

# Premiers programmes Ada

## Notions acquises à l'issue du TP :

- Savoir compiler et exécuter un programme ADA.
- Comprendre l'intérêt d'un compilateur
- Comprendre les entrée-sorties en Ada
- Savoir choisir la « bonne » structure de contrôle dans un algorithme.

## Exercice 1 : Compiler et exécuter un programme Ada.

Un programme Ada est écrit dans un fichier dont l'extension est `.adb`, comme par exemple `premier_programme.adb` (listing 1). Ce programme est dans le dossier `tp1`.

Nous utiliserons GNAT (GNU Ada), le compilateur Ada du projet GNU, pour traduire un programme Ada en un exécutable. Il suffit de taper dans un terminal (il faut bien sûr être dans le dossier qui contient le fichier à compiler) :

```
> gnatmake -gnatwa premier_programme.adb
```

Cette commande produit un fichier exécutable nommé `premier_programme1` (le nom du fichier contenant le source Ada sans l'extension `.adb`). Pour l'exécuter, on fait :

```
> ./premier_programme
```

La commande `gnatclean` efface les fichiers engendrés.

```
> gnatclean premier_programme
```

1. D'autres fichiers sont engendrés. La commande `ls` permet de les lister.

```
1  with Text_IO;
2  use Text_IO;
3
4  -- Programme minimal qui affiche juste un message.
5  procedure Premier_Programme is
6  begin
7      Put_Line ("Bravo ! Vous avez réussi à exécuter le programme.");
8  end Premier_Programme;
```

Listing 1 – Le fichier Ada `premier_programme.adb`

**Attention :** L'option `-gnatwa` demande au compilateur de signaler davantage de messages d'avertissement. Même s'ils n'empêchent pas le compilateur d'engendrer un exécutable, les avertissements correspondent généralement à des erreurs ou des maladroites qu'il faut corriger.

### Exercice 2 : Intérêt d'un compilateur et du typage statique

Les fichiers `min_max_serie.py` (listing 2) et `min_max_serie.adb` (listing 3) contiennent respectivement un programme Python et Ada affichant le plus petit et le plus grand élément d'une série d'entiers naturels. L'utilisateur indique la fin de la série en tapant un nombre strictement négatif.

1. Exécuter le programme Python<sup>2</sup> en tapant les entiers 2 9 3 6 3 et 0. Quelles erreurs sont signalées ? Les corriger et recommencer. Est-ce que le programme contient d'autres erreurs ?

2. Compiler le programme Ada. Quelles erreurs sont signalées ? Les corriger et recompiler.

Ces erreurs étaient aussi présentes dans le programme Python. Avaient-elles été détectées ? Dans la négative, comment les mettre en évidence ?

3. Indiquer l'intérêt du typage statique et du compilateur.

### Exercice 3 : Comprendre les saisies d'entiers et de caractères

Le programme du listing 4 permet de saisir une longueur sous la forme d'une valeur (entier) et d'une unité (caractère). Par exemple 52c correspond à la valeur 52 et l'unité c (pour centimètre).

1. Compiler ce programme. L'exécuter avec les entrées suivantes : 52cm, \_52mn puis 1..n.

Nous avons fait deux appels à `Get`, le premier sur `Valeur` de type entier et le second sur `Unite` de type caractère. Derrière ce même nom `Get`, il y a deux procédures différentes : c'est la *surcharge*. L'une est définie dans le paquetage `Ada.Integer_Text_IO`, l'autre dans `Ada.Text_IO`.

Les sous-programmes de lecture consomment les caractères que l'utilisateur saisit au clavier (sur l'entrée standard du programme) et les utilisent pour produire la valeur attendue. Pour la lecture d'un entier, les blancs sont ignorés et les chiffres sont assemblés pour produire l'entier. Par exemple, si l'utilisateur a saisi `_52cm`, l'espace initial est ignoré, les caractères '5' et '2' sont reconnus comme faisant partie d'un entier, le caractère suivant 'c' ne fait pas partie de l'entier. La procédure `Get(Integer)` consomme donc l'espace, '5' et '2' pour reconnaître l'entier 52. Les caractères suivants restent sur l'entrée standard : 'c', 'm' et « retour à la ligne ». `Get(Character)` récupère le prochain caractère sur l'entrée standard qui n'est pas un retour à la ligne et le fournit à l'appelant. Le caractère 'c' sera utilisé les deux autres restant sur l'entrée standard.

