Programmation impérative (avancé)

Projet

Licence informatique 2ème année

Université de La Rochelle

La Rochelle Université

2023-2024_1

Les lignes commençant par $ représente une commande *unix*. Le $ ne fait pas partie de la commande proprement dite mais symbolise l'invite de commande.

Table des matières

1 Consignes

1.1 Groupes

Le projet se fait par groupe de maximum 6 étudiants et est à rendre par un dépôt *moodle* le vendredi 24 mai 2024 à 23h00. Un retard raisonnable est toléré mais il sera pénalisé.

L'utilisation des générateurs de code de type *copilot* est fortement encouragée.

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/
- https://chat.openai.com/

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 (−) (notation https://en.wikipedia.org/wiki/Letter_case#Kebab_case);
- 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 (notation https://en.wikipedia.org/wiki/Letter_case#Camel_case). Les structures devront commencer par un sous-tiret (_) puis suivre la notation *Camel case* 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 (_) (notation https://en.wikipedia.org/wiki/Letter_case#Snake_case).

1.3.4 Variables et fonctions

Les variables et les fonctions s'écrivent toutes en minuscules en séparant les mots par des sous-tirets (notation https://en.wikipedia.org/wiki/Letter_case#Snake_case).

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 Structuration du code

Les champs des structures seront protégés à la manière de la librairie `fraction` vue en travaux pratiques. Un soin sera tout particulièrement apporté à la structuration du code notamment en ce qui concerne les structures de données utilisées.

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-arithmetic.

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

2 Sujet

Le but du projet est de poursuivre le travail commencé lors de l'**É**valutation **F**inale et de produire d'une part un exécutable permettant de résoudre un *Nonogramme* et d'autre part un exécutable permettant de générer un *Nonogramme* résolvable.

Le projet pourra fournir un fichier `README.md` expliquant les instructions à exécuter avant de le configurer. Il devra fournir une documentation produite avec

```
$ make docs
```

Il pourra être installé avec

```
$ make install
```

2.1 Description du problème

2.1.1 Exécutable `nonogram-solve`

Il s'agit de construire un exécutable `nonogram-solve` permettant de résoudre un *Nonogramme*. Les données en entrées sont :

- un fichier au format **JSON** contenant les indices (tels que calculés par la librairie de l'**É**valuation **F**inale). Pour lire ce fichier, vous **utiliserez** la librairie dont le code est donné à https://github.com/DaveGamble/cJSON.
- un fichier **optionnel** au format ASCII **PBM** (paramètre `--board`) qui contient certaines cases déjà noircies (mais pas forcément toutes). Pour lire ce fichier, vous **utiliserez** la librairie dont le code est donné à https://github.com/nkkav/libpnmio

L'exécutable produit en sortie une image au format ASCII **PBM** (paramètre `--output`) contenant la résolution du problème. Si l'algorithme n'arrive pas à résoudre le puzzle en utilisant un raisonnement simpliste (sur une seule ligne ou une seule colonne), le message `"Unsolvable puzzle"` sera affiché dans la **sortie d'erreur**. Si le paramètre `--output` n'est pas présent, la résolution du problème sera affichée sur la **sortie standard**.

```
$ ./nonogram-solve hints.json --board board.pbm --output solved.pbm
```

2.1.2 Exécutable `nonogram-create`

Il s'agit de construire un exécutable `nonogram-create` permettant de produire un *Nonogramme*. Les données en entrées sont :

- un fichier au format ASCII **PBM**

L'exécutable produit en sortie :

- un fichier **optionnel** au format **JSON** (paramètre `--hints`) contenant les indices (tels que calculés par la librairie de l'**É**valuation **F**inale)
- un fichier **optionnel** au format ASCII **PBM** (paramètre `--board`) ajoutant certaines cases noircies **si** la connaissance des indices est insuffisante à la résolution du puzzle. Les cases additionnelles noircies pourront être différentes d'une exécution à l'autre.
- un fichier au format **SVG** (paramètre `--output`) contenant une image permettant à un humain de résoudre le puzzle. Voir fig. 1 extrait de la page https://fr.wikipedia.org/wiki/Picross. Si le

paramètre `--output` n'est pas présent, le problème au format **SVG** sera affiché sur la **sortie standard**.

