EPITA SPE promo 2014
Programmation - Épreuve machine

Marwan Burelle*

Vendredi 7 janvier 2011

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, en particulier pour les problèmes de compilation de votre code ou l'existence de sous-répertoire dans votre rendu.

Dans le répertoire `sujet` *vous trouverez un sous-répertoire* `Skel` *vous devez copier le* **contenu** *de ce répertoire dans votre répertoire de rendu. Dans le répertoire* `sujet` *vous trouverez également deux scripts* `startemacs` *et* `startemacs-nw` *qui permettent de démarrer* `emacs` *avec le mode* `tuareg` *chargé.*

Pour information, la copie des fichiers du répertoire d'origine vers votre répertoire de rendu, ce fait à l'aide des commandes :

```
> cd
> cd sujet
> cp Skel/* ~/rendu/
```

Dans ce répertoire 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` *ou* `questionXX.ml`*. Seuls les fichiers de question peuvent être modifiés, les autres seront remplacés lors de la correction par les originaux, toute modification sera perdue.*

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****.*

Pour chaque question le nombre de points est indiqué en face de son numéro sur la droite de la page et entre parenthèse. ***Ce barème est donné à titre indicatif et peut être sujet à modifications.***

À la fin de ce document vous trouverez des annexes décrivants :

- *Quelques consignes sur les programmes de tests*
- *La liste des fichiers à rendre (ceux modifiables et ceux à ne pas toucher.)*

Il y a 25 points et 12 questions dans cet examen.

*marwan.burelle@lse.epita.fr

Programmation OCaml : Récursions arithmétiques classiques

Question 1 (1)

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

```
val fact : float -> float
```

`fact f` renvoie factorielle de `f`. On suppose que `f` (de type `float`) est un entier positif ou nul.

Exemple 1.1:

```
> ./question01 0 5
fact 30.0 = 265252859812191032188804700045312.0
fact 22.0 = 1124000727777607680000.0
fact 1.0 = 1.0
fact 4.0 = 24.0
fact 18.0 = 6402373705728000.0
> ./question01 42 5
fact 11.0 = 39916800.0
fact 10.0 = 3628800.0
fact 9.0 = 362880.0
fact 11.0 = 39916800.0
fact 10.0 = 3628800.0
```

Question 2 (1)

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

```
val fibo : int -> int
```

`fibo n` calcule le n^{ieme} terme de la suite de Fibonacci. `n` est un entier positif ou nul. Attention, les valeurs testées peuvent être *relativement* grandes, essayez d'écrire votre factorielle en linéaire.

On rappelle la définition de la suite de Fibonacci :

$$\text{fibo}(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{fibo}(n-1) + \text{fibo}(n-2) & \text{sinon} \end{cases}$$

Exemple 2.1:

```
> ./question02 0 5
fibo 30 = 832040
fibo 22 = 17711
fibo 1 = 1
fibo 4 = 3
fibo 18 = 2584
> ./question02 42 5
```

```
fibo 11 = 89
fibo 10 = 55
fibo 9 = 34
fibo 11 = 89
fibo 10 = 55
```

Programmation OCaml : Listes

Question 3 (1)

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

```
val max : 'a list -> 'a
```

`max l` renvoie l'élément maximal de la liste polymorphe `l` passée en paramètre. Lève l'exception `Not_found` si la liste est vide. On rappelle qu'il **existe** une fonction :

```
val max : 'a -> 'a -> 'a
```

Exemple 3.1:

```
> ./question03 0 0
l = [ ]
Liste vide
> ./question03 0 5
l = [ 338; 292; 449; 086; 414; ]
max l = 449
> ./question03 42 5
l = [ 010; 203; 361; 234; 011; ]
max l = 361
```

Question 4 (2)

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

```
val max_range : int list -> int
```

`max_range l` renvoie l'écart entre l'élément le plus grand et l'élément le plus petit de la liste d'entier `l`. Renvoie `0` si la liste est vide.

