Mini-projet 1 : Précisions et compléments

Ce document décrit les attendus concernant les raffinages de manière générale, précise quelques points de vigilance sur le mini-projet ainsi que quelques compléments.

1 Qualités d'un raffinage

1.1 Règles

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

```
1 Ri : Comment « Action complexe » ?
2     actions...
3     liées par des...
4     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.

1.2 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 !

1.3 Remarque

— Certaines de ces règles sont subjectives !

— L'important est de pouvoir expliquer et justifier les choix faits

2 Vérification d'un raffinage

2.1 Nom des actions

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

2.2 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)

2.3 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).

2.4 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.

3 Compléments sur le sujet du mini-projet.

Levons quelques ambiguïtés dans le sujet du mini-projet et apportons quelques précisions...

1. Quand on demande à l'utilisateur « On continue ? », l'utilisateur répond avec un seul caractère. S'il répond 'n' ou 'N' le programme se termine, sinon le programme lui demande à nouveau une table à réviser... Le message affiché pourrait donc être « On continue (o/n) ? ».
2. Le programme peut poser 10 fois la même multiplication (par exemple 7 * 3).
3. Quand on signale une hésitation sur une table, c'est que l'utilisateur a mis du temps pour répondre à une multiplication, peu importe s'il a donné une mauvaise réponse à cette question ou aux autres.
4. On doit vérifier que la table que demande de réviser l'utilisateur est bien comprise entre 1 et 10. Si ce n'est pas le cas, il faut lui expliquer pourquoi on lui demande à nouveau la table.
5. On considérera que l'utilisateur entre toujours des entiers quand on lui demande une table ou le résultat d'une multiplication. Le programme ne sera donc pas robuste s'il saisit des caractères quelconques.

4 Points de vigilance sur le Mini-projet

1. Il faut respecter la présentation des raffinages utilisée en cours.

2. Il faut respecter les règles rappelées ci-avant.
3. On doit toujours utiliser la structure de contrôle adaptée : dans le langage algorithmique et en Ada, il y 2 conditionnelles et 3 répétitions. Il faut utiliser la bonne !
4. Avoir du code redondant est un défaut important. Il faut donc le limiter au maximum.
5. Dès le R1 on doit voir que l'utilisateur pourra réviser plusieurs tables.
6. Même si un niveau de raffinage n'a que des actions élémentaires, il peut être utile de le faire apparaître pour montrer la structure de contrôle utilisée.

5 Autres points

1. Il est important de lire l'intégralité des documents fournis, en particulier tout le fichier `LISEZ-MOI.txt`.
2. Dans le fichier `LISEZ-MOI.txt`, les temps indiqués ne sont bien sûr pas pris en compte dans la notation. En revanche, s'ils ne sont pas renseignés, une pénalité sera appliquée.
3. Le programme rendu doit compiler. Sinon, vous aurez une forte pénalité (5 points).
4. Un travail rendu hors délai sera réputé non rendu et donc, non corrigé. Vous pouvez déposer plusieurs versions (c'est fortement conseillé et vous y êtes incités en TP puisqu'on vous demande de pousser vos modifications après chaque question) et seule la dernière version (la plus aboutie normalement) sera évaluée. N'hésitez donc pas à pousser vos modifications régulièrement !
5. Comment être sûr que mon travail a bien été poussé sur SVN ?
   (a) Faire « `gnatclean *.adb` » pour supprimer les fichiers intermédiaires, « `svn update` » pour être à jour avec la dernière version du dépôt central puis « `svn status` ». Cette dernière commande indique l'état des fichiers locaux. Rien ne devrait être affiché mis à part les fichiers qui n'ont pas à être poussés (exécutable, fichiers intermédiaires, etc.), ils sont précédés d'un « `?`. » S'il y a des « `A` » ou des « `M` » en première colonne, c'est que les modifications n'ont pas été poussées.
   (b) On peut aussi mettre l'URL du dépôt dans un navigateur internet et vérifier que les fichiers sont bien là.
   (c) On peut enfin créer une nouvelle copie du dépôt dans un nouveau dossier (par exemple `/tmp`), vérifier les fichiers qui remontent et en profiter pour les compiler (on est ainsi sûr que la version déposée compile) et les exécuter.