Rappels de PASCAL Ve. Beta2.2
HEC 922, 2001-2002

Laure Danthony
http://www.ens-lyon.fr/~ldanthon

13 novembre 2001

Ce document dont le seul objectif au début était de n'être qu'un aide mémoire (utile pour des questions de syntaxe), devient de plus en plus conséquent au fur et à mesure des versions. Ceci dit, il conserve son nom "rappels" car en aucun cas il ne se veut un cours de Pascal exhaustif. L'objectif est qu'il réponde au fur et à mesure de l'année aux questions des 922, qu'ils peuvent me poser en TP ou par mail à `ldanthon@ens-lyon.fr`. Merci de me signaler toute coquille et toute erreur éventuelles.

Table des matières

1 Le début du TP ...

Comment travailler avec Pascal dans les salles 714/715 ?

1. Démarrer Windows. Le mot de passe est trivial.
2. Après avoir démarré Windows, la première chose à faire est de s'assurer que le disque dur de son poste (*i.e.* l'ordinateur que l'on a devant soi) contient un répertoire à son nom comme sous-répertoire de `C:\Travail\922\`. Sinon, on le crée en se mettant dans le répertoire `C:\Travail\922\` et en faisant Fichier-Nouveau Dossier, et en changeant le nom de ce dossier en son nom perso.
3. Si le TP consiste à remplir un squelette, c'est-à-dire un programme à trous, il faut au préalable le récupérer à l'aide du réseau. Sinon, on passe à l'étape 4. En général, le texte se trouve sur un ordinateur dont le nom est donné au début du TP au tableau (Fuissé en salle 714, Saint Emilion en 715), dans le répertoire `Travail\922\Danthony\numero_tp\`. Pour le récupérer, il faut aller ouvrir le répertoire en question : cliquer sur l'icône Voisinage réseau, sélectionner le bon ordinateur (Fuissé par exemple), puis aller dans le répertoire `Travail\922\Danthony\numero_tp\`. Pour copier un fichier vers son répertoire, le sélectionner (simple clic), puis faire Edition-Copier, se placer dans son réperoire (de la machine en face de soi), et le coller (Edition-Coller). A ce moment, on possède sa propre version du squelette.
4. Pour travailler avec Pascal, on ouvre l'éditeur DOS en cliquant sur l'icône présente sur le bureau. Si on ne démarre pas avec un squelette, mais avec une feuille vide, JUSTE ENSUITE, on sauve la feuille courante en faisant File-Save, et en cherchant dans l'arborescence son répertoire personnel, puis en donnant un nom au fichier : par exemple `poly.pas`, et enfin en cliquant sur OK. Si on travaille avec un squelette, on fait File-Open et on cherche dans son répertoire perso le squelette en question.
5. Se mettre au travail.
6. Ne pas oublier de sauver (Touche F2) très régulièrement.
7. Pour compiler un programme, faire Alt-F9, pour l'exécuter, faire CRTL-F9.
8. A la fin du TP, enregistrer, puis sortir de l'éditeur DOS. Pour imprimer son travail, l'ouvrir avec le programme maison Imprime Pascal (icône sur le bureau). Il met les mots-clefs de Pascal en forme et l'impression est donc plus lisible.
9. Ne pas oublier de sortir de Windows.

2 Structure d'un programme Pascal

Un programme PASCAL possède la structure suivante :

```
program toto; {nom du programme}

uses crt ;    {pour avoir les entrées/sorties}
```

```
...            {code éventuel de procédures et
                de fonctions auxillaires}

var ... ;      {déclaration des variables du programme}

BEGIN
   ...         {le CORPS du programme}
END.
```

EXEMPLE 1 *Ainsi le programme suivant fonctionne bien :*

```
program bonjour;
uses crt;
BEGIN
   writeln('hello world');
   readln;
END.
```

... c'est-à-dire qu'il **compile** *avec succès et qu'à l'***exécution***, il écrit la chaîne* `hello world` *et demande à l'utilisateur de taper un caractère quelconque, puis* **termine** *(ce qui se manifeste dans votre éditeur Pascal par un retour à l'écran bleu).*

Si vous désirez un programme PASCAL plus complexe, rien ne vous empêche de remplir directement le corps du programme avec des boucles, des `writeln` *etc.*, mais on comprend très bien que cela peut vite devenir illisible. Il convient donc de segmenter son programme en sous-fonctions et sous-procédures, qui seront appelées dans le corps du programme principal.

