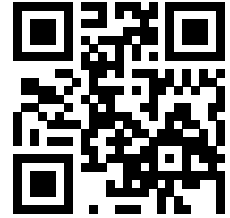


NOM : Hanon Ymous  
(000000)  
Place : 0

#0000



## Programmation Orientée Système (IN/SC) : SÉRIE NOTÉE

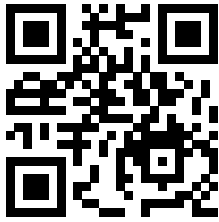
11 avril 2022

SUJET A

### INSTRUCTIONS (à lire attentivement)

**IMPORTANT!** Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre série annulée dans le cas contraire.

1. Vous disposez de 1h45 pour faire cette série notée (8h15 - 10h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur. N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.  
En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée; utilisez aussi le verso des feuilles, **MAIS** n'utilisez *que* le verso de la feuille sur laquelle se trouve la question, et non **pas** celui de la feuille précédente!  
Ne joignez aucune feuille supplémentaire; **seul ce document sera corrigé**.  
Si nécessaire, il y a des pages supplémentaires en fin de copie.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. Cette série notée ne comporte qu'un seul exercice en six questions.



## 1 – Optimisation [sur 70 points]

Les algorithmes génétiques sont une technique d'optimisation de fonctions à valeurs réelles, par recherche pseudo-aléatoire. On cherche ici à écrire des *parties* d'un programme d'optimisation par algorithmes génétiques.

Pour simplifier nous supposons ici vouloir maximiser une fonction de  $\mathbb{R}^2$  dans  $\mathbb{R}$ , accessible au travers d'un pointeur sur fonction, `global_goal`, global à tout le programme.

Pour trouver un maximum de cette fonction, un algorithme génétique utilise un ensemble (`Population`) d'« individus » (`Individual`) qui, dans notre cas simple, sont des points de  $\mathbb{R}^2$  (deux `double`, donc).

Un algorithme génétique « mélange » alors itérativement une telle `Population` pour en créer une nouvelle, dont il ne garde à chaque étape que les meilleurs individus.

Nous ne vous demandons dans cet exercice que de ne coder que *certaines parties spécifiques* d'un tel programme, lesquelles seront détaillées par la suite.

### 1.1 – Exemple d'utilisation [sur 6 points]

Afin d'illustrer le contexte global, voici un exemple de `main()` possible :

```
int main(void)
{
    Target choices[] = { f1, f2 };
    const size_t nb_choices = 2;

    global_goal = choose(choices, nb_choices);

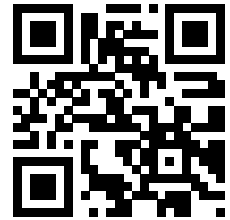
    Population points = create_random_initial_population(12);
    evolve(&points, 10);
    display_bests(&points, 2);

    return 0;
}
```

**[3 points]** Sachant que la fonction `choose()` permet de choisir un élément dans un tableau, proposez un type possible pour `Target`. Écrivez le `typedef` correspondant :

**[3 points]** Complétez si nécessaire ce `main()` (à droite avec une/des flèche(s)) ou indiquez : « rien à ajouter ». (Si ce n'est pas déjà fait, lisez peut être *toute* la question suivante avant de répondre ici.)

Les seules fonctions qui nous préoccupent dans la suite sont `create_random_initial_population()` et `evolve()`, ainsi que les types et quelques fonctions auxiliaires nécessaires à leur fonctionnement.



---

## 1.2 – Types de données [sur 16 points]

Avant tout, il faut définir les types de données utilisés.

**[1.5 points]** Définissez ci-dessous à gauche le type `Individual` comme simplement deux `double`.

**[2.5 points]** Définissez ensuite (à droite) le type `Population` comme un tableau dynamique d'`Individuals`.



**[5 points]** Définissez ensuite une fonction

```
Population create_zero_initial_population(size_t nb);
```

qui crée une population de taille donnée en paramètre, en initialisant tous les individus à deux `0.0`<sup>1</sup> :

**[4 points]** En supposant qu'il existe une fonction `void random_init(Individual*)` qui initialise un individu au hasard, définissez ensuite une fonction

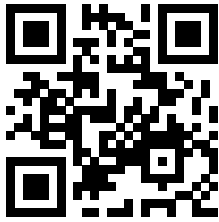
```
Population create_random_initial_population(size_t nb);
```

qui crée une population de taille donnée en paramètre, en initialisant tous ses individus au hasard.