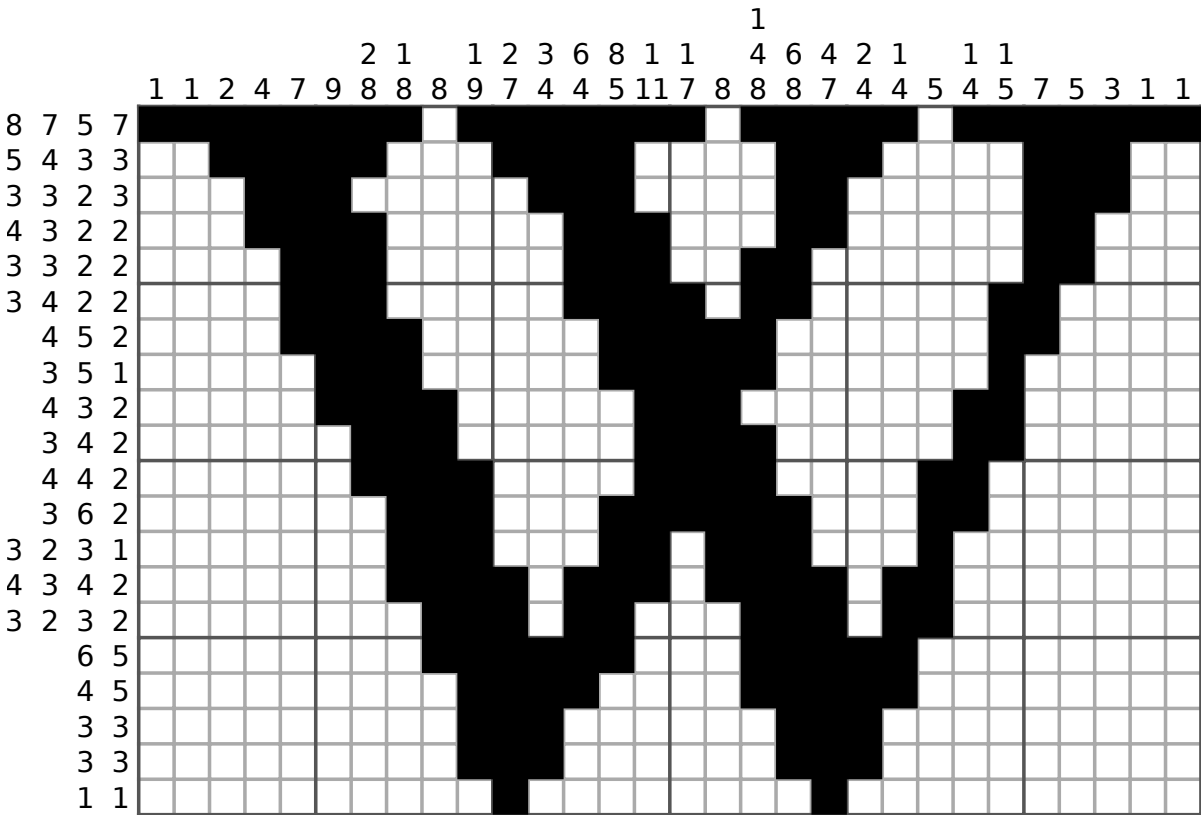

Figure 1: Nonogramme

```
$ ./nonogram-create \
    image.pbm \
    --hints hints.json \
    --board board.pbm \
    --output puzzle.svg
```

L'archive déposée avec le projet montre le contenu des différents fichiers incriminés.

2.2 Rapport

Le rapport est à produire au moyen de la commande

```
$ make docs
```

et devra contenir votre approche ainsi que vos difficultés et le cas échéant, ce qui ne fonctionne pas.

Historique des modifications

2023-2024_1 *Dimanche 17 mars 2024*

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

- Version initiale

