Réalisation d'une carte de la constellation ORION en utilisant le langage de programmation ADA

I. Pour comprendre de quoi il est question

L'extrait de carte ci-contre représente la constellation Orion contenue dans le catalogue de l'astronome Hevelius (1611-1687) .
Auteur d'un catalogue de **1564** étoiles, il avait gravé (ou faire graver) des planches de cartes pour les joindre à l'édition.
A cette époque, tout se faisait à la main (si ce n'est les calculs)...

A l'instant présent, des catalogues de plusieurs millions d'étoiles, résultant d'observations au sol et de missions spatiales, sont accessibles via le réseau internet.

La puissance de calcul des ordinateurs personnels permet de réaliser « par programme » des cartes de champs stellaires d'une grande précision. De tels programmes peuvent être achetés voire utilisés gratuitement.
Ceux qui sont plus attachées au résultat qu'à la façon dont on l'atteint ont ainsi accès à cet aspect de la connaissance.

Pour qui est intéressé par le « comment ? », les pages qui suivent proposent le compte-rendu d'une démarche faite de petits pas et d'expériences, d'avancées parfois minuscules, mais qui importent pour qui a décidé d'avancer dans la direction d'un but fixé.

L'auteur, initié à la programmation, mais qui est débutant complet dans le langage ADA, a choisi ce thème de travail pour le « faire fonctionner » et en comprendre les avantages, les contraintes.
Même si ce document n'a pas vocation à servir de support pédagogique pour le langage, il permettra, à travers la lecture de commentaires sur un code de débutant, de présenter ADA en action.
Le format graphique SVG, autre sujet d'intérêt personnel, est utilisé dans la « construction » des cartes.

I.1. Matériel de base, références

- Un fichier de données, au format texte comprenant 1628 étoiles, de magnitudes inférieures à 5.
- Le compilateur GNAT dans la version accessible librement pour Linux ou autres systèmes.
- Un éditeur de texte. Un grand choix est possible ici et chacun a ses préférences. J'ai choisi de conserver Geany, déjà expérimenté avec les langages Python, C et Java.
- Les documents de référence sont ceux qui sont accessibles, en français et en anglais, sur internet.

I.2. Le choix du format de représentation SVG

Le langage ADA n'a pas de bibliothèque graphique propre, aussi a-t-on recherché des bibliothèques graphiques libres, utilisables par l'intermédiaires d'une mise en relation avec ce langage (« binding » en anglais). **Plplot**, **Cairo** et **GTK+ ont été explorées brièvement.**
Plplot est la plus simple. Cairo et GTK+ nécessitent un apprentissage difficile avant tout travail effectif.
Un autre aspect à prendre en compte : si le lecteur souhaite expérimenter le code présenté ici, il faut lui

permettre de se contenter de n'installer que « l'ADA libre de base » et pas l'embarquer dans des installations avec des paramétrages complexes.
Ces raisons ont poussé à s'intéresser au format de réprésentation graphique SVG.

I.3. Réaliser un document SVG

Réaliser un document SVG, c'est écrire dans un fichier de texte, ce qu'ADA fait aussi simplement que d'autres langages de programmation tels que C, Basic, Java ou Python.
Pour **visualiser** le document obtenu, il faut l'ouvrir avec un **navigateur** récent (Firefox, Chromium ...).
Le format SVG présente plusieurs avantages supplémentaires :

- Le document est « zoomable » sans perte de qualité.
- Si le contenu est simple (et c'est le cas des cartes de constellation) les fichiers obtenus sont petits et affectent peu la bande passante du réseau.

Compte tenu de ses avantages, le format SVG sera utilisé pour réaliser les cartes.

I.4. Le catalogue utilisé

Il fallait un catalogue suffisamment réduit, mais comportant un nombre d'étoiles approchant de celui qui est visible dans des conditions moyennes. Dans les banlieues pavillonaires par exemple, on n'atteint pas souvent la magnitude 5. Au cœur des villes, la multiplication des éclairages réduit plus encore le nombre d'étoiles visibles.

Le « Catalogue des étoiles les plus brillantes » est accessible à cette URL :
ftp://cdsarc.u-strasbg.fr/cats/V/53A/
Sous la forme d'un fichier texte, il contient **1628** étoiles (ce qui se rapproche du catalogue d'Hevelius), de magnitudes comprises entre -1,44 (Sirius) et 5.
Comme il s'agit d'un fichier texte, il est constitué d'un ensemble de lignes (1628) ayant toutes la même longueur et la même organisation de l'information.
Par contre, il ne contient ni amas globulaire ni nébuleuse et ne contient rien qui permettrait de tracer les lignes imaginaires qui représentent les « personnages » de chaque constellation.
La description complète du contenu de ce fichier figure à la fin de ce document (**Annexe 1**).

II. Première lecture du fichier de données

II.1. Cahier des charges :

Ouvrir le catalogue et lister à l'écran :

- le nom la lettre grecque désignant l'étoile (quand il y en a un), sous la forme d'une abréviation de trois lettres,
- le nom de la constellation,
- l'ascension droite,
- la déclinaison (positive au Nord et négative au Sud),
- la magnitude,

pour les étoiles dont l'ascension droite est comprise entre 5 et 6 heures ET (en même temps) la déclinaison comprise entre + 13 degrés et =13 degrés ET ayant une magnitude < à 4.

II.2. Remarques sur le code en langage ADA :

(*Le code complet, très court, est joint en* ***annexe 2***).
Les explications données ici ne peuvent se substituer à un vrai cours sur le langage, mais peuvent aider à retrouver la logique de celui-ci.

Code	**Commentaires**
`Ligne_fichier  : String (1 .. 280);`	Déclaration d'une chaîne destinée à contenir chaque ligne lue dans le fichier de données.

