

Exam Machine Contrôle 1 Nov. 2017

Remarques :

Les consignes de rendu

Lisez les consignes : assurez-vous d'avoir bien tout compris !

Créez le dossier de rendu : *prenom.nom* (votre login).

Dans le dossier de rendu, vous devez fournir un fichier `exoi.ml` par exercice demandé contenant chacun les sources (**sans les évaluations de CAML !**) de l'exercice correspondant.

```
firstname.lastname.zip
+-- firstname.lastname/
|-- exo1.ml
|-- exo2.ml
|-- exo3.ml
```

Les rendus se font sur le site : `exam-sup.pie.cri.epita.net`, utilisez votre login et mot de passe CRI.

**Choisissez le bon exam (selon votre ville)
si vous voulez que votre rendu soit pris en compte !**

CAML

Lisez l'énoncé, puis relisez l'énoncé !

Vos fonctions doivent **impérativement** respecter les exemples d'application donnés : noms et modes de passage des paramètres.

Si vous avez été amené à définir des fonctions intermédiaires pour résoudre le problème, celles-ci doivent bien entendu être intégrées au fichier (dans l'ordre !) accompagnées de leurs spécifications (en commentaire !) : ce que fait la fonction, que signifient les paramètres.

Il va de soi que votre code doit être indenté !

En dehors d'indication dans l'énoncé, vous ne pouvez utiliser aucune fonction prédéfinie de CAML, à l'exception de `failwith` et `invalid_arg`. **Tout manquement à cette règle entraînera une note nulle.**

Vos fonctions doivent prendre en compte tous les cas : y compris ceux où les paramètres ne respectent pas les spécifications.

L'optimisation est notée, il est donc très fortement déconseillé d'utiliser l'opérateur `@` !

Fin de l'épreuve

Assurez-vous d'avoir bien enregistré vos fichiers, selon les consignes.

Important :

- Si vous avez le moindre souci avec cette procédure, les ACDCs sont là pour y répondre.
- **Avant de rendre votre archive, vérifiez bien que celle-ci ne contient pas uniquement un raccourci vers le dossier rendu, mais bien tout le dossier et les fichiers que vous devez rendre...**

Exercice 1 : PGCD par décomposition en facteurs premiers

Le pgcd de deux entiers u et v peut se calculer de la manière suivante :

- en décomposant les nombres u et v en facteurs premiers,
- puis en calculant le produit des facteurs communs.

product

Écrire la fonction `product` qui calcule le produit de tous les éléments d'une liste d'entiers.

```
val product : int list -> int = <fun>

# product [1; 2; 3; 4; 5] ;;
- : int = 120

# product [];;
- : int = 1
```

decompose

On désire construire la liste des facteurs premiers d'un entier naturel n . Une méthode simple consiste à diviser successivement l'entier par tous les entiers d tels que $2 \leq d < n$.

Utiliser cette méthode pour écrire la fonction `decompose n` qui retourne la liste des facteurs premiers de n triés en ordre croissant lorsque n est supérieur à 2.

```
val decompose : int -> int list = <fun>

# decompose 45 ;;
- : int list = [3; 3; 5]

# decompose 150 ;;
- : int list = [2; 3; 5; 5]

# decompose 0 ;;
Exception: Invalid_argument "decompose: parameter <= 1".
```

shared

Soient deux listes d'éléments croissantes. Écrire une fonction qui construit la liste des éléments communs aux deux listes.

```
val shared : 'a list -> 'a list -> 'a list = <fun>

# shared [1; 3; 6; 6; 8; 9] [2; 5; 6; 6; 8; 9] ;;
- : int list = [6; 6; 8; 9]

# shared [2; 3; 5; 5; 5] [3; 3; 5] ;;
- : int list = [3; 5]

# shared [3; 3; 5; 7] [2; 3; 3; 3; 7; 7] ;;
- : int list = [3; 3; 7]
```

gcd

Écrire la fonction `gcd` qui calcule le pgcd de deux entiers supérieurs à 2 en calculant le produit de leurs facteurs communs.

```
val gcd : int -> int -> int = <fun>

# gcd 45 150;;
- : int = 15

# gcd 100 7;;
- : int = 1
```

Exercice 2 : Immédiatement décodable

Ici, un ensemble de symboles codés est dit *immédiatement décodable* si aucun code des symboles ne sert de préfixe à un autre code du même ensemble.

