Algorithmique et Programmation 2

Licence 1 – MPI&MASS

Section d'Informatique

U.F.R. Sciences Appliquées et Technologie (SAT)

Université Gaston Berger de Saint-Louis

cheikh-tidiane.dieng@ugb.edu.sn

Sommaire

1. Procédures et Fonctions (4h)
2. Récursivité (5h)
3. Algorithmes de Tri (6h)
4. Enregistrements et structure de données avancées (Union, Champ de bits, Enumérations) (5h)
5. Fichiers (4h)

Durée totale: 24h

Chapitre 1:
Procédures et Fonctions

Procédures et Fonctions : ça sert à quoi ?

Jusqu'ici nous n'avons travaillé que dans un seul fichier appelé programme principal. Pour le moment, c'était acceptable car nos programmes étaient tout petits et très simples, mais ils vont bientôt être composés de milliers voire de **millions** de lignes.

Principe ?

Le principe est tout simple : plutôt que de placer tout le code de notre programme dans un seul fichier, nous le « séparons » en plusieurs sous-programmes ou fichiers.

Procédure

- Une procédure est un sous-programme.
- Ecrire des procédures permet de **découper** un programme en **plusieurs morceaux plus simples**.
- Chaque procédure définit une nouvelle instruction, que l'on peut appeler n'importe quel endroit du programme.
- On peut ainsi **réutiliser** le code d'un sous-programme.

Procédure

Lorsqu'on découpe un problème en terme de procédures, puis qu'on implémente ces procédures, on fait ce qu'on appelle une **analyse descendante** : on va du plus général au détail.

Procédure sans parametres

Il s'agit simplement de donner un nom a un groupe d'instructions. Ensuite, l'appel de ce nom à divers endroits du programme provoque à chaque fois l'exécution de ce groupe d'instructions.

Procédure sans paramètres

{ Déclaration de la procédure nom_procedure }

PROCEDURE nom_procedure()

VARIABLE {Definition des variables}

DEBUT

{ Corps de la procédure }

FIN;

Procedure sans parametres

{ Déclaration de la procédure Echange_xy }

PROCEDURE echange_xy()

DEBUT

{ Corps de la procedure }

t ← x; x ← y; y ← t;

FIN;

*//**Attention:** il ne faut pas oublier de déclarer x, y et t dans le programme principal (variables globales). Nous verrons plus tard comment déclarer des variables locales.*

Procedure sans parametres

```
ALGORITHME exemple1;
VARIABLE x, y, t : entier;
{ Déclaration de la procédure echange_xy }
PROCEDURE echange_xy()
DEBUT
        { Corps de la procedure }
        t ← x; x ← y; y ← t;
FIN;
DEBUT
        { Programme principal }
        x ← 3; y ← 4;
        ecrire (x, ‘  ', y);
        echange_xy() { 1er appel de la procedure }
        ecrire (x, ‘  ', y);
        echange_xy() { 2eme appel de la procedure }
        ecrire (x, ‘  ', y);
FIN.
```

Procedure sans parametres

```
ALGORITHME exemple1;
VARIABLE x, y, t : entier;
{ Declaration de la procedure echange_xy }
PROCEDURE echange_xy()
DEBUT
      { Corps de la procedure }
      t ← x; x ← y; y ← t;
FIN;
DEBUT
      { Programme principal }
      x ← 3; y ← 4;
      ecrire (x, ' ', y);
      echange_xy(); { 1er appel de la procedure }
      ecrire (x, ' ', y);
      echange_xy(); { 2eme appel de la procedure }
      ecrire (x, ' ', y);
FIN.
```

Ce programme affiche

```
3 4
4 3
3 4
```

Procedure sans parametres

- **Attention :**
  - Le nom de la procedure est un **identificateur**.
  - On déclare toute procédure avant le **DEBUT** du programme principal.

Appel d'une procédure

- On peut très bien appeler une procédure P1 depuis une procédure P2, mais il faut que la procédure P1 ait été déclarée avant la procédure P2.

