Faculté des Sciences de Gabès

Cours Atelier de Programmation

Chapitre 5 : Structures en C

Faîçal Felhi

felhi_fayssal@yahoo.fr

Itération

- Une itération correspond à la **répétition** d'une séquence de calcul.
- Une des 4 structures de base des algorithmes : les boucles (ou structures répétitives, ou structures itératives).
- Exemple :
  - On pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non).
  - - Mais l'utilisateur risque de taper autre chose.
  - Dès lors, le programme peut planter ou produire des résultats fantaisistes.

=> Alors, on peut mettre en place un **contrôle de saisie (pour vérifier que les données** entrées correspondent bien à celles attendues par l'algorithme).

Itération

- On peut faire cela avec une boucle SI :

Algorithme Test
Var Rep: Caractère
Debut
Ecrire "Voulez vous un café ? (O/N)"
Lire Rep
Si Rep <> "O" ET Rep <> "N" Alors
Ecrire "Saisie erronnée. Recommencez"
Lire Rep
FinSi

- Si l'utilisateur ne se trompe qu'une seule fois => OK
- Mais pour prévoir le cas de deuxième erreur, il faut rajouter un SI.
- Et ainsi de suite….

=> impasse…

Itération

- Une écriture concise pour la répétition de blocs d'instructions
- Il existe 2 sortes d'itération :
  - 1. Le nombre de répétitions est fixe
    - faire N fois
    - en C : **for**
  - 2. Le nombre de répétitions n'est pas fixe
    - Il dépend des calculs effectués par les instructions répétées
    - si ... alors repartir
    - en C : **while** ou **do**...**while**

Itération-while

- Le nombre de répétitions n'est pas fixe

  while (condition) bloc-instruction;
  - condition est une expression booléenne
- L'exécution de cette itération s'effectue par :
  - 1. évaluation de la condition
  - 2.
    - si la condition est vraie alors exécution du bloc-instruction **et** repartir (recommencer) en 1.
    - si la condition est fausse alors l'exécution est terminée.
- Tant que la condition est vraie, le bloc d'instructions est exécuté.
- A la sortie de la boucle, la condition est toujours fausse (… ne pas tester)

Itération - do...while

Le nombre de répétitions n'est pas fixe

do liste-instruction **while** (condition);

L'exécution de cette itération s'effectue par :

1.- exécution de la liste d'instructions

2.- évaluation de la condition

3.-

- si la condition est vraie alors repartir (recommencer) en 1.
- si la condition est fausse alors l'exécution est terminée.

- Exécuter la liste d'instructions tant que la condition est vraie et toujours au moins une fois !!!

Itération - do...while

Exemple

```
Main () {
   scanf("%d", &N);
   Res = 1;
   do {
         Res = Res * N ;
         N = N -1 ;                          // N est modifié
   } while (N > 0) ;                         // N est testé
   printf("%d \n", Res);
}
```

Simulation de l'exécution

- pour la saisie de 0 et pour la saisie de 5

Itération - for

Le nombre de répétitions est fixe

for (expr1; expr2; expr3;) {bloc-instructions}

L'exécution de cette itération s'effectue par :

1. initialiser le compteur d'itération par expr1
2. Tant que la condition expr2 est vraie, exécuter le bloc d'instructions
3. Le compteur est modifié par expr3 à chaque itération

Utilisation typique (boucle pour du langage de description d'algorithme)

for (i=0; i<N; i=i+1) {bloc-instructions}

Remarque générale d'utilisation

Veiller à ce que l'intervalle [début, fin] soit identifiable avant l'itération.

Itération - for

Exemple

```
scanf("%d", &N) ; res = 1 ;
for (I = 2; I <=N; I = I + 1)
        res = res * I ;
printf("%d \n", res);
```

Simulation de l'exécution (pour la saisie de 0 et pour la saisie de 5)

Simulation par while

```
scanf("%d", &N) ; res = 1 ; I = 2 ;
while (I <= N)
        { res = res * I ;
        I = I + 1;}
printf("%d", res);
```

Itération - for

Exemple d'itération avec décrémentation

```
scanf("%d", &N) ; res = 1 ;
for (I = N; I >=2; I = I - 1)
        res = res * I ;
printf ("%d", res);
```

Simulation de l'exécution pour la saisie de 0 et pour la saisie de 5

Remarque générale

- pour l'itération **for**, on peut mettre plusieurs autres variantes pour les expressions expr1, expr2 et expr3.

Programmation de Test-Carré-Parfait

Carré parfait: Quand on multiplie un nombre entier par lui-même, on obtient un carré parfait. Par exemple, 7*7=49 est un carré parfait

- Rappel de l'algorithme

Début

1. I←0

   Répéter

2. J←I*I

3. I←I+1

4. **jusqu'à** J ≥ N

5a. **Si** J = N **alors** Reponse ← Vrai

5b. **Sinon** Reponse ← Faux

   Finsi

Fin

Programme Test-Carré-Parfait

```
#include <stdio.h> ;                                    // bibliothèque
main() {
    /* Test-Carré-Parfait : Ce programme vérifie si l'entier N est ou non un carré
    parfait */
int  N, I, J  ;                                 // bloc déclaration
printf ("Donnez l'entier à tester : ");
scanf("%d", &N);                                // saisie de la valeur de N au clavier
I = 0;                                          //  initialisation de I
J = 0 ;
while (J<N)  {
    I = I+1 ;
    J = I * I  ;
}
if (J == N)
    printf("%d est un carré parfait", N);
else
    printf("%d n'est pas un carré parfait", N);
}
```

Imbrication de boucles

- Supposons que la multiplication soit interdite !

```
main(){ // Test-Carré-Parfait
   ...
   I = 0;                        // initialisation de I
   do {

         for (J=0; J<N; J=J+1) {
            J = J + I;           // calcul de J=I*I par additions successives
           ….
         }
          I = I+1;               // incrémentation de I
        }
   while (J<N);
...
```

- La boucle « for » est imbriquée dans la boucle « do...while ».