`Open (F,In_File,"catalog.dat" );`	Ouverture d'un fichier texte en lecture
`Get_line(F,Ligne_fichier,lg);`	Lecture d'une ligne du fichier et chargement dans la variable définie plus haut
`Ligne_fichier(39..48)...Ligne_fichier(51..58)...`	Contiennent ascension droite et déclinaison (sans le signe, inutile ici)
`if Ligne_fichier(172..176) < " 4.00" ...`	Test sur la magnitude
`Put(Ligne_fichier(23..25) & " ");`	Affiche le nom de la constellation
`Put(Ligne_fichier(50..50)) ;` (à retenir!!!)	Affiche le signe seul. Comme il est codé sur un seul caractère, la valeur de début dans la chaîne est la même que la valeur de fin.

Résultat obtenu :

```
bet  Ori   5 14 32.2  - 8 12  6 - mag:  0.13
tau  Ori   5 17 36.3  - 6 50 40 - mag:  3.59
eta  Ori   5 24 28.6  - 2 23 49 - mag:  3.35
gam  Ori   5 25  7.8  + 6 20 59 + mag:  1.64
...
```

Commentaires :

Code court et efficace. Le décodage du signe sur un seul octet (en gras plus haut) n'était pas évident à trouver. Retenir le codage des lettres grecques sur trois lettres.

II.3. Considérations méthodologiques :

Faute de maîtriser à l'instant l'ensemble des connaissances qui permettraient la réalisation du document final, il a été décidé de découper le traitement en plusieurs étapes, enregistrant à chaque fois le fruit du travail dans un fichier intermédiaire.

III. Création d'un fichier intermédiaire

III.1. Cahier des charges :

Lire certaines zones du catalogue et les écrire, modifiés ou non, dans un fichier séquentiel.

III.2. Remarques sur le code en langage ADA :

```
type Rec_Etoile is   record
      Lettre_Grecque : String(1..4) ;
      Constellation  : String(1..3) ;
      AD_secondes    : float ;
      Decl_secondes  : float ;
      Magnitude      : float ;
   end record ;
```

Les données que l'on souhaite sélectionner dans le fichier sourcevont être écrites dans une succession d'enregistrements de structure identique.
Pour le langage ADA (et d'autres langages), c'est un « Record » (un enregistrement = une fiche).

Ce que l'on définit ainsi, c'est une sorte de matrice à dupliquer, appelée **type**. Pour pouvoir l'utiliser, la dupliquer autant que nécessaire, il faut l'associer à une **variable**, qui pourra - elle - recevoir ou restituer les données.

Le programme ajoute de nouvelles notions, qui sont un peu détaillées ci-dessous.

Code	**Commentaires**
`Etoile   : Rec_Etoile ;`	Une **variable**, appelée Etoile est associée au **type** défini plus haut.

`package Fichier_IO is new ada.sequential_io(Rec_Etoile) ;` `use Fichier_IO ;`	Très important et spécifique. Une bibliothèque mère existe pour les fichiers séquentiels « en général ». En initialisant une copie de cette bibliothèque avec le type de données à manipuler on passe du général au particulier.
`Catalogue_Etoiles : Fichier_IO.file_type ;`	Nom de la variable, permettant de manipuler le type de contenu défini plus haut.
`Create(Catalogue_Etoiles,out_file,"transformer.dat");`	Avant de pouvoir écrire dans le fichier, il faut lui ouvrir une porte sur notre disque.
`Hr := Float'value(Ligne_fichier(39..40));`	ADA oblige à faire un transtypage explicite. Ici on transforme un texte de 2 caractères en nombre « à virgule flottante » (flottant = float).
`Hr := Hr * 900.0 ;`	Une heure de l'ascension droite, c'est 15 degrés d'arc. Et 15 degrés, c'est 15 * 60 soit 900 minutes d'arc. Comme le nombre est un flottant, il faut ajouter le point et le zéro.
`if Ligne_fichier(50..50) = "-" then Etoile.Decl_minutes := Etoile.Decl_minutes * (-1.0); end if ;`	Dans le cas d'une déclinaison Sud, (reconnu par le signe « - »), il faut multiplier la valeur par -1 pour obtenir des valeurs négatives. La valeur négative doit être entre ( )

`write(Catalogue_Etoiles,Etoile);`
`…`
`Close(Catalogue_Etoiles);`
	Syntaxe pour le fichier séquentiel : écriture, fermeture.

III.3. Le programme en quelques mots :

Ouvre fichier texte source en lecture
Crée le fichier cible en écriture
Tant que pas arrivé à la fin du fichier source, répéter
    Lit une ligne du fichier source
    Remplit la fiche « virtuelle » (avec les données récupérées puis transformées)
    Ecrit la fiche « virtuelle »dans le fichier cible (elle devient une fiche réelle)
    Passe à la ligne suivante dans le fichier source
On retourne au « tant que » si le fichier source n'est pas fini. Sinon on passe au traitement décrit ci-dessous.
Fermer les deux fichiers.
C'est fini.
Code de l'ensemble du programme en ***annexe 3***

Commentaires :

Le programme fonctionne. La compréhension de la syntaxe permettant de « décliner » un « paquetage » a nécessité du temps pour être comprise et acceptée. C'est un des mécanismes importants du langage et il faudra se familiariser avec. Le résulat souhaité a été atteint.

IV. Contenu minimal d'un fichier SVG.

Le fichier devra comporter au moins trois lignes.
En voici un exemple :

Code	Commentaires
`<?xml version="1.0" encoding="iso-8859-1"?>`	Le format SVG est un sous-ensemble du format XML. On précise ici le jeu de caractères adapté au monde occidental.
`<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">`	Début de la partie SVG. L'image sera encadrée.
….	Contenu utile variable.
`</svg>`	Fin du fichier SVG

Contenu utile : quelques exemples de figures dessinées en SVG

Code	Commentaires
`<text x="0" y="60" fill="red"> 800</text>`	Positionne un texte au point de coordonnées X et Y. Le texte sera écrit en rouge selon la police et la taille standard.
`<line x1="50" y1="50" x2="400" y2="50" stroke="pink" stroke-width="2" />`	Trace un segment entre les points de coordonnées x1,y1 et x2,y2. Il sera rose et de 2 de large (Notion à préciser).
`<circle cx="200" cy="50" r="40" stroke="black" style="fill:blue; stroke:pink;stroke-width:5; fill-opacity:0.5;stroke-opacity:0.9"/>`	Trace un cercle de centre x,y et de rayon r. Le contenu sera bleu, la circonférence rose, de largeur 5. Contenu et circonférence seront d'opacité partielle.
`<rect x="100" y="30" width="80" height="80" style="fill:red;stroke-width:2;stroke:black" />`	Trace un rectangle dont le coin haut gauche est en x,y de largeur et hauteur définies, rempli de rouge. Le contour de largeur 2 sera noir.

