TP1 : premiers pas en C

Algorithmique et projet de programmation — M1 mathématiques

20 septembre 2016

1 Intéraction avec l'utilisateur

Rappelons que l'on utilise la fonction `printf`, de la bibliothèque `stdio.h`, pour afficher des chaînes de caractères à l'écran. Le nom de cette fonction signifie *print formatted*, ce qui signifie qu'elle attend une chaîne de cractère formattée : en termes plus simples, son argument est une châine de caractères pouvant contenir des "jokers". Par exemple :

```
printf("Hello, World!\n");

int n = 10;
while(n>0){
  printf("%d...", n);
  n--;
}
printf("Go!");
```

La fonction `C` qui demande une entrée clavier à l'utilisateur est `scanf` (pour *scan formatted*), située dans la bibliothèque `stdio.h`. On se contentera de l'utiliser pour demander des *nombres* à l'utilisateurs, comme suit :

```
int n ;
printf("Choose a number: ");
scanf("%d", &n); //remark the &
```

Remarquez, par rapport à l'utilisation de la fonction `printf`, que l'argument est préfixé par une esperluette "`&`", ajoutée **juste devant** le nom de variable (ici `n`). C'est qu'il faut passer en paramètres à `scanf` l'*adresse* en mémoire de la variable `n` et non la variable elle-même : on dit que `&n` est une référence sur `n`. Vous comprendrez l'exacte signification de tout cela quand vous aborderez les pointeurs ; en attendant vous pouvez juste prétendre que c'est une curiosité de la fonction `scanf`.

Exercice 1. Écrire un programme qui demande deux entiers `n` et `m` à l'utilisateur et lui renvoie le quotient et le reste de la division euclidienne de `n` par `m`.

2 Conditions booléennes

Les structures de contrôle `if-then-else` permettent d'influencer le comportement du programme en fonction de la valeur de vérité d'un booléen. Typiquement, ces valeurs booléennes seront de comparaisons numériques. Exemple :

```c
if(n!=0){
  printf("Division by %n allowed.\n", n);
}
else {
  printf("Division by 0 is not allowed.\n");
}
```

Le tableau ci-dessous donne les relations de comparaison numérique en C :

Relation	Nom	Comportement attendu
`a<b`	lesser than	vrai ssi `a` strictement inférieur à `b`
`a<=b`	lesser than	vrai ssi `a` inférieur ou égal à `b`
`a>b`	lesser than	vrai ssi `a` strictement supérieur à `b`
`a>=b`	lesser than	vrai ssi `a` supérieur ou égal à `b`
`a==b`	lesser than	vrai ssi `a` égal à `b`
`a!=b`	lesser than	vrai ssi `a` différent de `b`

On peut également combiner des booléens pour obtenir des conditions plus complexes. Les opérations (les plus courantes) sur les booléens en C sont les suivantes :

Opération	Nom	Comportement attendu
`bool1 && bool2`	conjunction	vrai ssi `bool1` et `bool2` sont vrais
`bool1 || bool2`	disjunction	vrai ssi l'un de `bool1` et `bool2` est vrai
`!bool`	negation	vrai ssi `bool` est faux

Exercice 2. Écrire un programme qui demande à l'utilisateur d'entrer une année et détermine si celle-ci est bissextile. Rappelons qu'une année est bissextile si elle est divisible par 4 mais pas par 100, ou alors par 400.

3 C'est l'histoire d'un type...

Remarque. La manipulation d'entiers en C ne se limite pas au type `int`. Ci-dessous un tableau récapitulatif des différents types, leurs domaines et leurs jokers dans les chaînes formatées :

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 $2^{63}-1$	`%ld`
`unsigned long`	64 bits	de 0 à $2^{64}-1$	`%lu`

Exercice 3. La suite de Fibonacci est définie par $u_0 = 1, u_1 = 1$ et $u_n = u_{n-1} + u_{n-2}$ pour tout $n \geq 2$. Écrire un programme qui calcule toutes les valeurs possibles de la suite de Fibonacci et s'arrête quand le résultat dépasse le domaine du type choisi pour représenter les entiers dans votre programme.

Exercice 4. Modifier le programme précédent pour laisser le choix à l'utilisateur du type à utiliser. Indice : on affichera en début de programme un menu de la forme

```
1. short
2. unsigned short
3. int
4. unsigned int
5. long
6. unsigned long
Type to use (enter corresponding number):
```

4 Let's play

La bibliothèque `stdlib.h` contient une fonction qui génère un nombre aléatoire : `rand()` renvoie un entier aléatoire entre `0` et une (grande) valeur abitraire fixée par `C` et appelée `RAND_MAX`. Pour que `rand` se comporte réellement comme un génrateur aléatoire, il faut l'initialiser avec une *graine*, par exemple l'heure :

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

int main(int argc, char** argv){

  srand(time(NULL)); //initialize random generation
  r = rand(); //r is now a random number

  return EXIT_SUCCESS;
}
```

Exercice 5. Écrire un programme qui génère un nombre aléatoire entre 0 et un entier `n` fixé (exclu).

Exercice 6. Écrire un jeu du *plus ou moins*. Le jeu se déroule comme suit :

(1) un nombre `bound` est fixé à l'avance,

(2) un nombre de coups `chances` est fixé à l'avance,

(3) le programme génère un nombre aléatoirement entre `0` et `bound`,

(4) l'utilisateur est invité à entrer un nombre et le programme indique si le nombre cherché est plus grand ou plus petit (ou égal),

(5) on répète cette instruction jusqu'à ce que le joueur trouve le nombre (et dans ce cas il gagne) ou que le nombre de coups soit écoulé (et dans ce cas il perd).

Exercice 7. Améliorer le programme en proposant différent niveau à l'utilisateur. On pourra par exemple afficher un menu en début de programme de la forme :

```
1. Level 1 -- number between 0 and 10 -- 5 chances
2. Level 2 -- number between 0 and 100 -- 20 chances
3. Level 3 -- number between 0 and 100 -- 5 chances
4. Level 4 -- number between 0 and 1000 -- 10 chances
Level (enter corresponding number):
```

Exercice 8. Améliorer encore un peu le jeu en ajoutant une dernière option au menu où on laisse le joueur libre de rentrer lui même les nombres `bound` et `chances`.

5 À rendre pour le 27 septembre 2016

Exercice 9. Écrire un programme demandant à l'utilisateur de rentrer un entier et retournant si celui-ci est premier ou non. Dans le cas où le nombre n'est pas premier, on affichera le plus petit diviseur (strictement supérieur à 1).

Exercice 10. Compiler sans le flag `-Wall` et exécuter le programme suivant :

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

int main(int argc, char** argv){
  int n=0;
  printf("The value of n is %d.\n", n);

  if(n=1){
    printf("This case is obviously useless. Or is it?\n");
  }
  else {
    printf("This is what is gonna be printed!\n");
  }

  return EXIT_SUCCESS;
}
```

Le corriger pour qu'il ait le comportement attendu et expliquer (en commentaire de votre code) ce qu'il se passe dans le programme original. Indice : recompiler avec `-Wall`.