Premiers programmes Ada

Notions acquises à l'issue du TP :

- Savoir utiliser Subversion (SVN) de manière minimale.
- 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.
- Savoir utiliser les variables booléennes.

Exercice 1 : Subversion

Subversion (SVN) est un outil de gestion de versions qui permet de conserver l'historique des versions de fichiers. Les outils de ce type sont essentiels, en particulier dans les développements informatiques.

Subversion vous permettra de récupérer les fichiers fournis et de rendre le travail fait.

1. *Créer un dossier pour PIM.* Commençons par créer un dossier `pim` dans lequel seront faits les TP de cette matière. Exécuter dans un terminal les commandes suivantes :

```
> cd          # aller dans votre dossier racine (home directory)
> mkdir pim   # créer le dossier pim
> cd pim      # se déplacer dans le dossier pim
```

Vous pouvez bien sûr placer le dossier `pim` à un autre endroit en fonction des habitudes ou préférences que vous avez pour organiser vos fichiers.

2. *Récupérer une copie du dépôt.* Subversion est un gestionnaire de version centralisé : les fichiers sont stockés sur un serveur. On appelle *dépôt* ou *référentiel* les fichiers stockés sur ce serveur. Les utilisateurs travaillent sur une copie locale.

La commande `svn` permet de manipuler le référentiel et la copie locale via des commandes, en fait des sous-commandes, en précisant éventuellement des options et des arguments suivant la forme suivante : `svn commande [options] [arguments]`

La première chose à faire est de créer une copie locale du dépôt SVN. Ceci se fait grâce à la commande svn `checkout` (abréviation `co`).

```
> svn co http://cregut.svn.enseeiht.fr/2019/1sn/pim/tps/$USER tp
```

`$USER` donne l'identifiant qui sert à se connecter sur une station de travail (généralement initiale du prénom et début du nom). Il est possible de vérifier sa valeur en tapant dans un terminal :

```
> echo $USER
```

Le dossier `tp` est créé. Il contient une copie du dépôt, en particulier des sous-dossiers `tp0` et `tp1`. Vérifions-le en tapant les commandes suivantes.

```
> ls
tp
> cd tp
> ls
tp0 tp1
> ls -A        # -A affiche les fichiers cachés sauf '.' et '..'
.svn tp0 tp1
```

Il ne faut surtout pas modifier ou effacer le contenu du fichier caché `.svn`. C'est dans ce dossier que sont conservées toutes les informations liées au bon fonctionnement de SVN.

Quand on a récupéré une copie du dépôt, on n'a plus besoin de faire `svn checkout`.

3. *Modifier un fichier.* Aller dans le dossier tp0 et modifier le fichier texte.txt en ajoutant une nouvelle ligne à la fin. Vous avez maintenant un fichier local qui est différent du fichier récupéré du dépôt SVN. On peut le constater avec la commande `status`.

```
> svn status
M texte.txt
```

Le caractère `M` indique que le fichier a été modifié (Modified).

Si rien ne s'affiche, c'est que la copie locale est identique à la dernière copie récupérée.

On peut aussi voir les différences entre les deux fichiers en faisant :

```
> svn diff texte.txt
```

4. *Pousser les modifications sur le dépôt SVN.* On utilise `svn commit` pour valider les modifications locales et les pousser sur le dépôt SVN. Une nouvelle version est créée sur le dépôt et son numéro est affiché (*revision*). Lorsque l'on fait un commit il faut toujours expliquer de manière concise les modifications faites grâce à l'option `-m` (`m` comme message). Si cette option n'est pas précisée, l'éditeur par défaut [1] est lancé pour saisir le message.

```
> svn commit -m "Ajout d'une ligne en fin"
```

Par défaut, toutes les modifications du dossier courant et de ses sous-dossiers seront validées et poussées. Si on ne veut valider et pousser que certains fichiers/dossiers, il faut les lister explicitement sur la ligne de commande.

Cette commande échoue si les fichiers modifiés localement ont déjà été modifiés sur le dépôt. Dans ce cas là, il faut commencer par mettre à jour sa copie locale du dépôt.

