Rappel sur les bases

- Les variables, les opérateurs et les opérations
- Structures de contrôles
- Tableaux et pointeurs
- **Fonctions et récursivité**
- Quelques algorithmes de tri et de recherche
- Fichiers
- Structures
- Listes chainées, Piles, Files

Rappel

Fonctions et récursivité

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

- Il est donc 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-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.

Rappel

Fonctions et récursivité

- 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 variables. 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).

Rappel

Fonctions et récursivité

- **Syntaxe d'une fonction (en Algorithme)**

```
Fonction Nom_Fonction (Nom_Paramètre:Type_paramètre;…): 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>) ;

Rappel

Fonctions et récursivité

- **Exemple de fonction (en Algorithme)**

```
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 à 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*/
```

Rappel

Fonctions et récursivité

- **Syntaxe d'une procédure (en Algorithme)**

```
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.

Rappel

Fonctions et récursivité

- **Exemple de procédure (en Algorithme)**

```
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*/
```

Rappel

Fonctions et récursivité

- **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, mais 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 ;

Rappel

Fonctions et récursivité

- **Mode de passages de paramètres:** 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.

Rappel

Fonctions et récursivité

- **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 ;

Rappel

Fonctions et récursivité

- **Mode de passages de paramètres:** 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.

Rappel

Fonctions et récursivité

Fonctions en c++

- 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;
}
```

Rappel

Fonctions et récursivité

- **Exemple**:

```
#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;
}
```

- L'instruction ***return*** permet à la fois de fournir une valeur de retour et à mettre fin à l'exécution de la fonction.
- Dans la déclaration d'une fonction, il est possible de prévoir pour un ou plusieurs arguments (obligatoirement les derniers de la liste) des ***valeurs par défaut*** ;
  - elles sont indiquées par le signe **=**, à la suite du type de l'argument.
    Exemple: float op(char, float=1.0, float=1.0);

- Une fonction peut ne pas renvoyer de valeur. Dans ce cas, le type de la fonction est void.
  - **Exemple**:

```
void abs(int nombre)
{
    if (nombre<0)
              nombre=-nombre;
}
```

- Lorsqu'une fonction s'appelle elle-même, on dit qu'elle est « ***récursive*** ».

Rappel

Fonctions et récursivité

Fonctions en c++: passage par valeur

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

- **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.

Rappel

Fonctions et récursivité

Fonctions en c++: passage par adresse

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;
}
```

Rappel

Fonctions et récursivité

Fonctions en c++: passage par référence

On peut également passer les paramètres par référence:

- **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;
}
```

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

Rappel

Fonctions et récursivité

Fonctions en c++: passage par référence

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

- **Exemple**: 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;

}
```

- **Exemple**: 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;

}
```

Rappel

Fonctions et **récursivité**

Récursivité: définitions

On appelle récursivité tout sous-programme qui s'appelle dans son traitement.

- Il est impératif de prévoir une condition d'arrêt puisque le sous-programme va s'appeler récursivement. sinon, il ne s'arrêtera jamais.
  - On teste la condition,
  - Si elle n'est pas vérifié, on lance à nouveau le sous-programme.

Rappel

Fonctions et **récursivité**

Exemples

Algorithme:

```
fonction factoriel(n : entier): entier
Début
   Si(n<2) alors
      retourner 1
   Sinon
      retourner n*factoriel(n-1)
   Fin si
Fin
```

C++:

```
int factoriel(int n)
{
        if(n<=1)
                return 1;
        else
                return(n*factoriel(n-1));
}
```

Rappel

Fonctions et **récursivité**

Il également possible qu'un sous-programme appel un second qui a son tour appel le premier. On dit que la récursivité est indirecte, cachée, croisée ou mutuelle.

Exemples

Algorithme:

```
fonction pair (n : entier) : booleén
Début
   Si(n=0) alors
      retourner VRAI
   Sinon Si(n=1) alors
      retourner FAUX
   Sinon
      retourner impair(n-1)
   Fin si
Fin
```

```
fonction impair (n : entier) : booleén
Début
   Si(n=1) alors
      retourner VRAI
   Sinon Si(n=0) alors
      retourner FAUX
   Sinon
      retourner pair(n-1)
   Fin si
Fin
```

Rappel

Fonctions et récursivité

EXERCICES D'APPLICATIONS

- Application 18 :

Ecrire un programme qui appel trois fonctions:

- Une fonction affiche « Toc toc ! » et qui ne possède ni argument, ni valeur de retour;
- Une deuxième qui affiche « entrée » un ou plusieurs fois (une valeur reçu en argument) et qui ne renvoie aucune valeur;
- Une troisième qui fera comme la première mais en un ou plusieurs fois (une valeur reçu en argument) et qui retourne cette fois ci la valeur de 0.

- Application 19 :

a) Ecrire un programme utilisant une fonction qui reçoit en argument 2 nombres flottants et un caractère (opération), et qui fournit le résultat du calcul demandé.

b) Proposer le même programme mais cette fois ci, la fonction ne disposera plus que de 2 arguments en nombres flottants . L'opération est précisé, cette fois, à l'aide d'une variable globale.

- Application 20 :

Ecrire un programme utilisant une fonction qui fournit en valeur de retour la somme des éléments d'un tableau d'entiers. Le tableau ainsi que sa dimension sont transmis en argument.

- Application 21 :

Ecrire un programme faisant appel à une fonction qui ne renvoie aucune valeur et qui détermine la valeur maximale et la valeur minimale d'un tableau d'entiers. proposer deux solutions: l'une utilisant effectivement cette notion de référence, l'autre la « simulant » à l'aide de pointeurs.