Difficulté particulière à la syntaxe SVG :

Les langages de programmation tels que ADA et d'autres permettent d'affecter des chaines de caractères à des variables. La syntaxe est en général :
`variable = « ma chaîne de caractères »`
En ADA, ce sera
`variable:= « ma chaine de caractères » ;`
Cette variable peut alors être écrite dans un fichier, textuel ou séquentiel. Dans le programme précédent, on a ainsi pu écrire le nom de la constellation dans l'enregistrement.
La difficulté apparaît quand on a des guillements dans la chaîne
`variable := <text x="0" y="60" fill="red"> 800</text>`
...génèrera un message d'erreur : on ne peut avoir des guillemets seuls au milieu d'une chaîne.

Pour résoudre la difficulté, plusieurs logiques peuvent être utilisées.

Ici, on a choisi de définir une variable qui ne contient qu'un seul guillement. Cela se fait de la façon suivante :
`guil : string(1..1) :=""";`
`1..1` signifie commence à 1 et finit à 1 = ne contient qu'un seul caractère.
`"""` pour entrer UN SEUL guillemet. Cela semble étrange. Mais c'est comme cela.
La chaîne va être constituée par concaténation (assemblage de morceaux) à l'aide du signe & . Exemple :
`variable := « <text x= » & guil & « 0 » & guil & «  y= « ….`
Ce n'est pas une partie de plaisir. Heureusement que c'est un programme qui va faire le travail.

V. Exemples de mise en œuvre des procédures

Dans ce qui suit, les chaînes de caractères sont écrites dans un fichier texte appelé `fichier` en utilisant des procédures qui acceptent des paramètres quand c'est nécessaire.

- Commençons par le plus simple : l'écriture de la fin du fichier SVG.

```
procedure fin is
   begin
      Put_line(Fichier,"</svg> " );
   end fin ;
```

L'instruction Put_line écrit une ligne dans le fichier de texte. Cette ligne contiendra `"</svg> "`

- Plus compliqué : l'en-tête du document XML :

```
 Procedure en_tete is
   begin
      Put (Fichier,"<?xml version = ");
      Put(Fichier,guil); Put(Fichier,"1.0"); Put(Fichier,guil);
      Put(Fichier," encoding = ");  Put(Fichier,guil);
      Put(Fichier,"iso-8859-1"); Put(Fichier,guil);
      Put_line(Fichier," ?> ") ;
   end en_tete ;
```

Remarquer l'introduction d'un guillemet à la fois par `Put(Fichier,guil);`

- Beaucoup plus compliqué : générer un cercle (seul un extrait est donné ici mais le programme est en annexe).

```
 Procedure cercle(x0  : in integer;  y0 : in integer ;
                  r   : in integer ; couleur : in string ;
                  l   : in integer ; tour : in string ) is
   begin
      Put(Fichier,"<circle cx=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(x0))) ; Put(Fichier,guil);
      Put(Fichier," cy=");  Put(Fichier,guil);....
```

On remarque ici le passage des paramètres qui sont nécessaires.

- Exemple d'appel de cette procédure :

```
    cercle(x,y,d,"white",1,"black");
```

Ce qui est beaucoup plus facile que d'entrer le texte à la main.

VI. Et maintenant, un pas de géant : réalisation d'un paquetage

Dans les programmes suivants, on aura souvent à écrire dans des fichiers SVG. Ce serait une perte de temps que de devoir reproduire dans chacun le code qui permet de tracer un cercle, un rectangle, d'écrire un texte. C'est la où un paquetage va être utile.
Dans les faits, ce sont des lignes de code compilées séparément qui contiennent des fonctions et procédures qui peuvent être appelées depuis un programme via l'instruction `with` .
Réaliser un paquetage est simple une fois que l'on a compris qu'il se compose de deux fichiers :

- un fichier avec l'extension ADS
- un fichier avec l'extension ADB, les deux ayant le même nom.

Par exemple fairesvg.ADS et fairesvg.ADB
Ces deux fichiers ont chacun leur spécificité.

extension	Commentaires
ADS	Contient les types et variables utilisés par le paquetage. Contient l'en-tête de toutes les fonctions et procédures du paquetage mais pas de code.
ADB	Contient tout le code.

Fichier ADS :

```
with Ada.Text_Io; use Ada.Text_Io;

package fairesvg is
   Fichier  : File_type ;
   guil : string(1..1) :="""; -- guillemet
   function eliminer(C:Character:=' ';T:String) return String ;
   procedure en_tete ;
   procedure cadre(w  : in integer ; h  : in integer ) ;
…
end fairesvg ;
```

Fichier ADB :

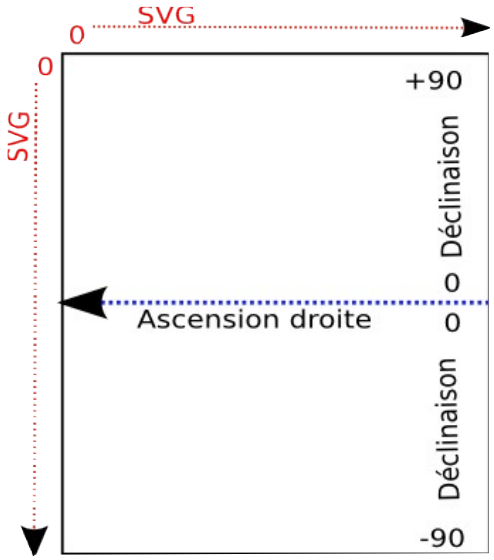
```
with Ada.Text_Io; use Ada.Text_Io;

package body fairesvg is
   Procedure en_tete is
   begin
      Put (Fichier,"<?xml version = ");  Put(Fichier,guil); Put(Fichier,"1.0");
….
   end en_tete ;
….
….
end fairesvg ;
```

Le code complet de fairesvg.ads est ***en annexe 4***
Le code complet de fairesvg.adb est ***en annexe 5***

VII. Le programme qui réalise la carte

Il est accessible en ***annexe 6***
Bien que court, il comporte quelques astuces qui seront mieux comprises avec le dessin ci-dessous.