Exemple 4.1:

```
> ./question04 0 0
l = [ ]
max_range l = 0
> ./question04 0 5
l = [ 082; 036; 193; 086; 158; ]
max_range l = 157
```

```
> ./question04 42 5
l = [ 010; 203; 105; 234; 011; ]
max_range l = 224
```

– *Pour le deuxième exemple, on vérifie bien que* 193 – 36 = 157
– *Pour le troisième exemple, on vérifie bien que* 234 – 10 = 224

Programmation OCaml : Arbres binaires

Question 5 (4)

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

```
val ordering : tree -> (string, int * int) Hashtbl.t * int
```

`ordering t` effectue un parcours profondeur de l'arbre `t` et associe dans une table de hash la clef de chaque nœud, avec son ordre préfixe et son ordre suffixe calculer avec le même compteur. La fonction renvoie un couple contenant la table et le compteur. Le type des arbres est décrit par :

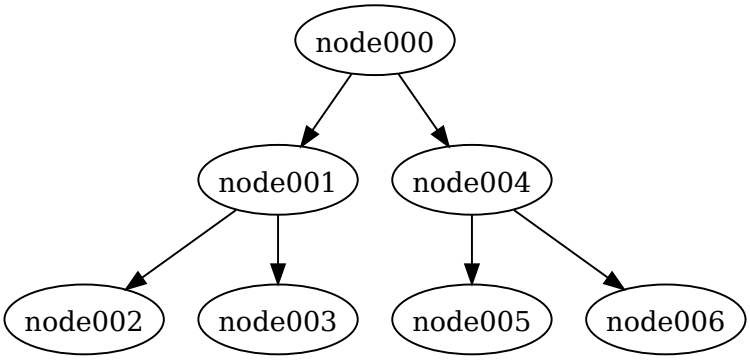

FIG. 1 – Exemple d'arbre

La table de hash renvoyer aura pour clef des chaines de caractères (clef de l'arbre) et pour valeur associer des couples d'entier (représentant l'ordre préfixe et l'ordre suffixe du nœud.) On rappelle les opérations de base sur les table de hash d'OCaml :

```
val create : int -> ('a, 'b) Hashtbl.t
val add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit

type tree =
    Empty
  | Node of tree * string * tree
```

Exemple 5.1:

```
> ./question05 0 0
t =
```

```
  Empty

(h,c) = ordering t
c = 0 (should be 0)
h = {
}
> ./question05 0 1
t =
  Node(Empty , node000, Empty )
(h,c) = ordering t
c = 2 (should be 2)
h = {
     (node000,1,2)
}
> ./question05 0 3
t =
  Node(
       Node(Node(Empty , node002, Empty ), node001,
            Node(Empty , node003, Empty )
       ), node000,
       Node(Node(Empty , node005, Empty ), node004,
            Node(Empty , node006, Empty )
       )
  )
(h,c) = ordering t
c = 14 (should be 14)
h = {
     (node000,1,14)
     (node001,2,7)
     (node002,3,4)
     (node003,5,6)
     (node004,8,13)
     (node005,9,10)
     (node006,11,12)
}
```

L'arbre du troisième exemple est représenté graphiquement sur la figure 1 page précédente.

Programmation C : bases

Question 6 (1)

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

```
size_t fact(size_t n);
```

`fact n` renvoie factorielle de `n`.

Exemple 6.1:

```
> ./question06 0 5
fact(01) = 1
fact(07) = 5040
fact(03) = 6
fact(05) = 120
fact(09) = 362880
```

Question 7 (1)

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

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

`my_strlen(s)` renvoie le nombre de caractère (différent du caractère `0`) dans la chaîne `s`. Le pointeur `s` est supposé différent de `NULL`.

Exemple 7.1:

```
> ./question07 0 0
my_strlen("") = 0
> ./question07 0 5
my_strlen("8xVnF") = 5
> ./question07 0 10
my_strlen("8xVnFbq[;$") = 10
```

Question 8 (2)

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

```
void str_replace(char *s, char o, char n, size_t len);
```

`str_replace(s, o, n len)` remplace dans la chaîne `s`, toutes les occurrences du caractère `o` par le caractère `n`. Le remplacement s'arrête à la fin de la chaîne (caractère de code `0`) ou lorsqu'elle a parcouru `len` caractères.

