Les fonctions

5 Trace de programme contenant plusieurs fonctions

Il faut faire un tableau de trace par appel de fonction. Les tableaux sont fait au fur et à mesure de l'exécution du programme. On nomme chaque tableau par l'appel de la fonction et on ajoute une dernière ligne indiquant la valeur de la fonction renvoyée.

```
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* déclaration de fonctions utilisateurs */
/* renvoie le carre de x */
double carre(double x);

int main()
{
	double x = 3.1;

	printf(«le carre de %d est %d.\n »,x,carre(x));
	printf(«le carre de %d est %d.\n »,carre(3.1),carre(carre(3.1));

	/* valeur fonction */
	return EXIT_SUCCESS; /* la fonction vaut la valeur de l'expression EXIT_SUCCESS */
}

/* implantation de fonctions utilisateur */
double carre(double x)
{
	/* valeur fonction */
	return x * x;
}

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

void modification_param(int i);

int main()
```

```
{
  int i = 0;

  /* représenter l'occupation mémoire */

  modification_param(i);

  printf("i = %d\n",i);

  return EXIT_SUCCESS;
}

void modification_param(int i) {
  /* représenter l'occupation mémoire */

  i = 10;

  /* représenter l'occupation mémoire */
}
```

Exercice : Réécrire la fonction puissance sans utiliser de variables locales, autres que les paramètres formels.

6 Fonctions sans valeur de retour

Il est possible de définir des fonctions sans valeur de retour. Le résultat de l'exécution de la fonction doit donc être communiqué par effets de bord : écriture à l'écran, dans un fichier, sur límprimante, modifications de varibales définies a l'extérieur de la fonction (semestre 2). L'analogie avec la notion de fonction mathérmatique est perdue, et l'on préfère parler de procédure ou de routine en informatique pour ce type de fonctions en C. Une prototype de procédure est de la forme :

```
void nom_fonction(type1 paramètre1,type2 paramètre2,...,typen paramètren);
```

void = néant, vide.

Exemple :

```
void affichage_rectangle_d_etoiles(int longueur,int largeur)
{
   int i; /* var. de boucle */
   int j; /* var. de boucle */
```

```
    for(i = 0;i < largeur;i = i + 1) /* largeur fois */
    {
        /* affiche longueur etoiles */
        for(j = 0;j < longueur;j = j + 1) /* longueur fois */
        {
            /* affiche une etoile */
            printf("*");
        }
        /* j >= longueur */

        /* passe a la ligne suivante */
        printf("\n");
    }
    /* i >= largeur */
}
```

La fin d'exécution est implicite lorsqu'on arrive à la fin du corps de la procédure. Il est cependant possible de la marquer explicitement si plusieurs fins sont possibles avec l'instruction :

return;

Remarque : Les procédures peuvent vues comme la définition de nouvelles instructions de haut niveau, correspondant à la réalisation de certaines tâches. Certains langages de programmation sont construits autour de la notion de procédures, on parle de programmation procédurale (http://en.wikipedia.org/wiki/Procedural_programming).

à voir, plutôt E/S avancées au semestre 2 ?:

En C, il est de tradition de définir des fonctions plutôt que des procédures...

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

Upon successful return, these functions return the number of characters printed (not including the trailing '\0' used to end output to strings).

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

These functions return the number of input items successfully matched and assigned, which can be fewer than provided for, or even zero in the event of an early matching failure.

7 Fonctions sans paramètres formels

Il est également possible de définir des fonctions sans entrée, bien que leur utilisation soit rare. Sans entrée, leur exécution produit toujours le même résultat et on préfère définir des constantes symboliques, ce qui est plus clair et plus rapide à l'exécution.

Exemple : on préfère

#define PI 3.1415926 /* approximation de PI avec 6 décimales */

à

```
double pi()
{
        return  3.1415926;
}
```

Utilisation possible :

```
int lit_entier()
{
        int a;

        scanf(« %d »,&a);

        return a;
}
```

Dans la pratique, elles sont souvent définie sans valeur de retour et ne sont utilisées que pour leur effets de bord : une procédure.

Exemple :

```
void affichage_menu()
{
        printf(« Menu : \n »);
        printf(« 1) Cassoulet\n »);
        printf(« 2) Saucisses-lentilles\n »);
        printf(« 3) Choucroute\n »);
}
```

8Erreurs courantes

Oublier de déclarer un nom de variable dans la définition d'une fonction :

```
int addition(int, int) /* oh, on a oublie les noms de variables ! */
{
        int c;
        c = a+b;
    return c;
```

```
}
```

Il y a aussi les `;` en trop :

```
int addition(int a, int b);     /* il y a un ';' en trop !!! */
{
        return a+b;
}
```

Une autre erreur très fréquente est de redéclarer les noms de variables :

```
int addition(int a, int b)
{
        int a, b;       /*  variables deja declarees dans l'en-tete de
la fonction */
        return a+b;
}
```

Si une fonction n'est pas de type `void`, attention à bien retourner une valeur :

```
int addition(int a, int b)
{
        int c;
        c = a + b;
        return; /* on oublie de retourner un entier ! */
}
```