---

1. Si jamais, la représentation de `0.0` correspond à toute la mémoire à `0`.

suite au dos 



---

[3 points] Définissez enfin une fonction `release()` qui prend une population en paramètre et libère son contenu. Elle ne retourne rien.

---

### 1.3 – Copie d’une population [sur 8 points]

Dans la suite, nous aurons besoin de recopier des (sous-)populations dans d’autres populations.

Définissez ici une fonction `copy_subpopulation()` qui ne retourne rien, mais prend en paramètres :

- la population à copier (origine) ;
- l’index `start` de début de copie (origine) ;
- le nombre total `nb` d’individus à copier ;
- la population d’arrivée (modifiée) ;
- l’index `where`, dans la population d’arrivée, où placer le premier individu copié.

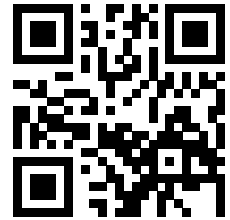
Si `start` est plus grand que la taille de la population de départ, on ne fait rien.

Si la taille à copier (`nb`) est trop grande par rapport à `start` et à la taille de la population de départ, on la réduit à la taille maximale possible. Par exemple, si l’on demande de copier `nb=10` individus, à partir de l’index `start=3`<sup>2</sup> d’une population de départ de 5 individus, il faudra réduire `nb` à 2.

Si, depuis `where`, il n’y a pas assez de place dans la population d’arrivée pour copier les individus finalement demandés, on ne fait rien.

---

2. Les index commencent bien sûr à 0.



---

### 1.4 – Mutation d'un individu [sur 4 points]

Les algorithmes génétiques utilisent deux opérations pour générer de nouveaux individus : la mutation et le croisement. Nous nous intéressons ici à la mutation. La question suivante portera sur le croisement.

Pour muter un `Individual`, nous considérons celui-ci simplement comme une suite d'octets dont nous allons, pour simplifier, ne modifier qu'un seul octet.

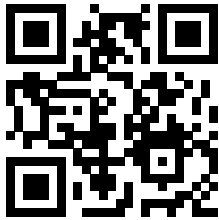
Définissez ci-dessous la fonction `mutate_i()` qui ne retourne rien, mais prend en paramètre un `Individual` et ajoute la valeur 1 (00000001 en binaire si vous préférez) à son troisième octet.

Illustration :

0      1      2      3      .....      mutation

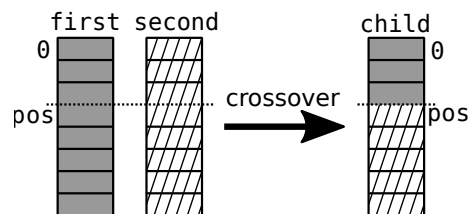
0110110100101010**11001101**10010010 ..... 10101001      →      0110110100101010**11001110**10010010 ..... 10101001

suite au dos 



### 1.5 – Croisement de deux individus [sur 11 points]

Nous nous intéressons maintenant aux croisements entre individus. Un croisement considère la représentation mémoire de deux individus et la croise (= échange une sous-partie) pour créer un nouvel individu :



En supposant qu'il existe une fonction

```
size_t random_size_t(size_t max);
```

retournant un `size_t` au hasard entre 0 inclus et `max exclu`, définissez ici une fonction

```
Individual crossover_i(const Individual* first, const Individual* second)
```

qui prend deux individus et retourne un nouvel individu, en (voir illustration ci-dessus) :

1. tirant au hasard une position `pos` entre 1 (inclus) et le nombre maximal d'octets d'un `Individual` (exclu) ;
2. copiant les *octets* 0 (inclus) à `pos` (exclu) du premier individu (dans le nouvel individu) ;
3. copiant les octets jusqu'à la fin du second individu à partir de `pos` (inclus).


Ne pas écrire dans cette zone.

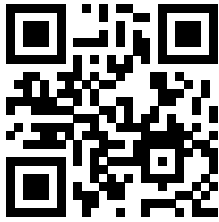
question 1.5

Anonymisation : #0000  
p. 7



Ne pas écrire dans cette zone.

suite au dos 



### 1.6 – Évolution [sur 25 points]

La dernière étape que nous vous demandons de coder, plus conséquente, consiste à gérer l'évolution pas à pas d'une population (on parle de « générations ») en vue de maximiser la fonction `global_goal()` (revoir l'introduction si nécessaire).

Cette partie pourra nécessiter l'écriture de fonctions auxiliaires.

