Chapitre 1

Les sous-programmes : Fonctions et Procédures

Les procédures et les fonctions (le terme routine rassemble les deux notions) sont des notions qui permettent de structurer les programmes. Ces notions sont présentes dans la plupart des langages de programmation. La déclaration d'une fonction permet d'associer un nom à une suite d'instructions, et d'utiliser ce nom comme abréviation chaque fois que la suite apparaît dans le programme. Cette utilisation du nom à la place de la suite d'énoncés s'appelle un appel de fonction.

Le premier intérêt évident des routines est donc de réduire la taille du code des programmes écrits, puisque les appels de procédure ou de fonction éviteront la duplication de plusieurs dizaines de lignes de code. Mais bien plus qu'une façon d'abréger le texte du programme, les procédures et les fonctions permettent :

- **La structuration :** les fonctions définissent des composants opérationnels fermés et cohérents qui représentent des parties ou sous-parties du problème à résoudre et qui structurent ainsi logiquement le programme.
- **La localisation :** les fonctions sont des unités textuelles dans lesquelles sont définies, non seulement les actions à exécuter, mais aussi des données ou objets locaux dont l'existence est limitée à celle de la procédure ou de la fonction dans laquelle ils sont définis.
- **Le paramétrage :** grâce à la notion de paramètre, il sera possible d'appeler une routine pour exécuter une même suite d'énoncés sur des valeurs de données différentes.

1.1 Déclaration

1.1.1 Déclaration d'une procédure

Le rôle de la déclaration d'une procédure (ou d'une fonction) est d'associer un nom à une suite d'actions (instructions) qui porte sur des objets formels (variables) qu'on qualifie de paramètres. Ces paramètres prendront des valeurs effectives lors de l' appel de la procédure (lors de la déclaration de la procédure on utilise des variables dont les valeurs ne sont pas connus jusqu'à ce qu'on fasse l'appel). La déclaration est constituée de deux parties : l'en-tête et le corps.

- **L'en-tête indique :**
  - le nom de la routine ;
  - les noms des paramètres formels et leurs types : pour chaque paramètre il faut donner le mode passage de paramètre (cette partie sera traitée en détail dans la section 1.3)
- **Le corps contient :**
  - des objets locaux (déclaration de paramètres connus et utilisés uniquement dans la procédure),
  - une suite d'actions ou d'instructions,

Dans l'entête de la routine (procédure ou de la fonction), chaque paramètre est associé à un mode de passage d'un paramètre, ce dernier permet de déterminer la manière avec laquelle la valeur du paramètre va être modifiée ou non pendant l'exécution de la fonction (cette partie sera traitée en détail dans la section 1.3)

PROCEDURE $Nom\_de\_la\_procedure$ (
$< mode\_de\_passage > / < parametre1 >:< type >$,
$< mode\_de\_passage > / < parametre2 >:< type >$,
$\ldots$,
$< mode\_de\_passage > / < parametreN >:< type >$)

Declaration des variables locales
Début
Instruction 1
Instruction 2
...
Instruction N
Fin.

Exemple 1 *On définit une fonction factorielle qui calcule la factorielle d'un entier naturel N :*

PROCEDURE *Factorielle* (
E/ N : *Entier*
)

VAR : $i, Fact$: *Entier* ;
Début
$Fact \leftarrow 1$;
Pour i **de** 1 **à** N
$Fact \leftarrow Fact * i$;
fpour;
Ecrire("Le factorielle =",Fact) ;
Fin.

En C :

```c
void Factorielle (int N, int &fact)
{
      int i;
      fact=1;
      for (i=2;i<N;i++)
      fact=fact*i;
      printf("Le factoriel =%d",fact);
}
```

1.1.2 Déclaration d'une fonction

Une fonction est une procédure qui renvoie un seul résultat. Comme pour la procédure, la déclaration est composée de deux parties, l'en-tête et le corps.

