Variables et types de données

1 Motivation

1.1 Un problème

- État initial : Tas 1 un nombre quelconque de cartes, les autres tas vides.
- État final : peu importe
- **But :** afficher le nombre de cartes situées initialement sur le tas 1.

1.2 Algorithme

Une solution du problème consiste à déplacer (une à une) toutes les cartes du tas 1 vers un autre tas (le tas 2 par exemple), et à compter chaque déplacement effectué.

```
mise à zéro du compteur c

tant que le tas 1 n'est pas vide faire
  déplacer la carte au sommet du tas 1 vers le tas 2
  ajouter 1 au compteur
fin tant que
```

2 Types de données

En programmation, les données manipulées peuvent être de natures différentes : nombres entiers, nombres réels, chaînes de caractères, booléens, ... Ces différentes natures de données sont appelées *types de données*.

Le langage Pascal dispose d'un certain nombre de types prédéfinis.

2.1 Les booléens

Le type `BOOLEAN` est le type des expressions booléennes permettant d'écrire les conditions des instructions **`if`** et **`while`**. Les données de ce type ne peuvent prendre que deux valeurs.

Nom : `BOOLEAN`

Ensemble des valeurs : **`true`**, **`false`**

Littéraux : **`true`** et **`false`**

Opérateurs : **`and`**, **`or`**, **`not`**

2.2 Les nombres entiers

Free Pascal permet de manipuler plusieurs types d'entiers qui diffèrent entre eux par l'intervalle représenté. Nous ne citerons que deux d'entre eux : les types `CARDINAL` et `INTEGER`.

Le type CARDINAL Ce type permet de manipuler des nombres entiers positifs ou nuls.

Nom : `CARDINAL`

Ensemble des valeurs : entiers n non signés sur 32 bits : $0 \leq n \leq 2^{32} - 1 = 4294967295$

Littéraux : les entiers dans leur forme écrite usuelle. Par exemple : `24`.

Opérateurs arithmétiques : `+`, `-`, `*`, **`div`**, **`mod`**

Opérateurs de comparaison : `=`, `<>`, `<`, `<=`, `>`, `>=`

Remarque : le type `CARDINAL` est limité, il y a un plus grand entier : $2^{32} - 1 = 4294967295$. Si on ajoute 1 à cet entier, le résultat n'est pas $2^{32} = 4294967296$ mais 0. L'arithmétique sur les entiers de type `CARDINAL` doit être comprise modulo 2^{32}.

Le type INTEGER Ce type permet de manipuler des nombres entiers aussi bien positifs que négatifs.

Nom : `INTEGER`

Ensemble des valeurs : entiers n signés sur 32 bits[1] : $-2^{31} = -2147483648 \leq n \leq 2^{31} - 1 = 2147483647$

Littéraux : les entiers dans leur forme écrite usuelle. Par exemple : `-24`.

Opérateurs arithmétiques : `+`, `-`, `*`, **`div`**, **`mod`**

Opérateurs de comparaison : `=`, `<>`, `<`, `<=`, `>`, `>=`

2.3 Types définis par l'unité cartes

L'unité cartes définit deux types `COULEURS` et `TASPOSSIBLE`.

Le type COULEURS Les données de ce type ne peuvent prendre que quatre valeurs notées : `CARREAU`, `TREFLE`, `PIQUE` et `COEUR`.

Un appel à la fonction `couleurSommet` donne une valeur de ce type.

Le type TASPOSSIBLES Les données de ce type ne peuvent prendre que quatre valeurs : 1, 2, 3 et 4. Ce type représente les numéros des tas.

Lorsqu'on fait appel à la procédure `DeplacerSommet`, on lui passe deux données de ce type.

3 Variables

3.1 Notion de variable

Les variables servent à nommer les objets.

Elles sont caractérisées par leur nom ou *identificateur* et leur *type*.

3.2 Déclaration de variables

La déclaration des variables utilisées par un programme, une procédure ou une fonction se fait à l'aide du mot réservé **`var`**

```
var
  <identificateur> : <type>;
```

où `<identificateur>` est le nom de la variable, et `<type>` son type.

Les déclarations de variables se font dans la partie déclaration des programmes[2].

Exemple : Voici la déclaration d'une variable nommée `compteur` et de type `CARDINAL`

```
var
  compteur : CARDINAL;
```

3.3 Affectation

L'*affectation* est une instruction qui permet d'attribuer une valeur à une variable.

La syntaxe de cette instruction est

```
  <id> := <expr>
```

Le symbole `:=` signifie que la valeur v de l'expression `<expr>` est affectée à la variable désignée par l'identificateur `<id>`. Après l'exécution de cette instruction la variable `<id>` vaut v.

