FONCTIONS, PROCÉDURES

FONCTIONS & PROCÉDURES

DÉFINITIONS

Lorsque l'Algorithme à écrire devient de plus en plus important (volumineux), des difficultés d'aperçu global sur son fonctionnement se posent. Il devient donc très difficile de coder et de devoir traquer les erreurs en même temps.

- Il devient utile de découper le problème en de sous problème,
- de chercher à résoudre les sous problèmes (sous-algorithmes),
- puis de faire un regroupement de ces sous sous-algorithmes pour reconstituer une solution au problème initial.

Un sous-algorithme est une partie d'un algorithme. Il est d'habitude déclaré dans la partie entête et est réutiliser dans le corps de l'algorithme.

- Un sous-algorithme est un algorithme. Il possède donc les même caractéristiques d'un algorithme.

FONCTIONS & PROCÉDURES

DÉFINITIONS

- Un sous-algorithme peut utiliser les variables déclarés dans l'algorithme. Dans ce cas, ces variables sont dites globales. Il peut également utiliser ses propres . Dans ce cas; les variables sont dites locales. Ces dernières ne pourront alors être utilisable qu'à l'intérieur du sous-programme et nulle part ailleurs (notion de visibilité). Ce qui signifie que leur allocation en mémoire sera libérer à la fin de l'exécution du sous-programme.
- Un sous-programme peut être utilisable plusieurs fois avec éventuellement des paramètres différents.
- **Un sous-algorithme peut se présenter sous forme de fonction ou de procédure:**
  - Une fonction est un sous-algorithme qui, à partir de donnée(s), calcul et rend à l'algorithme Un et Un seul résultat;
  - alors qu'en général, une procédure affiche le(s) résultat(s) demandé(s).

FONCTIONS

- Syntaxe d'une fonction:

```
Fonction Nom_Fonction (Nom_Paramètre:Type_praramètre;...): type_Fonction ;
Variable
        Nom_variable : Type_variable ;      } Variables locales
        ...
Début
        ...
        Instructions ;                      }
        ...                                 } Corps de la fonction
        Nom_Fonction←resultat;              }
Fin ;
```

Un appel de fonction est une expression d'affectation de manière à ce que le résultat soit récupéré dans une variable globale de même type: Nom_variable-globale ← Nom_Fonction (paramètres) ;

FONCTIONS
EXEMPLE D'APPLICATION

```
Algorithme Calcul_des_n_premiers_nombres_entiers;
Variable
        I, Som, N : entier;
Fonction Somme: entier ;
Variable
        S : entier ;
Debut /*Début de la fonction*/
        S ←0 ;
        Pour I ← 1 a N Faire
                        S ←S + I;
        FinPour ;
        Somme ← S
Fin /*Fin de la Fonction */
Debut /*Début de l'algorithme*/
        Som ← Somme ;
        Ecrire ('La somme des ', N, 'premiers nombres est', Som) ;
Fin. /*Fin de l'algorithme*/
```

PROCÉDURES

- Syntaxe d'une procédure:

```
Fonction Nom_Procedure (Nom_Paramètre:Type_praramètre;...) ;
Variable
        Nom_variable : Type_variable ;      } Variables locales
        ...
Début
        ...
        Instructions ;                      } Corps de la fonction ...

Fin ;
```

L'appel d'une procédure peut être effectué en spécifiant, au moment souhaité, son nom et éventuellement ses paramètres; cela déclenche l'exécution des instructions de la procédure.

PROCÉDURES
EXEMPLE D'APPLICATION

```
Algorithme Calcul_des_n_premiers_nombres_entiers;
Variable
        I, Som, N : entier;
Procedure Somme ;
Debut /*Début de la procédure*/
        Som ←0 ;
        Pour I ← 1 a N Faire
                Som ←Som + I;
        FinPour ;
        Ecrire ('La somme des ', N, 'premiers nombres est', Som) ;

Fin /*Fin de la Fonction */
Debut /*Début de l'algorithme*/
        Somme ;
Fin. /*Fin de l'algorithme*/
```

MODE DE PASSAGES DE PARAMÈTRES

PASSAGE PAR VALEUR

On distingue deux types de passage de paramètres : par valeur et par variable (dite aussi par référence ou encore par adresse).

- Le mode de passage par valeur qui est le mode par défaut, consiste à copié la valeur des paramètres effectifs dans les variables locales issues des paramètres formels de ma fonction ou de la procédure appelée.
  - Dans ce mode, nous travaillons pas directement avec la variable, mis avec une copie. Ce qui veut dire que le contenu des paramètres effectifs n'est pas modifié. À la fin de l'exécution du sous-programme, la variable conservera sa valeur initial.
  - **Syntaxe**:
    - Procedure nom_procédure (param1:type1 ; param2, param3:type2) ;
    - Fonction nom_fonction (param1:type1 ; param2:type2):Type_fonction ;

PASSAGE PAR VALEUR
EXEMPLE D'APPLICATION

```
Algorithme valeur_absolue_d-un_nombre_entier;
Variable
        val: entier;
Procedure Abs(nombre: entier);
Debut /*Début de la procédure*/
        Si nombre<0 Alors
                nombre ← - nombre;
        FinSi ;
        Ecrire (nombres) ;
Fin /*Fin de la Fonction */
Debut /*Début de l'algorithme*/
        Lire (val);
        Abs (val);
        Ecrire (val);
Fin. /*Fin de l'algorithme*/
```

- Ici, val reprend sa valeur initiale. Il a juste servi de données pour Abs.