- **L'en-tête indique :**
  - le nom de la fonction ;
  - les noms des paramètres formels et leurs types : pour chaque paramètre il faut donner le mode passage de paramètre (cette partie sera traitée en détail dans la section 1.3)
  - le type du résultat
- **Le corps contient :**
  - des objets locaux (déclaration de paramètres connus et utilisés uniquement dans la fonction),

– une suite d'actions ou d'instructions,

FONCTION $Nom\_de\_la\_fonction$ (
$< parametre1 >:< type >$,
$< parametre2 >:< type >$,
$< parametreN >:< type >$):$< type >$

Declaration des variables locales
Début
Instruction 1
Instruction 2
...
Instruction N
Renvoyer X
Fin.

En C, les fonctions et les procédures sont désignées de la même façon. La seule différence, c'est que l'en-tête d'une fonction commence par le type du résultat de la fonction, suivi de son nom et terminé par les paramètres formels entre parenthèses.

Par contre, lorsqu'il s'agit d'une procédure (donc pas de valeur de retour), l'entête de la fonction commence par le mot clé **void** qui signifie rien à renvoyer.

La syntaxe fonctions en C est présentée comme suit :

```
1
2  <type> nom_fonction (
3  <type> <mode\_de\_passage> <parametre1>   ,
4  <type> <mode\_de\_passage> <parametre2>   ,
5  ...
6  <type> <mode\_de\_passage> <parametreN>   )
7  {
8   //instructions
9         Instruction 1;
10        Instruction 2;
11        ...
12        Instruction N;
13 }
```

Pour une procédure :

```
1
2  void nom_procedure (parametres){
3  //instructions
4         Instruction 1;
```

```
5         Instruction 2;
6         ...
7         Instruction N;
8   }
```

Exemple 1 (suite) On définit une fonction factorielle qui calcule la factorielle d'un entier naturel N :

FONCTION *Factorielle(E/ N : Entier) : Entier*

VAR : $i, Fact$: *Entier* ;
Début
$Fact \leftarrow 1$;
Pour i **de** 2 **à** N
$Fact \leftarrow Fact * i$;
fpour;
Renvoyer Fact ;
Fin.

En C :

```
1   int Factorielle (int N)
2   {
3         int i,fact=1;
4         for (i=2;i<N;i++)
5         fact=fact*i;
6         return fact;
7   }
```

1.1.3 Appel d'une routine (fonction ou procédure)

L'appel d'une routine (fonction ou procédure), est une action élémentaire qui consiste à nommer la routine avec des paramètres effectifs (les valeurs des paramètre dans la routine ne sont connus que à ce moment). L'appel provoque l'exécution des actions (les instructions) qui composent le corps de la routine.

Exemple 1 (suite) L'appel de la routine *Factorielle* se fait comme suit :

Algorithme Appel_Factorielle
VAR : $i, fact$: *Entier* ;
Début
Factorielle(10, *fact*); //appel 1 de la procédure
$Fact \leftarrow$ *Factorielle*(10);//appel 2 de la fonction **Fin.**

En C :

```c
main()
{
int fact;
factorielle(10,&fact); //appel 1 de la proc dure
fact=factorille(10); //appel 2 de  la fonction
printf("Le fact de 10=%d",fact);
}
```

```
$> Le fact de 10=3628800
```

1.1.4 Prototype d'une routine

Comme tout objet (variables) en C, une routine doit être déclarée avant son utilisation. Cette déclaration est nommée aussi le prototype de la fonction. Le prototype doit indiquer au compilateur le nom de la fonction, le type de la valeur de retour et le type des paramètres. Un prototype est une signature de la routine, ça consiste à donner l'entête de la fonction suivi par " ;".

Exemple 2 *Considérons la fonction factorielle, voilà son prototype :*

```c
int Factorielle ( int N);
```

Remarque 1 *Quand l'utilisation (l'appel) de la routine se fait après la définition (le corps de la routine) de cette dernière ; le prototype n'est pas obligatoire.*

Les noms de paramètre sont optionnels, mais il est fortement conseillé de les laisser. Cela donne une bonne indication sur leurs rôles.