Appel d'une procédure

Exemple donnant le même resultat que l'exemple précédent.

```
ALGORITHME exemple2;
VARIABLE x, y, t : entier;
PROCEDURE affiche_xy()
DEBUT
        ecrire (x, ' ', y);
FIN;
PROCEDURE echange_xy()
DEBUT
        t ← x; x ← y; y ← t;
        affiche_xy();
FIN;
```

```
DEBUT
    x ← 3; y ← 4;
    affiche_xy();
    echange_xy();
    echange_xy();
FIN.
```

Procédure

- Remarque
  - On peut aussi appeler une procedure depuis elle-même : c'est la recursivite, que l'on etudiera plus tard.

Procédure et variable locale

- Les objets du programme qui ne sont utilisés que dans la procédure peuvent être définis dans les déclarations locales de la procédure.

Procédure et variable locale

Exemple Reprenons exemple1 et changeons t :

```
ALGORITHME exemple1;
VARIABLE x, y, t : entier;
{ Declaration de la procedure echange_xy }
PROCEDURE echange_xy()
DEBUT
      { Corps de la procedure }
      t ← x; x ← y; y ← t;
FIN;
DEBUT
      { Programme principal }
      x ← 3; y ← 4;
      ecrire (x, ' ', y);
      echange_xy() { 1er appel de la procedure }
      ecrire (x, ' ', y);
      echange_xy() { 2eme appel de la procedure }
      ecrire (x, ' ', y);
FIN.
```

Procédure et variable locale

Exemple Reprenons exemple1 et changeons t :

```
ALGORITHME exemple3;
VARIABLE x, y : entier;
PROCEDURE echange_xy()
VARIABLE t : entier; { Déclaration locale }
DEBUT
        t ← x; x ← y; y ← t;
FIN;
DEBUT
        { ... }
FIN.
```

Procédure et variable locale

- Une variable déclarée localement n'existe que pendant l'exécution de la procédure, et ne sert que à cette procédure.
- Le programme principal n'a jamais accès à une variable locale de procédure.
- Une procédure n'a jamais accès à une variable locale d'une autre procédure
- Améliore la lisibilité du programme.

Procédure : portée des variables

- Les variables déclarées dans la section VARIABLES du programme principal sont appelées variables globales. Elles existent pendant toute la durée du programme et sont accessible de partout.
- Une variable locale a une procédure P, portant le même nom x qu'une variable globale, masque la variable globale pendant l'execution de P.

Procédure : portée des variables

Exemple

```
ALGORITHME ex4;
VARIABLE x : entier;
PROCEDURE toto()
VARIABLE x : entier;
DEBUT
    x ← 4;
    ecrire ('toto x = ', x);
FIN;
```

```
DEBUT
    x ← 2;
    ecrire ('glob x = ', x);
    toto();
    ecrire ('glob x = ', x);
FIN.
```

Affichage ??

Procedure : portée des variables

Exemple

```
ALGORITHME ex4;
VARIABLE x : entier;
PROCEDURE toto()
VARIABLE x : entier;
DEBUT
    x ← 4;
    ecrire ('toto x = ', x);
FIN;
```

```
DEBUT
    x ← 2;
    ecrire ('glob x = ', x);
    toto();
    ecrire ('glob x = ', x);
FIN.
```

Ce programme affiche

glob $x = 2$
toto $x = 4$
glob $x = 2$

Procédure : portée des variables

Voici le scenario catastrophe :

- On est dans une procedure P et on veut modifier une variable x locale a P.
- Il existe déjà une variable globale ayant le même nom x.
- On oublie de déclarer la variable locale x au niveau de P.
- A la compilation tout va bien !
- A l'exécution, P modifie le x global alors que le programmeur ne l'avait pas voulu.
- Conséquence : le programme ne fait pas ce qu'on voulait, le x global à l'air de changer de valeur tout seul !
- Erreur très difficile a détecter ; il faut être très rigoureux et prudent !

Procédure avec parametres

Il est tres utile de paramétrer une procedure.

Pseudo-passage de paramétres

Ecrivons une procédure **produit** qui calcule z = x*y.