REMARQUE 1 **IMPORTANT** : on aura noté que l'*écriture du code* d'une procédure -ou d'une fonction- (déclaration du nom, des paramètres, de ce qu'elle fait, le tout terminé par un `;`) est distincte de son *EVENTUEL appel* par une autre fonction, une autre procédure, ou à l'intérieur du corps du programme, et aussi de son *exécution* qui ne s'effectue que si, d'une part, on l'a effectivement appelée à l'intérieur du corps du programme (ou qu'on fait appel à une procédure, une fonction, qui y fait appel, *etc.*), et si d'autre part on lance l'exécution par `RUN`.

Lorsque l'on aprend le PASCAL, on utilise le plus souvent la structure d'un programme PASCAL comme suit : les procédures et les fonctions que l'on nous demande sont codées dans la première partie du programme (avant le corps), et on utilise le corps pour les tester, *i.e.* les appeler sur des exemples pertinents pour savoir si elles effectuent bien le travail demandé.

REMARQUE 2 Si on a bien suivi ce qui précède, en TP Pascal on écrit les procédures et les fonctions demandées AVANT le corps du programme, et on les appelle A L'INTERIEUR du code du programme. Evidemment, rien ne nous empêche (et même quelquefois cela nous est demandé) d'appeler des fonctions ou des procédures à l'intérieur d'autres fonctions ou d'autres pocédures[1].

[1] dans le jargon, on appelle ça des sub-routines

3 La syntaxe Pascal

Lorsque l'on programme, un certain nombre d'instructions sont utiles : les opérations usuelles, les entrées-sorties qui permettent l'impression de texte à l'écran, les boucles qui permettent de faire un nombre de fois paramétrables une certaine chose, *etc.*

Ici on présente avec des exemples les différentes possibilités offertes par le langage Pascal :

- les types de base sont les entiers : `integer`, les flottants : `real` (attention, le nombre de chiffres après la virgule est borné !), les tableaux : `array`. Un tableau d'entiers de taille 10 indicé de -16 à 60 sera déclaré `array[-16..60] of integer`. On accède à l'indice i du tableau (si sa valeur existe !) avec `nom_du_tableau[i]`. Remarquons que l'on peut aussi déclarer des tableaux de tableaux (matrices), à l'aide de la syntaxe `array[1..6,1..6] of integer` par exemple.
- les opérations d'addition (`+`), de multiplication (`*`), de soustraction, de division sont possibles : il faut toutefois distinguer la division réelle : `/`, de la division entière : `DIV`. On peut faire des opérations "modulo" sur les entiers : par exemple `n mod 2` vaut 1 si n est impair, 0 sinon. Les fonctions de puissance et de factorielle n'existent pas (c'est-à-dire que si elles existent, on vous interdit de les utiliser).
- L'opération d'affectation : "x reçoit 4" s'effectue à l'aide de l'opérateur `:=`, ici on écrit : `x:=4;`
- Les impressions à l'écran ont la syntaxe suivante : `write('toto')` permet d'écrire la *chaîne* toto, c'est à dire la lettre "t", suivie de la lettre "o",*etc.* Si on écrit `writeln('toto')`, après avoir écrit la chaîne "toto" il y aura un passage à la ligne effectué. **Attention**, `write(t)` (sans quote) n'écrit pas la lettre (le caractère) "t" mais la *valeur* que possède "t" à ce moment là (2, par exemple). Noter aussi que `write('t',i)` ; est un raccourci pour `write('t');write(i)`
- Si l'on veut demander à l'utilisateur un renseignement, on utilise `read` (ou `readln`). Par exemple, on peut demander à l'utilisateur un chiffre qui sera stocké dans la variable `x` *préalablement déclarée*, on écrit `readln(x)`.
- Notion de test : si on veut effectuer une instruction *seulement si* une certaine condition est réalisée, on utilise `if`. Par exemple, si on peut ajouter 2 à x seulement si x est pair, on écrit : `if x mod 2 =0 then x:=x+2`. Si on veut retrancher 3 dans le cas contraire, on écrit
  `if x mod 2 =0 then x:=x+2 else x:=x-3`. Noter que le test d'égalité se fait à l'aide du signe `=`, le test d'inégalité à l'aide de `<>` (il y a aussi `<=`, `>=` dont je vous laisse deviner la signification). Si après le `then` ou le `else` on veut faire plusieurs instructions, on encadre celles-ci par `begin` et `end;`.
- Lorsque l'on désire faire une série d'instruction un certain nombre de fois ("boucler"), il y a deux possibilités :
  - Si on connaît ce nombre de fois à l'avance (par exemple, c'est n), on utilise une boucle `for` donc la syntaxe est `for i:=0 to n do begin ... end;`, où les ... désignent les instructions à effectuer n fois. A chaque fin de boucle, la variable i est automatiquement incrémentée de 1. Les instructions internes à la boucle peuvent ou non dépendre de la va-

