courante.
   Solution :

```
#include <stdio.h>

int main() {
  int annee_naissance, annee_courante;
  printf("Votre annee de naissance : ");
  scanf("%d", &annee_naissance);
  printf("L'annee courante : ");
  scanf("%d", &annee_courante);
  if (annee_naissance > annee_courante)
    printf("Vous vous moquez de moi ! \n");
  else
    printf("Vous avez %d ans. \n", annee_courante-annee_naissance);
  return 0;
}
```

3. Si une année est divisible par 4, alors elle est bissextile sauf si elle est divisible par 100 et pas par 400. On rappelle que le reste de la division euclidienne de `a` par `b` s'écrit `a % b` en C. Écrire un programme qui demande une année à l'utilisateur et indique si elle est bissextile ou pas. Essayez de n'utiliser qu'un seul bloc `if-then-else`.
   Solution :

```
#include <stdio.h>

int main() {
  int a;
```

```
  printf("Annee : ");
  scanf("%d", &a);
  if (a%4==0 && (a%100 != 0 || a%400==0))
    printf("%d est bissextile.", a);
  else
    printf("%d n'est pas bissextile.", a);
  return 0;
}
```

4. À votre avis, que fait ce programme :

```
#include <stdio.h>

int main() {
  int x=0;
  if (x=5) {
    printf("La valeur de x est %d.\n", x);
  }
  else {
    printf("On sait bien que x vaut 0.\n");
  }
  return 0;
}
```

Exécutez-le sur votre machine. Quoi ?!

Solution : Le programme affiche "La valeur de x est 5." contre toute (?) attente. Si vous avez compilé avec l'option `-Wall`, gcc a dû vous indiquer un warning mais sinon, il reste totalement silencieux. Il ne faut pas confondre l'affectation `=` avec la comparaison d'égalité `==`. Si on veut tester si x vaut 5, il faut taper `x==5`. Si on tape `x=5` dans une condition, on donne la valeur 5 à x et on retourne la valeur 5. Comme c'est une valeur non-nulle, C interprète cela comme une condition vraie et exécute la branche positive du `if`. D'où ce comportement bizarre : dorénavant, **pas de = dans les conditions**.

Exercice 2. On considère le programme suivant :

```
#include <stdio.h>

int main() {
  int n = 1;
  while(n != 0) {
    printf("Entrez un nombre (0 pour quitter) : ");
    scanf("%d", &n);
    printf("%d*%d vaut %d.\n", n,n,n*n);
  }
  return 0;
}
```

1. Que fait ce programme ? Modifiez-le pour qu'il s'arrête lorsque l'utilisateur entre un entier divisible par 3.

   Solution : Ce programme demande une valeur en boucle à l'utilisateur jusqu'à ce qu'il entre 0. À chaque tour de boucle, le programme affiche le carré du nombre saisi. Pour modifier la condition d'arrête, il faut remplacer la ligne `while(n!=0)` par `while(n%3 != 0)` : on change la condition d'arrêt de la boucle.

2. Écrire un programme qui demande en boucle un entier à l'utilisateur jusqu'à ce qu'il entre l'entier 0. Votre programme affiche alors la somme de tous les entiers saisis jusque-là.

   Solution :

```
#include <stdio.h>

int main() {
  int somme = 0, n = 1;
  while(n != 0) {
    scanf("%d",&n);
    somme += n;
  }
  printf("La somme vaut %d\n", somme);
```

```
  return 0;
}
```

3. La machine à café : un café à la machine vaut 40 centimes. Écrire un programme qui demande en boucle à l'utilisateur la valeur (en centimes) des pièces qu'il entre dans la machine. Lorsqu'il a mis assez de monnaie, la machine arrête de demander des pièces et affiche la monnaie due. Par exemple :

   ```
   Entrez une pièce : 5
   Entrez une pièce : 20
   Entrez une pièce : 50
   Merci, je vous dois 35 centimes.
   ```

   Solution : voir plus bas.

Lorsqu'il faut gérer plusieurs cas en C, on peut utiliser la construction `switch` ainsi :

```
switch(x) { // x est une variable ; int par exemple
 case 0:
   // cas x=0...
   break;
 case 1:
   // cas x=1
 case ...
 default:
   // comportement par defaut
}
```

4. Améliorez ce programme pour qu'il propose d'abord à l'utilisateur de choisir entre un café (40c), un chocolat (45c) et un capuccino (50c). Le programme doit aussi refuser les pièces qui n'existent pas.