Procédure avec parametres

```
ALGORITHME exemple5;
VARIABLE x, y, z, a, b, c, d : reel;
PROCEDURE produit()
DEBUT
        z ← x * y;
FIN;
```

Procédure avec paramètres

On veut se servir de Produit pour calculer c = ab et d = (a-1)(b + 1).

```
DEBUT
    ecrire ('a b ? ‘);
    lire (a, b);
    x ← a; y ← b; { donnees }
    produit();
    c ← z; { resultat }
    x ← a-1; y ← b+1; { donnees }
    produit();
    d ← z; { resultat }
    ecrire ('c = ', c, ' d = ', d);
FIN.
```

Procédure avec paramètres

Remarques

- L'écriture est un peu lourde.
- Il faut savoir que la procedure < communique > avec les variables x, y, z.
- Cela interdit de se servir de x, y, z pour autre chose que de communiquer avec la procédure ; sinon gare aux effets de bord !
- Deux sortes de paramètres : données et résultats.

Procedure avec paramètres

La solution élégante consiste à déclarer des paramètres au niveau de la procédure :

PROCEDURE AVEC PARAMETRES

Syntaxe

PROCEDURE nom_proced ([{[params] : [types_params],}])

VARIABLE [{[var_locales] : [type],}];

DEBUT

{ instructions };

FIN;

Tout ce que l'on a dit sur le paramétrage des procédures reste valable pour les fonctions.

Procédure avec paramètres

```
ALGORITHME exemple5;
VARIABLE x, y, z, a, b, c, d : reel;
PROCEDURE Produit()
DEBUT
                z ← x * y;
FIN;
DEBUT
      ecrire ('a b ? ‘);
       (a, b);
      x ← a; y ← b; { donnees }
      Produit();
      c ← z; { resultat }
      x ← a-1; y ← b+1; {donnees }
      Produit();
      d ← z; { resultat }
      ecrire ('c = ', c, ' d = ', d);
END.
```

ALGORITHME exemple5bis;

VARIABLE a, b, c, d : reel;

PROCEDURE Produit **(x, y : reel; variable z: reel)**
{ parametres }

DEBUT

z ← x * y;

FIN;

DEBUT

ecrire ('a b ? ‘); lire (a, b);

Produit (a, b, c); { passage de }

Produit (a-1, b+1, d); { parametres }

ecrire ('c = ', c, ' d = ', d);

FIN.

Paramètres comment ca marche ?

- A l'appel, on donne des paramètres dans les parenthèses, séparés par des virgules, et dans un certain ordre (ici a puis b puis c).
- L'exécution de la procédure commence ; la procédure reçoit les paramètres et identifie chaque paramètre a une variable dans le même ordre (ici x puis y puis z).
- Les types doivent correspondre ; ceci est vérifié a la compilation.

Paramètres : types de passage

- Il y a deux sorte de passage de paramètres : le passage par valeur et le passage par référence.
- Passage par valeur : à l'appel, le paramètre est une variable ou une expression.
- C'est la valeur qui est transmise, elle sert à initialiser la variable correspondante dans la procédure (ici x est initialise à la valeur de a et y à la valeur de b).
- Passage par **référence ou adresse** : à l'appel, le paramètre est une variable uniquement (jamais une expression). C'est l'adresse mémoire (la référence) de la variable qui est transmise, non sa valeur. La variable utilisée dans la procédure est en fait la variable de l'appel, mais sous un autre nom (ici z désigne la même variable (zone mémoire) que a).
- C'est le mot-clé **variable** qui dit si le passage se fait par valeur (pas de variable) ou par **référence ou adresse** (présence de variable).

Paramètres : types de passage

- Pas de variable = données ; présence de variable = données/résultat.
- **Erreurs classiques**
  - Mettre variable quand il n'en faut pas : on ne pourra pas passer une expression en paramètre.
  - Oublier variable quand il en faut : la valeur calculée ne pourra pas < sortir > de la procédure.

