Mini-projet 1 (Raffinages)

1 Cahier des charges

On souhaite écrire un programme qui aide à **réviser les tables de multiplication**. Voici les attentes concernant ce programme.

1. L'interface avec l'utilisateur sera en mode texte (console) et devra respecter l'exemple donné au listing 1 et expliqué ci-après.

```
Table à réviser : 7

(M1) 7 * 1 ? 7
Bravo !

(M2) 7 * 3 ? 24
Mauvaise réponse.

...

(M10) 7 * 5 ? 35
Bravo !

4 erreurs.  Il faut encore travailler la table de 7.

On continue (o/n) ? _
```

Listing 1 – Exemple d'exécution du programme de révision des multiplications

2. Le principe général du programme est le suivant. L'utilisateur commence par choisir la table à réviser (entre 0 [1] et 10), le programme lui pose 10 multiplications dont le nombre de gauche est la table à réviser et le nombre de droite est un nombre choisi au hasard (entre 0 et 10 aussi). Le programme demande à l'utilisateur le résultat de cette multiplication et lui indique s'il a bien répondu ou pas (« Bravo ! » ou « Mauvaise réponse. »). À la fin de la série de multiplications, un message est affiché en fonction du nombre d'erreurs commises :

— « Aucune erreur. Excellent !» si toutes les réponses sont justes,

— « Une seule erreur. Très bien. » s'il n'a commis qu'une seule erreur,

— « Tout est faux ! Volontaire ? » si toutes les réponses sont fausses.

— Le nombre de bonnes réponses et une incitation à apprendre cette table s'il y a moins de la moitié de bonnes réponses. Par exemple : « Seulement 2 bonnes réponses. Il faut apprendre la table de 7 ! ».

1. Ce pourrait être entre 1 et 10 mais choisir la table 0 facilitera la réalisation des tests.

— Le nombre d'erreurs et une incitation à retravailler cette table dans les autres cas. Par exemple « 4 erreurs. Il faut encore travailler la table de 7. ».

3. La table à réviser saisie par l'utlisateur doit être entre 0 et 10 sinon on lui redemande.

```
Table à réviser : 11
Impossible.  La table doit être entre 0 et 10.
Table à réviser : _
```

4. À la fin d'une série de multiplications, on demandera à l'utilisateur s'il veut continuer à s'entraîner ou s'il arrête. Seul le premier caractère de la réponse est pris en compte. Si l'utilisateur répond `'o'` ou `'O'` (en majuscule ou en minuscule), on recommence. Dans tous les autres cas, le programme se termine.

Deux extensions sont envisagées. Elles ne seront pas prises en compte dans les raffinages demandés mais devront être implantées dans la version finale du projet.

1. Le programme ne posera pas deux fois de suite la même multiplication.
2. Avant de demander si on continue, on conseillera de réviser la table correspondant au nombre de droite de la multiplication pour laquelle l'utilisateur a mis le plus de temps pour répondre, à condition que ce temps soit supérieur d'une seconde au temps moyen. Peu importe que la réponse soit juste ou fausse ; ce qui est pris en compte ici, c'est l'hésitation à répondre. Il est possible qu'aucun conseil ne soit affiché.

```
Des hésitations sur la table de 8 : 6.405475000 secondes contre 1.798116600
en moyenne.  Il faut certainement la réviser.
```

Indication : En Ada, on utilisera le paquetage `Alea` fourni (fichiers `alea.ads` et `alea.adb` dont il n'est pas utile de regarder le contenu). Le fichier `exemple_alea.adb` (listing 2) est un exemple d'utilisation du paquetage `Alea`. `Alea` est un paquetage générique [2] paramétré par deux entiers correspondant aux bornes de l'intervalle dans lequel les nombres aléatoires seront tirés. Pour l'utiliser, le `use Alea;` en début de fichier n'est pas suffisant. Il faut aussi instancier le paquetage pour donner une valeur aux bornes. Ceci se fait dans la partie déclarations :

```
package Mon_Alea is new Alea (5, 15);
    -- Les nombres aléatoires seront dans [5..15]
use Mon_Alea;  -- on peut alors utiliser Get_Random_Number
```

Indication : Le programme `mesure_temps.adb` (listing 3) montre comment on peut se servir du module `Calendar` pour obtenir l'heure actuelle et mesurer la durée utilisateur d'une opération.