leur de i. Quelquefois, on peut avoir besoin de la syntaxe suivante : `for i:=58 downto -3 do begin ... end;` et ici, à chaque tour de boucle, la variable i est décrémentée de 1.

– Si on ne connaît pas ce nombre de fois mais que par contre on connaît une condition de sortie (par exemple une certaine valeur est à 0, une certaine variable booléenne est à `true`, *etc.*), on utilise une boucle `while` dont la syntaxe est la suivante : `while condition do begin ... end;` La condition est de type booléenne, c'est-à-dire que la réponse au test condition est `true` ou `false`. La condition peut être une conjonction de condition (utiliser `AND`) ou une disjonction de conditions (utiliser `OR`). La boucle `repeat ... until` peut aussi être utilisée, à condition que le programmeur qu'il l'utilise soit certain de la pertinence de son code (on peut tout-à-fait s'en passer). Bien noter que dans le cas d'une telle boucle, le corps de la boucle sera de toute façon effectuée une fois, même si le test suivant `until` est faux.

4 Les Booléens

Un **booléen** est une variable qui ne peut prendre que deux valeurs différentes : `true` ou `false`. On déclare une telle variable de la façon suivante : `toto : boolean`, et l'on peut initialiser une telle variable *après l'avoir déclarée, évidemment !* en faisant `toto:=true` ou `toto:=false`.

L'utilisation de ces variables peut se faire de façon analogue à l'utilisation de variable de type `integer` :

– Comme on dispose de fonctions binaires (ex : '+') pour les entiers, on dispose de fonctions binaires pour les booléens : ce sont les fonctions AND et OR dont on rappelle les tables de vérité :

AND	`true`	`false`
`true`	`true`	`false`
`false`	`false`	`false`

OR	`true`	`false`
`true`	`true`	`true`
`false`	`true`	`false`

On en déduit par exemple que si x vaut `true` et y vaut `false`, x `AND` y vaut `false`

– Comme la fonction '-' est une fonction unaire qui calcule l'opposé, on dispose de la fonction `NOT` qui réalise : si x vaut `true`, alors `NOT x` vaut false et vice-versa.

Retour sur la syntaxe du `if` et du `while` en Pascal : par exemple, lorsque l'on écrit `if (x<5) then y:=3 then z:=5`, que réalise Pascal ? En fait, il *commence par évaluer* la condition suivant le "if", ici $x <$?5 ("évaluer $x < 5$" signifie donner une valeur booléenne à "$x < 5$", c'est-à-dire `true` ou `false`). Si cette condition est vraie **A CE MOMENT PRECIS**, on réalise ce qu'il y a après le then, sinon on réalise ce qu'il y a après le else. On peut donc tout-à-fait réaliser des conditions du type `if (x<5 and y>7) then ...`, pourvu que ce qu'il y a entre le of et le then soit de type booléen. On peut donc aussi faire `if ai_fini then ...`, pourvu que `ai_fini` ait précédemment été déclarée avec le type `boolean`. La conditition du `while` et du `repeat` est aussi de type booléen.

5 Fonctions et procédures

5.1 Fonction vs Procédure

Une fonction RETOURNE un résultat (*i.e.* une valeur) alors qu'une procédure ne peut que modifier un paramètre. Une fonction est en général utilisée pour retourner un entier que l'on peut ensuite stocker dans une variable. Une procédure peut modifier ses paramètres d'entrée, mais si l'on veut que les modifications qui affectent un certain paramère x soient toujours valables APRES la sortie de l'appel à cette procédure, il faut que dans la déclaration de cette procédure la variable x soit précédée du mot-clef `var`.