Remarque 2 *C'est remarques sont valable qu'en C :*

- *Les fichiers .h en C par convention contiennent les prototypes des fonctions, exemple les prototypes de la bibliothèque stdio se trouvent dans le fichier stdio.h qu'on inclus dans nos programme C pour que les fonctions de cette bibliothèque soient reconnus.*
- *Si la fonctionne est placée avant sont appel pas besoin de faire de prototype, puisque le compilateur la déjà vu.*

1.2 Les variables locales et les variables globales

Jusque là, nous avions vu comment il est possible d'échanger des informations (des variables) entre différentes fonction grâce au passage de paramètres et la récupération d'une valeur en retour. Mais il est possible aussi, que plusieurs fonctions (ainsi que le programme principal) partagent des variables communes qu'on qualifie alors de "variables globales" (externes).

- **Les variables globales** En programmation, une variable globale est partagée par plusieurs fonctions. Elle est déclaré en dors de toute fonction.
- **Les variables locales** Une variable est dite locale à une fonction (ou une procédure) dans laquelle elle est déclarée si elle est définie au sein de cette dernière. Les variables locales ne sont connues qu'à l'intérieur de la fonction (ou de la procédure) où elles sont déclarées (en dehors de la fonction la variable n'est pas reconnue).

Remarque 3 *Il faut noter que :*

- *Les variables locales ont une "durée de vie" limitée à celle d'une exécution du bloc dans laquelle elles figurent. Un nouvel espace mémoire leur est alloué à chaque entrée dans ce bloc et libéré à chaque sortie.*
- *Les valeurs transmises en arguments à une fonction sont traitées de la même manière que les variables locales. Leur durée de vie correspond également à celle de la fonction.*
- *Quand une variable globale est définit avec un identifiant donné, aucune autre variable que ça soit locale ou globale ne peut être définit avec le même identifiant.*

Exemple 1 (suite) Cette fois on va utiliser une variable globale pour récupérer la valeur résultante :

```
1 #include stdio.h
2 int fact=1; //c'est une variable globale
3 void Factorielle (int N)
```

```
4  {
5        int i;
6        for (i=2;i<N;i++)
7        fact=fact*i;
8        return fact;
9  }
10 main()
11 {
12 Factorielle(10);
13 printf("Le fact de 10=%d",fact);
14 }
```

```
$> Le fact de 10=3628800
```

1.3 Le passage des paramètres

Un paramètre formel est un nom (une variable) avec lequel une routine va reconnaitre et manipuler une donnée lors de sa déclaration.

Un paramètre effectif est l'entité fournie au moment de l'appel de la routine, sous la forme d'un nom, d'une expression ou d'une valeur constante. Nous distinguerons deux types de paramètres : les données et les résultats.

- Les paramètres données fournissent les valeurs utilisées par routine pour effectuer le calcul.
- Les paramètres résultats sont les valeurs de sorties calculées par la routine.

Le passage de paramètre donc c'est le remplacement des paramètres formels par les paramètres effectifs lors de l'appel de la routine. Ce passage se fait selon des règles strictes de passage des paramètres.

En algorithmique, ça peut être :

- **E/ :** Si le paramètre est une entrée c.à.d il fourni les valeurs à partir les quelles les énoncés du corps de la routine effectueront leur calcul
- **S/ :** Si le paramètre est une sortie c.à.d le paramètre rend les valeurs calculées par la procédure.
- **ES/ :** Si le paramètre est à la fois une entrée et une sortie
- **Ref/ :** Si c'est un paramètre référence, dans ce cas on va utiliser l'adresse de l'argument, les modifications effectuées à l'intérieur de la fonction sont également répercutées en dehors de la fonction

Et en C, ça peut être :

- **Rien devant le paramètre :** Si le paramètre est une entrée
- *** :** Si le paramètre est une sortie ou entrée et une sortie

- & : Si c'est un paramètre référence

Remarque 4

- *En C, les vecteurs sont des cas exceptionnels pour le passage de paramètres. Un tableau à une démission se transmis soit par adresse soit avec des crochets vides*
- *Le passage d'un tableau comme paramètre d'une fonction est impossible en tant que valeur : la recopie du tableau prendrait trop de temps et de place. On passe donc à la fonction l'adresse du tableau, ce qui permettra à la fonction d'effectuer des lectures et des écritures* ***DIRECTEMENT DANS LE TABLEAU****.*

Exemple 3

```
1 // la fonctionne1 prend un tableau d'entier en param tre
2 int fonction1 ( int * tab);
3 // la fonctionne2 prend aussi un tableau d'entier en
    param tre
4 int fonction2 ( int tab[]);
5 //Exemple d'apple
6 int tab[10];
7 fonction1(tab);
8 fonction2(tab);
```

Un tableau à 2 démissions se transmis soit par adresse (double étoile) soit avec 2 crochets le premier contiennent une valeur et le deuxième est vide.