2 Contraintes

1. Le programme doit fonctionner sur les salles d'enseignement de l'ENSEEIHT.
2. **Il est interdit de définir des sous-programmes et d'utiliser des tableaux.**

2. Cette notion sera vue plus tard...

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

-- Procédure qui illustre l'utilisation du paquetage Alea.
procedure Exemple_Alea is

    package Mon_Alea is
        new Alea (5, 15);  -- générateur de nombre dans l'intervalle [5, 15]
    use Mon_Alea;

    Nombre: Integer;
begin
    -- Afficher 10 nombres aléatoires
    Put_Line ("Quelques nombres aléatoires : ");
    for I in 1..10 loop
        Get_Random_Number (Nombre);
        Put (Nombre);
        New_Line;
    end loop;
end Exemple_Alea;
```

Listing 2 – Le fichier `exemple_alea.adb`

3 Documents à rendre

Les **documents à rendre**, dits *livrables*, sont :

— un rapport sous la forme d'un Google Doc. Il faudra faire une **copie** du document `https://docs.google.com/document/d/1sqj96bBI8qyenIPxWVf5srBNnt_Py5kuw3nJUz1rsI0`

Ce document doit être partagé avec votre enseignant de TP en lui donnant au moins les droits de commentateur. Il pourra ainsi vous faire des commentaires sur vos raffinages. Vous ne devez pas fermer ces commentaires mais répondre à chaque commentaire en indiquant comment il a été traité (ou en demandant des précisions ou compléments). C'est l'enseignant de TP qui pourra éventuellement fermer un commentaire une fois satistait de son traitement.

Les différentes rubriques de ce document devront être remplies.

Il inclut en particulier la grille d'évaluation des raffinages et la grille d'évaluation du code. Ces deux grilles seront complétées par l'étudiant et prises en compte dans la notation.

Les explications sur la grille des raffinages est à cette URL : `https://docs.google.com/spreadsheets/d/1A42zW2va97LPAyp27Ue6v16QKlQhqbsiLh0oG8NekpM`.

— un export PDF du Google Doc, poussé sur SVN dans le dossier pr01/. Ceci permet de garder une trace du travail rendu.

— le fichier source du programme : `multiplications.adb`.

```
1  with Ada.Text_IO;           use Ada.Text_IO;
2  with Ada.Integer_Text_IO;   use Ada.Integer_Text_IO;
3  with Ada.Calendar;          use Ada.Calendar;
4
5  procedure Mesure_Temps is
6      N: Integer;          -- un entier lu au clavier
7      Debut: Time;         -- heure de début de l'opération
8      Fin: Time;           -- heure de fin de l'opération
9      Duree : Duration;    -- durée de l'opération
10 begin
11     -- récupérer l'heure (heure de début)
12     Debut := Clock;
13
14     -- réaliser l'opération
15     Put_Line ("Début");
16     Put ("Valeur : ");
17     Get (N);
18     Put_Line ("Fin");
19
20     -- récupérer l'heure (heure de fin)
21     Fin := Clock;
22
23     -- calculer la durée de l'opération
24     Duree := Fin - Debut;
25
26     -- Afficher la durée de opération
27     Put_Line ("Durée :" & Duration'Image(Duree));
28
29     -- Initialisation directe
30     Duree := Duration (0.5);
31     Put_Line ("Durée = " & Duration'Image(Duree));
32 end Mesure_Temps;
```

Listing 3 – Le fichier `mesure_temps.adb`

4 Principales dates

— lundi 19 septembre 2022 : publication du sujet (Moodle) et des fichiers fournis (SVN).

— fin première séance de TP (Séance TP 2) : partage des raffinages réalisés en TP.

— samedi 24 septembre 2022 : rendu des raffinages et de la grille d'évaluation des raffinages (version finale), Les deux extensions ne doivent pas être décrites par ces raffinages.

— fin deuxième séance de TP (séance TP 4) : rendu du code source écrit en TP.

— samedi 8 octobre 2022 : date limite pour rendre la version finale des livrables : toutes les rubriques du rapport, les fichiers sources, la version PDF du rapport sur SVN. Les deux extensions doivent être implantées dans le code et expliquées en réponse aux questions 7 et 8.

5 Barème (indicatif)

1. Raffinages : 8 points
2. Programmation : 6 points
3. Correction du programme : 3 points
4. Réponses aux questions du Google Doc : 3 points

Attention, les pénalités suivantes peuvent s'appliquer sur la note obtenue :

— 5 points : le programme ne compile pas.

— 2 points : le programme rendu contient des messages d'avertissement.

— 2 points : mauvaise présentation du rapport. Il s'agit de conserver la structure du rapport et de le compléter.

— 1 point : orthographe, construction des phrases...

6 Qualités d'un raffinage

6.1 Règles

1. Un raffinage doit être bien présenté (indentation).