1. L'éditeur par défaut est souvent `vi` (ou maintenant `vim`, la version terminal de `Gvim`), un éditeur modal (mode commande et mode édition) qui est déroutant quand on a l'habitude d'un éditeur je-clique-je-tape. Si vous ne le connaissez pas, sachez qu'on peut le quitter en faisant `:q!`. Si vous êtes curieux, vous pouvez faire `:help`.

Il est important de faire des `svn commit` pour conserver les différentes versions de vos fichiers (au moins après chaque question terminée). Vous pourrez ainsi les récupérer en cas de perte accidentelle. **Il est en particulier IMPÉRATIF de faire un commit en fin de séance.** Bien sûr, si vous n'avez pas terminé le TP en séance, vous pouvez continuer à y travailler après. Il faudra faire un commit pour rendre votre travail. Votre enseignant de TP regardera après chaque séance de TP le travail de quelques étudiants pour leur faire un retour.

5. *Récupérer la dernière version du dépôt.* Pour récupérer localement la dernière version du dépôt et donc mettre à jour la copie locale, il suffit d'utiliser la commande `svn update`.

```
> svn update
```

La commande n'affiche rien si la copie locale est à jour.

Au fur et à mesure que nous avancerons dans les TP de PIM, de nouveaux dossiers seront ajoutés au dépôt. `svn update` permettra de les récupérer.

Si des fichiers modifiés localement ont aussi été modifiés sur le dépôt (certainement par d'autres personnes), `svn update` essaie de fusionner les modifications. S'il n'y arrive pas, il indique un conflit qu'il vous faudra résoudre. Nous y reviendons plus tard.

Toujours faire `svn update` avant de commencer à modifier ses fichiers pour être sûr que la copie locale est à jour et limiter le risque de conflits.

6. *Historique des versions.* Dans le contexte de PIM, Subversion conserve l'historique de toutes les modifications validées. `svn log` affiche la liste de ces versions avec son auteur, sa date, son message, etc.

```
> svn log
```

7. *Et la suite....* Il a y bien d'autres commandes proposées par subversion : add, mv, rm... Mais aussi la gestion des conflits, etc. Un petit guide avec un tutoriel est disponible dans les ressources de la page du module sous Moodle. Il est conseillé de les consulter.

Exercice 2 : 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) :

```
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`

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

Cette commande produit un fichier exécutable nommé `premier_programme` [2] (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
```

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 maladresses qu'il faut corriger.

Exercice 3 : 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 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 4 : Raffinage
Donner le raffinage (fichier `min_max_serie.raff`) à l'origine des programmes des listings 2 et 3.

Exercice 5 : 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`, `␣␣␣123mn` 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.

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

```python
# Afficher le plus petit et le plus grand élément d'une série d'entiers
# naturels lus au clavier.  La saisie de la série se termine par 0
# (qui n'appartient pas à la série).
# Exemple : 2, 9, 3, 6, 3, 0 -> min = 2 et max = 9

# afficher la consigne
print('Saisir les valeurs de la série (-1 pour terminer) :')

# saisir un premier entier
entier = int(input())

if entier == 0:  # entier n'est pas une valeur de la série
    print('Pas de valeurs dans la série')
else:   # entier est le premier élément de la série
    # initialiser min et max avec le premier entier
    min = entier
    max = entier

    # traiter les autres éléments de la série
    entier = int(input())   # saisir un nouvel entier
    while entier != 0:      # entier est une valeur de la série
        # mettre à jour le min et le max
        if entier > max    # nouveau max
            # mettre à jour le max avec entier
            max = entier
        elif entier < min:  # nouveau min
            # mettre à jour le min avec entier
            min = enlier
        else:
            pass

        # saisir un nouvel entier
        entier = int(input())

    # afficher le min et le max
    print('min =', min)
    print('max =', max)
```