Paramètres : passage et erreur

Exemples d'erreurs a l'appel de Produit (a-1, b+1, d);

- PROCEDURE Produit (variable x : réel; y : réel; variable z : réel);

ne compile pas a cause du paramètre 1, ou une variable est attendue et c'est une expression qui est passée.

- PROCEDURE Produit (x, y, z : réel);

produit une erreur a l'exécution : d ne reçoit jamais le résultat z car il s'agit de 2 variables distinctes.

- Portée des variables :

dans Exemple5bis, les paramètres x, y, z de la procédure Produit sont des variables locales a Produit.

Paramètres : passage et erreur

Leur nom n'est donc pas visible de l'extérieur de la procédure.

Attention : redéclarer un paramètre comme variable locale ! erreur a la compilation. Exemple :

```
PROCEDURE Produit (x, y : réel; variable z : réel);
VARIABLE
    t : réel; { déclaration d'une variable locale : permis }
    x : réel; { redéclarations d'un paramètre : interdit }
DEBUT
    z ← x * y;
FIN;
```

Paramètres : passage et erreur

- Le seul moyen pour une procédure de communiquer avec l'extérieur, c'est-à-dire avec le reste du programme, ce sont les variables globales et les paramètres.
- Il faut toujours éviter soigneusement les effets de bords.
- Le meilleur moyen est de paramétrer complètement les procédures, et d'éviter la communication par variables globales.
- Les variables de travail tels que compteur, somme partielle, etc doivent être locales à la procédure, surtout pas globale.
- Prendre l'habitude de prendre des noms de variables différents entre le programme principal et les procédures : on détecte plus facilement à la compilation.
- Chaque fois que l'on appelle une procédure, on vérifie particulièrement le bon ordre des paramètres et la correspondance des types.
- La compilation est très pointilleuse sur les types, mais par contre elle ne détecte pas les inversions de paramètres de même type.

FONCTION

- Une fonction est une procédure qui renvoie un résultat, de manière à ce qu'on puisse l'appeler dans une expression.
- Exemple: y ← cos(x) + 1;

FONCTION SANS PARAMETRES

Syntaxe

FONCTION nom_fonction(): type_resultat

VARIABLE {Definition des variables}

DEBUT

{ ... Corps de la fonction... }

{ Résultat de la fonction, de type type_resultat }

RETOURNER expression;

FIN;

FONCTION SANS PARAMETRES

- Une fonction doit toujours retourner un resultat (i.e on ne peut pas le laisser indétermine).

FONCTION SANS PARAMETRES

```
ALGORITHME ex1;
VARIABLE x : type_resultat;
{ ici declaration de la fonction }
DEBUT
    { appel fonction et stockage du resultat dans x }
    x ← nom_fonction();
FIN.
```

FONCTION AVEC PARAMETRES

Syntaxe

FONCTION nom_fonction ([{[params] : [types_params],}]) : type_resultat

VARIABLE [{[var_locales] : [type],}];

DEBUT

{ ... }

RETOURNER expression;

FIN;

Tout ce que l'on a dit sur le paramétrage des procedures reste valable pour les fonctions.

PROCEDURE VERSUS FONCTION

```
ALGORITHME exemple5bisProc;
VARIABLE a, b, c, d : reel;
PROCEDURE Produit (x, y : reel;
variable z: reel); { parametres }
DEBUT
z ← x * y;
FIN;
DEBUT
        ecrire ('a b ? '); lire (a, b);
        Produit (a, b, c); { passage de }
        Produit (a-1, b+1, d); { parametres }
        ecrire ('c = ', c, ' d = ', d);
FIN.
```

```
ALGORITHME exemple5bisFunc;
VARIABLE a, b, c, d : reel;
FONCTION Produit (x, y : reel; ):reel; {
    parametres }
Variable z;
DEBUT
z ← x * y;
RETURN z;
FIN;
DEBUT
        ecrire ('a b ? '); lire (a, b);
        c←Produit (a, b); { passage de }
        d←Produit (a-1, b+1); { parametres }
        ecrire ('c = ', c, ' d = ', d);
FIN.
```

Procedure et fonction en langage C ?

