

# EPITA S4 Promo 2019

## Programmation - Épreuve machine

Marwan Burelle \*

9 mai 2016

### Instructions :

**Vous devez lire l'intégralité du sujet, ainsi que l'ensemble de ces instructions. Tout ce qui est explicitement demandé dans ce sujet est obligatoire. La perte de points due au non respect des consignes explicites ne sera pas contestable.**

*Votre répertoire pendant l'épreuve est temporaire, dans ce répertoire vous trouverez un répertoire `subject` et un répertoire `submission`. Dans le répertoire `submission` vous trouverez tous les fichiers pour faire votre exam (squelette des questions.)*

**Vous devez effectuer des rendus réguliers pour être sûr de ne pas perdre votre travail. Pour effectuer un rendu, il vous suffit d'appeler la commande `submission`.**

*Dans le répertoire `submission` vous trouverez : un fichier `Makefile` permettant de compiler votre code, un certain nombre de fichiers annexes, un fichier pour chaque question de la forme `questionXX.c`. Seuls les fichiers de question peuvent être modifiés, les autres seront remplacés lors de la correction par les originaux.*

*Le `Makefile` permet d'engendrer un petit programme de test pour chaque question. Le programme de test se charge des interactions avec l'utilisateur et appelle votre fonction avec les paramètres attendus. Pour obtenir le programme de test de chaque question il faut faire la commande : `make questionXX`.*

***La compilation lors de la correction utilisera ce `Makefile`, par conséquent vous n'aurez les points à la question XX que si la commande "`make questionXX`" réussit et bien sûr que le résultat est correct.***

***Le barème est donné à titre indicatif et peut être sujet à modifications.***

*À la fin de ce document vous trouverez des annexes décrivant quelques consignes sur les programmes de tests.*

***Il y a 13 points et 12 questions dans cet examen.***

---

\*marwan.burelle@lse.epita.fr

### Question 1

(1)

Écrire la(les) fonction(s) suivante(s):

```
unsigned long my_intsqrt(unsigned long n);
```

`my_intsqrt(n)` calcule la racine carrée entière de  $n$ .

La racine carrée entière est la solution (entière)  $x$  à l'inéquation :

$$x^2 \leq n < (x + 1)^2$$

Pour résoudre ce problème on utilise la méthode de Héron (une variante de la méthode de Newton). On considère une première approximation  $x$  de la racine (cette approximation doit être supérieur à la racine et inférieur à  $n$ , on prendra donc  $n$  comme première valeur). On calcule la prochaine approximation comme étant la moyenne arithmétique entre  $x$  et  $n/x$  et on continue tant que  $x$  est plus grand que  $n/x$  (c'est à dire tant que  $x$  est plus grand que la racine cherchée).

#### Exemple 1.1:

```
shell> ./question01 0 5
```

Fixed tests:

<code>my_intsqrt( 0)</code>	<code>=</code>	<code>0</code>	<code>[OK]</code>
<code>my_intsqrt( 1)</code>	<code>=</code>	<code>1</code>	<code>[OK]</code>
<code>my_intsqrt( 4)</code>	<code>=</code>	<code>2</code>	<code>[OK]</code>
<code>my_intsqrt( 16)</code>	<code>=</code>	<code>4</code>	<code>[OK]</code>
<code>my_intsqrt( 64)</code>	<code>=</code>	<code>8</code>	<code>[OK]</code>
<code>my_intsqrt( 256)</code>	<code>=</code>	<code>16</code>	<code>[OK]</code>
<code>my_intsqrt( 1024)</code>	<code>=</code>	<code>32</code>	<code>[OK]</code>
<code>my_intsqrt( 4096)</code>	<code>=</code>	<code>64</code>	<code>[OK]</code>
<code>my_intsqrt(16384)</code>	<code>=</code>	<code>128</code>	<code>[OK]</code>
<code>my_intsqrt(65536)</code>	<code>=</code>	<code>256</code>	<code>[OK]</code>

Random tests:

<code>my_intsqrt(1804289383)</code>	<code>=</code>	<code>42476</code>	<code>[OK]</code>
<code>my_intsqrt( 846930886)</code>	<code>=</code>	<code>29102</code>	<code>[OK]</code>
<code>my_intsqrt(1681692777)</code>	<code>=</code>	<code>41008</code>	<code>[OK]</code>
<code>my_intsqrt(1714636915)</code>	<code>=</code>	<code>41408</code>	<code>[OK]</code>
<code>my_intsqrt(1957747793)</code>	<code>=</code>	<code>44246</code>	<code>[OK]</code>

### Question 2

(1)

Écrire la(les) fonction(s) suivante(s):

```
void array_reverse(int *begin, int *end);
```

`array_reverse(begin, end)` inverse le contenu des cases du tableau entre `begin` (inclus) et `end` (exclus.)