[1] dans le mode ObjFPCou Delphi uniquement, sinon ces entiers sont codés sur 16 bits

[2] et plus tard des procédures et fonctions

Exemples :

1. Pour mettre à zéro la variable `compteur` précédemment déclarée

```
{compteur =??}
compteur := 0 ;
{compteur = 0}
```

2. ajouter 1 au compteur

```
{compteur = n}
compteur := compteur + 1 ;
{compteur = n + 1}
```

Remarques :

1. Attention à ne pas confondre le symbole `:=` utilisé en PASCAL pour l'instruction d'affectation, avec le symbole `=` opérateur de comparaison.
2.

4 Afficher des données

En PASCAL, lorsqu'on veut afficher des données, on utilise l'instruction **`write`** ou bien **`writeln`**.

La différence entre ces deux instructions réside dans le passage à la ligne ou non après l'affichage des données : **`writeln`** provoque un passage à la ligne, **`write`** n'en provoque pas.

On utilise l'une ou l'autre de ces deux instructions en mettant entre parenthèses les données à afficher séparées par des virgules.

Exemples :

1. affichage du message `nbre de cartes` (notez les apostrophes autour du message) sans passage à la ligne

```
write('nbre␣de␣cartes␣=') ;
```

2. affichage de la valeur du compteur (notez l'absence d'apostrophes) avec passage à la ligne

```
writeln(compteur) ;
```

3. bien entendu il est possible de mettre les deux instructions l'une à la suite de l'autre

```
write('nbre␣de␣cartes␣=') ;
writeln(compteur) ;
```

et on obtient alors l'affichage (si le compteur vaut 10) :

```
nbre de cartes =10
```

Notez l'absence d'apostrophes à l'affichage, et l'absence d'espace entre = et 10.

4. On peut réunir les deux instructions de l'exemple précédent en une seule

```
writeln('nbre␣de␣cartes␣=',compteur) ;
```

Remarques :

1. Si on veut afficher une apostrophe, il faut la doubler :

```
writeln('L''UE␣InitProg␣est␣passionnante␣!') ;
```

2. On ne peut pas afficher tout type de données. C'est le cas du type `COULEURS`. Par exemple, lors de la compilation du programme

```
program essai ;
uses cartes;
var
  x : COULEURS;
begin
  x := PIQUE;
  writeln(x);
end.
```

on obtient le message d'erreur

```
essai.pas(7,12) Error: Can't read or write variables of this type
```

qui signale l'impossibilité d'afficher la valeur de la variable x.

5 Le programme solution du problème introductif

```
// auteur : EW
// date   : sept 05
// objet  : compter les cartes du tas 1
program compter;

uses
  cartes;

  var
    compteur : CARDINAL;

begin
  // Initialisation des tas
  InitTas(1,'[T]');

  // initialisation du compteur
  compteur := 0;

  {Soit N le nbre initial de cartes sur le tas 1
   et n1 le nombre actuel de cartes sur le tas 1
   on a n1 + compteur = N }
  // action : vider le tas 1 en comptant les cartes
  while tasNonVide(1)  do
  begin
     { n1 + compteur = N}
     DeplacerSommet(1,2);
     { n1 + compteur = N - 1}
     compteur := compteur + 1;
     { n1 + compteur = N}
  end {while};
  { n1 = 0, n1 + compteur = N}
  {donc compteur = N}

  // affichage du nombre de cartes
  writeln('nbre de cartes = ', compteur);
end.
```

6 Exercices

Exercice 1. *Échange de variables*

Donnez une séquence d'instructions qui échange les valeurs de deux variables numériques.

Exercice 2. *Afficher une table de vérité*

Réalisez un programme qui affiche la table de vérité de l'opérateur **and**.

Exercice 3. *Limitation des entiers*

Question 1. Réalisez un programme qui affiche le carré du nombre entier 1234.

Question 2. Modifiez votre programme afin qu'il affiche le carré de 123456. Expliquez l'affichage obtenu.

Exercice 4. *Calcul de la valeur d'une fonction polynomiale*

Soit $f(x) = x^4 - 3x^3 - 2x^2 + x + 1$.

Réalisez un programme qui affiche la valeur de $f(x)$ lorsque $x = 13$. Concevez votre programme pour qu'il soit facile de le modifier pour le calcul de $f(x)$ pour d'autres valeurs de x.

Exercice 5. *Avec les cartes*

Réalisez des programmes qui, à partir de la situation initiale

Situation initiale :

Tas 1 : `'[K+T+P+C]'` Tas 2 : ”

Tas 3 : ” Tas 4 : ”

1. calcule le nombre de trèfles,
2. calcule le nombre de cartes de couleur noire,
3. répartit équitablement les cartes rouges sur les tas 3 et 4, et les cartes noires sur les tas 1 et 2.