Éléments d'informatique – Cours 6. Boucle while, expressions booléennes. Algorithmes élémentaires.

Pierre Boudes

22 octobre 2011

Ce semestre 

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectaction
  - Évaluation d'expressions, expressions booléennes
  - Instructions de contrôle : if, for, while
  - *Types de données :* entiers, *caractères, réels,* tableaux, *struct*
  - Fonctions d'entrées/sorties (scanf/printf)
  - *Écriture et appel de fonctions*
  - *Débogage*
- Notions de compilation
  - Analyse lexicale, analyse syntaxique, analyse sémantique ; préprocesseur du C (include, define) ; Édition de lien
- Algorithmes élémentaires
- Méthodologie de résolution, manipulation sous linux

L'instruction de contrôle while

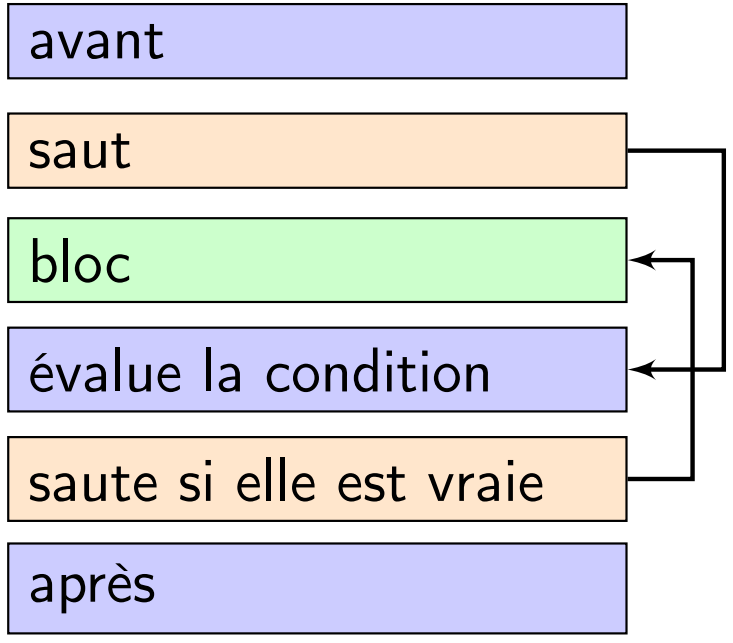
Syntaxe :

`while (`*condition*`) {` *bloc* `}`.

Code source

```
/* avant */
while ( ... )
{
  ...
}
/* après */
```

Schéma de traduction



Trace

```
1   int main()
2   {
3       /* Declaration et initiali
4       int x;
5       int nb = 1; /* nombre de c
6
7       printf("Entrer un nombre p
8       scanf("%d", &x);
9
10      while (x > 9) /* tant que
11      {
12          x = x / 10; /* enlever
13          nb = nb + 1; /* compter un chiffre de plus */
14      }
15      printf("ce nombre a %d chiffres decimaux\n", nb);
16
17      /* Valeur fonction */
18      return EXIT_SUCCESS;
19  }
```

L'utilisateur saisit 6071.

ligne	x	nb	sortie écran
initialis.	?	1	
7			Entrer...
8	6071		
12	607		
13		2	
12	60		
13		3	
12	6		
13		4	
15			ce nombre a 4...
18	Renvoie EXIT_SUCCESS		

For ou while ?

- Un `for` peut toujours être simulé par un `while` et le code machine sera identique. Il suffit d'introduire un compteur de boucle (la variable de boucle du `for`).
- Par convention, les programmeurs préfèrent utiliser un `for` lorsque le nombre d'itérations est connu à l'avance. Par exemple, pour faire la somme des éléments d'un tableau. Dans le cas contraire, les programmeurs utilisent un `while`. Par exemple, pour chercher un élément dans un tableau.
- Maintenant que nous avons le `while`, il est possible qu'un programme ne termine jamais (`Ctrl-C`).

