Saé S1.01 : implémentation d'un besoin client

Documenter le code source – générer une documentation

Introduction

« Commenter son code, c'est un cadeau que vous envoyez dans le futur ».
Quentin Busuttil (BUZUT.net)

Une question revient souvent chez le développeur débutant : *« pourquoi commenter mon programme alors que c'est moi qui l'ai écrit, je sais donc bien ce que j'ai programmé ? »*. Et pourtant !
De même qu'il est indispensable de donner des noms explicites aux constantes et aux variables d'un programme, il est absolument indispensable d'insérer des commentaires afin d'expliquer le code que l'on a écrit et pourquoi on l'a écrit.

Au cours de cette séance nous allons :
- revenir sur la nécessité de commenter les programmes,
- normaliser les commentaires d'un programme,
- générer une documentation à partir de ces commentaires.

1 Pourquoi commenter son code ?

Exercice 1

Ouvrez le progamme `prog1.c` *fourni sur Moodle et analysez ce code source.*

Que fait ce programme ?

À quoi sert le type `typTab` *?*

Quel est le rôle de la constante `TMAX` *?*

Que réalise la fonction `ide()` *?*

Dans la procédure `ord(…)` *à quoi sert la conditionnelle* `if (N!=-1)` *?*

Quel est l'inconvénient de la fonction `ana(…)` *?*

Exercice 2

Répondez exactement aux mêmes questions que précédemment mais en utilisant cette fois le programme `prog2.c`

Un programme bien commenté facilite le travail de ceux qui voudraient lire, comprendre, utiliser, voire maintenir votre travail, à commencer par vous-même.

La plupart du temps, vous ou un de vos collègues aurez besoin de relire un programme que vous avez écrit il y a quelque temps. En utilisant les commentaires, vous ou votre collègue gagnerez du temps pour vous rappeler/comprendre ce que fait le programme et pour le corriger (vous ne serez pas obligé de vous "replonger dans le code").

2 Les commentaires dans un programme C

Le langage C permet bien sûr l'ajout de commentaires à un programme. Il propose même deux moyens pour cela.

2.1 Le commentaire « ligne »

Pour ajouter un commentaire sur une ligne, ou à la fin d'une ligne de code, il faut le faire précéder de `//`

Exemple

```
if (a < 0) {
 // Traite le cas d'erreur
 printf("Erreur\n");
}
```

ou

Exemple

```
if (bof > 1) {
     return TRUE; // cas particulier
} else {
     return isPrime(a);  // Attention :que pour a impair
}
```

2.2 Le commentaire « bloc »

Pour ajouter un commentaire sur plusieurs lignes (un « bloc ») il faut commencer le bloc par **`/*`** et le terminer par **`*/`**

Exemple

```
/* fonction estValide()
 cette fonction permet de contrôler
 que le dénominateur est non nul
 elle renvoie un booléen */

bool estValide(int n) {
...
}
```

ou

Exemple

```
/*

 fonction estValide() : cette fonction permet de contrôler
 que le dénominateur est non nul.
 Elle renvoie un booléen

*/

bool estValide(int n) {
...
}
```

Remarquez bien qu'on peut utiliser un commentaire « bloc » pour commenter une seule ligne.

Exemple

```
if (a < 0) {

 printf("Erreur\n");    /* Traite le cas d'erreur */

}
```

Sachez enfin que l'imbrication de commentaires n'est pas possible.

Exemple

```
 /* Ceci  /*Traite le cas d'erreur */  n'est pas permis */

if (a < 0) {

 printf("Erreur\n");

}
```

3 Que faut-il commenter ?

Pour rester efficace, le commentaire doit être pertinent. En effet, il ne faut pas noyer le lecteur avec du commentaire insignifiant, inutile, qui lui ferait perdre du temps. Dans les deux exemples qui suivent, les commentaires n'ont strictement aucun intérêt.

Exemple

```
int total; // variable entière
```

ou encore

Exemple

```
i = i + 1;  // on incrémente i de 1
```

Dans ces deux exemples le commentaire n'apporte aucune information nouvelle, rien d'autre que ce que le lecteur ne puisse voir dans le code.

En commentant son code il faut se mettre en tête qu'on est en train d'expliquer ce que l'on fait à un autre développeur qui maitrise le langage, donc commenter les parties du programme qui nécessitent des précisions supplémentaires pour le futur lecteur.