Les coordonnées extraites du fichier sont l'ascension droite et la déclinaison.

L'ascension droite croît de droite à gauche, alors que la valeur x de l'image SVG croît de gauche à droite.

La déclinaison est égale à 0 sur l'équateur et elle est de +90 au pôle Nord et de -90 au pôle Sud.

Dans une image SVG, la valeur y augmente en continu depuis le coin haut gauche.
Les plus fortes valeurs positives seront « en haut ».Les plus fortes valeurs négatives seront « en bas ».

Code	Commentaire

`with Ada.Text_Io; use Ada.Text_Io;`
`with Ada.Sequential_IO ;`
`with fairesvg ; use fairesvg ;`
	Déclaration des ressources extérieures utilisées.
	En particulier le paquetage défini page précédente.

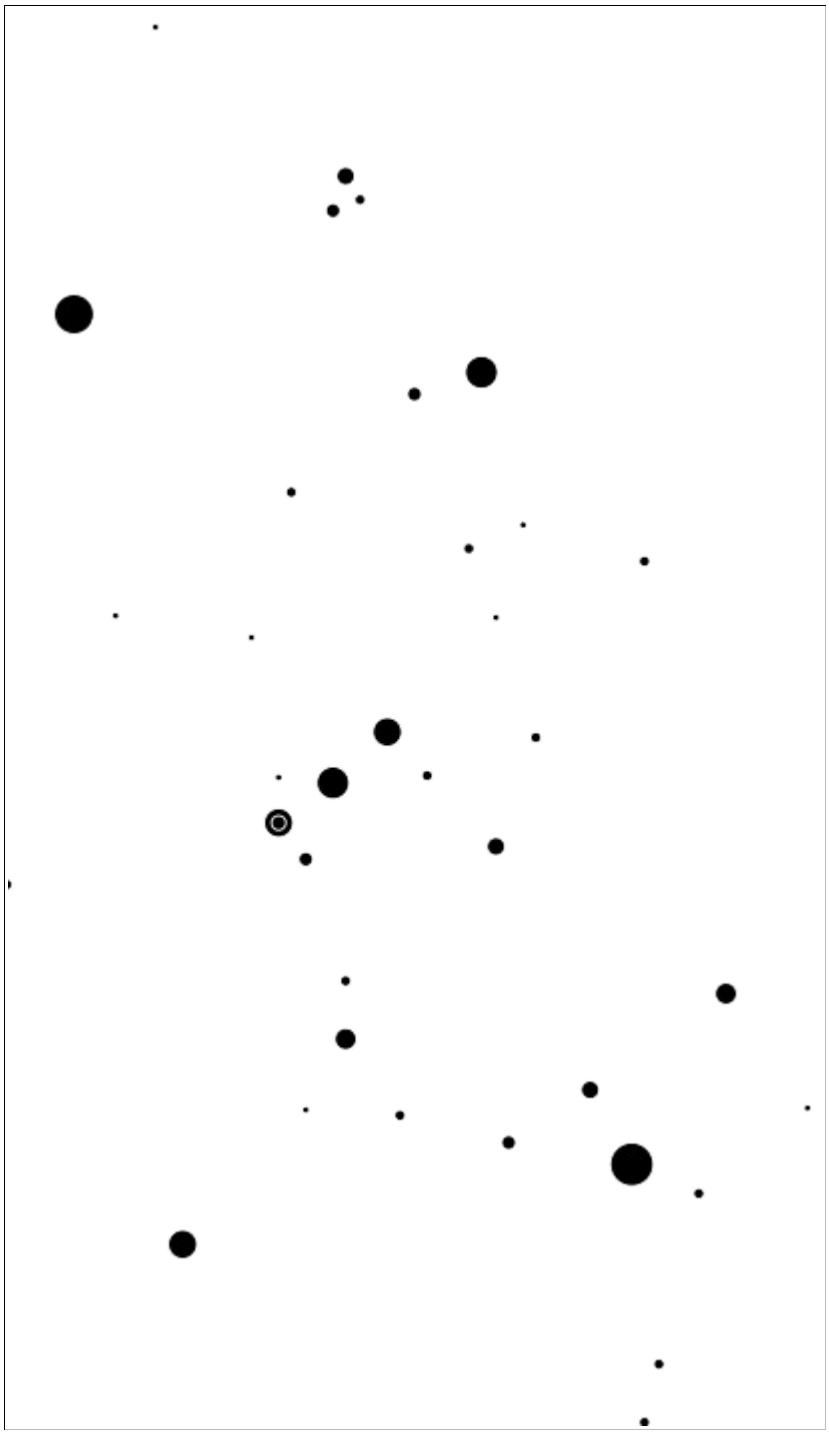
`x, y , d : integer ;`	Coordonnées et diamètre du cercle qui représente l'étoile.
`if etoile.ADs >= 4500.0 and etoile.ADs <= 5400.0 and etoile.Dcls > -780.0 and etoile.Dcls < 780.0 then`	Test : l'étoile entre-t-elle dans le cadre ? Si oui alors « Then... et la suite »
`prov :=((etoile.ADs-5400.0) * (-1.0));`	L'ascension droite va de « droite à gauche » alors que les coordonnées de la carte partent du point haut gauche. Il faut mettre la valeur dans le bon sens.
`x := integer(prov) /2 ;`	Division par deux = 2 minutes pour 1 pixel.