On supposera par contre fournies<sup>3</sup> les fonctions suivantes :

- `void mutate_p(Population* p, size_t from, size_t to)`  
qui applique la mutation `mutate_i()` à tous les individus de `p` entre l'index `from` (inclus) et `to` (exclu) ;
- `void crossover_p(const Individual* first, const Individual* second, size_t nb, Individual* children)`  
qui suppose trois tableaux (`first`, `second` et `children`) de tailles au moins `nb` et met dans `children[i]` le résultat du croisement de `first[i]` avec `second[i]`.

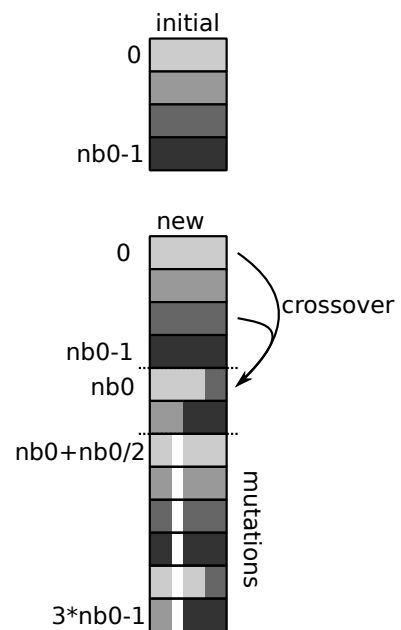
Définissez ci-contre une fonction

```
void evolve(Population* initial, unsigned int nb_generations)
```

qui, partant d'une population initiale va la modifier en une population finale après `nb_generations` cycles d'évolution.

L'algorithme est le suivant (voir illustration ci-contre) :

- créer une population `new`, initialisée à zéro, trois fois plus grande que la population initiale (appelons `nb0` la taille de la population initiale) ;
- copier la population initiale au début de cette population `new` (à ce stade elle est donc au deux-tiers encore à zéro) ;
- pendant `nb_generations` itérations :
  - met à partir de la position `nb0` de `new`, le croisement de ses `nb0 / 2` premiers individus avec les `nb0 / 2` suivants ; il y a donc à ce stade `nb0 + nb0 / 2` individus mis à jour dans `new` ;
  - ajoute à la fin de `new` la mutation de tous les individus ci-dessus (de 0 à `nb0 + nb0 / 2 - 1`) ; autrement dit : ajoute dans `new`, à partir de `nb0 + nb0 / 2`, la mutation des `nb0 + nb0 / 2` premiers individus de `new` ; il y a donc bien à ce stade deux fois `nb0 + nb0 / 2` individus (c.-à-d. trois `nb0` en tout, d'où la taille de `new`) mis à jour dans `new` ;
  - trie le tableau `new` suivant `global_goal()` : de sorte à ce que les premiers individus soient ceux qui sont les meilleurs pour `global_goal()` : on peut ainsi recommencer le cycle pour une nouvelle génération ;
- au final, recopie les `nb0` premiers individus de `new` dans `initial`.



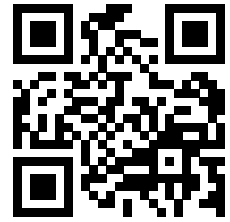
Ne pas écrire dans cette zone.

3. **NE PAS** les écrire.

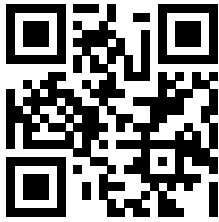


question 1.6

Anonymisation : #0000  
p. 9



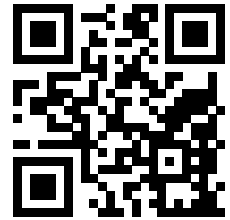
Ne pas écrire dans cette zone.



Anonymisation : #0000  
p. 10

---

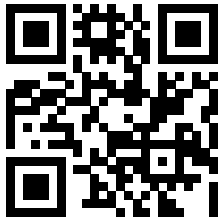
Ne pas écrire dans cette zone.



---

Place supplémentaire pour répondre à n'importe quelle question si nécessaire. Mais  
**VEUILLEZ INDIQUER LE NUMÉRO DE LA QUESTION TRAITÉE.**

Ne pas écrire dans cette zone.



Anonymisation : #0000  
p. 12

---

Place supplémentaire pour répondre à n'importe quelle question si nécessaire. Mais  
**VEUILLEZ INDIQUER LE NUMÉRO DE LA QUESTION TRAITÉE.**

Ne pas écrire dans cette zone.