Exemple 4

```
1 // la fonctionne1 prend une matrice d'entier en param tre
2 int fonction1 ( int ** Mat);
3 // la fonctionne2 prend aussi une matrice d'entier en
    param tre
4 int fonction2 ( int Mat[100][]);
5
6  //Exemple d'apple
7 int mat[10][4];
8 fonction1(mat);
9 fonction2(mat);
```

1.4 La récursivité

La récursivité est une démarche qui consiste à définir quelque chose en fonction d'elle-même. En mathématiques, les définitions récursives sont très courantes. Par exemple la fonction factorielle se définit comme :

$$n! = n * (n-1)!$$

La précédente définition de factorielle est infinies, en ce sens qu'elles ne s'achèvent pas. Pour être calculables, c'est-à-dire qu'elle puisse être exécutées sur un ordinateur, les définitions récursives ont besoin d'une **condition d'arrêt**. Si on reprend l'exemple de factorielle, $0! = 1$ qui représente notre condition d'arrêt.

Exemple 1 (suite) Reprenant la fonction factorielle :

FONCTION *Factorielle* (
E/ N : *Entier* ,
):*Entier*

Début
Si$(N = 0)$ **Alors**
Renvoyer 1 ;
FinSi
$Fact \leftarrow N * Factorielle(N - 1)$;
Renvoyer *Fact* ;
Fin.

En C :

```
1
2  int Factorielle (int N)
3  {
4        //la condition d'arret
5        if(N==0){
6        printf("1\n");
7        return 1;
8        }
9        //calcul du factorielle du N en cour
10       printf("%d*",N);
11       return N*Factorielle(N-1);
12 }
13 main()
14 {
15 int fact=Factorielle(10);
16 printf("Le fact de 10=%d",fact);
17 }
```

```
$> 10*9*8*7*6*5*4*3*2*1*1
$> Le fact de 10=3628800
```

Chapitre 2

Les chaînes de caractères

Les tableaux à une dimension sont aussi utilisés pour représenter les chaînes de caractères. Une *chaîne de caractères* est une suite de caractères successifs formant un texte qui se terminant par le caractère spécial '\0' (qui a 0 pour code ASCII).

a	b	c	d	e	\0

Le caractère '\0' sert à repérer la fin de la chaîne, ce qui nous permis d'éviter d'avoir à chaque fois avec la chaîne nombre de caractères. On peut ainsi passer une chaîne de caractères en paramètre à une fonction, sans avoir besoin de passer un deuxième paramètre contenant le nombre de catactères à chaque fois.

2.1 déclaration d'une chaîne de caractères

Déclaration de chaînes de caractères en langage algorithmique se fait comme suit :

Syntaxe

<NomChaine> [<tailleMaxChaine>] : chaine ;

En C :

```
1 char <NomChaine> [<tailleMaxChaine>];
```

Remarque 5

- En C, les chaînes de caractères sont représentées sous forme de tableaux de type **char[]**. Les chaînes de caractères se terminent par un caractère spécial noté ' \0 '. Ce caractère ne s'imprime pas. Son rôle est simplement d'indiquer la fin de la chaîne. Il permet donc d'augmenter ou de diminuer la taille de la chaîne sans modifier la taille du tableau.
- Il est important, lors de la déclaration de la chaîne, de déclarer une taille de tableau qui est supérieure à la taille du texte qu'on souhaite introduire.Sinon, une partie de la chaîne sera perdue.