`if etoile.Dcls > 0.0`
` then`
`  prov := (etoile.Dcls - 780.0)* (-1.0) ;`
`  y := integer(prov)/2;`
` else`
`  prov := (etoile.Dcls * (-1.0)) + 780.0 ;`
`  y :=Integer(prov)/2;`
`end if ;`
	La déclinaison est positive ou négative, alors que la coordonnée y progresse depuis l'origine.
	Il faut effectuer une transformation d'une logique dans une autre.
	La division par deux a la même cause que plus haut.

`prov := (etoile.Mag -6.0) *(-2.0) ;`	Formule « au pif » pour définir le rayon du cercle.
`d := integer(prov);`	La valeur est transformée en nombre entier.
`cercle(x,y,d,"white",1,"black");`	Appel de la procédure de tracé.

Le résultat obtenu figure sur la page suivante.

Le résultat obtenu ;



ANNEXES

Annexe 1. Description du catalogue d'étoiles utilisé.

V/53A Catalogue of the Brightest Stars (Ochsenbein+ 1988)

```
================================================================
Le catalogue des étoiles les plus brillantes
Catalogue of the brightest stars
    Ochsenbein F., Halbwachs J.L.
   <Bull. Inform. CDS 32, 83 (1987)>
   =1987BICDS..32...83O
    Ochsenbein F., Acker A., Legrand E., Poncelet J.M., Thuet-Fleck E.
   <A.Acker (ed.), Strasbourg Obs. (1988)>
================================================================
```

ADC_Keywords: Stars, bright ; Combined data

Introduction au catalogue (version francaise):

Ce catalogue fournit des données sur les 1628 étoiles brillantes de magnitude allant jusqu'à 5.01 contenues dans le "catalogue des étoiles les plus brillantes" de F. Ochsenbein et co. (1984), lui-même dérive de la 4ème édition du "Bright Star Catalogue" (Hoffleit 1982). Il diffère de la version originale imprimée par les points suivants:

- les désignations des étoiles dans le catalogue Hipparcos, ainsi que les positions Hipparcos, exprimées dans l'International Celestial Reference System (ICRS), ont été rajoutées (A)
- les valeurs des parallaxes et mouvements propres ont été remplacées par les valeurs du catalogue Hipparcos pour la quasi-totalite des étoiles (A)
- les distances en années-lumières ont été calculées (A)
- les coordonnées écliptiques ont été ajoutées aux données astrométriques.
- la photométrie UBVRI a été extraite du catalogue de Lanz (1986) ; ceci explique que trois étoiles soient de magnitude superieure à 5.00 (HR 3229, HR 4392 et HR 6161). De plus, afin de faciliter les traitements par programme, on donne la valeur 99.99 en cas de mesure manquante (99.999 pour l'uvbyHbeta).
- la lettre grecque "delta", qui apparaissait dans les types spectraux sous la forme codée "<04>" est maintenant écrite en toutes lettres.
- une colonne donnant la classe de luminosite en chiffre arabe a été ajoutée a l'ecriture classique du type spectral.
- le nombre de composantes des étoiles doubles visuelles (2) est donne explicitement.

(A) indique les données ajoutées dans la version 'A'

File Summary:

FileName	Lrecl	Records	Explanations
ReadMe	80	.	This file
catalog.dat	278	1628	The Catalogue
names.dat	18	80	Star names / Noms des étoiles

See also:

V/50 : The Bright Star Catalogue, 5th Edition

I/239 : The Hipparcos and Tycho Catalogues (ESA 1997)

