Sous-programmes

Procédures et fonctions

Dr Khadim DRAME
kdrame@univ-zig.sn

Département d'Informatique
UFR des Sciences et Technologies
Université Assane Seck de Ziguinchor

7 juillet 2021

Plan

Introduction

Problèmes et solutions

- Problèmes
  - **Problème de lisibilité** dans les gros programmes ;
  - Difficultés de trouver des **erreurs**.
- Solutions
  - Décomposition du problème en sous problèmes plus simples ;
  - Un sous-programme pour résoudre un sous problème.
- C'est le principe de la **programmation modulaire**.
- Il permet de **réduire la taille des programmes** mais surtout de mieux **organiser le code**.

Introduction

Sous-programmes

- Un **sous-programme** est une partie d'un programme conçue pour faire un traitement bien défini.
- Un sous-programme est généralement défini **indépendamment du contexte** particulier du programme.
- On distingue deux types de sous-programmes : les **procédures** et les **fonctions**.
- Avantages des sous programmes :
  - une grande **lisibilité** des programmes ;
  - la **réutilisabilité** de sous-programmes existants ;
  - une **maintenance** facile des programmes.

Plan

Fonctions

Notion de fonction

- Une **fonction** est un sous-programme qui
  - prend en entrée des valeurs appelées **paramètres** ;
  - applique un traitement (instructions) sur ces valeurs ;
  - **retourne un** (et un seul) **résultat** au programme appelant.
- Une fonction peut contenir une partie déclaration.
- On distingue la **définition** (déclaration) de l'**appel** d'une fonction.

Fonctions

Définition de fonction

function
<nomf>(<par_1> :<type_1> ;.. ;<par_n> :<type_n>) :<type>
begin
<instructions> ;
<**nomf**> :=<expression> ;
end ;

- <nomf> est le **nom** de la fonction ;
- <par_i> sont les **paramètres** de la fonction ;
- <type_i> sont les **types des paramètres** ;
- <type> est le **type de résultat** retourné par la fonction ;
- le **corps** de la fonction est entre **begin** et **end** ;
- <expression> est la valeur que la fonction va retourner.

- Dans sa définition, une fonction peut faire appel à d'autres fonctions et/ou procédures.

Fonctions

Définition de fonction

- Les paramètres (formels) sont séparés par des points virgules.

  function puiss(x : real **;** n : integer) : real ;

- Les paramètres de même type, on peut les mettre ensemble en les séparant par des virgules.

  function somme(x**,**y : real) : real ;

- Une fonction peut contenir une partie déclaration (avant begin).
- La fonction retourne une valeur qui porte son nom, donc elle doit contenir dans son corps une instruction permettant d'affecter cette valeur à la fonction (nomf :=expression).

Fonctions

Définition de fonction

- Exemples

```
1 function cube(x:integer):integer;
2 begin
3   cube:=x*x*x;
4 end;
```

```
1 function moyenne(x,y,z:integer):real;
2 begin
3   moyenne:=(x+y+z)/3;
4 end;
```

Fonctions

Appel de fonction

- <nomf>(<arguments>) ;
  - <nomf> est le **nom** de la fonction ;
  - <arguments> sont les **arguments** (paramètres effectifs ou réels) de la fonction ;
  - Les argument sont séparés par des virgules.
- Le **nombre** et l'**ordre** des arguments dans la définition de la fonction doivent être **les mêmes** dans l'appel.
- L'appel de fonction se fait généralement dans une affection ou une écriture.
- Exemples

```
m1 :=moyenne(11,10,12) ;
write(cube(2)) ;
z :=cube(2+3) ;
```

Fonctions

Appel de fonction

- Exemple

```
1  program calcul_moyenne;
2  var a, b, c : integer;
3    m1, m2:real;
4  {Définition de la fonction moyenne}
5  function moyenne(x,y,z:integer):real;
6  begin
7    moyenne:=(x+y+z)/3;
8  end;
9  begin
10   a:=12;
11   b:=10;
12   c:=14;
13   m1:=moyenne(a,b,c); {appel de fonction}
14   m2:=moyenne(14,12,16); {appel de fonction}
15   writeln(m1,' ',m2); {12 14}
16 end.
```

Plan

Procédures

Notion de procédure

- Une **procédure** est un sous-programme qui
  - prend en entrée des valeurs appelées **paramètres** ;
  - applique un traitement sur ces valeurs ;
  - **ne retourne pas directement** un résultat au programme appelant.
- Comme une fonction, une procédure peut contenir une partie déclaration.
- Une procédure modifie, en général (pas toujours), les valeurs de ses paramètres.