Une bonne pratique est de commenter au minimum ces éléments :

- **le programme** : indiquer des informations générales notamment ce qu'il est censé faire,
- **chaque type de données** : la raison d'être de chaque type de données (tableau, structure cartésiennes, fichier, nouveau type, énumération, etc.) doit être expliquée,
- **chaque procédure ou fonction** : ce qu'elle est censée faire, le rôle de ses paramètres et, pour une fonction, le rôle de son résultat,
- **toute constante ou variable** qui joue un rôle important dans une fonction,
- **tout autre élément** non trivial nécessaire à la compréhesion du programme : une technique de programmation particulière, le traitement d'un cas particulier, une condition importante, etc.

Enfin il ne faut pas hésiter, surtout si votre programme est long, à le structurer en mettant en évidence les différentes parties, grâce à des commentaires du style :

```
/*****************************************************
 * FONCTIONS UTILITAIRES POUR MANIPULER LES PRODUITS *
 *****************************************************/
```

ou encore :

```
/*****************************************************
 *                PROGRAMME PRINCIPAL                *
 *****************************************************/
```

4 Doxygen

Doxygen (pour DOCS GENerator) est un outil d'auto-documentation pour le langage C (notamment, car il est prévu pour beaucoup d'autres langages). Il permet de générer automatiquement une documentation à partir des commentaires d'un programme. L'intérêt, outre le gain de temps, est d'assurer une certaine cohérence entre le programme et la documentation.

Pour générer automatiquement une documentation il faut bien sûr que les commentaires puissent être interprétés correctement et donc qu'ils **respectent un format particulier**.

4.1 Une nouvelle façon de commenter

Avec Doxygen, le principe est d'insérer des blocs de commentaires sous ce format (remarquez les deux « étoiles » de la première ligne) :

```
/**
*
*   Ceci est un commentaire multiligne
*
*/
```

ou bien sous ce format (remarquez ici le **!** de la première ligne) :

```
/*!
*
*   Ceci est un commentaire multiligne
*
*/
```

Il est aussi possible de commenter une seule ligne (mais toujours avec les deux "*" ou le " !") :

```
int x ; /** Ceci est le commentaire d'une ligne */

int y ; /*! Ceci est le commentaire d'une ligne */
```

4.2 Les balises Doxygen

Doxygen permet d'ajouter aux commentaires des éléments particuliers en utilisant des balises. Ces éléments seront interprétés d'une certaine façon lors de la génération de la documentation.

Exemple : avec la balise ***\brief***

```
/**
*
* \brief Programme de tests
*
*/
```

Voici les principales balises que nous utiliserons (vous retrouverez l'ensemble des balises sur www.doxygen.nl/manual/commands.html) :

`\brief`	:	pour donner une description courte, un résumé
`\struct`	:	pour documenter une structure cartésienne
`\fn`	:	pour documenter une procédure ou fonction
`\var`	:	pour documenter une variable
`\def`	:	pour documenter un #define
`\typedef`	:	pour documenter la définition d'un type
`\param`	:	pour documenter un paramètre de fonction
`\author`	:	pour donner le nom de l'auteur
`\return`	:	pour documenter les valeurs de retour d'une fonction
`\date`	:	pour indiquer une date
`\version`	:	pour indiquer une version de programme
`\see`	:	pour indiquer une référence à un autre élément

A NOTER : les balises peuvent aussi s'écrire avec le caractère @. On peut donc indifféremment utiliser `\brief` ou bien `@brief`, utiliser `\struct` ou bien `@struct`, etc.

4.3 Mise en place de la documentation

On l'a vu, il est impératif de commenter votre code source. Ce qui suit est une proposition de mise en place de commentaires qui vous contraint à expliquer ce que vous faites et qui vous permettra d'obtenir une documentation digne de ce nom. Elle concerne des informations générales sur le programme, sur les données qu'il utilise et sur chaque procédure/fonction qu'il contient. Bien sûr, vous pouvez commenter en plus toute partie de votre programme que vous jugez nécessaire.

4.3.1 Informations d'entête

Votre programme doit commencer par un bloc d'information contenant au minimum :

- une brève description de ce que réalise le programme,
- le nom de l'auteur du programme,
- sa date de création,
- sa version.

