Chapitre 8: LES FONCTIONS

La structuration de programmes en sous-programmes se fait en C à l'aide de ***fonctions***. Les fonctions en C correspondent aux fonctions *et* procédures en Pascal et en langage algorithmique.

Fonctions prédéfinies dans des bibliothèques standard (**printf** de *<stdio>*, **strlen** de *<string>*, **pow** de *<math>*, etc.).

8.1. Modularisation de programmes

8.1.1. La modularité et ses avantages

Modules

Un ***module*** désigne une entité de données et d'instructions qui fournissent une solution à une (petite) partie bien définie d'un problème plus complexe. Un module peut faire appel à d'autres modules, leur transmettre des données et recevoir des données en retour. L'ensemble des modules ainsi reliés doit alors être capable de résoudre le problème global.

***Avantages* d'un programme modulaire:**

* ***Meilleure lisibilité***

* ***Diminution du risque d'erreurs***

* ***Possibilité de tests sélectifs***

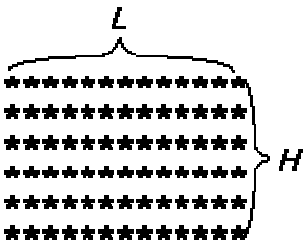
* ***Réutilisation de modules déjà existants***

* ***Simplicité de l'entretien***

* ***Favorisation du travail en équipe***

8.1.2. Exemples de modularisation en C

a) Exemple 1: Afficher un rectangle d'étoiles



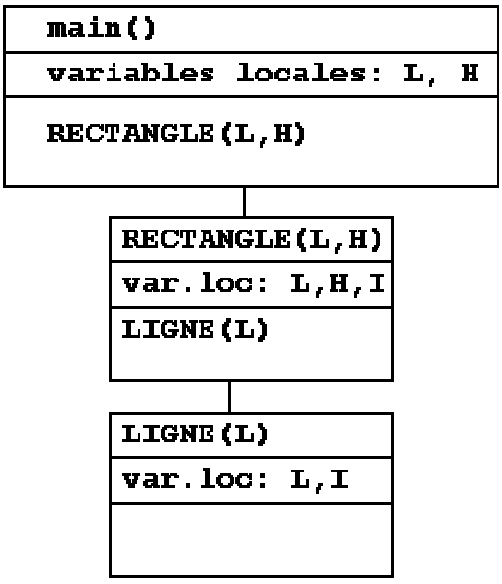

```
#include <stdio.h>
main()
{
 /* Prototypes des fonctions appelées par main */
 void RECTANGLE(int L, int H);
 /* Déclaration des variables locales de main */
 int L, H;
 /* Traitements */
          printf("Entrer la longueur (>= 1): ");
          scanf("%d", &L);
         printf("Entrer la hauteur  (>= 1): ");
         scanf("%d", &H);
 /* Afficher un rectangle d'étoiles */

RECTANGLE(L,H);

return 0;
}
```

Pour que la fonction soit exécutable par la machine, il faut encore spécifier la fonction RECTANGLE:

```
void RECTANGLE(int L, int H)
{
 /* Prototypes des fonctions appelées */
 void LIGNE(int L);
 /* Déclaration des variables locales */
 int I;
/* Afficher H lignes avec L étoiles  */
 for (I=0; I<H; I++)
     LIGNE(L);
}
```

Pour que la fonction RECTANGLE soit exécutable par la machine, il faut spécifier la fonction LIGNE:

```
void LIGNE(int L)
{
 /* Affiche à l'écran une ligne avec L étoiles */
 /* Déclaration des variables locales */
 int I;
 /* Traitements */
 for (I=0; I<L; I++)
     printf("*");
 printf("\n");
}
```



b) Exemple 2: Tableau de valeurs d'une fonction

Soit F la fonction numérique définie par $F(X) = X^3-2X+1$. On désire construire un tableau de valeurs de cette fonction. Le nombre N de valeurs ainsi que les valeurs de X sont entrés au clavier par l'utilisateur.