Procédures

Définition et appel de procédure

- Définition de procédure

  procedure
  <nomProc>(<par_1> :<type_1> ;.. ;<par_n> :<type_n>) ;
  begin
  <instructions> ;
  end ;

- Les paramètres de même type, on peut les mettre ensemble en les séparant par des virgules.
- Dans sa définition, une procédure peut faire appel à d'autres fonctions et/ou procédures.
- L'appel d'une procédure se fait comme l'appel d'une fonction.

  <nomProc>(<arguments>) ;

Procédures

Définition et appel de procédure

- Exemple

```
1  program affichage;
2  procedure affiche_somme(x,y:integer); {Défition d'
       une procedure}
3  var som:integer;
4  begin
5    som:=x+y;
6      writeln('La somme est : ',som);
7  end;
8  begin
9     affiche_somme(12,17); {La somme est : 29}
10    affiche_somme(2*5,13); {La somme est : 23}
11    affiche_somme(14,sqr(4)); {La somme est : 30}
12 end.
```

Procédures

Passage de paramètres par valeur/par adresse

- En général, les procédures modifient leurs paramètres.
- Dans certains cas, on ne veut pas que cette modification se répercute sur les paramètres réels du programme appelant.

  $\Rightarrow$ la procédure travaille sur une copie des paramètres réels : **passage par valeur**.
- Dans le cas où la procédure doit **modifier ses paramètres réels**.

  $\Rightarrow$ la procédure reçoit l'adresse mémoire des paramètres réels : **passage par adresse**.
- NB : Les paramètres d'une fonction (exceptés les tableaux) sont toujours passés par valeur.

Procédures

Passage de paramètres par valeur/par adresse

```
1  program modif_parametres;
2  var a, b : integer;
3  procedure double_val(x:integer);
4  begin
5    x:=2*x;
6  end;
7  procedure double_adr(var x:integer);
8  begin
9    x:=2*x;
10 end;
11 begin
12   a:=12;
13   b:=12;
14   double_val(a); {appel de double_val}
15   double_adr(b); {appel de double_adr}
16   writeln(a,' ',b); {12 24}
17 end.
```

