Licence 2 — Université de Nantes
Algorithmique et Structures de données 1 — 2016/2017

Feuille de travaux pratiques n° 1
Doxygen, assertions et tests unitaires

2 heures

L'objectif de ce TP est de mettre en œuvre la démarche de fiabilité des algorithme lors de leur implémentation en programmes C++. Cela passe par l'ajout de commentaires stipulant la spécification de chaque routine et du programme principal, l'ajout d'*assertions* pour vérifier les pré-post-conditions à l'exécution du programme, et une démarche de tests unitaires implémentant le jeu de tests défini pour chaque routine.

1 Documentation de code

Depuis que vous apprenez à programmer, on vous l'a répété cent fois. Voici la cent unième : *un programme doit être bien commenté !* Outre les commentaires dans le corps du programme qui précisent les détails techniques de son implémentation, la spécification du programme, c'est à dire le rôle, les entrées/sorties et les pré-/post-conditions de l'algorithme qu'il implémente, doit également être fournie en commentaires. Il est également utile de fournir en commentaires le jeu de tests représentatif retenu, ainsi que le résultat de son exécution.

Ces nombreuses informations permettent alors d'engendrer la *documentation du code source* du programme. En C++, *Doxygen* est l'un des outils permettant d'engendrer automatiquement cette documentation. Pour reconnaître les différents commentaires, cet outil emploie quelques *balises* simples. Il est capable d'engendrer la documentation dans de nombreux formats, y compris sous forme de pages HTML comme nous allons le voir.

Considérez le programme simple donné dans le fichier `tp1-1.cpp` sur Madoc et qui implémente l'algorithme de calcul de valeur absolue vu en cours.

Les commentaires de ce programme destinés à Doxygen sont organisés comme suit :

- les zones de commentaires prises en compte par Doxygen doivent commencer par `/**` et terminer par `**/` ; les lignes blanches dans ces zones sont importantes pour la mise en page de la documentation.
- préambule du fichier (e.g., lignes 1–7 de `tp1-1.cpp`) ; cette zone de commentaires doit comporter les quatre informations suivantes :
  — le nom du fichier précédé de la balise `@file`
  — le nom de l'auteur du fichier précédé de la balise `@author` ; plusieurs auteurs peuvent être indiqués
  — les dates de création/modification du fichier précédées de la balise `@date`
  — une brève description du contenu du fichier précédée de la balise `@brief`

- commentaire de chaque routine (e.g., lignes 15–37 de `tp1-1.cpp`) ; cette zone doit comporter les six informations suivantes :
  — une brève description de la routine précédée de la balise `@brief` ;
  — son rôle précédé de la balise `@b Role` ;
  — ses entrées sous forme de liste précédée de la balise `@b Entrées` ;
  — ses sorties sous forme de liste précédée de la balise `@b Sorties` ;
  — sa pré-condition précédée de la balise `@pre` ;
  — sa post-condition précédée de la balise `@post` ;
  elle peut également comporter les informations suivantes lorsque c'est pertinent :
  — le jeu de tests et ses résultats précédés de la balise `@test`. Afin de rendre plus lisible les tests, ceux-ci sont organisés dans un tableau comportant 4 colonnes, *Entrées*, *Sorties* (attendues), *Justification* et *Résultat* ;
  — une explication détaillée (algorithme, ...) sous forme d'un paragraphe de texte séparé du reste des informations par une ligne blanche.

Afin d'engendrer la documentation correspondante, il faut utiliser la commande `doxygen` dans un terminal. Celle-ci s'applique en trois étapes :

1. Dans le terminal, taper la commande
   ```
   doxygen -g doxygen.config
   ```
   qui engendre un fichier de configuration `doxygen.config` ; celui-ci contient les réglages par défaut pour la génération de la documentation du code source.
2. Dans un éditeur de texte, modifier les réglages dans le fichier `doxygen.config`. Il est *indispensable* de régler les paramètres suivants :
   — `OUTPUT_LANGUAGE` : English par défaut, French possible ;
   — `INPUT` : indéfini par défaut ; préciser le chemin vers le code source à documenter, e.g., `/home/~jerm ann-c/X3I0010/tp1-1.cpp`. Vous pouvez également engendrer la documentation de tous les fichiers source d'un même dossier en précisant seulement un dossier et en réglant le paramètre `FILE _PATTERNS` à la valeur `*.cpp`.
   — `SOURCE_BROWSER` : NO par défaut, YES pour permettre l'accès au code source directement dans la documentation
   Vous pouvez bien entendu changer d'autres réglages, par exemple pour définir les formats de documentation souhaités (par défaut HTML et LaTeX). Le fichier de configuration contient des indications (en anglais) sur chaque réglage possible. *N'oubliez pas de sauvegarder vos modifications !*
3. Dans le terminal, taper la commande
   ```
   doxygen doxygen.config
   ```
   qui engendre la documentation dans les formats demandés, chacune dans un dossier nouvellement créé. La documentation HTML est créé dans le dossier `html` et peut être consultée en ouvrant le fichier `index.html` dans un navigateur.