Ce comportement permet à l'utilisateur de saisir plusieurs informations d'un coup comme ici une longueur (entier) suivie d'une unité (caractère) que l'utilisateur peut saisir comme 52c.

Souvent, ce n'est pas le comportement souhaité : on préfère repartir avec une nouvelle saisie à chaque nouvelle sollicitation de l'utilisateur. Ceci évite que les caractères non consommés par la saisie précédente soient utilisés pour la saisie suivante. Pour ce faire, la procédure `Skip_Line` consomme les caractères de l'entrée standard jusqu'au prochain retour à la ligne inclus.

**Remarque :** la fonction `Get_Line`, utilisée dans le dernier `Put_Line`, retourne la chaîne constituée des caractères de l'entrée standard jusqu'au prochain retour à la ligne exclu mais consommé.

2. Décommenter les `Skip_Line` de `comprendre_get_skip_line.adb` (listing 4) et mettre en commentaire le dernier `Put_Line`. Compiler et exécuter les exemples précédents. Exécuter en saisissant 52ccc, deux lignes vides et mno.

---

2. Pour exécuter un programme Python depuis le terminal : `python3 min_max_serie.py`

```

1  # Afficher le plus petit et le plus grand élément d'une série d'entiers
2  # naturels lus au clavier. La saisie de la série se termine par 0
3  # (qui n'appartient pas à la série).
4  # Exemple : 2, 9, 3, 6, 3, 0 -> min = 2 et max = 9
5
6  # afficher la consigne
7  print('Saisir les valeurs de la série (0 pour terminer) :')
8
9  # saisir un premier entier
10 entier = int(input())
11
12 if entier == 0: # entier n'est pas une valeur de la série
13     print('Pas de valeurs dans la série')
14 else: # entier est le premier élément de la série
15     # initialiser min et max avec le premier entier
16     min = entier
17     max = entier
18
19     # traiter les autres éléments de la série
20     entier = int(input()) # saisir un nouvel entier
21     while entier != 0: # entier est une valeur de la série
22         # mettre à jour le min et le max
23         if entier > max: # nouveau max
24             # mettre à jour le max avec entier
25             max = entier
26         elif entier < min: # nouveau min
27             # mettre à jour le min avec entier
28             min = entier
29         else:
30             pass
31
32     # saisir un nouvel entier
33     entier = int(input())
34
35     # afficher le min et le max
36     print('min =', min)
37     print('max =', max)

```

Listing 2 – Le fichier Python min\_max\_serie.py

```

1  with Text_IO;
2  use Text_IO;
3  with Ada.Integer_Text_IO;
4  use Ada.Integer_Text_IO;
5
6  -- Afficher le plus petit et le plus grand élément d'une série d'entiers
7  -- naturels lus au clavier. La saisie de la série se termine par 0
8  -- (qui n'appartient pas à la série).
9  -- Exemple : 2, 9, 3, 6, 3, 0 -> min = 2 et max = 9
10 procedure Min_Max_Serie is
11     Entier: Integer;    -- un entier lu au clavier
12     Min, Max: Integer;  -- le plus petit et le plus grand élément de la série
13
14 begin
15     -- Afficher la consigne
16     Put("Saisir les valeurs de la série (0 pour terminer) : ");
17
18     -- Saisir un premier entier
19     Get(Entier);
20
21     if Entier = 0 then --{ entier n'est pas une valeur de la série }
22         Put_Line("Pas de valeurs dans la série");
23     else --{ Entier est le premier élément de la série }
24         -- initialiser Min et Max avec le premier entier
25         Min := Entier;
26         Max := Entier;
27
28         -- traiter les autres éléments de la série
29         Get(Entier); -- saisir un nouvel entier
30         while Entier /= 0 loop -- Entier est une valeur de la série
31             -- mettre à jour le Min et le Max
32             if Entier > Max -- nouveau max
33                 -- mettre à jour le max avec Entier
34                 Max := Entier;
35             elsif Entier < Min then -- nouveau Min
36                 -- mettre à jour le min avec Entier
37                 Min := Entier;
38             else
39                 null;
40             end if;
41
42             -- saisir un nouvel entier
43             Get(Entier);
44         end loop;
45
46         -- afficher le min et le max de la série
47         Put("Min = ");
48         Put(Min, 1);
49         New_Line;
50
51         Put("Max = ");
52         Put(Max, 1);
53         New_Line;
54     end if;
55 end Min_Max_Serie;
    
```

