Annexe A

Syntaxe FreePascal

La syntaxe en FreePascal des instructions vues en pseudo-code en cours est donnée dans cette annexe.

A.1 Procédures et fonctions

Une procédure en Pascal se définit avec la syntaxe suivante :

```
procedure nom_procedure(parametres);
   declarations locales
begin
   corps de la procedure
end;
```

Les paramètres d'entrée sont définis par un nom de variable suivi de : puis du type de donnée correspondant. Si plusieurs paramètres d'entrée ont le même type, les noms de variable peuvent être tous mis avant le type de donnée, séparés par une virgule. Si des paramètres de différents types sont utilisés il faut les séparer avec un point-virgule. Les paramètres de sortie et d'entrée/sortie s'écrivent de la même manière sauf qu'ils sont précédés par le mot-clé **var**.

Les déclarations locales définissent les variables locales à la procédure en utilisant le mot-clé **var** suivi du nom de variables, de : et d'un type de donnée.

Le corps de la procédure contient les instructions exécutées lors d'un appel.

Exemple A.1 Une procédure permutant les valeurs de deux variables entières :

```
procedure swap(var a,b : Integer);
var temp :  Integer;
begin
  temp:=a;
  a:=b;
  b:=temp
end;
```

Une fonction en Pascal se définit avec la syntaxe suivante :

```
function nom_fonction(parametres): type de retour;
   declarations locales
begin
   corps de la fonction
   nom_fonction := valeur a retourner;
end;
```

Les paramètres d'entrée sont définis par un nom de variable suivi de : puis du type de donnée correspondant. Si plusieurs paramètres d'entrée ont le même type, les noms de variable peuvent être tous mis avant le type de donnée, séparés par une virgule. Si des paramètres de différents types sont utilisés il faut les séparer avec un point-virgule.

Le type de retour est le type de donnée de la valeur renvoyée par la fonction.

Les déclarations locales définissent les variables locales à la procédure en utilisant le mot-clé **var** suivi du nom de variables, de : et d'un type de donnée.

Le corps de la procédure contient les instructions exécutées lors d'un appel. Enfin, l'instruction **retourner** en Pascal s'écrit par une affectation sur le nom de la fonction. On utilise le nom de la fonction suivi de := suivi de la valeur à retourner.

Exemple A.2 Une fonction qui renvoie le minimum entre 2 valeurs entières :

```
function minimum(a,b : Integer): Integer;
begin
  if (a > b) then minimum := b
  else minimum := a;
end;
```

L'appel d'une procédure ou d'une fonction se fait par son nom suivi de parenthèses et des paramètres effectifs de l'appel. Avant d'appeler une procédure ou une fonction il faut qu'elle soit déclarée. Si elle a été préalablement définie, elle est déclarée. Si non, il est possible de déclarer une procédure ou une fonction avant de la définir en écrivant sa signature suivie du mot-clé **forward**.

A.2 Tableaux

Un type de donnée tableau à une dimension s'écrit avec la syntaxe suivante :

```
array [indiceDebut..indiceFin] of TypeDesElements
```

indiceDebut et *indiceFin* sont des constantes de type scalaire et du même type de donnée. TypeDesElements est le type de toutes les données stockées dans un tableau. On accède ensuite à un indice donné du tableau en l'indiquant entre crochets.

Exemple A.3

```
var tab : array[1..10] of Integer;
begin
```

```
  tab[1] := 10;
  tab[2] := tab[1]*2;
end;
```

Un tableau à plusieurs dimensions (≥ 2) se déclare de la même manière mais en séparant chaque dimension entre les crochets par un intervalle séparé des autres par des virgules. L'accès à des indices donnés se fait en mettant entre crochets un indice pour chaque dimension du tableau. La syntaxe de déclaration est :

```
array [indDeb1..indFin1, indDeb2..indFin2, ... ,
          indDebN..indFinN] of TypeDesElements
```

Exemple A.4 Un exemple de déclaration et d'utilisation d'un tableau à 2 dimensions est :

```
var tab : array[1..2,1..2] of Integer;
begin
  tab[2][1] := 42;
end;
```

Il est aussi possible de définir des tableaux à plusieurs dimensions en déclarant des tableaux de tableaux.

Exemple A.5 Même exemple avec un tableau de tableau :

```
var tab : array[1..2] of array[1..2] of Integer;
begin
  tab[2][1] := 42;
end;
```

A.3 Structures

Les structures en Pascal sont appelées des enregistrements et utilise le mot-clé **record**. Après le mot-clé **record**, les champs de l'enregistrement sont définis jusqu'à l'instruction **end**. La syntaxe générale pour définir un nouveau type enregistrement est :

```
Type nom du type = record
       champ1 : type du champ1;
       champ2 : type du champ2;
       ...
end;
```

Exemple A.6 Un exemple d'enregistrement représentant une personne est :

```
Type Personne = record
       prenom, nom : String;
       age : Integer;
end;
```

Pour accéder à un champ d'un enregistrement, on utilise le symbole point (.) après la variable enregistrement, suivi du nom du champ.