5.2 Déclaration d'une fonction

Voici la syntaxe PASCAL pour une fonction :

```
function toto(x:type1;y:type2;...):type; {nom, paramètres, types}
var ...                                  {variables locales éventuelles}
begin
   ...                                   {corps de la fonction}
toto:=truc;
end;
```

Avec :

- `toto` est le nom de la fonction
- `x,y`, ...sont les paramètres de la fonction, c'est à dire ses données : on considère `x` et `y` comme connus, et on peut les manipuler à sa guise : par exemple, si ce sont des entiers, on peut les ajouter, les diviser, *etc.*pour obtenir le résultat demandé. Par contre, DANS TOUS LES CAS, à la sortie de l'appel à une fonction, aucun des paramètres n'a été modifié, autrement dit, à la sortie d'un éventuel appel à la fonction `toto`, `x` et `y` ont la même *valeur* qu'à l'entrée, même si on leur a fait subir des trucs du type `x:=x+1` !
- `type1` est le type de la première variable, par exemple `x` peut être un entier (type `integer`), un tableau (type `array`), ... Les variables peuvent être de types tous différents.
- `type` est le type retourné par la fonction. En Turbo-Pascal on ne peut retourner que des entiers ou des flottants (`real`). En Delphi on peut retourner des tableaux.
- Après `var` on déclare les variables LOCALES à la procédure. Ce sont de nouvelles variables : dans le corps elles doivent obligatoirement être initialisées avant d'être utilisées. Leur valeur est inaccessible à l'extérieur de la fonction. En conséquence, si on déclare un même nom de variable à l'intérieur d'une autre fonction ou d'une autre procédure, il d'agira d'une variable différente que l'on pourra manipuler autrement. On peut ainsi déclarer une variable locale `x:integer` à l'intérieur d'une fonction `toto1` et une AUTRE variable locale (qui se trouve avoir le même nom !) `x` de type `array` à l'intérieur d'une fonction `toto2`. Il n'y aura aucun conflit ![2]

[2] si vous trouvez que c'est un peu bizarre, pensez aux homonymes : ce sont des personnes différentes qui ont le même nom : ce n'est pas un problème si elles vivent dans des pays différents !

- `begin` et `end;` délimitent le corps de la fonction, c'est-à-dire l'ensemble des instructions nécessaires pour obtenir ce que l'on veut.
- `toto:=truc` est l'affectation finale, c'est comme cela que l'on dit que le résultat de la fonction `toto` est `truc`, que l'on a auparavant calculé à l'aide des variables locales et des paramètres.

5.3 Déclaration d'une procédure

Voici la syntaxe PASCAL pour une procédure :

```
procedure titi(x:type;y:type;...;var z:type);
var ...
begin
   ...
end;
```

Avec :

- `titi` est le nom de la procédure ;
- `x,y,z` sont ses paramètres. On peut modifier `x,y,z` à l'intérieur de la procédure. Une différence cependant : la valeur de `x` et de `y` à la fin de l'exécution de la procédure sera la même qu'au début, alors que les modifications apportées au paramètre `z` seront enregistrées (on a utilisé le mot-clef `var`[3]).
- On ne retourne aucun résultat, par contre on peut IMPRIMER (à l'écran) à certains moments de l'exécution la VALEUR de certaines variables au moment où l'on appelle `write` par exemple.
- Pareillement à une fonction, on peut déclarer des variables locales et le corps est délimité par `begin` et `end;`.

5.4 Appel à une fonction

Problématique : on désire avoir accès au résultat d'une fonction à l'intérieur d'une autre fonction, d'une autre procédure ou du corps du programme. Comment faire ?

Solution

- Comme le résultat de la sous-fonction est une VALEUR qui est retournée, il suffit de stocker à l'intérieur d'une variable déclarée dans la sur-fonction cette valeur.
- Syntaxe : (par exemple)

```
function sous_fonction(...):integer;
begin
   ...
   sous-fonction := truc;
end;

function sur_fonction(...):...;
var temp : integer;
begin
    ...
```

[3] attention, il n'a pas la même signification à cet endroit que lorsque on déclare des variables locales !

```
    temp:=sous_fonction(...);
    ...
    sur_fonction:=bidule;
end;
```

REMARQUE 3 On peut évidemment faire de même à l'intérieur d'une procédure !