Plan

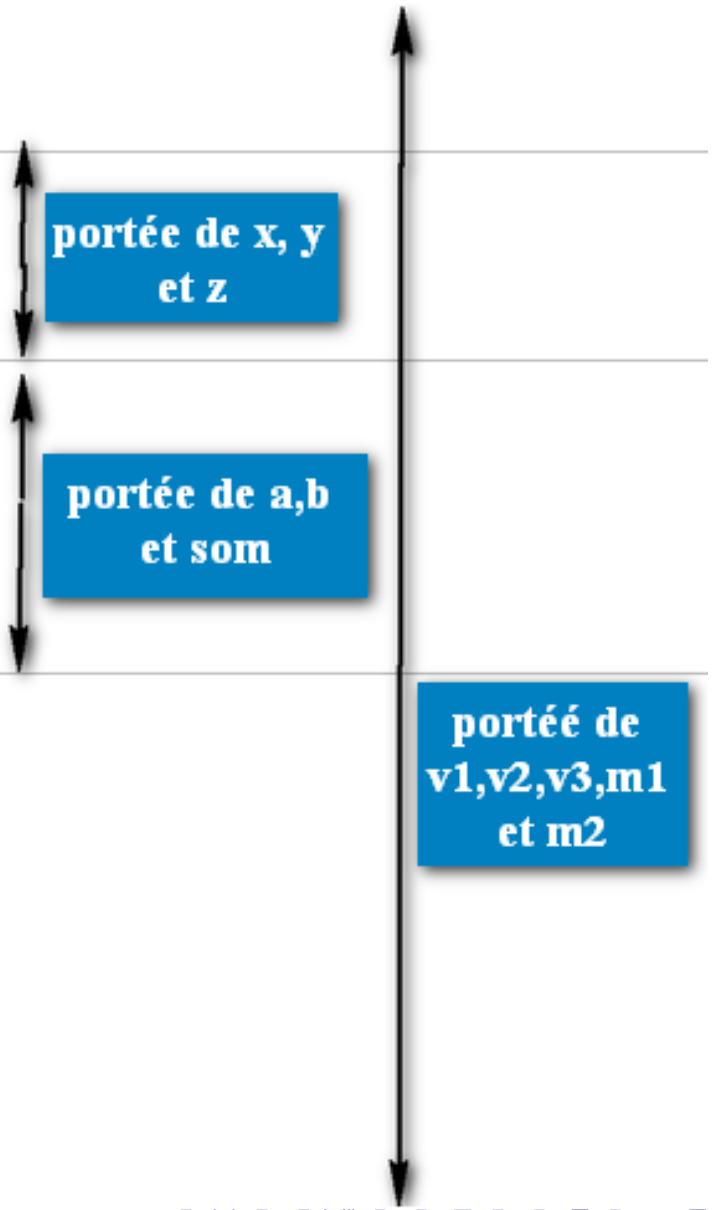
Portée d'une variable

Exemple

- La **portée** d'une variable est l'ensemble des endroits où cette variable existe et peut être utilisée.
- Elle peut être **locale** ou **globale**.
- Une variable définie au niveau d'une fonction ou d'une procédure a une portée **locale**.
  - Elle ne peut être utilisée qu'à l'intérieur de cette fonction ou procédure.
- Une variable **globale** est déclarée en dehors des fonctions et procédures.
  - Elle est disponible tout au long du programme et accessible à tous les sous-programmes.

Portée d'une variable

```
program caccul;
var v1, v2, v3:integer;
  m1, m2:real;
function moyenne(x,y,z:integer):real;
begin
     moyenne:=(x+y+z)/3;
end;
procedure affiche_somme(a,b:integer);
var som:integer;
begin
     som:=a+b;
     writeln('La somme est : ',som);
end;
begin
     v1:=12;
     v2:=10;
     v3:=14;
     m1:=moyenne(v1,v2,v3);
     m2:=moyenne(14,12,16);
     writeln(m1,' ',m2);
     affiche_somme(12,17);
     readln();
end.
```



Portée d'une variable

Effet de bord

- Modification d'une variable globale dans un sous-programme.

```
1  program portee_variables;
2  var a,b : integer;
3  procedure affiche_carre(x:integer);
4  begin
5    b:=x*x;
6    writeln('Le carré est : ',b);
7  end;
8  begin
9    a:=12;
10   b:=50;
11   affiche_carre(5); {Le carré est : 25}
12   writeln(b); {b vaut 25}
13   affiche_carre(a); {Le carré est : 144}
14   writeln(b); {b vaut 144}
15 end.
```

Exercices d'applications

- Exercice 1 : Qu'affiche le programme suivant ?

```
1  program calcul;
2  var x,y,s : integer;
3  procedure somme(a, b:integer);
4  begin
5    s:=a+b;
6    a:=a+b;
7    b:=b+4;
8  end;
9  begin
10   x:=2;
11   y:=9;
12   somme(x,y);
13   writeln(x,'+',y,'=',s);
14 end.
```

Exercices d'applications

- Exercice 1 : Qu'affiche le programme suivant ?

```
1  program calcul;
2  var x,y,s : integer;
3  procedure somme(a, b:integer);
4  begin
5    s:=a+b;
6    a:=a+b;
7    b:=b+4;
8  end;
9  begin
10   x:=2;
11   y:=9;
12   somme(x,y);
13   writeln(x,'+',y,'=',s);
14 end.
```

- Correction
  Le programme affiche : **2+9=11**

Exercices d'applications

- Exercice 2
  1. Écrire une fonction qui teste si un nombre est premier ou pas.
  2. Écrire un programme Pascal qui détermine tous les nombres premiers entre 1 et N, N étant fourni par l'utilisateur.

Exercices d'applications

- Correction de l'exercice 2

```
1  program nombres_premiers;
2  var i,n : integer;
3  function premier(a :integer):boolean;
4  var j:integer;
5  begin
6    premier:=true;
7    for j:=2 to a div 2 do
8      if a mod j = 0 then
9        premier:=false;
10 end;
11 begin
12   write('Donner la valeur de N : ');
13   readln(n);
14   for i:=2 to n do
15     if(premier(i)=true) then
16       write(i, ' ');
17 end.
```

A retenir

- En Pascal, une *fonction* retourne toujours une valeur.
- Une fonction ***nom_f*** contient toujours une instruction de la forme *nom_f := exp*.
- L'utilisation d'une fonction se fait, en général, par **affectation** (y := carre(x)) ou **écriture** (write(carre(x)).
- Il faut **éviter** des instructions de la forme *nom_f(args) := exp*.
- Une *procédure* ne retourne pas directement de valeur.
- Il faut **éviter** des instructions de la forme *v := nom_proc(args)*.
- Une procédure modifie, en général, la valeur de ses paramètres.