Language C - Projet

Licence informatique 2ème année

Université de La Rochelle

<christophe.demko@univ-lr.fr>

2019-2020_2

Table des matières

1 Consignes **3**
1.1 Groupes . . . 3
1.2 Langue utilisée . . . 3
1.3 Nommage . . . 3
1.3.1 Fichiers . . . 3
1.3.2 Types . . . 3
1.3.3 Macros . . . 3
1.3.4 Variables . . . 4
1.4 Style . . . 4
1.4.1 Marque d'inclusion unique . . . 4
1.4.2 Ordre des inclusions . . . 4
1.4.3 Indentation . . . 4
1.5 Champs protégés . . . 4
1.6 Documentation . . . 4
1.7 Tests . . . 5

2 Sujet **5**

Historique des modifications **8**

Liste des exercices

1 Génération des individus . . . 5
2 Création et destruction d'une population . . . 6
3 Population suivante . . . 6
4 Résolution de sudokus . . . 7

1 Consignes

1.1 Groupes

Le projet se fait par groupe de 4 à 6 étudiants et à rendre le vendredi 20 décembre 2019 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

Les variables 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. Pour l'installer sur les machines virtuelles étudiantes, vous aurez besoin avant toute chose d'exécuter :

```
$ export HOME="/media/Qi/$USER"
$ export PATH="/media/Qi/$USER/.local/bin:$PATH"
```

avant les instructions décrites dans le projet.

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 implémentant les algorithmes génétiques [1] ;
- un logiciel capable de résoudre des problèmes de sudoku [2] en utilisant la librairie précédemment décrite.

La librairie est la suite du contrôle continu du 16 novembre 2019.

Le projet devra fournir une documentation produite avec

```
$ make docs
```

Il pourra être installé avec

```
$ make install
```

Exercice 1 *(Génération des individus)*
Écrire une fonction

```
extern unsigned int* genetic_generator_individual(
  const GeneticGenerator* generator
);
```

permettant de générer un individu.
Un individu est un tableau dynamique d'entiers non signés représentant les valeurs prises par chacune

1. https://fr.wikipedia.org/wiki/Algorithme_génétique
2. https://fr.wikipedia.org/wiki/Sudoku

des inconnues. Initialement les valeurs prises par ces inconnues sont tirées au sort (entre 0 et $n-1$, n étant la cardinalité maximum associée à cette inconnue).

Exercice 2 *(Création et destruction d'une population)*
Une population est un ensemble d'individu. C'est une structure contenant une taille, un générateur et un tableau dynamique d'individus.
Écrire une fonction

```
extern Population* ga_population_create(
  const GeneticGenerator* generator,
  unsigned int size
);
```

permettant de générer une population de taille `size`. Cette fonction fera intelligement appel à `genetic_generator_individual`. La taille `size` devra être un nombre pair (comme nous le verrons plus loin).
Écrire une fonction

```
extern void ga_population_destroy(
  Population* population
);
```

permettant de libérer en mémoire une population d'individus.

Exercice 3 *(Population suivante)*
Les algorithmes génétiques utilisent des populations d'individus successives pour rechercher une solution à un problème donné.
Une des fonction essentielles consiste à passer d'une population à une autre.
Écrire une fonction

```
extern Population* ga_population_next(
  Population* population,
  const float cross_over,
  const float mutation,
  unsigned int (*evaluate)(unsigned int *, const void*)
);
```

permettant de faire évoluer la population.

- `population` est une population à faire évoluer ;
- `cross_over` est une probabilité de croisement, i.e. la probabilité qu'un gène (la valeur d'une inconnue dans un individu) soit échangé avec le gène d'un autre individu. On ne peut échanger

que des gènes ayant la même position.

— `mutation` est une probabilité de mutation, i.e. la probabilité qu'un gène change de valeur et prenne une valeur entre 0 et sa cardinalité maximum (-1).

— `evaluate` est une fonction permettant d'évaluer la pertinence d'un individu. Les valeurs retournées par la fonction `evaluate` passée en argument servent de proportion dans la roue de la fortune biaisié utilisée pour sélectionner les individus à fort potentiel. Le pointeur de fonction `evaluate` reçoit 2 arguments :

— un individu à évaluer

— un pointeur vers des données représentant le problème à optimiser.

L'algorithme de la fonction `ga_population_next` consiste à effectuer la boucle suivante:

— sélection et clonage de 2 individus en utilisant une roue de la fortune biaisée ;

— pour chaque gène, la probabilité de croisement est testée afin de savoir si le gène en question est échangé entre les 2 individus;

— pour chaque gène et pour les 2 individus, la probabilité de mutation est testée afin de savoir si le gène change de valeur.

La boucle s'arrête lorsque le nombre d'individus nouvellement générés atteint la taille de la population. Les anciens individus sont alors détruits et remplacés par les nouveaux.

Exercice 4 *(Résolution de sudokus)*

Écrire un programme permettant de résoudre des problèmes de sudoku en utilisant la librairie des algorithmes génériques écrite.

Ce programme prend en argument :

— un nom de fichier au format `yaml` indiquant le problème de sudoku à résoudre. Vous pourrez utiliser la librairie https://github.com/yaml/libyaml/ permettant de lire ou d'écrire des données au format yaml.

```
- [   2, null, null, null, null, null, null,    4, null]
- [null, null, null, null, null, null,    5, null, null]
- [null,    1, null,    6, null,    4, null,    3,    8]
- [   1, null, null,    7,    4, null, null,    8,    9]
- [null, null,    9, null,    8, null,    1, null, null]
- [   3,    7, null, null,    9,    5, null, null,    4]
- [   8,    9, null,    4, null,    7, null,    1, null]
- [null, null,    2, null, null, null, null, null, null]
- [null,    3, null, null, null, null, null, null,    7]
```

représente un sudoku avec 52 inconnues. Les sudoku sont toujours de taille $n^2 \times n^2$ avec n étant un nombre entier.

— une probabilité de croisement ;

— une probabilité de mutation ;

> — un nombre d'individus par population ;
> — un nombre d'itération maximum.
>
> Exemple
>
> ```
> $ ./sudoku sudoku.yaml 0.2 0.05 100 1000
> ```
>
> va tenter de résoudre le problème décrit dans le fichier `sudoku.yaml` avec une probabilité de croisement de 0.2, une probabilité de mutation de 0.05, 100 individus par population et une suite de 1000 populations.
> Le résultat devra être affiché sous la forme `yaml` :
>
> ```
> - [2, 8, 3, 5, 1, 9, 7, 4, 6]
> - [9, 6, 4, 8, 7, 3, 5, 2, 1]
> - [5, 1, 7, 6, 2, 4, 9, 3, 8]
> - [1, 5, 6, 7, 4, 2, 3, 8, 9]
> - [4, 2, 9, 3, 8, 6, 1, 7, 5]
> - [3, 7, 8, 1, 9, 5, 2, 6, 4]
> - [8, 9, 5, 4, 3, 7, 6, 1, 2]
> - [7, 4, 2, 9, 6, 1, 8, 5, 3]
> - [6, 3, 1, 2, 5, 8, 4, 9, 7]
> ```
>
> En cours de résolution, le programme affichera la numéro de population courante ainsi que la fonction d'évaluation du meilleur individu.

Historique des modifications

2019-2020_1 *Vendredi 15 novembre 2019*

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

— Version initiale

2019-2020_2 *Vendredi 22 novembre 2019*

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

— `population_destroy` renvoie `void`.