Bases du langage Pascal

Un programme doit en général recevoir des données, les traiter, puis fournir le résultat du traitement effectué.

A - Les variables

Les données sont représentées par des variables qui ont un nom, un type et un contenu.

1) Nom

Le nom d'une variable est un identificateur qui est une suite de caractères respectant les règles suivantes :

- il commence par une lettre (a..z ou A..Z), les caractères suivants sont d'autres lettres, ou des chiffres ou le caractère souligné (_).
- il n'y a pas de distinctions entre majuscules et minuscules
- la longueur n'a pas d'importance
- un identificateur ne peut pas être un des mots clés du langage.

2) Type

Le type d'une variable indique la nature des données. Les types élémentaires suivants sont prédéfinis :

- nombres entiers : selon le nombre d'octets utilisés et le caractère signé ou non signé on distingue les types **byte** (entier de 0 à 255 codé sur 1 octet), **integer** (entier de -32768 à 32767 codé sur 2 octets), **word** (entier de 0 à 65535 codé sur 2 octets) et **longint** (entier de -2147483648 à 2147483647 codé sur 4 octets).
- nombres en virgule flottante : type **real** (codé sur 6 octets) donnant 11 chiffres significatifs et des exposants compris entre 10^{-38} et 10^{38}.
- caractères et chaînes de caractères : type **char** (codé sur 1 octet) pour un seul caractère, type **string** (codé sur 256 octets) pour les chaînes de caractères qui sont limitées à 255 caractères.
- valeurs booléennes : type **boolean** (codé sur 1 octet)

3) Contenu

Le contenu d'une variable est une suite d'octets qui sera interprétée selon son type. On peut le fixer en suivant les règles suivantes :

- nombres entiers : même écriture qu'en mathématiques
- nombres en virgule flottante : notation décimale classique avec le point comme séparateur ou notation scientifique avec le caractère E séparant la mantisse et l'exposant.
- caractères ou chaînes de caractères : placés entre des apostrophes; si une chaîne de caractères doit contenir une apostrophe, celle-ci doit être doublée.
- valeurs booléennes : true et false

B - Instruction d'affectation

Pour définir le contenu d'une variable on utilise le symbole **:=** suivant le schéma :

```
nom_de_variable :=  contenu;
```

où *contenu* est une expression du même type que la variable.

Pour les variables numériques entières on peut utiliser des expressions mathématiques formées avec les opérateurs +, - , *, div et mod, des nombres entiers, des variables numériques de type nombre entier et des parenthèses.

Pour les variables numériques en virgule flottante, on peut utiliser des expressions mathématiques formées avec les opérateurs +, -, * et /, des nombres (entiers ou en virgule flottante), des variables numériques, des parenthèses et les fonctions sqrt, ln, exp, sin, cos, arctan, round. La constante pi représente le nombre π.

C - Instructions d'entrées et sorties

1) Procédures write et writeln

Ces procédures permettent d'afficher des messages à l'écran. Elles suivent le schéma :

```
writeln(element1, element2, ...).
```

Le nombre d'éléments à afficher est quelconque. Chaque élément est une expression qui est évaluée, et c'est le résultat qui est affiché. Les éléments s'affichent les uns derrière les autres sans espacement sur une même ligne. Avec l'instruction **writeln** un passage à la ligne sera effectué lors du prochain affichage. L'instruction **write** permet d'éviter ce passage à la ligne.

Les chaînes de caractères et les nombres entiers sont affichés de façon naturelle.
Les valeurs numériques en virgule flottante sont affichées par défaut sous forme scientifique.
Pour obtenir la notation décimale classique on utilise le schéma writeln(element:largeur:chiffres)où *largeur* indique le nombre total de caractères à utiliser (ou 0 pour un affichage optimisé de la partie entière) et *chiffres* indique le nombre de chiffres à faire apparaître après la virgule.
Par exemple :
- `writeln(12.5)` donnera 1.2500000000E001,
- `writeln(12.5:0:2)` donnera 12.50.