Exemple

```
Entrez un entier entre 1 et 100 :
Entrez 9 nombres réels :
 -4 -3 -2 -1 0 1 2 3 4

X    -4.0  -3.0  -2.0  -1.0  0.0  1.0  2.0  3.0  4.0
F(X) -55.0 -20.0 -3.0  2.0   1.0  0.0  5.0  22.0 57.0
```

```
main()
{
 float X[100];                  /* valeurs de X */
 float V[100];                  /* valeurs de F(X) */
 int N;
 ACQUERIR(&N);                  /* 1 <= N <= 100 */
 LIRE_VECTEUR(X, N);
 CALCULER_VALEURS(X, V, N);
 AFFICHER_TABLE(X, V, N);
 return 0;
}
```

Pour que la machine puisse exécuter ce programme, il faut encore implémenter les modules **ACQUERIR**, **LIRE_VECTEUR**, **CALCULER_VALEURS** et **AFFICHER_TABLE**. Ces spécifications se font en C sous forme de *fonctions* qui remplacent les fonctions *et* les procédures que nous connaissons en langage algorithmique et en Pascal. Une 'procédure' est réalisée en C par une fonction qui fournit le résultat **void** (vide). Les fonctions sont ajoutées dans le texte du programme au-dessus ou en-dessous de la fonction **main**.

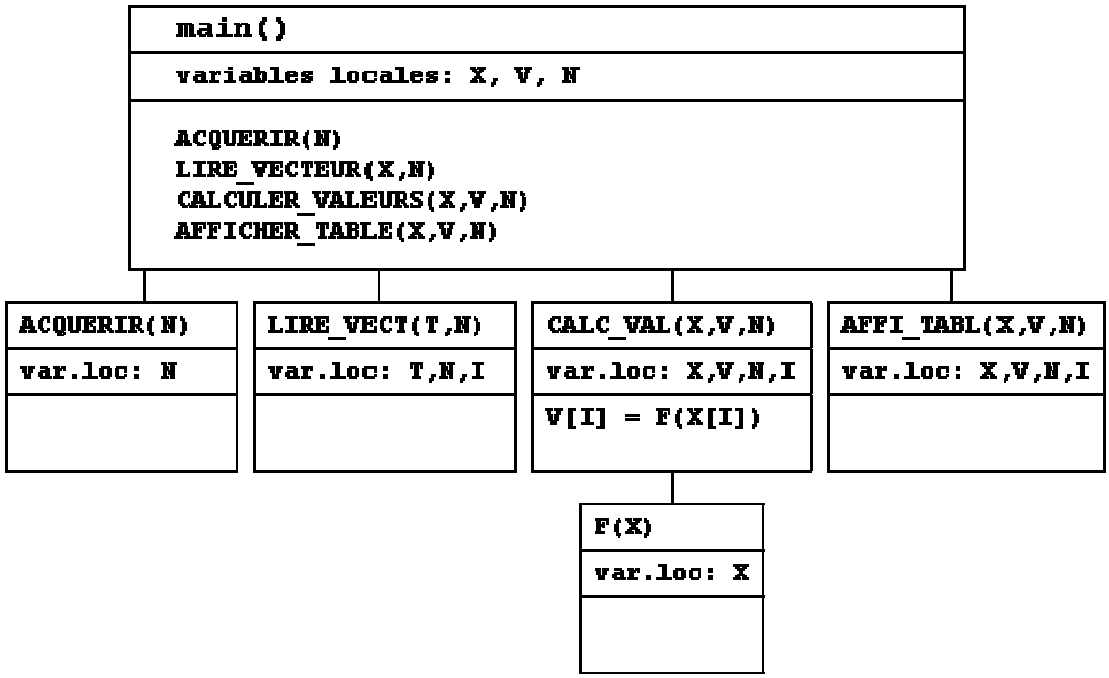
```
#include <stdio.h>

main()
{
 /* Prototypes des fonctions appelées par main */
 void ACQUERIR(int *N);
 void LIRE_VECTEUR(float T[], int N);
 void CALCULER_VALEURS(float X[], float V[], int N);
 void AFFICHER_TABLE(float X[], float V[], int N);
 /* Déclaration des variables locales de main */
 float X[100];              /* valeurs de X    */
 float V[100];              /* valeurs de F(X) */
 int N;

 /* Traitements */
 ACQUERIR(&N);                          /* 1 <= N <= 100   */
 LIRE_VECTEUR(X, N);
 CALCULER_VALEURS(X, V, N);
 AFFICHER_TABLE(X, V, N);
 return 0;
}
```

```
void ACQUERIR(int *N)
{
 do
   {
    printf("Entrez un entier entre 1 et 100 : ");
    scanf("%d", N);
   }
 while (*N<1 || *N>100);
}

void LIRE_VECTEUR(float T[], int N)
{
 /* Remplit un tableau T d'ordre N avec des nombres
    réels entrés au clavier */
 /* Déclaration des variables locales */
 int I;
 /* Remplir le tableau */
 printf("Entrez %d nombres réels :\n", N);
 for (I=0; I<N; I++)
     scanf("%f", &T[I]);
}

void CALCULER_VALEURS(float X[], float V[], int N)
{
 /* Remplit le tableau V avec les valeurs de */
 /* F(X[I]) pour les N premières composantes */
 /* X[I] du tableau X */
 /* Prototype de la fonction F */
 float F(float X);
 /* Déclaration des variables locales */
 int I;
 /* Calculer les N valeurs */
 for (I=0; I<N; I++)
     V[I] = F(X[I]);
}

float F(float X)
{
 /* Retourne la valeur numérique du polynôme défini
    par F(X) = X^3-2X+1 */
 return (X*X*X - 2*X + 1);
}
```

```
void AFFICHER_TABLE(float X[], float V[], int N)
{
 /* Affiche une table de N valeurs :
    X contient les valeurs données et
    V contient les valeurs calculées. */
 /* Déclaration des variables locales */
 int I;
 /* Afficher le tableau */
 printf("\n X   : ");
 for (I=0; I<N; I++)
    printf("%.1f", X[I]);
 printf("\n F(X): ");
 for (I=0; I<N; I++)
    printf("%.1f", V[I]);
 printf("\n");
}
```