   Solution :

```
#include <stdio.h>

int main() {
  int prix, boisson = 3, piece;

  while(boisson > 2) {
    printf("Choisissez une boisson en tapant son numero :\n \t 0 : Cafe \n\t 1 :
        Chocolat \n\t 2 : Cappuccino\n>");
    scanf("%d", &boisson);

    switch(boisson) {
    case 0:
      prix = 40;
      break;
    case 1:
      prix = 45;
      break;
    case 2:
      prix = 50;
      break;
    default:
      printf("Choix non disponible.\n");
      break;

    }
  }
  while(prix > 0) {
    printf("Il faut encore payer %d centimes. \n", prix);
    printf("Inserez une piece : ");
    scanf("%d", &piece);
    if (piece == 1  || piece == 2  || piece == 5  ||
        piece == 10 || piece == 20 || piece == 50 ||
        piece == 100 || piece == 200)
      prix -= piece;
    else
      printf("C'est une fausse piece.\n");
```

```
  }

  if (prix < 0)
    printf("Merci. Je vous rends %d centimes.\n", -prix);
  else
    printf("Merci.\n");
}
```

Exercice 3. Il existe plusieurs façons de représenter les entiers en C. C'est à vous de prévoir les besoins de votre programme pour savoir quelle représentation choisir.

Nom	Taille	Entiers représentés	Utilisation avec `printf`
`short`	16 bits	de -2^{15} à $2^{15}-1$	%hd
`unsigned short`	16 bits	de 0 à $2^{16}-1$	%hu
`int`	32 bits	de -2^{31} à $2^{31}-1$	%d
`unsigned int`	32 bits	de 0 à $2^{32}-1$	%u
`long`	64 bits	de -2^{63} à $2^{63}-1$	%l
`unsigned long`	64 bits	de 0 à $2^{64}-1$	%lu

1. Écrire un programme qui a une variable `short s` qui vaut $32767 = 2^{15}-1$. Affichez $s+1$. Qu'observez-vous ?

   Solution : On observe que la valeur affichée est -32768 soit -2^{15}. C'est ce qu'on appelle un overflow, un dépassement de capacité. Il faut imaginer les entiers répartis sur un cercle. Quand on arrive à la plus grande valeur et qu'on ajoute un, on recommence à la plus petite.

2. Changer le type de `s` en `unsigned short`. Qu'observez-vous ?

   Solution : Si vous avez oublié de modifier le `%hd` dans le `printf`, votre programme affichera toujours -32768. Cela vient du fait que la valeur est convertie en `short` avant d'être affichée ; il y a overflow lors de la conversion. Si en revanche, vous avez pensé à modifier `%hd` en `%hu` pour lui indiquer qu'on souhaite afficher un `unsigned short`, vous verrez le bon résultat : il n'y a plus d'overflow.

3. La suite de Fibonacci est définie ainsi : $u_0 = 1$, $u_1 = 1$ et $u_{n+2} = u_{n+1} + u_n$. Écrire un programme qui a une variable `int n` et qui calcule et affiche u_n. À partir de quelle valeur de n votre programme n'est-il plus correct ? (**Astuce** : utilisez une boucle `while`).

   Solution : Avec des entiers `long`, on peut aller jusqu'à $n = 91$ et $n = 92$ avec des `unsigned long`. On détecte l'overflow lorsque $u_{n+1} \leq u_n$.

```
#include <stdio.h>

int main() {
  int n = 0;
  unsigned long un=1,un1=1;
  while(un <= un1) {
    unsigned long tmp = un;
    printf("%d : %lu\n",n,un);
    un = un1;
    un1 += tmp;
    n++;
  }
  printf("Fin : %d : %lu \n",n,un);
  return 0;
}
```

À rendre pour le 28 Septembre

Envoyez-moi par mail (fcapelli@math.univ-paris-diderot.fr) pour le 28 Septembre, les réponses à l'exercice suivant :

Exercice 4. Avec une boucle `while`, écrire un programme `premier.c` qui demande un entier à l'utilisateur et indique si ce nombre est premier. S'il n'est pas premier, affichez son plus petit diviseur après 1.