**Exemple 2.1:**

```
shell> ./question02 0 5
```

Fixed tests:

array:

```
| 1 | 2 | 3 | 4 | 5 |
```

after reverse:

```
| 5 | 4 | 3 | 2 | 1 |
```

Random tests:

array:

```
| 383 | 886 | 777 | 915 | 793 |
```

after reverse:

```
| 793 | 915 | 777 | 886 | 383 |
```

```
shell> ./question02 0 6
```

Fixed tests:

array:

```
| 1 | 2 | 3 | 4 | 5 | 6 |
```

after reverse:

```
| 6 | 5 | 4 | 3 | 2 | 1 |
```

Random tests:

array:

```
| 383 | 886 | 777 | 915 | 793 | 335 |
```

after reverse:

```
| 335 | 793 | 915 | 777 | 886 | 383 |
```

**Question 3**

(1)

Écrire la(les) fonction(s) suivante(s):

```
void array_merge(int *dst, int *a1, int *a2, int *e1, int *e2);
```

`array_merge(dst, a1, a2, e1, e2)` fusionne les deux tableaux triés `a1` et `a2` dans le tableau `dst`. `e1` (respectivement `e2`) est le pointeur de fin (exclus) du tableau `a1` (respectivement `a2`). La zone mémoire pointée par `dst` est réputée suffisamment grande pour contenir le contenu des deux tableaux (la fin du tableau `dst` se trouve à l'adresse `dst + e1 - a1 + e2 - a2`).

Il n'y a aucune contrainte de taille entre `a1` et `a2`.

Le résultat de la fusion (dans `dst`) doit bien évidemment être trié.

**Exemple 3.1:**

```
shell> ./question03 0 7
```

Fixed tests:

array1:

```
| 1 | 2 | 3 |
```

```

array2:
  | 4 | 5 | 6 | 7 |
array_merge(dst, array1, array2, ...)
dst:
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```

Fixed tests (2):

```

array1:
  | 1 | 3 | 5 | 7 |
array2:
  | 2 | 4 | 6 |
array_merge(dst, array1, array2, ...)
dst:
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```

Random tests:

```

array1:
  | 10 | 28 | 44 |
array2:
  | 19 | 26 | 39 | 49 |
array_merge(dst, array1, array2, ...)
dst:
  | 10 | 19 | 26 | 28 | 39 | 44 | 49 |

```

#### Question 4

(1)

Écrire la(les) fonction(s) suivante(s):

```
size_t mystrlen(char *s);
```

mystrlen(s) renvoie le nombre de caractères de la chaîne s (le pointeur ne sera pas NULL.) Vous devrez respecter le comportement attendu pour la fonction strlen(3).

##### Exemple 4.1:

```

shell> ./question04 0 5
s = "n{6\P"
mystrlen(s) = 5 -- check: [OK]

```

#### Question 5

(1)

Écrire la(les) fonction(s) suivante(s):

```
char *mystrncpy(char *dst, char *src, size_t len);
```

`mystrncpy(dst,src,len)` : copie au plus `len` caractères de la chaîne `src` dans la chaîne `dst`. On suppose que `src` et `dst` sont non `NULL`, `src` est terminée par un `'\0'` et `dst` est de taille suffisante.

Dans tous les cas, `mystrncpy(dst,src,len)` écrit exactement `len` caractères dans `dst`. Si le nombre de caractères de `src` est inférieur à `len`, alors votre fonction devra remplir la fin de `dst` par des `'\0'`. Sinon (`src` plus grand que `len`) votre fonction ne doit pas mettre de `'\0'` à la fin de `dst` (voir `strncpy(3)`).

Je vous conseille fortement de lire la page de manuel de la fonction `strncpy(3)` qui fournit une description complète de cette fonction.

