Programmation en language C - Projet

Licence informatique 2ème année

Université de La Rochelle

. Ce document est distribué sous la licence CC-by-nc-nd (https://creativecommons.org/licenses/by-nc-nd/4.0/deed.fr)

© 2020-2021 Christophe Demko

<christophe.demko@univ-lr.fr>

2020-2021_1

Table des matières

Liste des exercices

1 Consignes

1.1 Groupes

Le projet se fait par groupe de 4 à 6 étudiants et à rendre par un dépôt *moodle* le vendredi 18 décembre 2020 à 23h00. Il n'y aura pas de délai supplémentaire.

1.2 Langue utilisée

La langue utilisée dans le code et la documentation devra être exclusivement l'anglais. Si vous avez des difficultés dans la langue de Shakespeare, vous pourrez utiliser les traducteurs automatiques :

- https://www.deepl.com/translator
- https://translate.google.fr/

1.3 Nommage

1.3.1 Fichiers

- les noms de fichiers du langage C seront tous en minuscules et en anglais. S'ils sont composés de plusieurs mots, ils devront être séparés par un tiret (-) ;
- les fichiers contenant le code devront avoir l'extension `.c` ;
- les fichiers d'en-têtes (exportables) devront avoir l'extension `.h` ;
- les fichiers destinés à être inclus dans votre code mais non exportables devront avoir l'extension `.inc`

1.3.2 Types

Les noms de types devront faire commencer chaque mot qui les compose par une majuscule. Il n'y a pas de sous-tirets. Les structures et les énumérations devront commencer par un sous-tiret (_) pour ne pas les confondre avec les noms de types.

1.3.3 Macros

Les macros (avec ou sans arguments) s'écrivent tout en majuscule en séparant les mots par des sous-tirets (_).

1.3.4 Variables et fonctions

Les variables et les fonctions s'écrivent toutes en minuscules en séparant les mots par des sous-tirets.

1.4 Style

1.4.1 Marque d'inclusion unique

Chaque fichier d'en-tête devra posséder une marque permettant d'éviter les conséquences d'un fichier inclus plusieurs fois. Voir https://google.github.io/styleguide/cppguide.html#The__define_Guard

1.4.2 Ordre des inclusions

L'inclusion des fichiers d'en-tête devra respecter la logique suivante :

1. Inclusion du fichier directement lié au fichier `.c` qui l'inclut suivi d'une ligne vide ;
2. inclusion des fichiers d'en-tête du C standard suivis d'une ligne vide ;
3. inclusion des fichiers d'en-tête provenant d'autres librairies suivis d'une ligne vide ;
4. inclusion des fichiers d'en-tête du projet suivi d'une ligne vide ;
5. inclusion des fichiers d'inclusion (extension `.inc`)

1.4.3 Indentation

Le style d'indentation devra être celui préconisé par Google https://google.github.io/styleguide/cppguide.html#Formatting. L'utilitaire `clang-format` (https://clang.llvm.org/docs/ClangFormat.html) supporte le style `Google`.

Vous pourrez utiliser l'utilitaire `cclint` pour vérifier votre code.

1.5 Champs protégés

Les champs des structures seront protégés à la manière de la librairie `fraction` vue en travaux pratiques.

1.6 Documentation

La documentation sera générée avec l'outil `sphinx` et les fonctions seront documentées avec la norme de `doxygen`.

1.7 Tests

Des tests unitaires devront être implémentés, ils testeront chaque fonction et s'efforceront de vérifier que la mémoire est bien libérée au moyen de l'utilitaire `valgrind`.

Vous pourrez vous inspirer du projet https://github.com/chdemko/c-test.

D'une manière générale, toutes les options possibles décrites dans ce projet devront être implémentées.

2 Sujet

Le but du projet est de produire :

— une librairie permettant de représenter un nombre entier en toute lettres ;
— un logiciel capable d'afficher dans un dialecte donné la représentation d'un nombre.

Le projet devra fournir une documentation produite avec

```
$ make docs
```

Il pourra être installé avec

```
$ make install
```

Le logiciel devra s'appeler `spell-number` (et devra utiliser une librairie partagée de même nom) et devra prendre en argument le dialecte à utiliser et le nombre à épeler :

```
$ ./spell-number --fr_FR 4798221
quatre million sept cent quatre-vingt dix huit mille deux cent vingt-et-un
$ ./spell-number --fr_BE 4798221
quatre million sept cent nonante huit mille deux cent vingt-et-un
$ ./spell-number --fr_FR 4798222
quatre million sept cent quatre-vingt dix huit mille deux cent vingt deux
$ ./spell-number --fr_FR_african 4798222
quatre million sept cent quatre-vingt dix huit mille deux cent vingt-et-deux
$ ./spell-number --fr_FR 1700
mille sept cents
$ ./spell-number --roman 1700
MDCC
$ ./spell-number --fr_FR_date 1700
dix-sept cents
$
```

Comme on peut le voir dans les exemples précédents, chaque nombre peut être représenté différemment selon le dialecte utilisé. Dans les exemples, nous avons utilisé :

— `fr_FR` pour la langue française en France ;
— `fr_FR_african` pour l'usage de certaines exceptions dans les pays francophones en Afrique (notamment la liaison avec le *et* pour les unités différentes de *un*) ;
— `fr_BE` pour la langue française en Belgique ;
— `fr_FR_date` pour les dates en langue française en France ;
— `roman` pour les chiffres romains.

L'idée générale est d'arriver à trouver une structure générique de représentation d'un dialecte permettant de convertir n'importe quel nombre vers n'importe quel dialecte. La richesse des structures des dialectes devra permettre d'en ajouter de nouveaux. Votre librairie devra au moins implémenter les dialectes cités.

Historique des modifications

2020-2021_1 *Mercredi 30 septembre 2020*

Dr Christophe Demko <christophe.demko@univ-lr.fr>

— Version initiale