Exemple

```
/**
*
* \brief Programme de manipulation de chaînes de caractères.
*
* \author LE GUIC
*
* \version 1.0
*
* \date 13 septembre 2021
*
* Ce programme propose plusieurs opérations de manipulation
* de chaînes de caractères où les chaînes sont implémentées
* par des listes chainées de caractères.
*
*/
```

4.3.2 Documentation des structures de données

Chaque donnée ou structure de données du programme doit être documentée, que ce soit une constante, une variable globale, un tableau, une structure cartésienne, une énumération, un fichier, etc.

Au-dessus de la donnée ou structure de données, on ajoutera un bloc de commentaire qui pourra contenir la balise adéquate (`\var`, `\typedef`, `\struct`, `\def`, etc.).

Exemple : documenter une constante

```
/**
*
* \def TMAX
*
* \brief constante pour la Taille Max d'un tableau.
*
*/
```

Exemple : documenter un nouveau type

```
/**
*
* \typedef typTab
*
* \brief type tableau de TMAX caractères
*
* Le type typTab sert de stockage provisoire pour ordonner
* les éléments (caractères) d'une liste.
*
*/
```

Cas particulier des structures cartésiennes

Chaque champ de la structure peut recevoir un commentaire. Remarquez le caractère « < » dans le commentaire de chaque champ.

Exemple : cas d'une structure cartésienne

```
/**
*
* \struct fiche
*
* \brief structure des informations sur un patient
*
*/
typedef struct {
    chaine15 no;    /*!< n° de sécu du patient */
    chaine15 n;     /*!< son nom */
    chaine15 p;     /*!< son prénom */
    int a;          /*!< son âge */
} fiche;
```

4.3.3 Documentation d'une procédure/fonction

On utilisera la balise **\fn** pour documenter une fonction (ou une procédure, mais en langage C une procédure est une fonction qui ne retourne rien) :

`\fn` *<entête de la fonction>*

On précisera le rôle de chaque paramètre grâce à la balise **\param** :

`\param` *<rôle du paramètre>*

Enfin, on précisera le rôle du résultat avec la balise **\return** :

`\return` *<rôle du résultat>*

Exemple : bloc placé au dessus de la définition de la fonction estV().

```
/**
*
* \fn bool estV(chaine ch)
*
* \brief Fonction qui indique si une chaine est vide.
*
* \param ch : la chaine à tester.
*
* \return true si la chaine est vide, false sinon.
*
* Consiste à vérifier si la tête de liste est à NULL.
*
*/
```

Résumé du rôle de la fonction

Commentaire détaillé sur le rôle de la fonction

Exercice 3

Reprenez le progamme `prog2.c` *fourni sur Moodle. L'exercice consiste à adapter les commentaires façon Doxygen.*

1) Ajoutez à la documentation d'entête la date d'aujourd'ui et la version (1.0 par exemple), à l'aide des balises adéquates.

2) En utilisant la balise adéquate, documentez la constante ***TMAX****.*

3) En utilisant les balises adéquates, documentez la structure ***element****, en précisant le rôle de chaque champ (*`Lettre` *correspond au caractère mémorisé et* `svt` *est un pointeur sur le caractère suivant de la liste chainée).*

4) En utilisant les balises adéquates, documentez la fonction ***ana(…)****, en précisant le rôle de la fonction, le rôle de chaque paramètre et celui du résultat.*

5 Générer la documentation

Doxygen permet de générer une documentation relative à un programme, ou à un ensemble de programmes. Pour cela, il faut créer un **fichier de configuration** qui contient un ensemble de paramètres. C'est grâce à ce fichier que l'on "prépare" la génération de la documentation.

5.1 Générer un fichier de configuration

La première chose à faire est d'établir le fichier de configuration. Heureusement, Doxygen permet de générer un fichier template que vous n'aurez plus qu'à modifier.

Pour générer le fichier de configuration, la commande est :

```
doxygen -g <nom du fichier de configuration>
```

Exercice 4

Pour générer un fichier de configuration, tapez dans votre console :

```
doxygen -g maConfig
```

5.2 Étudier le fichier de configuration

Ouvrez (par exemple avec VSC) le fichier de configuration que vous venez de générer. Il contient normalement :

- 2495 lignes,
- 16 "sections",
- 268 paramètres.

Une "section" est identifiable par son titre encadré par deux lignes de tirets.

Exemple

```
#---------------------------------------------------------------
#  Project related configuration options
#---------------------------------------------------------------
```

Un paramètre est identifiable par son nom, suivi du signe **=**, et éventuellement de sa valeur. En effet certains paramètres ont des valeurs par défaut.