Par exemple : Si l'on considère un alphabet constitué des symboles {A,B,C,D},

— l'ensemble de codes suivant est *immédiatement décodable* :

A:01 B:10 C:0010 D:0000

— mais celui-ci ne l'est pas :

A:01 B:10 C:010 D:0000 (A sert de préfixe à C)

Chaque code sera représenté par une liste d'entiers. Un ensemble de codes sera une liste de codes.

prefix

Écrire une fonction `prefix` qui prend en paramètre un couple de listes et retourne *true* si l'une des listes est préfixe de l'autre, *false* sinon.

```
val prefix : 'a list * 'a list -> bool = <fun>

# prefix ([1; 2; 3], [1; 2]) ;;
- : bool = true
# prefix ([1; 2], [1; 2; 3; 4]) ;;
- : bool = true
# prefix ([1; 2], [1; 3; 4]) ;;
- : bool = false
# prefix ([1; 2; 4], [1; 2; 3; 4; 5]) ;;
- : bool = false
# prefix ([], [1]) ;;
- : bool = true
```

is_prefix

Écrire une fonction `is_prefix` qui prend en paramètre une liste *l* et une liste de listes *ll* et qui vérifie si *l* est préfixe d'une des sous-listes de *ll*, ou si une des sous-listes de *ll* est préfixe de *l*.

```
val is_prefix : 'a list -> 'a list list -> bool = <fun>

# is_prefix [1;2] [[2]; [2;1]; [1;2;3]] ;;
- : bool = true
# is_prefix [1;2] [[3;1]; [1;3]; [2;1]] ;;
- : bool = false
# is_prefix [1;2] [[3;1]; [1]; [2;1]] ;;
- : bool = true
# is_prefix [] [] ;;
- : bool = false
# is_prefix [] [[]] ;;
- : bool = true
```

decodable

Écrire une fonction `decodable` qui prend en paramètre un ensemble de codes (représenté par une liste de codes) et qui détermine si cet ensemble est *immédiatement décodable*.

```
val decodable : 'a list list -> bool = <fun>

# decodable [[0;1]; [1;0]; [0;0;1;0]; [0;0;0;0]] ;;
- : bool = true
# decodable [[0;1]; [1;0]; [0;1;0]; [0;0;0;0]] ;;
- : bool = false
# decodable [[0]; [1;0]; [1;1;0]; [1;1;1]] ;;
- : bool = true
# decodable [[1;0]; [0]; [1;0;0]] ;;
- : bool = false
```

Exercice 3 : Les entiers longs

On peut représenter un entier naturel comme étant une liste de chiffres, commençant par le chiffre des unités, puis celui des dizaines...

Par exemple l'entier 987654321 sera représenté par la liste : [1; 2; 3; 4; 5; 6; 7; 8; 9]. Dans la suite cette représentation sera appelée *entier long*.

bigint_of_int et int_of_bigint

1. Écrire la fonction `int_of_bigint` qui convertit un *entier long* en entier.

```
val int_of_bigint : int list -> int = <fun>

# int_of_bigint [0; 1; 2; 3; 4; 5; 6; 7; 8; 9] ;;
- : int = 9876543210
# int_of_bigint [] ;;
- : int = 0
```

2. Écrire la fonction `bigint_of_int` qui convertit un entier naturel en sa représentation sous forme d'*entier long* (de liste de chiffres).

```
val bigint_of_int : int -> int list = <fun>

# bigint_of_int 9876543210 ;;
- : int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]

# bigint_of_int 0 ;;
- : int list = []
```

bigint_sum

Écrire la fonction `bigint_sum` qui additionne deux *entiers longs* (attention aux retenues).

```
val bigint_sum : int list -> int list -> int list = <fun>

# bigint_sum [0; 1; 2; 5] [8; 2; 9] ;;
- : int list = [8; 3; 1; 6]
```

$$\begin{array}{r} 1 \\ 5 \ 2 \ 1 \ 0 \\ + \quad 9 \ 2 \ 8 \\ \hline 6 \ 1 \ 3 \ 8 \end{array}$$

bigint_mult

Écrire une fonction `bigint_mult` qui multiplie un *entier long* par un entier naturel (`int`).

```
val bigint_mult : int list -> int -> int list = <fun>

# bigint_mult [1; 2; 3; 4; 5] 5 ;;
- : int list = [5; 0; 6; 1; 7; 2]
```

$$\begin{array}{r} 2 \ 1 \ 1 \\ 5 \ 4 \ 3 \ 2 \ 1 \\ \times \quad \quad \quad 5 \\ \hline 2 \ 7 \ 1 \ 6 \ 0 \ 5 \end{array}$$

bigint_times

Écrire une fonction `bigint_times` qui multiplie deux *entiers longs*.

```
val bigint_times : int list -> int list -> int list = <fun>

# bigint_times [0; 1; 2; 5] [8; 2; 9] ;;
- : int list = [0; 8; 8; 4; 3; 8; 4]
```

Rappel :

$$\begin{array}{r} 5210 \times 928 = 834880 \\ \quad 8 \times 5210 \\ + \quad (2 \times 5210) \times 10 \\ + \quad (9 \times 5210) \times 100 \\ \hline = 834880 \end{array}$$