A.4 Fichiers

Les types de donnée pour représenter des variables assignées à des fichiers sont :
— **Text** : pour des fichiers texte
— **File of** type de donnée : pour des fichiers typés sur un type de donnée précis

L'assignation d'une variable de type fichier se fait avec l'instruction **assign** prenant 2 paramètres : la variable de type fichier et une chaîne de caractères représentant le chemin d'accès vers un fichier.

Selon le mode d'ouverture souhaité du fichier, il faut utiliser une des instructions suivantes :
— **reset** : pour une ouverture en lecture
— **rewrite** : pour une ouverture en écriture qui créé un fichier vide
— **append** : pour une ouverture en écriture d'un fichier existant à compléter (uniquement pour les fichiers textes)

Chacune de ces instructions prend un seul paramètre : une variable de type fichier.

Les opérations d'écritures ou de lecture se font respectivement par les instructions **read** et **write** prenant 2 paramètres : une variable fichier puis la variable à modifier par la lecture du fichier ou la valeur à écrire dans le fichier. Pour le cas de fichiers texte les instructions **readln** et **writeln** existent pour prendre en compte un saut de ligne après l'opération de lecture/écriture.

La fermeture d'un fichier se fait avec l'instruction **close** prenant 1 seul paramètre : la variable de type fichier.

Exemple A.7 L'exemple ci-dessous est un programme complet qui lit intégralement un fichier contenant des entiers et les affiche à l'écran :

```
program lecture;

var fichier : File of Integer; var nb : Integer;
begin
   assign(fichier,'unFichier.dat');
   reset(fichier);
   while not(eof(fichier)) do
      begin
         read(fichier,nb);
         writeln(nb);
      end;
   close(fichier);
end.
```

Exemple A.8 L'exemple ci-dessous est un programme complet qui créé un nouveau fichier texte et le remplit avec des phrases donnant le factoriel des entiers allant de 1 à 7 :

```
program ecriture;

var fichier : Text; var i,fact : integer;
begin
   assign(fichier,'unFichier.txt');
   rewrite(fichier);
   fact := 1;
   for i := 1 to 7 do
     begin
       fact :=fact *i;
       write(fichier,'factoriel de ');
       write(fichier,i);
       write(fichier,' = ');
       writeln(fichier,fact);
     end;
   close(fichier);
end.
```

A.5 Unités

Le code source d'une unité en Pascal doit se situer dans un fichier d'extension .pas ayant le même nom que l'unité.

L'écriture de ce code doit respecter la structure suivante :

```
unit nom_de_l'unite;

interface

{declaration des types, constantes et signatures de
 sous-programmes utilisables par les programmes autres que
 l'unite}

implementation

{declaration des types, constantes, ... uniquement visibles dans
 l'unite}

{implementation des sous-programmes}
end.
```

Exemple A.9 L'exemple ci-dessous est une unité fournissant un type de donnée et 2 fonctions pour manipuler des dates

```
unit dates;

interface

Type Jour = 1..31;
```

```
function estBissextile (annee : integer): boolean;
function estUneDateValide(jour:Jour;mois,annee:integer):boolean;

implementation

{renvoie vrai si l'annee designee en parametre d'entree
est bissextile, faux sinon}
function estBissextile (annee : integer): boolean;
begin
   estBissextile := ((annee mod 400) = 0) or
   (((annee mod 4) = 0) and ((annee mod 100) <> 0));
end; { estBissextile }

{renvoie vrai si les parametres d'entree correspondent a
une date valide, faux sinon}
function estUneDateValide(jour:Jour;mois,annee:integer):boolean;
var valide :  boolean;
begin
   valide := true;
   case mois of
     1,3,5,7,8,10,12 : valide := true;
     4,6,9,11        :
     if (jour <> 31) then valide := true;
     2 :
         if (jour < 29) then valide := true
         else if (estBissextile(annee)) and (jour = 29) then
            valide := true
         else
            valide :=false;
   end; { case }
   estUneDateValide := valide;
end; { estUneDateValide }

end.
```

Pour qu'un programme ou une unité utilise une autre unité, il faut l'indiquer avec le mot-clé **uses** suivi du nom de l'unité utilisée, juste après la déclaration du nom du programme ou de l'unité utilisatrice.

Exemple A.10 L'exemple ci-dessous est un programme utilisant l'unité `dates`

```
program bissextile;

uses dates;

var i : integer;
begin
```

```
   for i := 1900 to 2013 do
      if (estBissextile(i)) then
         writeln(i, ' est bissextile');
end.
```