Byte-by-byte Description of file: catalog.dat

Bytes	Format	Units	Label	Explanations
1- 4	I4	---	HR	[1/9110]+ "Bright Star" catalog <V/50>
6- 11	I6	---	HD	[1/225300] Henry Draper" catalog <III/135>

```
 13- 15  I3     ---     Flamstéed  ? Flamstéed number
 18- 21  A4     ---      Bayer     Gréek letter (lettre grecque)
 23- 25  A3     ---      Const     Constellation (3 characters)
 28- 37  A10    ---      VarName   Designation as Variable Star
 39- 40  I2     h        RAh       Right Ascension J2000 (hours)
 42- 43  I2     min      RAm       Right Ascension J2000 (minutes)
 45- 48  F4.1   s        RAs       Right Ascension J2000 (seconds)
     50  A1     ---      DE-       Declination J2000 (sign)
 51- 52  I2     deg      DEd       Declination J2000 (degrées)
 54- 55  I2     arcmin   DEm       Declination J2000 (minutes)
 57- 58  I2     arcsec   DEs       Declination J2000 (seconds)
 61- 66  F6.1   0.1s/a   precRA    Precession in RA for 10 years
 68- 71  I4 0.1arcsec/a precDE    Precession in DE for 10 years
 74- 79  F6.2   deg      ELON      Ecliptic longitude, J2000
 81- 86  F6.2   deg      ELAT      Ecliptic latitude, J2000
 88- 93  F6.2   deg      GLON      Galactic longitude
 95-100  F6.2   deg      GLAT      Galactic latitude
102-105  I4     ---      HRm       [0,9110] 2nd HR number when combined (4)
107-112  I6     ---      HIP       ?=0 Hipparcos (Cat. <I/239>) designation (4)
    114  A1     ---    n_HIP       [*] Note on Plx, pmRA, pmDE (4)
115-117  I3     mas      Plx       ? Parallax from Hipparcos (1) (4)
119-124  I6     mas/yr   pmRA      Proper motion (mouvement propre) in RA (4) (9)
126-131  I6     mas/yr   pmDE      Proper motion (mouvement propre) in DE (4)
133-142  F10.6  deg      RA(ICRS)  ? Right ascension from Hipparcos (10)
144-153  F10.6  deg      DE(ICRS)  ? Declination from Hipparcos (10)
155-159  I5     al       Dist      Distance of star in light-years (5)
    160  A1     ---    n_Dist      [ts] Trigonometric/Spectroscopic distance (5)
162-165  I4     km/s     RVel      Radial velocity (vitesse radiale)
167-170  A4     ---    n_RVel      [VSB12O? ]  Comment on RVel (3)
172-176  F5.2   mag      Vmag      [-1.46/5.03] Visual magnitude
                                       (magnitude visuelle)
178-182  F5.2   mag      B-V       ?=99.99  B-V (Johnson) index (2)
183-187  F5.2   mag      U-B       ?=99.99  U-B (Johnson) index (2)
188-192  F5.2   mag      V-R       ?=99.99  V-R (Johnson) index (2)
193-197  F5.2   mag      R-I       ?=99.99  R-I (Johnson) index (2)
199-204  F6.3   mag      b-y       ?=99.999 Stromegren index (2)
205-210  F6.3   mag      m1        ?=99.999 Stroemgren index (2)
211-216  F6.3   mag      c1        ?=99.999 Stroemgren index (2)
217-222  F6.3   mag      Hbeta     ?=99.999 Stroemgren index (2)
    226  A1     ---    l_vsini     [:<] Limit flag for vsini (6)
227-229  I3     km/s     vsini     ? Projected rotational velocity
                                    (projection de la vitesse de rotation
                                     sur la ligne de visée) (1)
    230  A1     ---    u_vsini     ':' for uncertainty (marque d'incertitude)
231-235  F5.1   0.1nm    Wgamma    ? Equivalent width of H-gamma line
                                     (largeur equivalente de la raie Hgamma)
                                     (en Angstroem = 10-10m) (1)
237-238  A2     ---      WilsonLC  Wilson Luminosity Class (7)
239-257  A19    ---      SpType    MK Spectral Type
    258  I1     ---      LClass    [1/6]? Numeric Luminosity Class (classe de
                                           luminosite exprimée en chiffre) (1)
260-261  I2     ---      nComp     [1/13]? Number of visual components
                                       (nombre de composantes visuelles)
263-266  F4.1   mag      Dmag      ? Magnitude difference betwéen components
                                       (difference de magnitude entre les
                                       composantes) Comp (1)
268-273  F6.1   arcsec   Sep       ? Separation betwéen Comp (1)
275-278  A4     ---      Comp      Designation of components (8)
--------------------------------------------------------------------------------
```

Note (1): (la donnée peut etre inexistante)
Note (2): (la valeur 99.99 ou 99.999 indique l'inexistance de la donnée)
Note (3): Note on RVel:
V: (vitesse variable sans raison détérminée)
SB: (binaire spectroscopique)
SB1: (binaire à un spectre)
SB2: (le deuxieme spectre est visible)
O: (les éléments orbitaux ont été détérmines)
Note (4):
(certaines binaires serrées ont des données Hipparcos qui combinent 2 étoiles, dont le numero HR est dans la colonne HRm; cette colonne HRm contient 0 pour les étoiles simples)
(Certaines étoiles brillantes n'ont pas de solution astrometrique dans Hipparcos; les parallaxes et mouvements propres affiches proviennent alors du "Bright Star" catalogue)
(la parallaxe Hipparcos a une precision generalement meilleure que 1mas)
Note (5):
(la distance, exprimée en années-lumières, est calculée par) :
- (la parallaxe trigonometrique quand n_Dist contient 't',
methode adoptée lorsque la parallaxe est d'au moins 5mas)
- (la distance spectroscopique, calculée a partir de la photometrie UBV et de la classification spectrale, a été adoptée pour des distances depassant 650al)
(La precision de la distance devrait etre meilleure que 20%)
Note (6):
(le symbole ':' signifie "inferieur ou égal")
Note (7):
(uniquement lorsque le type MK n'est pas complet)
Note (8):
(Désignation des composantes concernées par Dmag et Sep);
"O" (binaire découverte par occultation)
Note (9):
(ce mouvement propre est la variation annuelle de RA * cos(DE))
Note (10):
(les positions Hipparcos sont dans le systeme ICRS, qui est proche du systeme FK5 equinoxe J2000; les positions sont données pour l'epoque 1991.25, milieu de la mission Hipparcos)

```
--------------------------------------------------------------------------------
Byte-by-byte Description of file: names.dat
--------------------------------------------------------------------------------
   Bytes Format Units   Label     Explanations
--------------------------------------------------------------------------------
   1-  4  I4    ---     HR        Bright Star number
   7- 18  A12   ---     Name      Common name of the star / Nom usuel
--------------------------------------------------------------------------------
```

Annexe 2. Premier programme de lecture du catalogue.

```
— programme ADA lire_catalog01.adb
-- lecture d'un fichier texte contenant
-- un catalogue d'étoiles

with Ada.Text_Io; use Ada.Text_Io;

procedure lire_catalog01 is
F              : File_Type;
Ligne_fichier  : String (1 .. 280);
Lg             : Natural;
-- ascension_droite :

begin
Open (F,In_File,"catalog.dat" );
   while not End_of_file(F) loop
     Get_line(F,Ligne_fichier,lg);
     -- test sur ascension droite
     if Ligne_fichier(39..48) > " 5 00 00.0" and Ligne_fichier(39..48) < " 6 00
00.0"
       then -- ascension droite respectée
       -- test sur déclinaison
          if Ligne_fichier(51..58) < "13 00 00"
            then
            -- test sur magnitude
               if Ligne_fichier(172..176) < " 4.00"
                  then
                  Put(Ligne_fichier(18..21) & " ");    --    lettre grecque
abrégée
                  Put(Ligne_fichier(23..25) & "  ");   --    constellation
                  Put(Ligne_fichier(39..48) & "  ");   --    ascension droite
                  Put(Ligne_fichier(50..58) & " ");--    déclinaison
                  Put(Ligne_fichier(50..50) & " mag: ");
                  Put(Ligne_fichier(172..176) );--    magnitude
                  New_line;
               end if; -- test sur magnitude

            end if ; -- test sur déclinaison
       end if; -- test sur ascension droite
   end loop;   -- parcours du fichier
   Close (F);
end lire_catalog01 ;
```

