Travaux pratiques n°4 (2h): Les itératives simples en langage C

Objectifs

- Programmer des itératives simples en langage C (un seul niveau d'itération)

Documents nécessaires

- sujet du TD4 et notes de TD4
- support de cours

Partie 1: Introduction pratique au langage C

Les itératives en C:

Le cours présente comment traduire en C les itératives étudiées en algorithimique. La syntaxe générale des trois types d'itérative est la suivante:

1) Le while

1. Cas général:

```
while (expression)
        {
        instruction;
        ...
        }
/* fin du while */
```

Tant que l'expression est vraie (!=0), on effectue les instructions du bloc. L'expression est évaluée avant d'effectuer les instructions du bloc (si elle est fausse dès le début, les instructions ne sont jamais effectuée).

2. Cas particulier

```
while (expression) instruction; /* une seule instruction à effectuer */
/* fin du while */
```

exemple : Un programme qui affiche toutes les puissances de 2, jusqu'à une valeur maximale donnée par l'utilisateur.

```
#include <stdio.h>
void main(void)
 {
  int puissance=1,max;
  printf("nombre maximal désiré (ne pas dépasser 16000) ?");
  scanf("%d",&max);
  while (puissance<max) printf("%d\n",puissance*=2);
/* fin du while */
}
```

2) Le do ...while:

1. Cas général

```
 do
          {
           instruction;
          ...
          }
 while (expression);   /* attention au ; final /
 /* fin do ... while */
```

comme while, mais les instructions sont effectuées au moins une fois, avant la première évaluation de l'expression.

2. Cas particulier

```
 do instruction; while (expression) ; /* une seule instruction à effectuer */
 /* fin du do ... while */
```

exemple :

```
#include <stdio.h>
int main(void)
 {
  int a;
  do
        {
        printf("veuillez entrer le nombre 482");
```

```
        scanf("%d",&a);
        }
  while (a!=482);
  /* fin du do... while */
  printf("c'est gentil de m'avoir obéi");
 }
```

3) Le for:

1. Cas général

```
for ( expr_initiale; expr_condition; expr_incrémentation)
        {
                                                    instruction;
                                                    ...
        }
/* fin du for */
```

Cette boucle est surtout utilisée lorsque l'on connaît à l'avance le nombre d'itérations à effectuer. L'**expr_initiale** est effectuée une fois, en premier. Puis on teste la condition. On effectue les instruction puis l'incrémentation tant que la condition est vraie. L'instruction et l'incrémentation peuvent ne jamais être effectuées si **expr_condition** est fausse dès le début. Cette itérative est équivalente à :

```
expr_initiale;
while (expr_condition)
        {
        instruction;
        ...
        expr_incrémentation;
        }
```

3. Cas particulier

   Une ou plusieurs des trois expressions peuvent être omises, le bloc d'instruction peut être vide.

exemple :

```
#include                                    <stdio.h>
void                                        main(void)
{
 int i,n;
```

```
float note,somme=0,moyenne;
printf("nombre de notes ? ");
scanf("%d",&n);
for(i=0;i<n;i++)
        {
         printf("entrez votre %dième note",i+1);
         scanf("%f",&note);
         somme+=note;
        }
/* fin du for */
 moyenne=somme/n;
 printf("moyenne calculée :%5.2f\n",moyenne);
}
```

Précisions sur scanf:

Pour être honnête, **scanf** est difficile à utiliser correctement et pose de gros problèmes pour les programmeur débutant. Je vous incite à imprimer le man du scanf et à le lire en détail. Par exemple supposez que vous souhaitiez lire un entier positif avec filtrage, vous allez écrire une itérative du genre:

```
do
        {
        printf("Saisissez un nombre entier positif : ");
        scanf("%d", &valeur);
        }
while(valeur < 0);
```

Mais ce **do ... while** boucle à l'infini dans le cas par exemple ou l'utilisateur tape un caractère au lieu d'un entier!!! La raison est le suivante: le **scanf** ne retire pas le “retour chariot” (touche entrée: code ASCII 32) du tampon de saisie dans le cas ou le format attendu n'est pas celui entré. Du coup, à la prochaine itération **scanf** lit le “retour chariot” du premier tour tout en le laissant dans le tampon etc...

Une solution est de tester le retour de **scanf** et de vider son tampon grâce à **getchar();** qui lit un caractère dans le tampon d'entrée (donc de fameux “retour chariot”).

```
#include <stdio.h>

int main(void)
{
```

```
        int valeur, count;
        do
                {
                printf("Saisissez un nombre entier positif : ");
                count = scanf("%d", &valeur);
                /* si count == 0, alors erreur, mais il reste \n (retour chariot) dans le
buffer... */
                if ( count == 0 ) getchar();
                }
        while((count < 1) || (valeur < 0));
        printf("Valeur saisie : %d\n", valeur);
        return (0);
}
```

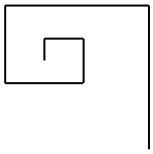
Vous pouvez par ailleurs utiliser **getchar()** (plus sain dans son utilisation que **scanf**) quand vous voulez lire un caractère au clavier.

```
#include <stdlib.h>   /* pour utiliser le getchar */

int main(void)
{

                char                                                    c;
                c = getchar(); /* c est initialisé avec le caractère lu au clavier par getchar()
                mais attention getchar laisse aussi le retour chariot dans le buffer car il n'y lit
                qu'un caractère */

}
```

Partie 2: Traduire des corps d'algorithmes détaillés avec itératives en langage C

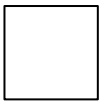
Exercice 1: La tortue ... le retour

- Au moyen de la tortue écrire un programme C traçant la spirale suivante sachant que :



1. le premier segment (au centre de la spirale) est de taille *longueurMin* (*longueurMin* > *0*)
2. la longueur de chaque segment est le double de celle du segment précédent
3. la longueur du dernier segment est inférieure ou égale à *longueurMax* (*longeurMax* ≥ *longueurMin*)

- (extrait du contrôle court de 2006) Écrire un programme en C qui fera dessiner à la tortue un polygone dont chaque coté à une longueur de 10 et dont le nombre de cotés sera demandé à l'utilisateur. Le nombre de cotés devra être supérieur à 2 et inférieur à 10 (il faut filtrer les mauvaises valeurs). Ainsi si l'utilisateur donne 4 pour le nombre de coté la tortue affichera un carré de coté 10:



Au départ, la tortue regarde vers le haut.

Exercice 2

- Écrire le programme C correspondant à l'algorithme de l'exercice 3 du TD4
- (Si vous avez le temps ou en travail personnel) Écrire le programme C correspondant à l'algorithme de l'exercice 2 du TD4