Exemples

```
PROJECT_NAME     =

DOXYFILE_ENCODING      = UTF_8
```

Important : chaque paramètre est précédé d'un commentaire qui en explique le rôle. Alors oui, c'est en anglais.

5.3 Paramétrer la configuration

Rassurez-vous, nous n'allons pas passer en revue tous les paramètres du fichier de configuration. Par contre, pour générer une documentation minimale il est nécessaire d'en définir quelques-uns.

*Dans la première section, intitulée "**Project related configuration options**", définissez les paramètres suivants.*

- le nom de votre documentation ou de votre projet logiciel.

ex : **PROJECT_NAME = "Ma Documentation"**

- un résumé du contenu

ex : **PROJECT_BRIEF = "Exemple de documentation Doxygen"**

- le répertoire dans lequel va être générée la documentation. S'il n'existe pas il sera créé, sinon il sera écrasé.

ex : **OUTPUT_DIRECTORY = "Docu/MaDoc"**

- la langue dans laquelle la documentation va être générée (`English` par défaut).

ex : **OUTPUT_LANGUAGE = French**

- pour adapter la documentation générée à un programme C.

OPTIMIZE_OUTPUT_FOR_C = YES

*Dans la deuxième section, intitulée "**Build related configuration options**", définissez les paramètres suivants.*

- indiquer à Doxygen de créer une rubrique pour tous les éléments du programme

EXTRACT_ALL = YES

- pour que les résumés soient triés (liste des résumés de fonction par exemple)

SORT_BRIEF_DOCS = YES

- pour que dans les informations de détail soient triées (liste des détails sur les fonctions par exemple)

SORT_MEMBER_DOCS = YES

*Dans la quatrième section, intitulée "**Configuration options related to the input files**", indiquez le nom du fichier C dont il faut générer la documentation.*

ex : **INPUT = prog2.c**

5.4 Choisir le type de documentation : **document texte (format RTF)**

*La neuvième section du fichier de configuration, intitulée "**Configuration options related to the RTF output**", est consacrée à la génération de documentation sous forme d'un fichier texte, au format RTF.*

Commencez par indiquer que vous souhaitez une génération de ce type :

```
GENERATE_RTF                = YES
```

Indiquez ensuite le répertoire qui contiendra cette documentation. Ce sera un sous-répertoire de l' **OUTPUT_DIRECTORY** indiqué dans la première section.

ex : **RTF_OUTPUT** `= "Doc RTF"`

Exercice 5

Votre fichier de configuration est prêt. Depuis votre console, lancez la commande :

```
doxygen maConfig
```

Normalement un fichier .rtf a été généré dans le répertoire indiqué dans la configuration (dans l'exemple ce serait le répertoire `Docu/MaDoc/Doc RTF`).

Ouvrez le fichier `.rtf` avec Libre Office et analysez-le. Faîtes le lien entre les commentaires que vous avez insérés dans le programme prog2.c et la documentation qui a été générée.

Exercice 6

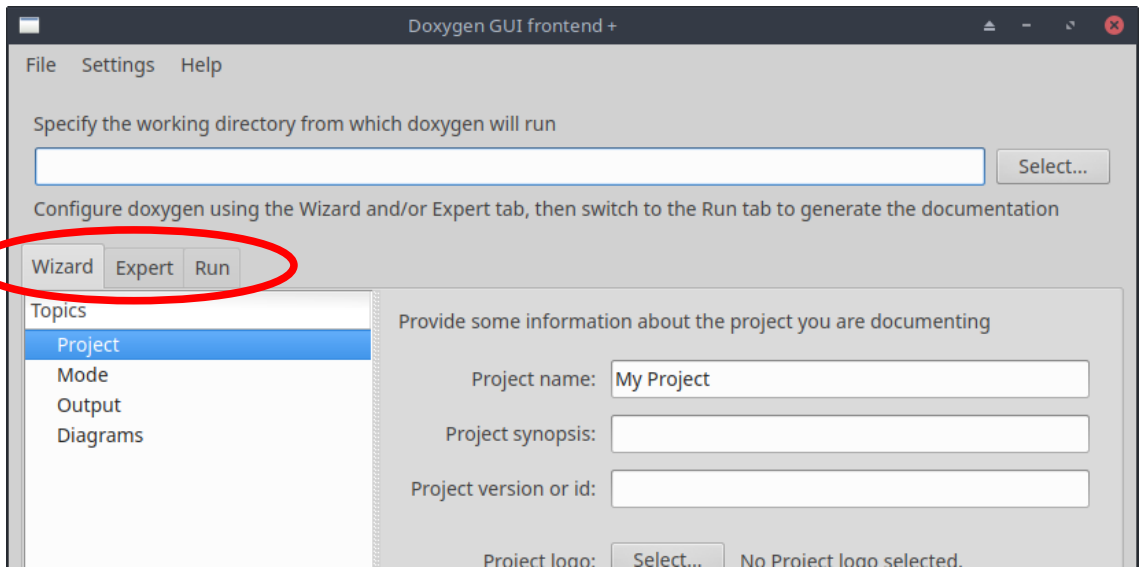
Modifiez votre fichier de configuration (ou créez-en un autre) afin de générer la documentation du programme `prog3.c`, disponible sur Moodle.