#### Exemple 5.1:

```
shell> ./question05 0 10
src = "n{6\Pavw[:"

test: mystrncpy(dst,src,11)
dst = "n{6\Pavw[:"
-- check:
  first char:  [OK]
  last char:   [OK]
  0 fill:      [OK]
  overflow:    [OK]

test: mystrncpy(dst,src,5)
dst = "n{6\P"
-- check:
  first char:  [OK]
  last char:   [OK]
  overflow:    [OK]

test: mystrncpy(dst,src,20)
dst = "n{6\Pavw[:"
-- check:
  first char:  [OK]
  last char:   [OK]
  0 fill:      [OK]
  overflow:    [OK]

test: mystrncpy(dst,src,0)
-- check:
  overflow:    [OK]
```

#### Question 6

(1)

Écrire la(les) fonction(s) suivante(s):

```
str_toupper(char *s);
```

`str_toupper(s)` transforme toutes les lettres minuscules de la chaîne `s` (terminée par le caractère `'\0'`) en majuscule et laisse inchangés les autres caractères.

On rappelle que les caractères en C ne sont que des entiers et que si `c` est une variable de type `char` contenant une lettre minuscule, alors `c` vérifie : `'a' <= c && c <= 'z'`. De même, si `c` est une lettre majuscule, alors : `'A' <= c && c <= 'Z'`.

#### Exemple 6.1:

```
shell> ./question06 0 26
```

Fixed tests:

```
s = "abcdefghijklmnopqrstuvwxyz"
```

```
str_to_upper(s)
```

```
s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Random tests:

```
s = "n{6\Pavw[:m04=ZvMD^bvU;e'R"
```

```
str_to_upper(s)
```

```
s = "N{6\PAVW[:M04=ZVMD^BVU;E'R"
```

#### Question 7

(2)

Écrire la(les) fonction(s) suivante(s):

```
struct matrix *matrix_mul(struct matrix *A, struct matrix *B);
```

`matrix_mul(A, B)` calcule le produit des matrices `A` et `B`. Le résultat est une nouvelle matrice qui devra être correctement créée et allouée.

Les matrices sont représentées par le type suivant :

```
struct matrix {  
    size_t lines, cols;  
    int *data;  
};
```

De manière évidente, `lines` représente le nombre de lignes de la matrice et `cols` son nombre de colonnes. Le champ `data` est un pointeur sur une zone mémoire contenant `lines * cols` entiers correspondant aux cases de la matrice stockées en mode *ligne d'abord*.

Pour accéder à la case  $(i, j)$  de la matrice `A`, on utilisera le décalage classique : `A->data[i * A->cols + j]`.

On rappelle que le produit d'une matrice  $A$  de dimensions  $n \times m$  par une matrice  $B$  de dimensions  $m \times p$  est la matrice  $AB$  de dimension  $(n \times p)$ , dont les cases sont données par la formule suivante :

$$AB_{i,j} = \sum_{k=0}^{m-1} A_{i,k} \times B_{k,j}$$

**Exemple 7.1:**

```
shell> ./question07 0 3 5
```

Test with id matrix:

A =

```
| 83 | 86 | 77 |
| 15 | 93 | 35 |
| 86 | 92 | 49 |
```

id =

```
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
```

C = A \* id

```
| 83 | 86 | 77 |
| 15 | 93 | 35 |
| 86 | 92 | 49 |
```

Random tests:

A =

```
| 21 | 62 | 27 | 90 | 59 |
| 63 | 26 | 40 | 26 | 72 |
| 36 | 11 | 68 | 67 | 29 |
```

B =

```
| 82 | 30 | 62 |
| 23 | 67 | 35 |
| 29 | 2 | 22 |
| 58 | 69 | 67 |
| 93 | 56 | 11 |
```

C = A \* B

```
| 14638 | 14352 | 10745 |
| 15128 | 9538 | 8230 |
| 11760 | 8200 | 8921 |
```

**Question 8**

(1)

Écrire la(les) fonction(s) suivante(s):

```
void list_insert(struct list *list, struct list *elm);
```

list\_insert(list,elm) insert le maillon (déjà alloué) elm dans la liste chaînée list à sa place. La liste est triée en ordre croissant et dispose d'une sentinelle.

Le type représentant les listes est le suivant :

```
struct list {
    struct list *next;
    int val;
};
```

**Exemple 8.1:**

```
shell> ./question08 0 5
```

Fixed tests:

```
list = EMPTY
```

```
    insert 1 in list
```

```
    insert 2 in list
```

```
    insert 3 in list
```

```
    insert 4 in list
```

```
    insert 5 in list
```

```
list = -> 1 -> 2 -> 3 -> 4 -> 5
```

Random tests:

```
list = EMPTY
```

```
    insert 383 in list
```

```
    insert 886 in list
```

```
    insert 777 in list
```

```
    insert 915 in list
```

```
    insert 793 in list
```

```
list = -> 383 -> 777 -> 793 -> 886 -> 915
```

**Question 9**

(1)

Écrire la(les) fonction(s) suivante(s):

```
void list_reverse(struct list *list);
```

`list_reverse(list)` renverse en place la liste `list`. Cette liste a une sentinelle qui doit être laissée en tête de la liste résultat.

**Attention :** il s'agit d'une liste intrusive (en gros vous n'avez pas accès au contenu) vous devez donc bien détacher les maillons de la liste et les racrocher en ordre inverse.

Le type représentant les listes est le suivant :

```
struct list {  
    struct list *next;  
};
```

**Exemple 9.1:**

```
shell> ./question09 0 5
```

Fixed tests:

```
list = -> 1 -> 2 -> 3 -> 4 -> 5
```

```
list_reverse(list)
```

```
list = -> 5 -> 4 -> 3 -> 2 -> 1
```



Random tests:

```
list = -> 793 -> 915 -> 777 -> 886 -> 383
```

```
list_reverse(list)
```

```
list = -> 383 -> 886 -> 777 -> 915 -> 793
```

## Programmation en Go

Pour cette partie, les squelettes de question se trouve dans le répertoire GoPart/src et vous disposez d'un sous répertoire par question.

Pour tester votre code (dans le répertoire de la question) il vous suffit de taper : `go test -v`. La sortie vous indiquera votre succès ou sinon quels tests ont échoués et affichera également quelques traces. Pour certaines questions ils sera possible de passer des paramètres supplémentaires dont vous pourrez voir les exemples pour chaque question.

Comme pour les questions de C, les tests disponibles en exam ne sont pas complets : **Le fait que le test renvoie OK n'indique pas que votre code réponde parfaitement à la question, vous devez inspecter les résultats affichés.**

### Question 10 (1)

Écrire la(les) fonction(s) suivante(s):

```
func Fact(n int) (r int)
```

Fact(*n*) calcule factorielle de *n*.

On rappelle la définition de factorielle :

$$\text{Fact}(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1 \\ n \times \text{Fact}(n - 1) & \text{otherwise} \end{cases}$$

Répertoire de la question : GoPart/src/question10

Fichier de la question : fact.go

#### Exemple 10.1:

```
shell> go test
=== RUN   TestFact0
--- PASS: TestFact0 (0.00s)
=== RUN   TestFact1
--- PASS: TestFact1 (0.00s)
=== RUN   TestFactn
--- PASS: TestFactn (0.00s)
question10_test.go:23: Fact(2) = 2
question10_test.go:23: Fact(3) = 6
question10_test.go:23: Fact(4) = 24
question10_test.go:23: Fact(5) = 120
question10_test.go:23: Fact(6) = 720
question10_test.go:23: Fact(7) = 5040
question10_test.go:23: Fact(8) = 40320
question10_test.go:23: Fact(9) = 362880
question10_test.go:23: Fact(10) = 3628800
question10_test.go:23: Fact(11) = 39916800
question10_test.go:23: Fact(12) = 479001600
```

```
question10_test.go:23: Fact(13) = 6227020800
question10_test.go:23: Fact(14) = 87178291200
PASS
ok    question10 0.002s
```

### Question 11

(1)

Écrire la(les) fonction(s) suivante(s):

```
func Sort(v []int) (sorted []int)
```

Sort(v) trie le tableau et renvoie le tableau trié en utilisant l'algorithme de tri fusion (*merge sort*.)

Vous aurez besoin d'implémenter une fonction de fusion (*merge*) de deux vecteurs déjà triés.

On rappelle le principe du tri fusion :

merge(v1, v2): returned a sorted merge of v1 and v2

Sort(v):

si longueur de v inférieure à 2:

renvoyer une copie de v

sinon:

mid <- longueur de v divisée par 2

renvoyer merge(Sort(v de 0 à mid), Sort(v de mid à fin de v))

Pour information, voici quelques constructions go qui pourraient être utiles :

```
// v a vector
// copy v in new vector r
r := make([]int, len(v))
copy(r, v)

// get a subrange from left to right of v
vsub := v[left:right]
// special cases
vsub := v[:right]
vsub := v[left:]
```

Répertoire de la question : GoPart/src/question11

Fichier de la question : merge\_sort.go

#### Exemple 11.1:

```
shell> go test -v -args 4
```

```

=== RUN    TestSorted
--- PASS: TestSorted (0.00s)
question11_test.go:47: After sort:  [0 1 2 3]
=== RUN    TestRevSorted
--- PASS: TestRevSorted (0.00s)
question11_test.go:59: After sort:  [0 1 2 3]
=== RUN    TestRandom
--- PASS: TestRandom (0.00s)
question11_test.go:72: After sort:  [11911 19767 32023 51502]
PASS
ok   question11 0.003s

```

## Question 12

(1)

Écrire la(les) fonction(s) suivante(s):

```
func next_permutation(v []int) (next bool)
```

`next_permutation(v)` permute les éléments de `v` en respectant l'ordre lexicographique et renvoie faux lorsque l'on revient à la première permutation et vrai sinon. L'ordre lexicographique correspond à un ordre de type dictionnaire : on compare les éléments depuis le début, tant qu'ils sont égaux on avance dans les deux tableaux et lorsque les éléments courants des deux tableaux sont différents on compare ces éléments. C'est exact le même ordre que pour les mots du dictionnaire ou l'ordre sur les chaînes. Par exemple, pour un tableau de 3 éléments, l'ordre lexicographique nous donne la séquence de permutations suivante :

- Tableau initial : 1,2,3
- Première permutation : 1,3,2
- Suivante : 2,1,3
- Suivante : 2,3,1
- Suivante : 3,1,2
- Dernière : 3,2,1

**Répertoire de la question :** GoPart/src/question12

**Fichier de la question :** next\_permutation.go

### Exemple 12.1:

```
shell> go test -v -args 4
```

## Remarques sur la session exam

À la fin de l'épreuve, vous devez quitter votre session en fermant l'horloge (et uniquement en fermant celle-ci.) Dans tous les cas, votre session sera également fermée à la fin du temps imparti.

Lorsque l'horloge est fermée (volontairement, ou en fin de session), la console qui vous a demandé le token vous demande votre mot de passe. **Vous devez rentrer votre mot de passe UNIX pour que le rendu ai bien lieu.**

Vous pouvez aussi rendre sans quitter : il y a une commande (dans le shell) appelée `submission` qui déclenche un rendu comme si vous quittiez (demande de mot de passe et envoi du rendu.)

Après la fin du test (dans les minutes qui suivent) vous pouvez redémarrer votre machine et vous connecter pour re-rendre (si par exemple il y a eu une erreur pendant votre premier rendu.)

## Remarques sur les squelettes de questions

Pour chaque question, un embryon de code vous est fourni. Ce code correspond au strict minimum pour que la question compile et que l'appel au programme de test échoue avec une erreur identifiable. Par conséquent, vous devez supprimer du corps des fonctions à écrire, le code provoquant l'erreur, sous peine de voir votre programme échouer lors des tests (ne laissez surtout pas la ligne `REMOVE_ME(...)`.)

### Exemple 1:

À titre d'exemple, si l'on vous demande la fonction C suivante :

```
int identity(int x);
```

`identity(x)` renvoi `x`.

Vous trouverez dans le fichier de question correspondant le code :

```
int identity(int x) {  
    /* FIX ME */  
    REMOVE_ME(x);  
}
```

Que vous devrez remplacer par :

```
int identity(int x) {  
    return x;  
}
```

## Remarques sur les programmes de test

La commande `make questionXX` produira un binaire `questionXX`. Ce binaire peut être utilisé pour tester votre réponse à la question X du sujet. Tous les binaires attendent des paramètres et affichent une aide succincte s'il sont appelés avec l'option `-help`.

### Exemple 2:

*Le binaire produit pour la question 1 (il s'agit d'un exemple qui ne correspond pas forcément au sujet)*

#### Question 1:

```
./question01 graine taille
-help    Display this list of options
--help   Display this list of options
```

Les deux paramètres sont donc `graine` et `taille`. Les paramètres nommés `graine` (présent pour chaque question, ou presque) font tous référence à l'initialisation d'un générateur de nombre aléatoire. Dans le cas présent la commande `./question03 X Y` va initialiser le générateur de nombre aléatoire avec X (ici, Y servant de taille à la liste générée.) Pour les mêmes valeurs de X on obtiendra les mêmes données (la séquence engendrée par une graine donnée est toujours la même.)

Le reste du temps les noms des paramètres donnés dans l'aide sont explicites (par rapport au sujet.) Si vous avez une doute sur l'utilisation vous pouvez jeter un oeil sur fichier contenant les tests de la question X (`test_qXX.c.`)