Boucles et conditions

Programmation séquentielle en C, 2024-2025

Orestis Malaspinas (A401)

2024-09-24

Informatique et Systèmes de Communication, HEPIA

Quiz

Que fait le code suivant?

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  printf("Enter n: ");
  int n = 0;
  scanf("%d", &n);
  int res = 0;
  for (int i = 0; i <= n; ++i) {
    res += i;
  }
  printf("For = %d, the result is %d\n", n, res);
  if (res != n * (n+1) / 2) {
    printf("Error: the answer is wrong.\n");
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Génération d'un exécutable

- Pour pouvoir être exécuté un code C doit être d'abord compilé (avec `gcc` ou `clang`).
- Pour un code `prog.c` la compilation "minimale" est

```
$ clang prog.c
$ ./a.out # exécutable par défaut
```

- Il existe une multitude d'options de compilation:

```
$ clang -std=c11 -Wall -Wextra prog.c -o prog
```

1. `-std=c11` utilisation de C11.
2. `-Wall` et `-Wextra` activation des warnings.
3. `-o` défini le fichier exécutable à produire en sortie.

Structures de contrôle: `if` .. `else if` .. `else` (1/2)

Syntaxe

```
if (expression) {
    instructions;
} else if (expression) { // optionnel
                         // il peut y en avoir plusieurs
    instructions;
} else {
    instructions; // optionnel
}
```

```
if (x) { // si x s'évalue à `vrai`
    printf("x s'évalue à vrai.\n");
} else if (y == 8) { // si y vaut 8
    printf("y vaut 8.\n");
} else {
    printf("Ni l'un ni l'autre.\n");
}
```

Structures de contrôle: `continue`, `break`

- `continue` saute à la prochaine itération d'une boucle.

```
int i = 0;
while (i < 10) {
    if (i == 3) {
        i += 1;
        continue;
    }
    printf("%d\n", i);
    i += 1;
}
```

- `break` quitte le bloc itératif courant d'une boucle.

```
for (int i = 0; i < 10; i++) {
    if (i == 3) {
        break;
    }
    printf("%d\n", i);
}
```

La fonction `main()` (1/2)

Généralités

- Point d'entrée du programme.
- Retourne le code d'erreur du programme:
  - 0: tout s'est bien passé.
  - Pas zéro: problème.
- La valeur de retour peut être lue par le shell qui a exécuté le programme.
- `EXIT_SUCCESS` et `EXIT_FAILURE` (de `stdlib.h`) sont des valeurs de retour **portables** de programmes C.

La fonction `main()` (2/2)

Exemple

```
int main() {
    // ...
    if (error) {
        return EXIT_FAILURE;
    } else {
        return EXIT_SUCCESS;
    }
}
```

- Le code d'erreur est lu dans le shell avec `$?`

```
$ ./prog
$ echo $?
0 # tout s'est bien passé par exemple
$ if [ $? -eq 0 ]; then echo "OK"; else echo "ERROR"; fi
ERROR # si tout s'est mal passé
```

Entrées/sorties: `printf()` (1/2)

Généralités

- La fonction `printf()` permet d'afficher du texte sur le terminal:

```
int printf(const char *format, ...);
```

- Nombre d'arguments variables.
- `format` est le texte, ainsi que le format (type) des variables à afficher.
- Les arguments suivants sont les expressions à afficher.

Entrées/sorties: printf() (2/2)

Exemple

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("Hello world.\n");
    int val = 1;
    // %d => int
    printf("Hello world %d time.\n", val);
    // %f => float
    printf("%f squared is equal to %f.\n",
           2.5, 2.5*2.5);
    // %s => string
    printf("Hello world %s.\n", "Hello world");
    return EXIT_SUCCESS;
}
```

Entrées/sorties: `scanf()` (1/2)

Généralités

- La fonction `scanf()` permet de lire du texte formaté entré au clavier:

```
int scanf(const char *format, ...);
```

- `format` est le format des variables à lire (comme `printf()`).
- Les arguments suivants sont les variables où sont stockées les valeurs lues.

Entrées/sorties: scanf() (2/2)

Exemple

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

int main() {
    printf("Enter 3 numbers: \n");
    int i, j, k;
    scanf("%d %d %d", &i, &j, &k); // !!! &
    printf("You entered: %d %d %d\n", i, j, k);

    return EXIT_SUCCESS;
}
```

Nombres aléatoires: rand(), srand() (2/2)

```
$ man 3 rand
The  rand()  function  returns  a  pseudo-random  integer
in  the  range  0  to RAND_MAX inclusive (i.e., the
mathematical range [0, RAND_MAX]).
```

Exemple

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    srand(0);           // every run will be identical
    srand(time(NULL)); // every run will be be different
    for (int i = 0; i < 50; ++i) {
        printf("%d\n", rand() % 50); // à quoi sert % 50?
    }
    return EXIT_SUCCESS;
}
```

Le cas du nombre secret

But du programme

- Faire chercher un nombre aléatoire,
- Le nombre est compris entre 0 et une borne max.

Quelles sont les étapes?

1. Donner une borne maximale à l'ordinateur, MAX.

Le cas du nombre secret

But du programme

- Faire chercher un nombre aléatoire,
- Le nombre est compris entre 0 et une borne max.

Quelles sont les étapes?

1. Donner une borne maximale à l'ordinateur, MAX.
2. Faire tirer un nombre aléatoire à l'ordinateur entre 0 et MAX-1.

Le cas du nombre secret

But du programme

- Faire chercher un nombre aléatoire,
- Le nombre est compris entre 0 et une borne max.

Quelles sont les étapes?

1. Donner une borne maximale à l'ordinateur, MAX.
2. Faire tirer un nombre aléatoire à l'ordinateur entre 0 et MAX-1.
3. Le · la joueur · se joue un nombre.

Le cas du nombre secret

But du programme

- Faire chercher un nombre aléatoire,
- Le nombre est compris entre 0 et une borne max.

Quelles sont les étapes?

1. Donner une borne maximale à l'ordinateur, MAX.
2. Faire tirer un nombre aléatoire à l'ordinateur entre 0 et MAX-1.
3. Le · la joueur · se joue un nombre.
4. L'ordinateur vérifie la réponse:
    - Si correct le jeu est fini,
    - Sinon indiquer si le nombre secret est plus grand ou plus petit et revenir à 3.

La fin

1. Y a-t-il des questions?

La fin

1. Y a-t-il des questions?
2. Si oui, répondre et revenir à 1, sinon aller faire l'exercice en A432/433/406.