Listing 3 – Le fichier Ada min\_max\_serie.adb

```

1  with Ada.Text_IO;           use Ada.Text_IO;
2  with Ada.Integer_Text_IO;   use Ada.Integer_Text_IO;
3
4  -- Objectif : Comprendre le comportement de Get et Skip_Line.
5
6  -- Saisir une longueur (valeur et unité) et l'afficher.
7  procedure Comprendre_Get_Skip_Line is
8      Valeur: Integer;         -- la valeur du longeur
9      Unite: Character;        -- l'unité de la longueur : (c)entimètre, (m)ètre...
10  begin
11      -- saisir la longueur
12      Put("Longueur = ");
13      Get(Valeur);
14      -- Skip_Line;
15      Get(Unite);
16      -- Skip_Line;
17
18      -- afficher la longueur
19      Put("Valeur = ");
20      Put(Valeur, 1);
21      New_Line;
22      Put_Line("Unité = >" & Unite & "<");
23
24      -- afficher les caractères en attente sur l'entrée standard (fin de ligne)
25      Put_Line("Reste de la dernière ligne saisie : " & "'" & Get_Line & "'");
26  end Comprendre_Get_Skip_Line;
    
```

Listing 4 – Le fichier Ada comprendre\_get\_skip\_line.adb

**Attention :** S’il n’y a pas de caractère en attente, et donc pas de retour à la ligne, Skip\_Line attend une nouvelle saisie de l’utilisateur. Ceci ne se produira jamais si on le fait après un Get d’un entier ou d’un caractère.

#### Exercice 4 : Structures de contrôle

L'objectif de cet exercice est de manipuler les structures de contrôle de notre pseudo-langage algorithmique et leur pendant en Ada. Les programmes listés ci-après sont à compléter dans l'ordre. Chaque programme commence par une description de son objectif et des exemples d'utilisation qui vous permettront de tester vos programmes. Il faudra faire attention à bien utiliser la bonne structure de contrôle.

On ne traitera pas la robustesse de ces programmes : on considère que l'utilisateur saisira toujours une donnée valide.

1. permuter\_caracteres.adb
2. tarif\_place.adb
3. classer\_caractere.adb
4. compte\_jules\_objectif.adb
5. chiffre\_significatif.adb
6. table\_7.adb
7. table\_pythagore.adb
8. score\_21.adb
9. somme\_serie\_double.adb

**Pour aller plus loin, pour les plus rapides ou ceux qui ont envie de s'entraîner voici quelques exercices supplémentaires et optionnels.**

#### Exercice 5 : Racine carrée d'un nombre (Méthode de Newton)

La  $k^{ieme}$  approximation de la racine carrée de  $x$  est donnée par  $a_{k+1} = (a_k + x/a_k)/2$  et  $a_0 = 1$ .

On arrête le calcul quand la distance entre  $a_{k+1}$  et  $a_k$  est inférieure à une précision donnée.

1. Écrire un programme (fichier `newton.adb`) qui affiche une valeur approchée de la racine carrée d'un nombre en utilisant la méthode précédente. Nombre et précision seront lus au clavier.
2. On peut aussi arrêter le calcul des  $a_k$  quand  $a_k^2$  est proche de  $x$  à la précision près. Modifier le programme précédent pour donner à l'utilisateur le choix entre ces deux conditions d'arrêt.

#### Exercice 6 : Puissance

Afficher la puissance entière d'un réel en utilisant somme et multiplication (`puissance.adb`). On traite d'abord le cas où l'exposant est positif avant de généraliser aux entiers relatifs.

#### Exercice 7 : Amélioration du calcul de la puissance entière

Améliorer l'algorithme de calcul de la puissance (`puissance_mieux.adb`) en remarquant que :

$$x^n = \begin{cases} (x^2)^p & \text{si } n = 2p \\ (x^2)^p \times x & \text{si } n = 2p + 1 \end{cases}$$

Ainsi, pour calculer  $3^5$ , on peut faire  $3 * 9 * 9$  avec bien sûr  $9 = 3^2$ .