```
Ri : Comment « Action complexe » ?
     actions...
     liées par des...
     structures de contrôle
```

2. **Rappel :** Un raffinage explique **comment** réaliser une action (ou un expression) complexe sous forme d'une **décomposition** en actions liées par des structures de contrôle (séquence, conditionnelle, répétition).
3. Le raffinage d'une action (sous-actions et structures de contrôle) doit décrire complètement cette action.
4. Le raffinage d'une action ne doit décrire que cette action.
5. Les structures de contrôle sont celles du langage algorithmique en respectant les normes de présentation vues en cours.
6. Éviter les structures de contrôle déguisées (si, tant que) : les expliciter en utilisant la structure de contrôle du langage algorithmique (Si, TantQue...)
7. Une action complexe commence par un verbe à l'infinitif
8. Une expression complexe décrit la valeur de cette expression
9. Attention : dans une séquence, l'exécution d'une action complexe ne commencera que lorsque l'action précédente sera terminée.

6.2 Compréhension

1. Le vocabulaire utilisé doit être précis et clair.
2. La formulation d'une action (ou une expression) complexe doit être concise et précise. Tout ne peut pas être écrit dans l'intitulé de l'action mais l'essentiel doit être présent et permettre d'identifier les éléments correspondants dans le cahier des charges ou la spécification.
3. Expliciter les données manipulées par une action permet de la préciser (flot de données).
4. Utiliser une contrainte (entre accolades) entre deux actions permet de préciser les obligations de l'action précédente et les hypothèses pour l'action suivante. Par exemple :

```
1   Demander un numéro de mois      Mois : out Entier
2   { Mois >= 1 ET Mois <= 12 }
3   ...
```

5. Chaque niveau de raffinage doit apporter suffisamment d'information (mais pas trop). Il faut trouver le bon équilibre ! Idéalement, on introduit plusieurs actions complexes qui devraient pouvoir être décomposées par des personnes différentes.
6. Les actions introduites devraient avoir un niveau d'abstraction homogène.
7. Ne pas utiliser de formulation « Comparer » car si on compare, c'est pour faire quelque chose ! On n'a pas exprimé l'intention mais le moyen.
8. Il ne devrait y avoir qu'une structure de contrôle (hors séquence) par raffinage.
9. On doit comprendre l'objectif d'une action complexe juste en lisant son intitulé et les données qu'elle manipule sans avoir à lire sa décomposition !

6.3 Remarque

— Certaines de ces règles sont subjectives !
— L'important est de pouvoir expliquer et justifier les choix faits

7 Vérification d'un raffinage

7.1 Nom des actions

— A-t-on utilisé des verbes à l'infinitif pour nommer les actions ?

7.2 Action et sous-actions

On appelle *sous-action* une action qui apparaît dans la décomposition d'une action complexe.

1. Pour être correct, un raffinage doit répondre à la question « COMMENT ? » !
2. Vérifier l'appartenance d'une action A au raffinage d'une action C en demandant « POURQUOI A ? ». Si la réponse n'est pas C :
   — soit on a identifié une action complexe intermédiaire
   — soit A n'est pas à sa place (ne fait pas partie des objectifs de C)

7.3 Flots de données

Utiliser les flots de données pour vérifier :

1. les communications entre niveaux :
   - les sous-actions doivent produire les résultats de l'action ;
   - les sous-actions peuvent (doivent) utiliser les entrées de l'action.
2. l'enchaînement des actions au sein d'un niveau :
   - une donnée ne peut être utilisée (**in**) que si elle a été produite (**out**) par une action précédente (ou si elle était en **in** de l'action complexe en cours de décomposition).

7.4 Raffinages et variables

1. Ne pas oublier de faire apparaître les flots de données.
2. On ne déclare pas une variable dans un raffinage. Elles apparaissent seulement dans les flots de données. Ensuite, on peut les rassembler dans un dictionnaire des données et, dans le cadre de ce mini-projet, elles seront déclarées comme variables locales du programme principal.