Le programme est composé de six fonctions dont quatre ne fournissent pas de résultat. La fonction F retourne la valeur de F(X) comme résultat. Le résultat de F est donc du type **float**; nous disons alors que *'F est du type **float**'* ou *'F a le type **float**'*.

Les fonctions fournissent leurs résultats à l'aide de la commande **return**. La valeur rendue à l'aide de **return** doit correspondre au type de la fonction, sinon elle est automatiquement convertie dans ce type.



8.2. La notion de blocs et la portée des identificateurs

Les fonctions en C sont définies à l'aide de blocs d'instructions.

Blocs d'instructions en C

```
{
 <déclarations locales>
 <instructions>
}
```

Par opposition à d'autres langages de programmation, ceci est vrai pour ***tous*** les blocs d'instructions, non seulement pour les blocs qui renferment une fonction. Ainsi, le bloc d'instructions d'une commande **if**, **while** ou **for**

Exemple

La variable d'aide I est déclarée à l'intérieur d'un bloc conditionnel. Si la condition (N>0) n'est pas remplie, I n'est pas défini. A la fin du bloc conditionnel, I disparaît.

```
if (N>0)
  {
   int I;
   for (I=0; I<N; I++)
     ...
  }
```

8.2.1. Variables locales

Les variables déclarées dans un bloc d'instructions sont *uniquement visibles à l'intérieur de ce bloc*. On dit que ce sont des ***variables locales*** à ce bloc.

Exemple

La variable NOM est définie localement dans le bloc extérieur de la fonction HELLO. A

+insi, aucune autre fonction n'a accès à la variable NOM:

```
void HELLO(void);
{
 char NOM[20];
```

```
 printf("Introduisez votre nom : ");
 gets(NOM);
 printf("Bonjour %s !\n", NOM);
}
```

Exemple

La déclaration de la variable I se trouve à l'intérieur d'un bloc d'instructions conditionnel. Elle n'est pas visible à l'extérieur de ce bloc, ni même dans la fonction qui l'entoure.

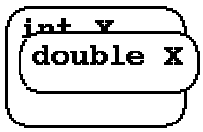
```
if (N>0)
  {
   int I;
   for (I=0; I<N; I++)
     ...
  }
```

Attention !

Une variable déclarée à l'intérieur d'un bloc ***cache*** toutes les variables du même nom des blocs qui l'entourent.

Exemple

Dans la fonction suivante,




```
int FONCTION(int A);
{
 int X;
 ...
 X = 100;
 ...
 while (A>10)
  {
   double X;
   ...
   X *= A;
     }
}
```

8.2.2. Variables globales

Les variables déclarées au début du fichier, à l'extérieur de toutes les fonctions *sont disponibles à toutes les fonctions du programme*. Ce sont alors des ***variables globales***. En général, les variables globales sont déclarées immédiatement derrière les instructions **#include** au début du programme.

Attention !

Les variables déclarées au début de la fonction principale **main**
ne sont pas *des variables globales, mais elles sont locales à* **main** ***!***

Exemple

La variable STATUS est déclarée globalement pour pouvoir être utilisée dans les procédures A et B.

```
#include <stdio.h>
int STATUS;

void A(...)
{
 ...
 if (STATUS>0)
     STATUS--;
 else
      ...
 ...
}

void B(...)
{
 ...
 STATUS++;
 ...
}
```