2) Procédure readln

Cette procédure permet de recueillir les entrées clavier de l'utilisateur et des les affecter à des variables. Elle suit le schéma :

```
readln(nom_variable);
```

L'utilisateur doit valider sa saisie en appuyant sur la touche entrée.

D - Structure d'un programme élémentaire

Un programme simple se divise en trois parties :
- un entête introduit par le mot clé **program** et donnant un nom au programme.
- une partie contenant les déclarations des divers éléments qui seront utilisés, en particulier les variables introduites par le mot clé **var**. On déclare une variable en donnant son nom et son type séparés par le symbole : . Attention, au moment de sa déclaration une variable a un contenu aléatoire; c'est le programme qui doit lui attribuer une valeur initiale.
- un corps de programme constitué par une suite d'instructions précédée par le mot clé **begin** et terminée par le mot clé **end** suivi d'un point final.

Les différentes déclarations et instructions sont séparées par des points virgules.
Exemple de programme calculant l'aire d'un carré.

```
program aire_carre;
var
 cote, aire : real;
begin
 write('Longueur du côté : '); readln(cote);
 aire := cote * cote;
 writeln('L''aire est ',aire:0:2);
end.
```

E - Instructions conditionnelles

1) Instruction if

L'instruction **if** permet de n'exécuter une partie de programme que si un condition est vérifiée.
Elle suit l'un des schémas suivants :

```
if expression then instruction;
```

ou

```
if expression then instruction1 else instruction2;
```

Les instructions sont soit une instruction simple, soit une suite d'instructions formant un bloc délimité par les mots clés **begin** et **end**.
L'expression est une expression booléenne, c'est à dire une expression ne pouvant prendre que les valeurs true et false.

2) Expressions booléennes

On forme des expressions booléennes simples en utilisant les opérateurs de comparaison <, <=, >, >=, = et <>. (attention à ne pas confondre l'opérateur de comparaison = et le symbole de l'affectation :=)
On dispose aussi des opérateurs booléens **and**, **or** et **not** pour former des expressions booléennes composées; attention il est nécessaire de placer les opérandes entre parenthèses s'ils sont formés à l'aide des opérateurs de comparaison.

F - Boucles

On utilise les boucles pour exécuter plusieurs fois une instruction ou un bloc d'instructions délimitées par les mots clés **begin** et **end**.

1) Boucle for

Elle utilise l'un des schémas :

```
for compteur:=val1 to val2 do instruction;
```

ou

```
for compteur:= val2 downto val1 do instruction;
```

où :

- *compteur* est une variable de type nombre entier qui est augmentée (to) ou diminuée (downto) d'une unité à chaque passage dans la boucle.
- *val1* et *val2* sont les valeurs initiales et finales de compteur
- *instruction* est une instruction simple ou un bloc d'instructions.

2) Boucle while

Elle utilise le schéma :

```
while expression do instruction;
```

où :

- *expression* est une expression booléenne
- *instruction* est une instruction simple ou un bloc d'instructions qui est exécuté tant que *expression* vaut true.

3) Boucle repeat

Elle utilise le schéma :

```
repeat
 instruction1;
 instruction2;
 ...
until expression;
```

Les différentes *instructions* sont répétées jusqu'à ce que l'expression booléenne *expression* prenne la valeur true.

G - Procédures et fonctions

On crée des procédures ou des fonctions lorsqu'on veut exécuter un même ensemble d'instructions à divers emplacements d'un programme.

Une procédure est une suite d'instructions qui forme une nouvelle instruction.

Une fonction est une suite d'instructions qui fournit un résultat qui pourra être utilisé dans une expression.

Procédures et fonctions peuvent utiliser d'une part des paramètres et d'autre part des variables locales.

Les procédures et les fonctions doivent être déclarées avant le corps du programme en utilisant les structures suivantes.