Procedure sans parametres en C

```
// Declaration de la procedure nom_procedure
void nom_procedure()
{
    //Declaration des variables locales
   // Corps de la procedure

}
```

Procedure sans parametres en C : Exemple

```
// Declaration de la procedure affichage
void affichage()
{
    printf("Bonjour tout le monde ! ");
}
```

Procédure sans paramètres en C
Exemple complet

```
#include <stdio.h>
// Declaration de la procédure Echange_xy
void echange_xy()
{
// Corps de la procédure
t = x; x = y; y = t;
};
int main(){
      int x, y, t;
       //Programme principal
       x = 3; y = 4;
       printf("%d %d ",x,y);
       echange_xy();  //1er appel de la procedure
       printf("%d %d ",x,y);
       echange_xy(); //2eme appel de la procedure
       printf("%d %d",x,y);
       return 0;
}
```

Procédure sans paramètres en C
Exemple complet

```
#include <stdio.h>
// Declaration de la procédure Echange_xy
void echange_xy()
{
// Corps de la procédure
t = x; x = y; y = t;
};
int main(){
        //Programme principal
        int x, y, t;
        x = 3; y = 4;
        printf("%d %d ",x,y);
        echange_xy();  //1er appel de la procedure
        printf("%d %d ",x,y);
        echange_xy(); //2eme appel de la procedure
        printf("%d %d",x,y);
        return 0;
}
```

Ce programme affiche

```
3 4
4 3
3 4
```

Procedure sans paramètres

```
#include <stdio.h>
// Declaration de la fonction echange_xy
void affiche_xy()
{
        printf("%d %d ",x,y);

}
void echange_xy()
{
    // Corps de la fonction
    t = x; x = y; y = t;
    affiche_xy();
};
```

```
int main(){
    //Programme principal
    int x, y, t;
    x = 3; y = 4;
    Affiche_xy();
    echange_xy();  //1er appel de la procédure
    echange_xy(); //2eme appel de la procédure

    return 0;
}
```

Procedure avec paramètres

VOID nomprocedure ([{[params] : [types_params],}])

```
{
 // corps de la fonction
}
```

Le passage par adresse en remplacant variable par *.

Procedure avec paramètres en C

```
#include <stdio.h>
VOID produit (float x, y, float *z)
{
        *z =x * y;
}
int main(void){
    float a, b, c, d;
     printf("a b ? "); scanf("%f %f", &a, &b);
     produit (a, b, &c); // passage de
     produit (a-1, b+1, &d); // parametres
     printf ("c = %d , d =  %d",c,d);
}
```

Procedure avec paramètres en C

Le langage C ne fait pas la différence entre procédure et fonction contrairement à certains langages comme le langage PASCAL. Cependant, il faut préciser le type retourné dans le préambule de la fonction à la place du VOID. De plus, il faut utiliser le mot clé **RETURN** pour retourner effectivement la valeur.

FONCTION SANS PARAMETRES

Syntaxe

```
type_retour nom_fonction()
{
    // ... Corps de la fonction...
    // Résultat de la fonction, du type type_resultat }
    RETURN valeur;
}
```

FONCTION AVEC PARAMETRES

```
#include <stdio.h>
float produit (float x, y)
{    float z;
     z= x * y;
     RETURN z;
}
int main(void){
      float a, b, c, d;
       printf("a b ? "); scanf("%f %f", &a, &b);
       c=produit (a, b); // passage de
       d=produit (a-1, b+1); // parametres
       printf ("c = %f , d =  %f",c,d);
}
```

FONCTION AVEC PARAMETRES

Syntaxe

type_retour **nom_fonction**([{[params] : [types_params],}])
{
    // ... Corps de la fonction...
    // Résultat de la fonction, du type type_resultat }
    RETURN valeur;
}

Tout ce que l'on a dit sur le paramétrage des procédures reste valable pour les fonctions.

EXERCICE D'APPLICATION

- Ecrire une fonction en C qui retourne x^y (x à la puissance y).
- Ecrire une fonction permettant de convertir un nombre écrit en binaire en décimal.
- Ecrire un programme principal permettant de tester ces deux fonctions.