Listing 2 – Le fichier Python `min_max_serie.py`

```
with Text_IO;
use Text_IO;
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;

-- Afficher le plus petit et le plus grand élément d'une série d'entiers
-- naturels lus au clavier.  La saisie de la série se termine par 0
-- (qui n'appartient pas à la série).
-- Exemple : 2, 9, 3, 6, 3, 0 -> min = 2 et max = 9
procedure Min_Max_Serie is
    Entier: Integer;    -- un entier lu au clavier
    Min, Max: Integer;    -- le plus petit et le plus grand élément de la série

begin
    -- Afficher la consigne
    Put("Saisir les valeurs de la série (-1 pour terminer) : ");

    -- Saisir un premier entier
    Put("Saisir les valeurs de la série (-1 pour terminer) : ");
    Get(Entier);

    if Entier = 0 then --{ entier n'est pas une valeur de la série }
        Put_Line("Pas de valeurs dans la série");
    else   -- Entier est le premier élément de la série
        -- initialiser Min et Max avec le premier entier
        Min := Entier;
        Max := Entier;

        -- traiter les autres éléments de la série
        Get(Entier);   -- saisir un nouvel entier
        while Entier /= 0 loop      -- Entier est une valeur de la série
            -- mettre à jour le Min et le Max
            if Entier > Max    -- nouveau max
                -- mettre à jour le max avec Entier
                Max := Entier;
            elsif Entier < Min then  -- nouveau Min
                -- mettre à jour le min avec Entier
                Min := Enlier;
            else
                null;
            end if;

            -- saisir un nouvel entier
            Get(Entier);
        end loop;

        -- afficher le min et le max de la série
        Put("Min = ");
        Put(Min, 1);
        New_Line;

        Put_Line("Max =" & Integer'Image(Max));
    end if;
end Min_Max_Serie;
```

Listing 3 – Le fichier Ada `min_max_serie.adb`

```ada
with Ada.Text_IO;          use Ada.Text_IO;
with Ada.Integer_Text_IO;  use Ada.Integer_Text_IO;

-- Objectif : Comprendre le comportement de Get et Skip_Line.

-- Saisir une longueur (valeur et unité) et l'afficher.
procedure Comprendre_Get_Skip_Line is
    Valeur: Integer;    -- la valeur du longeur
    Unite: Character;    -- l'unité de la longueur : (c)entimètre, (m)ètre...
begin
    -- saisir la longueur
    Put("Longueur = ");
    Get(Valeur);
    -- Skip_Line;
    Get(Unite);
    -- Skip_Line;

    -- afficher la longueur
    Put_Line("Valeur =" & Integer'Image(Valeur));
    Put_Line("Unité = >" & Unite & "<");

    -- afficher les caractères en attente sur l'entrée standard (fin de ligne)
    Put_Line("Reste de la dernière ligne saisie : " & '"' & Get_Line & '"');
end Comprendre_Get_Skip_Line;
```

Listing 4 – Le fichier Ada `comprendre_get_skip_line.adb`

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`.

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 6 : Drone commandé par un menu

On s'intéresse à la commande à distance d'un drone qui se déplace uniquement selon son axe vertical. On appelle « menu textuel » un affichage du type suivant permettant de prendre en

compte les choix successifs d'un utilisateur.

```
Altitude : 3

Que faire ?
    d -- Démarrer
    m -- Monter
    s -- Descendre
    q -- Quitter
Votre choix : _
```

Les propriétés suivantes doivent être satisfaites par le programme gérant le menu textuel :

1. Le drone ne peut monter et descendre que s'il a été démarré au préalable.
2. En fonctionnement nominal, l'action `monter` augmente l'altitude du drone d'une unité et l'action `descendre` la diminue d'une unité.
3. Le drone ne peut pas descendre à une altitude négative.
4. Le programme affiche le menu et traite chaque choix de l'utilisateur du programme jusqu'à ce que l'utilisateur choisisse de quitter (avec l'option 'q') ou jusqu'à ce que le drone monte à une altitude supérieure ou égale à 5 (le drone est alors hors de portée).
5. L'utilisateur pourra utiliser les minuscules ou les majuscules.

Écrire le programme correspondant (fichier `drone.adb`). Utiliser des variables booléennes est conseillé.

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

Exercice 7 : 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 8 : Puissance
Afficher la puissance entière d'un réel en utilisant somme et multiplication (`puissance.adb`).

On pourra commencer par traiter le cas où l'exposant est un entier naturel puis généraliser aux entiers relatifs.

Exercice 9 : 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 $9 = 3^2$.