5.5 Appel à une procédure

Problématique : on désire avoir accès au résultat d'une procédure à l'intérieur d'une autre procédure, d'une autre fonction ou du corps du programme. Comment faire ?

Solution

- Comme une procédure ne retourne rien, on utilise le fait qu'elle peut modifier certains de ses paramètres (ceux qui suivent le mot-clef `var`), la modification de ces paramètres restant visible à la sortie de son exécution.
- A l'intérieur de la sur-procédure ou de la sur-fonction, on utilise donc une(des) variable(s) qui sera(seront) destinée(s) à être modifiée(s) par la procédure. On pourra ensuite éventuellement calculer, imprimer, retourner (dans le cas d'une fonction) cette variable qui possèdera alors la bonne valeur.
- Syntaxe : (par exemple)

```
procedure sous_proc(x,y:integer;...;var z:integer);
begin
   ...
   z:=truc;
   ...
end;

procedure sur_proc(...):...;
var temp : integer;
begin
    ...
    sous_proc(1,2,...,temp); {la variable temp est modifiée}
    ...
    writeln(temp);
end;
```

REMARQUE 4 (IMPORTANT) Lorsque l'on fait appel à une fonction ou à une procédure, les paramètres de la fonction ou de la procédure sont remplacés par des trucs dont on sait la valeur, par exemple le chiffre 2, ou encore le mot *truc* à qui on a préalablement donné la valeur 3, *etc.*Le x paramètre de la fonction ou de la procédure `toto` est donc une variable "muette", c'est-à-dire que lorsque l'on appelle `toto(y)`, l'ordinateur fera subir à y les instructions de la procédure `toto` *comme si il s'appelait x.* On peut donc écrire tout un tas de procédures ou de fonctions en utilisant le même nom de paramètre `x`.

REMARQUE 5 Une procédure est souvent utilisée à la place d'une fonction lorsque l'on veut que le résultat soit un certain tableau. En effet, une fonction

Pascal ne peut retourner un tableau. On utilise une procédure qui modifie un de ses paramètres (déclaré avec var) de type tableau (`array`), et on appelle cette procédure sur un tableau déjà déclaré. Les modifications faites à ce tableau lors de l'exécution de la procédure seront enregistrées et gardées telles-qu'elles après cette exécution.

5.6 Exemple

Illustrons maintenant sur un exemple simple la différence entre une fonction et une procédure qui "font $x + y$" :

- La **fonction** va tout simplement stocker dans une variable le résultat de l'opération $x + y$, puis le retourner :

```
function addition(x,y:integer):integer;
var temp;
begin
    temp:= x+y;
    addition:=temp;
end;
```

Remarquer que dans la fonction précédente, on aurait pu se passer de l'utilisation de la variable auxillaire `temp`.

Si l'utilisateur veut visualiser le résultat de `addition(2,4)`, il faut qu'à l'intérieur du code du programme Pascal, il fasse appel à la fonction `addition(2,4)`, puis qu'il imprime le résultat :

```
program operations;
...
ici le code de la fonction addition
...

var result:integer;
BEGIN
   result:=addition(2,4);              {affectation de result}
   writeln(result);
END.
```

- La **procédure** va modifier la variable z de telle façon à ce qu'elle reçoive le résultat $x + y$.

```
procedure addition(x,y:integer;var z:integer);
begin
    z:=x+y;
end;
```

Pour visualiser le résultat, on va déclarer une variable `result` qui va être modifiée pour recevoir le résultat. Ensuite on l'imprimera :

```
program operations;
```

```
uses crt ;
...
code de la procedure addition
...
var result:integer;
BEGIN
    result:=0;               {ici result vaut 0}
    addition(2,4,result);    {modification de la valeur de result, qui
                              maintenant vaut 6}
    writeln(result);         {impression de la valeur 6}
END.
```

6 Déclaration de constantes, de types

6.1 Constantes globales

On trouve souvent en Pascal l'occasion de déclarer des constantes globales, par exemple pour utiliser une valeur numérique tout au long du programme, ou pour borner la taille des tableaux que l'on utilise.