MODE DE PASSAGES DE PARAMÈTRES

PASSAGE PAR ADRESSE

- Dans le mode de passage par variable , il s'agit pas simplement d'utiliser la valeur de la variable, mais également son emplacement mémoire.
  - Le paramètre formel se substitue au paramètre effectif tout au long de l'exécution du sous-programme et à la sortie il lui transmet sa nouvelle valeur.
  - Un tel passage se fait par l'utilisation du mot-clé **Var**.
  - **Syntaxe**:
    - Procedure nom_procédure (Var param1:type1 ; param2, param3:type2) ;
    - Fonction nom_fonction (Var param1:type1 ; param2:type2):Type_fonction ;

PASSAGE PAR ADRESSE
EXEMPLE D'APPLICATION

```
Algorithme valeur_absolue_d-un_nombre_entier;
Variable
        val: entier;
Procedure Abs(Var nombre: entier);
Debut /*Début de la procédure*/
        Si nombre<0 Alors
                nombre ← - nombre;
        FinSi ;
        Ecrire (nombres) ;
Fin /*Fin de la Fonction */
Debut /*Début de l'algorithme*/
        Lire (val);
        Abs (val);
        Ecrire (val);
Fin. /*Fin de l'algorithme*/
```

- Ici, val prend une nouvelle valeur.

FONCTIONS EN C++

RAPPEL

- Une fonction est un bloc paramétré et nommé
- Permet de découper un programme en plusieurs modules.
- Dans certains langages, on trouve *deux sortes de modules*:
  - Les fonctions, assez proches de la notion mathématique
  - Les procédures (Pascal) ou sous-programmes (Fortran, Basic) qui élargissent la notion de fonction.
- En C/C++, il n'existe qu'une seule sorte de module, nommé fonction
  - **Syntaxe:**

```
typeDeRetour nomFonction([arguments]){
        //instructions
}
```

FONCTIONS EN C++

EXEMPLE

- Une fonction peut ne pas renvoyer de valeur. Dans ce cas, le type de la fonction est void. **Exemple**: void *abs*(int nombre){…}.
- Lorsqu'une fonction s'appelle elle-même, on dit qu'elle est « récursive ».

```
#include <iostream>
using namespace std;

int abs(int nombre)
{
        if (nombre<0)
            nombre=-nombre;
        return nombre; // Valeur renvoyée
}

int main()
{
         int val, valAbs;
         cout << "Entrez un nombre : ";
         cin >> val;
         valAbs = abs(val); // Appel de la fonction et affectation
         cout << "La valeur absolue de" << val << "est" << valAbs << endl;
         return 0;
}
```

FONCTIONS EN C++

PASSAGE PAR VALEUR

Supposons que l'on souhaite faire une permutation de deux entiers a et a. exemple:

```
#include <iostream>
using namespace std;

void permute(int a, int b)
{
        int tempon = a;
        a = b;
        b =tempon;
}

int main()
{
        int a=2, b=6;
        cout <<  "a: " << a << " b: " <<b <<endl; // avant
        permute(a, b);
        cout <<  "a: " << a << " b: " <<b <<endl; // après
        return 0;
}
```

Après exécution, on constate qu'on a pas le résultat attendu. Par défaut, le passage des arguments à une fonction se fait par valeur. Pour remédier à cela, il faut passer par adresse ou par référence.

FONCTIONS EN C++

PASSAGE PAR ADRESSES

Pour modifier le paramètre réel, on passe son adresse plutôt que sa valeur. exemple:

```
#include <iostream>
using namespace std;

void permute(int *a, int *b)
{
        int tempon = *a;
        *a = *b;
        *b =tempon;
}

int main()
{
         int a=2, b=6;
         cout <<  "a: " << a << " b: " <<b <<endl; // avant
         permute(&a, &b);
         cout <<  "a: " << a << " b: " <<b <<endl; // après
         return 0;
}
```

FONCTIONS EN C++

PASSAGE PAR RÉFÉRENCE

On peut également faire ceci:

```
#include <iostream>
using namespace std;

void permute(int& a, int& b)
{
        int tempon = a;
        a = b;
        b =tempon;
}

int main()
{
         int a=2, b=6;
         cout <<  "a: " << a << " b: " <<b <<endl; // avant
         permute(a, b);
         cout <<  "a: " << a << " b: " <<b <<endl; // après
         return 0;
}
```

Ici, le compilateur se charge de la gestion des adresses: le paramètre formel est un alias de l'emplacement mémoire du paramètre réel.

FONCTIONS EN C++

PASSAGE D'UN TABLEAU

On peut faire passer un tableau en paramètre. Nous avons dans ce cas, deux cas de figure: par pointeur ou par semi-référence.

Par pointeur:

```
#include <iostream>
using namespace std;

void affiche(int *tableau, int taille)
{
        for(int i=0; i<taille; i++)
            cout << tableau[i] << "   " << endl;
}

int main()
{
        int tab[5] = {1, 2, 3, 4, 5 };
        affiche(tab, 5);
        return 0;
}
```

FONCTIONS EN C++

PASSAGE D'UN TABLEAU

Par semi-référence:

```
#include <iostream>
using namespace std;

void affiche(int tableau[], int taille)
{
        for(int i=0; i<taille; i++)
            cout << tableau[i] << "   " << endl;
}

int main()
{
        int tab[5] = {1, 2, 3, 4, 5 };
        affiche(tab, 5);
        return 0;
}
```