Annexe 3. Programme lire_catalog05.adb

```
-- Programme lire_catalog05.adb
-- Lerautal 01/2013
-- Thème de travail : la carte de la constellation Orion
-- intentions :
-- 1.
-- lecture d'un fichier texte contenant un catalogue d'étoiles
-- 2.
-- transformation de l'ascension droite en minutes d'arc (float)
-- transformation de la déclinaison en minutes d'arc (float)
-- transformation de la magnitude en float
-- on conserve lettre grecque et nom de constellation sous forme de string
-- 3.
-- création d'un fichier séquentiel en mode écriture selon une structure
-- définie par un "record"
-- 4.
-- enregistrement des "records" dans le fichier séquentiel
-- et fermeture de celui-ci.

with Ada.Text_Io; use Ada.Text_Io;
with ada.Sequential_IO ;

procedure lire_catalog05 is
-- variables pour la lecture du fichier texte
Fichier_texte  : File_Type;
Ligne_fichier  : String (1 .. 280);
Lg             : Natural;
i : integer := 1 ;
   package Entier IS NEW Ada.Text_IO.Integer_IO(Integer);
-- structure pour enregistrer ce qui résulte des transformations
   type Rec_Etoile is   record
      Lettre_Grecque : String(1..4) ;
      Constellation  : String(1..3) ;
      AD_minutes    : float ;
      Decl_minutes  : float ;
      Magnitude      : float ;
   end record ;

   Etoile   : Rec_Etoile ; -- chaque étoile en lecture écriture
   Hr , Mn         : float ;
   Deg , MnArc     : float ;
   Mag             : float ;

   package Fichier_IO is new ada.sequential_io(Rec_Etoile);
   use Fichier_IO ;
   Catalogue_Etoiles : Fichier_IO.file_type ;

begin
   Open (Fichier_texte,In_File,"catalog.dat" ); -- fichier à lire
   Create(Catalogue_Etoiles,out_file,"transformer.dat"); -- fichier cible
   while not End_of_file(Fichier_texte) loop  -- parcourir le fichier
     Get_line(Fichier_texte,Ligne_fichier,lg);
      Etoile.Lettre_Grecque  :=Ligne_fichier(18..21) ;
      Etoile.Constellation   :=Ligne_fichier(23..25) ;
      -- Ascension droite en minutes
```

```
      Hr := Float'value(Ligne_fichier(39..40));
      Hr := Hr * 900.0 ;-- heure en minutes d'arc
      Mn := Float'value(Ligne_fichier(42..43));
      Mn := Mn * 15.0 ;   -- minutes d'arc
      Etoile.AD_minutes := Hr + Mn ;
      -- Déclinaison en minutes
      Deg := Float'value(Ligne_fichier(51..52));
      Deg := Deg * 60.0 ;
      MnArc := Float'value(Ligne_fichier(54..55));
      MnArc := MnArc ;
      Etoile.Decl_minutes := Deg + MnArc ;
      -- tenir compte du signe moins
      if Ligne_fichier(50..50) = "-"
         then Etoile.Decl_minutes := Etoile.Decl_minutes * (-1.0);
      end if ;
      Mag := Float'value(Ligne_fichier(172..176));
      Etoile.Magnitude := Mag ;
      — écrire l'enregistrement
      write(Catalogue_Etoiles,Etoile);
      i := i + 1 ;
   end loop;   -- parcours du fichier
   Close (Fichier_texte);
   Close(Catalogue_Etoiles);
end lire_catalog05 ;
```

Annexe 4 : fairesvg.ads

```
with Ada.Text_Io; use Ada.Text_Io;

package fairesvg is

   Fichier  : File_type ;
   guil : string(1..1) :="""; -- guillemet

   function eliminer(C:Character:=' ';T:String) return String ;
   procedure en_tete ;
   procedure cadre(w  : in integer ; h  : in integer ) ;
   procedure rectangle(x0 : in integer;  y0 : in integer ;
                  w  : in integer ; h  : in integer ;
                  couleur : in string ;
                  l  : in integer ;
                  tour : in string ) ;
   Procedure cercle(x0  : in integer;  y0 : in integer ;
                     r  : in integer ; couleur : in string ;
                     l  : in integer ; tour : in string ) ;
   procedure ligne(x1 : in integer;  y1 : in integer ;
                   x2  : in integer ; y2  : in integer ;
                  couleur : in string ;
                  l  : in integer ) ;
   procedure texte(x1 : in integer;  y1 : in integer ;
                    couleur : in string ;
                    contenu : in string ) ;
   procedure fin ;

end fairesvg ;
```