2.2 Initialisation d'une chaîne de caractères

L'initialisation d'une chaîne de caractère peut être effectuée en utilisant la syntaxe classique des tableaux (en spécifiant que la dernière contient le caractère de fin de chaîne) ou bien en utilisant des guillemets. Dans l'exemple suivant le contenue des deux chaînes de caractère texte1 et texte2 est exactement le même, le mot Bonjour.

```
#include<stdio.h>
 main (){
 char texte1[]= {'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
 char texte2[8] = " Bonjour " ;
}
```

Pour stocker le mot "Bonjour"(qui comprend 7 lettres) en mémoire, il ne faut pas un tableau de 7 char, mais de 8 ! Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne '\0'.

2.3 Lecture et écriture d'une chaîne de caractères

La lecture de la chaîne de caractère s'arrête dès qu'elle rencontre un caractère de séparation : retour à la ligne, espace, tabulation, ...
On ne met pas de '&' dans scanf pour lire une chaîne de caractères. Ceci est dû au fait qu'une chaîne de caractères est un tableau ce qu'est déjà une adresse (une variable de type tableau contient l'adresse de la 1er case du tableau).

En C, écrire une chaîne de caractères se fait comme suit :

```
#include<stdio.h>
 main (){
 char texte[100];
```

```
4  printf("Donner une chaine de caract re\n");
5  scanf("%s",texte);
6  printf("Votre chaine est : %s \n", texte) ;
7 }
```

```
$> Donner une chaine de caract re
$> Bonjour
$> Votre chaine est : Bonjour
```

2.4 Opérations sur les chaînes de caractères (en C)

Les chaînes de caractères peuvent êtres manipulées comme des tableaux à une dimension, dans ce cas nous pouvons utiliser nos connaissances sur la manipulation des tableaux (vus précédemment). Un autre moyen est d'utiliser des fonctions définies dans la bibliothèque *string.h*. Cette dernière fournie un ensemble de fonctions qui vont nous aider à manipuler les chaînes. Elles permettent par exemple de comparer deux chaînes de caractères, les fusionner, connaître la longueur d'une chaîne ou la copier.

Ici, nous n'allons pas lister toutes ces fonctions, nous allons vous en présenter les principales dont vous aurez très certainement besoin dans vos programmes.

2.4.1 Comparaison de deux chaînes

En C, pour comparer deux variables C1 et C2 de type chaine, il suffit d'utiliser la fonctions strcomp ("string compare").

Elle est définie comme suit :

```
1 int strcmp(const char[] chaine1, const char[] chaine2);
```

La fonction compare les deux variable chaine1 et chaines 2 et revoie un entier. Il est important de récupérer ce que la fonction renvoie. En effet,strcmp renvoie :

- 0 si les chaînes sont identiques
- une autre valeur (positive ou négative) si les chaînes sont différentes.

```
1 #include<stdio.h>
2 #include<string.h>
3 int main(int argc, char *argv[])
4 {
5     char chaine1[] = "Texte de test", chaine2[] = "Texte de test"
    ;
```

```
6
7       if (strcmp(chaine1, chaine2) == 0) // Si chaines identiques
8               printf("Les chaines de caracteres sont identiques en
        utilisant strcmp\n");
9
10      else
11              printf("Les chaines de caracteres sont differentes en
        utilisant strcmp \n");
12
13      if(chaine1==chaine2)
14       printf("Les chaines de caracteres sont identiques en
        utilisant ==\n");
15       else
16        printf("Les chaines de caracteres sont differentes en
        utilisant == \n");
17      return 0;
18  }
```

```
$> Les chaines de caract res sont identiques en
 utilisant strcmp
$> Les chaines de caract res sont differentes en
 utilisant ==
```

2.4.2 Longueur d'une chaîne

Pour connaître la longueur d'une chaîne il suffit d'utiliser la fonction strlen défini comme suit :

```
1 int strlen(const char[] chaine);
```

Cette fonction renvoie la longueur de la variable chaîne.

```
1 #include<stdio.h>
2 #include<string.h>
3  main (){
4 int longueur ;
5 char texte[ ] = " Bonjour " ;
6 longueur = strlen(texte) ;
7 printf("La longueur = %d",longueur);
8 }
```

```
$ > La longueur = 7
```