Expressions booléennes

Les *conditions* employées dans les structures de contrôle (`if`, `for` ou `while`) sont des expressions booléennes, pouvant être *Vrai*, *Faux* ou :

- des inégalités entre expressions arithmétiques

$$\begin{aligned} \textit{inégalité} := {} & e_1 \texttt{ < } e_2 \mid e_1 \texttt{ > } e_2 \mid e_1 \texttt{ != } e_2 \\ & \mid e_1 \texttt{ <= } e_2 \mid e_1 \texttt{ >= } e_2 \mid e_1 \texttt{ == } e_2 \end{aligned}$$

- ou des combinaisons logiques d'expressions booléennes :

$$\begin{aligned} \textit{condition} := {} & (\textit{condition}) \texttt{ \&\& } (\textit{condition}) && \text{(et)} \\ & \mid (\textit{condition}) \texttt{ || } (\textit{condition}) && \text{(ou)} \\ & \mid \texttt{!}(\textit{condition}) && \text{(non)} \\ & \mid \textit{Vrai} \mid \textit{Faux} \mid \textit{inégalité} && \text{(cas de base)} \end{aligned}$$

Constantes booléennes

- Certains langages possédent un type booléen (admettant deux valeurs *true* et *false*) pour les expressions booléennes.
- En langage C, les expressions booléennes sont de type entier (`int`), l'entier *zéro* joue le rôle du Faux, l'entier *un* joue le rôle du Vrai et tout entier différent de zéro est évalué a vrai.
- On se donne deux constantes symboliques :

```
/* Declaration des constantes et types utilisateur */
#define TRUE 1
#define FALSE 0

int main()
{
    int continuer = TRUE; /* faut-il continuer ?*/

    while (continuer)
    {
        ...
```

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas
   - boucle `while`, conditions booléennes, `if`

2'. Parcourir/générer : une *ligne* mais aussi une *surface*, un *volume*...
   - imbriquer les boucles (var. $\neq$)

6. Boucle événementielle
   - boucle while

7. Attente active
   - boucle while

Recherche d'un diviseur (test de primalité)

Exemple

Le programme cherche un entier supérieur à 1 qui divise n, s'il n'en trouve pas de plus petit que n, alors n est premier (ou bien $n = 1$).

```
31          /* test de primalite */
32          d = 2;
33          while ( premier && (d < n) ) /* sans diviseur < d */
34          {
35              if (n % d == 0) /* d divise n */
36              {
37                  printf("divisible␣par␣%d\n", d);
38                  premier = FALSE;
39              }
40              d = d + 1; /* candidat diviseur suivant */
41          }
42
43          if (premier)
44          {
45              printf("%d␣est␣premier\n", n);
46          }
47          else
48          {
49              printf("%d␣n'est␣␣pas␣premier\n", n);
```

Boucle principale en programmation événementielle

Le programme lit les événements (les actions de l'utilisateur) et les traite, continuellement, dans une boucle.

Remarque

`scanf` est un appel système bloquant (processus en attente).

Exemple

Jeu de calcul mental (additionner deux nombres au hasard) 

```
16      int continuer = TRUE; /* TRUE s'il faut continuer a jouer */

22      /* initialisation du generateur de nombres pseudo-aleatoires *
23      srand(time(NULL)); /* a ne faire qu'une fois */

31      while(continuer) /* le joueur veut faire un nouvel essai */
32      {

34          /* tirage de x */
35          x = rand() % NBMAX + 1; /* entre 1 et NBMAX inclus */

53          printf("Continuer␣(1)␣ou␣arreter␣(0)␣?\n");
54          scanf("%d", &choix);
55          if (choix == 0)
```

Boucle d'attente active

La boucle sert à attendre qu'une condition externe devienne vraie, le programme teste continuellement cette condition.

Exemple

Tic tac boum, est-ce que 10 secondes sont écoulées ? 

```
19      printf("Auto-destruction␣en␣cours,␣ctrl-C␣pour␣desamorcer␣!\n"
20      debut = time(NULL); /* date de debut (secondes depuis 1/1/1970
21      while(duree < 10) /* pas encore 10 secondes */
22      {
23          duree = time(NULL) - debut; /* duree depuis debut (seconde

42      } /* fin des 10 secondes */
43      printf("**␣BOUM␣!␣**\n");
```

Démos

Premier partiel

- Au premier partiel, il est probable que vous ayez :
  - à compléter un programme fourni et question de cours
  - simuler l'exécution (faire une trace) d'un programme fourni et question de compréhension
  - Deux exercices avec des boucles (for ou while)
  - Un autre sur if et constantes symboliques (arbre de décision)
- Au second *partiel* (au moins la moitié de votre note d'UE) :
  - mêmes types d'exercices avec différents types de données (`int`, `char`, `double`, `struct`) et des fonctions
  - le programme à modifier et la trace sont avec fonctions
  - un exercice porte spécialement sur les structures
  - exercice supplémentaire : déclarer/définir des fonctions avec $\frac{1}{2}$pt par déclaration juste et $\frac{1}{2}$pt par définition juste.
- Gérez votre temps, apprenez à lire un sujet et ne rien oublier.

Nous testons vos acquis pas votre propension à paniquer !