Annexe 5 : fairesvg.adb

```
with Ada.Text_Io; use Ada.Text_Io;

package body fairesvg is

   function eliminer(C:Character:=' ';T:String) return String is
      X:String:=T;deb:Positive:=T'first;fin:Positive:=T'last;
      -- élimine l'espace qui bloque les paramètres numériques
      -- dans SVG
   begin
   -- on suppose que T n'est pas vide
         loop
            exit when deb>fin;
               if X(deb)=C
                  then
                     X(X'first..fin-1):=X(X'first..deb-1)&X(deb+1..fin);
                     fin:=fin-1;
                  else
                     deb:=deb+1;
                  end if;
         end loop;
         return X(X'first..fin);
   end eliminer;

   Procedure en_tete is
   -- écr ire l'en-tête d'un fichier SVG
   begin
      Put (Fichier,"<?xml version = ");
      Put(Fichier,guil); Put(Fichier,"1.0"); Put(Fichier,guil);
      Put(Fichier," encoding = ");  Put(Fichier,guil);
      Put(Fichier,"iso-8859-1"); Put(Fichier,guil);
      Put_line(Fichier," ?> ") ;
   end en_tete ;

   procedure cadre(w  : in integer ; h  : in integer )  is -- couleur du tour

   begin
      Put(Fichier,"<svg width =");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',integer'Image(w))) ; Put(Fichier,guil);

      Put(Fichier," height=");   Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(h))) ; Put(Fichier,guil);
      Put(Fichier," xmlns="); Put(Fichier,guil);
      Put(Fichier,"http://www.w3.org/2000/svg");
      Put(Fichier,guil);
      Put_line(Fichier,">");
   end cadre ;

   procedure fin is
   begin
      Put_line(Fichier,"</svg> " );
   end fin ;

   procedure rectangle(x0 : in integer;  y0 : in integer ;
                  w  : in integer ; h  : in integer ;
                  couleur : in string ;
```

```
                  l  : in integer ;
                  tour : in string ) is

   begin
      Put(Fichier,"<rect x =");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',integer'Image(x0))) ; Put(Fichier,guil);
      Put(Fichier," y =");   Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(y0))) ; Put(Fichier,guil);
      Put(Fichier," width=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(w))) ; Put(Fichier,guil);
      Put(Fichier," height=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(h))) ; Put(Fichier,guil);
      Put(Fichier," style=");  Put(Fichier,guil);
      Put(Fichier,"fill:"); Put(Fichier,couleur);
      Put(Fichier,";stroke-width:"); Put(Fichier,eliminer('
',Integer'Image(l)));
      Put(Fichier,";stroke:");Put(Fichier,tour);
      Put(Fichier,guil); Put_line(Fichier,"/>");
   end rectangle ;

   Procedure cercle(x0  : in integer;  y0 : in integer ;
                    r  : in integer ; couleur : in string ;
                    l  : in integer ; tour : in string ) is
   begin
--  <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2"
fill="red" />
      Put(Fichier,"<circle cx=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(x0))) ; Put(Fichier,guil);
      Put(Fichier," cy=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(y0))) ; Put(Fichier,guil);
      Put(Fichier," r=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(r))) ; Put(Fichier,guil);
      Put(Fichier," stroke=");  Put(Fichier,guil);
      Put(Fichier,couleur);Put(Fichier,guil);
      Put(Fichier," stroke-width=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(l))) ; Put(Fichier,guil);
      Put(Fichier," fill=");  Put(Fichier,guil);
      Put(Fichier,tour);Put(Fichier,guil) ;
      Put_line(Fichier,"/>");
   end cercle ;

  procedure ligne(x1 : in integer;  y1 : in integer ;
                   x2  : in integer ; y2  : in integer ;
                  couleur : in string ;
                  l  : in integer ) is
-- <line x1="50" y1="50" x2="400" y2="50" stroke="pink" stroke-width="2" />
   begin
      Put(Fichier,"<line x1=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(x1))) ; Put(Fichier,guil);
      Put(Fichier," y1=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(y1))) ; Put(Fichier,guil);
      Put(Fichier," x2=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(x2))) ; Put(Fichier,guil);
      Put(Fichier," y2=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(y2))) ; Put(Fichier,guil);
      Put(Fichier," stroke=");  Put(Fichier,guil);
      Put(Fichier,couleur); Put(Fichier,guil);
      Put(Fichier," stroke-width=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(l))) ; Put(Fichier,guil);
```

```
      Put_line(Fichier,"/>");
   end ligne ;

   procedure texte(x1 : in integer;  y1 : in integer ;
                   couleur : in string ;
                   contenu : in string )  is

--    <text x="0" y="60" fill="red"> 800</text>
   begin
      Put(Fichier,"<text x=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(x1))) ; Put(Fichier,guil);
      Put(Fichier," y=");  Put(Fichier,guil);
      Put(Fichier,eliminer(' ',Integer'Image(y1))) ; Put(Fichier,guil);
      Put(Fichier," fill=");  Put(Fichier,guil);
      Put(Fichier,couleur); Put(Fichier,guil);
      Put(Fichier,">");
      Put(Fichier,contenu);
      Put_line(Fichier,"</text>");
   end texte ;

end fairesvg ;
```

Annexe 6 : le programme faire_orion.adb

```
with Ada.Text_Io; use Ada.Text_Io;
with Ada.Sequential_IO ;
with fairesvg ; use fairesvg ;

procedure faire_orion is
-- structure pour enregistrer ce qui résulte des transformations
   type Ret is   record
      LG       : String(1..4) ;  Const    : String(1..3) ;
      ADs      : float ;         Dcls     : float ;
      Mag      : float ;
   end record ;
   etoile : ret ;
   package FSeq is new Ada.sequential_io(Ret);
   use FSeq ;
   Catalogue   :  FSeq.file_type ;
   prov : float ;
   x, y , d : integer ; -- coordonnées sur la carte et diamètre

begin
   Open(Catalogue,in_file,"transformer.dat"); -- en lecture
   Create(Fichier,Out_file,"orion01.svg");
   en_tete;
   cadre(450,780);

   while not End_of_file(Catalogue) loop  -- parcourir le fichier
         Read(Catalogue,etoile);
         -- il faut tester si les étoiles sont dans la fenêtre
         -- il faut ADs > 4500.0 et ADs < 5400.0
         -- et  Dcls > -780.0 et Dcls < 780.0
      if etoile.ADs >= 4500.0 and etoile.ADs <= 5400.0 and etoile.Dcls > -780.0
                      and etoile.Dcls < 780.0
      then
         prov :=((etoile.ADs-5400.0) * (-1.0));
         x := integer(prov) /2 ;
         if etoile.Dcls > 0.0
            then
               prov := (etoile.Dcls - 780.0)* (-1.0) ;
               y := integer(prov)/2;
            else
               prov := (etoile.Dcls * (-1.0)) + 780.0 ;
               y :=Integer(prov)/2;
         end if ;
         prov := (etoile.Mag -6.0) *(-2.0) ;
         d := integer(prov);
         cercle(x,y,d,"white",1,"black");
      end if ;
   end loop;   -- parcours du fichier
   Fin;
   Close(Fichier);
   Close(Catalogue);
end faire_orion ;
```