2.4.3 Copier une chaîne

Pour copier une chaîne de caractère dans une autre, il faut utiliser la fonction strcpy (« string copy »).

```
char[] strcpy(char[] copie_Chaine, const char[] chaine_ACopier);
```

Exemple 5 *Exemple d'utilisation la fonction strcpy*

```
#include<stdio.h>
#include<string.h>
int main(int argc, char *argv[])
{
    char chaine[] = "Texte", copie[100] = {0};
     strcpy(copie, chaine); // On copie "chaine" dans "copie"
     printf("chaine vaut : %s\n", chaine);
     printf("copie vaut : %s\n", copie);

     return 0;
}
```

```
$ > chaine vaut : Texte
    copie vaut : Texte
```

2.4.4 Concaténer deux chaînes

On appelle la concaténation le fait d'ajouter une chaîne à la suite d'une autre. Supposons que l'on ait les variables suivantes :

- chaine1 = "Salut "
- chaine2 = "Mateo21"

Si je concatène chaine2 dans chaine1, alors chaine1 vaudra "Salut Mateo21". En C, pour faire cette action on utilise la fonction strcat définie comme suit :

```
char[] strcat(char[] chaine1, const char[] chaine2);
```

La fonction ajoute à chaine1 le contenu de chaine2.

```
#include<stdio.h>
#include<string.h>
int main(int argc, char *argv[])
{

    char chaine1[100] = "Salut ", chaine2[] = "Mateo21";

    strcat(chaine1, chaine2); // On concatene chaine2 dans
   chaine1

    // Si tout s'est bien passe, chaine1 vaut "Salut Mateo21"
   printf("chaine1 vaut : %s\n", chaine1);
    // chaine2 n'a pas change :
    printf("chaine2 vaut toujours : %s\n", chaine2);
```

```
14
15    return 0;
16 }
```

```
$> chaine1 vaut : Salut Mateo21
$> chaine2 vaut toujours : Mateo21
```

2.4.5 Rechercher un caractère

La fonction strchr recherche un caractère dans une chaîne.

```
1 char[] strchr(const char* chaine, char caractereARechercher);
```

La fonction prend 2 valeurs :

- chaine : la chaîne dans laquelle la recherche doit être faite ;
- caractereARechercher : le caractère que l'on doit rechercher dans la chaîne.

La fonction renvoie une chaine où son contenu commence par le premier caractère qu'elle a trouvé. Elle renvoie NULL si elle n'a rien trouvé.

```
1  #include<stdio.h>
2  #include<string.h>
3  int main(int argc, char *argv[])
4  {
5      char chaine[] = "Texte de test", *suiteChaine = NULL;
6
7     suiteChaine = strchr(chaine, 'd');
8      if (suiteChaine != NULL) // Si on a trouve quelque chose
9     {
10        printf("Voici la fin de la chaine a partir du premier d :
      %s", suiteChaine);
11    }
12
13    return 0;
14 }
```

```
$ > Voici la fin de la chaine a partir du premier d :
  de test
```

2.4.6 Extraire les mots d'une chaine de caractère

```
1 char [] strtok( char [] string, const char [] separateurs );
```

Cette fonction permet d'extraire, un à un, tous les éléments syntaxiques (les tokens) d'une chaîne de caractères. Pour contrôler ce qui doit être extrait, un ensemble des caractères de séparateurs de tokens doit être spécifié.

```c
#include<string.h>
main()
{

  char texte[] = "voila une phrase qu'on va d composer en mots";
    // La d finitions de s parateurs connus.
    const char * separators = " ";

    // On cherche    r cup rer , un    un, tous les mots (token)
    de la phrase
    // et on commence par le premier.
    char * mot = strtok ( texte, separators );
    while ( mot != NULL ) {
        printf ( "%s\n", mot );
        // On demande le token suivant.
        mot = strtok ( NULL, separators );
    }
    }
```

```
$> voila
$> une
$> phrase
$> qu'on
$> va
$> d composer
$> en
$> mots
```