1) Procédure

```
procedure nom_procedure(param1: type1; param2 : type2; ...);
var
  nom_var1 : type_var1;
  nom_var2 : type_var2;
  ...
begin
  instruction1;
  instruction2;
  ...
end;
```

On utilise une procédure comme une instruction simple. On l'appelle en donnant son nom suivi des valeurs à utiliser pour les différents paramètres entre parenthèses et séparés par des virgules. On dit qu'il y a passage de paramètre par valeur.

Les variables déclarées dans la procédure sont locales, elles n'ont de sens que pendant l'exécution de la procédure.

2) Fonctions

```
function nom_fonction(param1: type1; param2 : type2; ...) : type-resultat;
var
  nom_var1 : type_var1;
  nom_var2 : type_var2;
  ...
begin
  instruction1;
```

```
    instruction2;
    ...
    nom_fonction:=resultat;
  end;
```
L'une des instructions doit indiquer le résultat renvoyé par la fonction; elle prend la forme :
```
    nom_fonction:=resultat;
```
On utilise une fonction comme une expression ayant le type du résultat. On l'appelle en donnant son nom suivi des valeurs à utiliser pour les paramètres entre parenthèses et séparés par des virgules.
Comme pour les procédures, les variables déclarées dans la fonction sont locales.

3) Passage de paramètre par adresse

Lorsqu'une procédure ou une fonction doit modifier le contenu de variables préalablement définies, on procède à un passage de paramètre par adresse. Cela signifie que le paramètre ne représente plus une simple valeur, mais une variable elle-même (c'est en fait l'adresse mémoire de la variable qui est fournie). Dans ce cas la déclaration du paramètre doit être précédée du mot clé **var**.

H - Tableaux et enregistrements

Pour traiter des données complexes on peut définir de nouveaux types de variables sous forme de tableaux ou d'enregistrements.

1) Tableaux

Un tableau est une suite indicée de données de même type. Celles-ci sont rangées en mémoire de façon contiguë ce qui permet d'y accéder simplement à partir de leur numéro d'ordre.
Pour utiliser des variables de type tableau, on commence par déclarer le type correspondant (dans la partie déclarations du programme, avant les variables). Cette déclaration utilise le mot clé **array** et prend la forme suivante :
```
  type
    nom_type = array[min..max] of type_var;
```
où
min et *max* sont les valeurs extrêmes des indices des éléments du tableau; *max*-*min*+1 donne le nombre d'éléments du tableau; *type-var* est le type des éléments contenus dans le tableau.
Lorsqu'un type tableau a été défini, on peut déclarer des variables de ce type.
Pour avoir accès à un élément du tableau, il suffit d'écrire le nom de la variable suivi de l'indice de l'élément visé entre crochets. Cet accès permet à la fois de lire et d'écrire cet élément.

2) Enregistrements

Il est parfois utile de regrouper des variables de types éventuellement différents. Ceci se fait grâce à la notion d'enregistrement. Un enregistrement contiendra donc plusieurs données qu'on appellera champs. Ceux ci seront inscrits en mémoire de façon contiguë. Pour utiliser des variables de type enregistrement, on commence par déclarer le type correspondant dans la partie des déclarations. Cette déclaration utilise le lot clé **record** et prend la forme suivante :
```
  nom_type = record
   champ1 : type1;
   champ2 : type2;
   .....
  end;
```
Par exemple, pour créer un type associant le nom d'une personne et sa date de naissance on pourra déclarer :
```
  anniversaire = record
   nom : String;
   jour : Byte;
   mois : Byte;
   an : Word;
  end;
```
Lorsqu'un type enregistrement a été défini, on peut déclarer des variables de ce type. Pour avoir accès à un champ d'une variable de ce type on écrit le nom de la variable suivi d'un point et du nom du champ visé. Cet accès permet à la fois de lire et d'écrire ce champ.
Par exemple, si unAmi est une variable du type anniversaire précédemment défini, on aura accès au nom en écrivant unAmi.nom.