6 Doxywizard

Vous en conviendrez, gérer "à la main" un fichier de configuration comme précédemment n'est pas chose aisée et peut être source d'erreur. L'application **Doxywizard** propose une interface graphique qui permet de paramétrer de manière plus conviviale un fichier de configuration Doxygen.

6.1 Les étapes de configuration

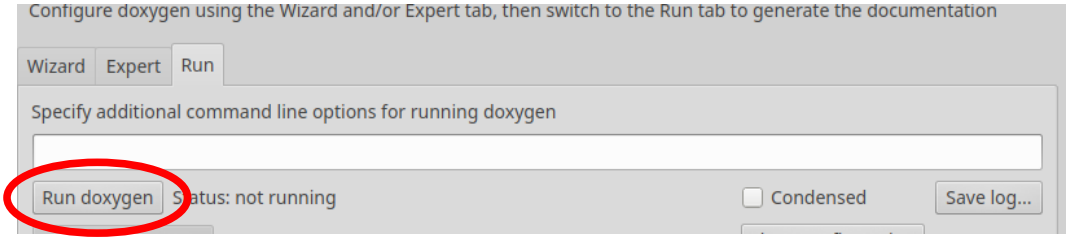
Quand vous lancez l'application Doxywizard, cette fenêtre principale apparaît avec ses trois onglets que nous allons détailler :



Vous avez la possibilité de configurer le fichier Doxygen :

- soit en mode rapide (onglet **Wizard**),
- soit en mode avancé (onglet **Expert**).

Ensuite vous utiliserez l'onglet **Run** pour lancer la génération de la documentation (voir bouton *Run doxygen* sur l'image ci-dessous).



6.2 Le mode rapide (**Wizard**)

Ce mode permet d'indiquer rapidement les informations minimales pour la génération de la documentation.