Par exemple, la phrase Pascal suivante : `CONST PI=3.1415` déclare que pour tout le reste du programme, *la chaîne* `PI` est un alias pour la *chaîne* `3.1415`, c'est à dire qu'à la compilation, toutes les occurences de la chaîne `PI` dans le programme seront remplacés par la chaîne `3.1415`. Dans cette exemple, on voit toute l'utilité de telles déclarations de constantes : si on décide en cours d'élaboration du programme Pascal d'augmenter la précision du calcul, il suffira de changer dans la déclaration `3.1415` par `3.1415926` une seule fois ! pour que la modification soit prise en compte dans tout le programme, pour peu que l'on ait utilisé la chaîne `PI` lorsque l'on voulait écrire une valeur approchée de π. De même en analyse, un "petit epsilon" pourra être au début fixé à `1e-6`, valeur qui pourra être facilement modifiée en fonction de la qualité des résultats obtenus.

Lorsque l'on utilise des tableaux en Pascal, on est obligé d'utiliser des tableaux de taille bornée fixée à l'avance. Il est alors intéressant de déclarer au début du programme une constante qui vaudra la taille maximale des tableaux utilisés. Comme cela, on pourra changer cette valeur autant que l'on veut, en ne modifiant qu'une seule ligne de code.

6.2 Déclaration de nouveaux types

Qu'est-ce qu'un type ? Un type ce n'est rien d'autre qu'un nombre de cases mémoires dont pourrait avoir besoin Pascal. On lui dit en quelque sorte à l'avance de préparer assez de place pour stocker ce que l'on va lui donner. Par exemple, lorsque l'on utilise le type déjà défini `integer` en déclarant une nouvelle variable à l'intérieur d'une procédure : `var toto:integer`, on dit à Pascal de préparer mettons 16 cases adjacentes en mémoire qui seront étiquetées par "toto", et tout calcul sur `toto` sera réalisé à partir de ces mêmes 16 cases. Il faut bien noter que ces mêmes cases sont détruites à l'extérieur de la procédure : en fait, Pascal sait qu'il peut les écraser à un autre moment où il peut avoir

besoin de place. C'est pour toutes ces raisons qu'il et nécessaire de bien déclarer ses variables et de bien vérifier qu'elles existent encore lorsqu'on les utilise.

Déclaration de types En Pascal, on n'a que peu de types à disposition : les différents types de nombres, les booléens et les tableaux (`array`). Il peut être intéressant de déclarer un type personnel pour un TP donné, par exemple si tout au long du TP on manipule des trucs du type `array[1..5] of array[1..5] of boolean` (on aura bien sûr reconnu des matrices booléennes de taille 5-5), il peut être intelligent pour des questions de nombre de caractères à taper et de lisibilité du code de déclarer `type mat_bool = array[1..5] of array[1..5] of boolean`! Remarquez au passage la syntaxe de déclaration de types.

7 Petits exercices

EXERCICE 1 *Ecrire un programme Pascal qui demande une certaine valeur à l'utilisateur, calcule le quadruple de cette valeur, et écrit cette valeur à l'écran.*

EXERCICE 2 *Ecrire un programme Pascal qui demande une certaine valeur i à l'utilisateur, calcule 2^i, et écrit cette valeur à l'écran. On interdit l'usage de l'opérateur* `**`*. On utilisera donc une boucle bornée.*

EXERCICE 3 *Même exercice sauf que l'on écrit à l'écran tous les 2^i, en passant à la ligne à chaque fois.*

EXERCICE 4 *Taper le programme suivant :*

```
program tab;
uses crt;

var P : array[6..12] of integer

begin
 ...
end;
```

Remplir les blancs de façon à remplir une à une les cases de P par des entiers demandés à l'utilisateur. On rappelle que l'accès à la case i d'un tableau T s'effectue avec `T[i]`*.*

EXERCICE 5 *Compléter le programme précédent de façon à ce qu'ensuite, les chiffres entrés par l'utilisateurs soient rendus visibles par une impression écran.*

EXERCICE 6 *Ecrire une fonction* `max(a,b:integer):integer` *qui retourne le maximum de a et b. On demande de mettre cette fonction AVANT le corps du programme et de la tester à l'intérieur du programme (l'appeler sur $a = 3$ et $b = -1$, sur $a = 6 = b$, etc.).*

EXERCICE 7 *Ecrire une procédure* `maxi(a,b:integer;var c:integer)` *qui met le maximum de a et de b dans la variable c. La tester.*