Notez que les étapes 1 et 2 ne sont à faire qu'une seule fois pour la documentation d'un fichier source (ou dossier source) donné. Plus d'information concernant Doxygen est disponible dans le manuel en ligne `http://www.stack.nl/~dimitri/doxygen/manual/`.

Dorénavant, vous commenterez vos programmes en suivant les indications données ci-avant afin de pouvoir engendrer la documentation de votre code source.

Exercice 1.1

1. Engendrez la documentation HTML du fichier `tp1-1.cpp` fourni. Consultez-la.
2. Modifiez les commentaires du programme et engendrez-la de nouveau pour observer les modifications.
3. Complétez la documentation de la procédure `inserer` du fichier `tp1-2.cpp` également fourni sur Madoc.
4. Engendrez une documentation qui comporte ces deux fichiers à la fois.

2 Assertions

La bibliothèque standard C++ propose un mécanisme d'assertion utile pour mettre en œuvre la validation automatique des pré- et post-conditions de votre algorithme lors de l'exécution de son implémentation en C++. Ceci vous aidera à valider vos programmes lors de l'exécution des jeux de tests représentatifs : si une assertion n'est pas vérifiée au cours de l'exécution du programme, celle-ci sera interrompue et un message d'erreur indiquant le fichier, la ligne et l'assertion concernés, sera affiché.

Ce mécanisme est rendu disponible par l'inclusion du fichier `assert.h` en préambule de vos programmes et bibliothèques :

```
#include <cassert>
```

Ce fichier définit une seule fonction nommée `assert` qui prend en paramètre une expression booléenne. Lorsque l'expression s'évalue à Vrai au moment de l'exécution du programme, celui-ci se déroule normalement. Lorsqu'elle s'évalue à Faux, il est interrompu et le message d'erreur est émis.

Dorénavant, vous insérerez des appels à `assert` validant les pré- et post-conditions en début et fin, respectivement, de vos programmes et routines (toujours avant le return, jamais après !).

Exercice 1.2

1. Compilez et exécutez les programmes fournis.
2. Modifiez les entrées de façon à violer certaines conditions et exécutez à nouveau ces programmes.
3. Étudiez les assertions définies dans le programme `tp1-2.cpp` et comparez les aux pré-/post-conditions données dans sa documentation.
4. L'assertion de post-condition vous paraît-elle fiable ?
5. Quelles limites générales en déduisez-vous pour l'utilisation d'assertions ?

3 Tests unitaires

La définition d'un jeu de tests représentatif permet de vérifier qu'un algorithme semble fonctionner correctement. Cela est normalement établi par la simulation de l'algorithme sur chacun des tests du jeu de tests. Cependant, cette vérification est bien souvent longue et fastidieuse, et n'assure pas, quand bien même l'algorithme est correct, que sa transcription en programme C++ l'est aussi. C'est pourquoi un jeu de tests doit aussi être programmé. Cela le rend automatisable et permet de vérifier, à chaque modification d'un programme, que chacune de ses fonctionnalités continue de se comporter correctement.

Un jeu de tests est défini pour chaque routine d'un algorithme, et chacune doit être testée séparément : c'est la notion de *test unitaire*. Afin de les automatiser, il faut associer à chaque routine un programme principal dont le seul but est d'exécuter son jeu de tests.

Dorénavant, vous écrirez des programmes de test unitaire pour chaque jeu de tests de chaque routine de vos programmes.

Exercice 1.3

Le fichier `tp1-2.cpp` fourni sur Madoc fourni un programme (partiel) de test unitaire de la procédure d'insertion dans un tableau vue en cours.

1. Étudiez le code de ce programme via sa documentation engendrée par Doxygen.
2. Observez comment est défini et exécuté le test proposé dans la fonction principale (`main`).
3. Ajoutez des tests fonctionnels et structurels, même non-représentatifs, recompilez et exécutez à nouveau le programme.
4. Complétez les commentaires Doxygen pour qu'ils incluent les tests supplémentaires réalisés et leur résultat.
5. Engendrez à nouveau sa documentation et vérifiez que vos nouveaux tests y apparaissent correctement.

4 Application

Exercice 1.4

Vous pouvez à présent vous exercer en programmant quelques-uns des algorithmes vus en TD en respectant les consignes données dans cette feuille : commentaires de documentation, assertions, test unitaires.