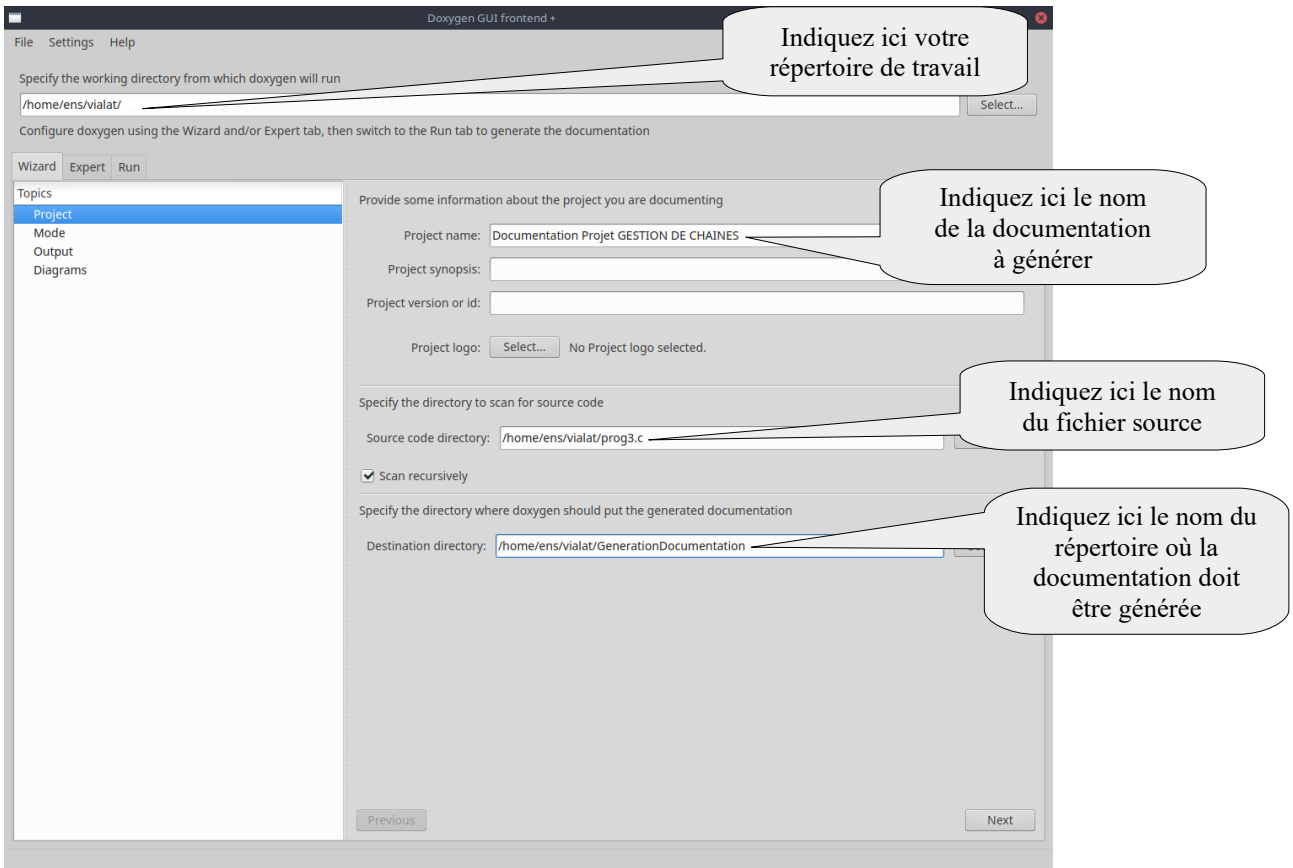
- L'item *Project* permet par exemple de fournir le nom du projet et les répertoires source et destination.
- L'item *Mode* permet notamment préciser le langage concerné à des fins d'optimisation.
- L'item *Output* permet d'indiquer dans quel format on souhaite générer la documentation (ex : HTML, LateX, RTF, etc).
- L'item *Diagrams* permet d'indiquer s'il faut intégrer des schémas à la documentation.

Exercice 7

Après avoir indiqué tout en haut de la fenêtre le nom de votre répertoire de travail (ici le répertoire qui contient votre programme source), vous indiquerez dans l'onglet Wizard :

- le fichier source (*Source code directory*) et le répertoire dans lequel vous souhaitez générer la documentation (*Destination directory*) comme montré sur l'image suivante,
- une optimisation pour le langage C,
- une génération en RTF ;
- pas de diagrammes.

Lancez ensuite la génération de la documentation RTF.



6.3 Le mode **Expert**

Ce mode "avancé" permet un paramétrage plus spécifique et plus complet. Les items et les paramètres sont en effet plus nombreux. Néanmoins, on se contentera ici des items **Project**, **Input** et **HTML**.

Exercice 8

Vous allez maintenant générer une documentation HTML. Dans l'onglet Expert, vous indiquerez les paramètres suivants :

- item **Project** :
  - le nom du projet (ou bien garder celui saisi précédemment)
  - le répertoire de destination (ou bien garder celui saisi précédemment)
  - la langue (choisir French par exemple)
  - et l'optimisation pour langage C si ce n'est pas déjà fait
- item **Build** :
  - cochez les options EXTRACT_ALL, SORT_BRIEF_DOCS et SORT_MEMBER_DOCS
- item **Input** :
  - indiquer le nom du programme source dans INPUT
- item **HTML** :
  - pensez bien sûr à cocher GENERATE_HTML
  - indiquez le nom du sous-répertoire qui contiendra la documentaiton (par défaut c'est *html*)
  - cochez aussi les options HTML_DYNAMIC_MENUS, GENERATE_TREEVIEW et SEARCHENGINE

Lancez ensuite la génération puis étudiez la documentation **HTML** ainsi générée (ouvrir le fichier index.html).

6.4 Aperçu du fichier de configuration

Vous pourrez constater qu'en cliquant sur le bouton *Show configuration* dans l'onglet **Run** vous verrez apparaître le contenu du fichier de configuration :