Exemple 8.1:

```
> ./question08 0 10 z A
s = "djrxzzgecf"
```

```
str_replace(s,'z','A',11);
s = "djrxAAgecf"
str_replace(s,'A','z',11);
s = "djrxzzgecf"
str_replace(s,'z','A',5);
s = "djrxAzgecf"
```

Programmation C : structures

Question 9 (3)

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

```
void insert(size_t x, list *l);
```

`insert(x,l)` insert `x` à sa place (dans l'ordre croissant) dans la liste `l`. La liste est supposée déjà triée.

Les listes sont représentées à l'aide de la structure suivantes :

```
typedef struct s_list  *list;

struct s_list
{
  size_t                data;
  list                  next;
};
```

Exemple 9.1:

```
> ./question09 0 5
l = [ 001; 018; 027; 038; 069;]

insert(077,l);
l = [ 001; 018; 027; 038; 069; 077;]

insert(000,l);
l = [ 000; 001; 018; 027; 038; 069; 077;]

insert(027,l);
l = [ 000; 001; 018; 027; 027; 038; 069; 077;]
```

Question 10 (3)

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

```
int not_circullar(list l);
```

`not_circullar(l)` renvoie vrai si la liste `l` n'est pas circulaire (le dernier élément pointe sur le premier) et faux (`0`) sinon. On rappelle que la liste vide n'est pas circulaire.

Exemple 10.1:

```
> ./question10 0 5
not_circullar(NULL) = 1 (sould be 1)
l = [ 049; 137; 075; 031; 039;]
not_circullar(l) = 1 (sould be 1)
l = [ 047; 002; 152; 240; 013; ... ]
not_circullar(l) = 0 (sould be 0)
```

Question 11 (4)

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

```
void *find(int k, blist l);
```

`find(k,l)` cherche la valeur associée à la clef `k` dans la liste de bloc `l`. Renvoie le pointeur `NULL` si la clef n'est pas présente. La liste de bloc `l` est une liste chaînée de blocs, où chaque bloc contient :

- un tableau de clefs (des entiers) ;
- un tableau de valeurs (des pointeurs) ;
- les valeurs de clefs minimale et maximale ;
- le nombre de clef dans le bloc.

Le type représentant tout ça :

```
typedef struct s_blist *blist;

struct s_blist
{
  blist                 next;
  int                   inf, sup;
  size_t                size;
  int                   keys[32];
  void                 *values[32];
};
```

Exemple 11.1:

```
> ./question11 0 2 0
l =
[inf=0845; sup=4085; size=08]
  {4085 -> byqz}
  {1250 -> adhc}
  {2528 -> abwe}
  {2683 -> dapg}
  {2529 -> heli}
  {2014 -> pmsg}
  {0845 -> jknz}
```

```
  {3969 -> dbol}
[inf=1538; sup=3721; size=02]
  {3721 -> rxzz}
  {1538 -> ecfp}
find(0000,l) = (null)
> ./question11 0 2 2529
l =
[inf=0845; sup=4085; size=08]
  {4085 -> byqz}
  {1250 -> adhc}
  {2528 -> abwe}
  {2683 -> dapg}
  {2529 -> heli}
  {2014 -> pmsg}
  {0845 -> jknz}
  {3969 -> dbol}
[inf=1538; sup=3721; size=02]
  {3721 -> rxzz}
  {1538 -> ecfp}
find(2529,l) = heli
```

Question 12 (2)

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

```
void fastcopy(void *src, void *dst, size_t len);
```

`fastcopy(src,dst,len)` copie `len` octets depuis la zone pointée par `src` vers la zone pointée `dst`. Les zones pointées par `dst` et `src` sont suffisament grandes. La taille `len` est forcément un multiple de **8**, il serait donc malin de ne pas faire de copie octet par octet mais plutôt 8 par 8 ou 4 par 4.

Attention, vous ne devez pas dépasser la taille passée en argument, ni modifier le contenu pointé par `src`.

Remarque 1:
Les questions sont compilées en C99 vous avez donc à votre disposition les types `uint32_t` *et* `uint64_t`.

Exemple 12.1:

```
> ./question12 0 5
size aligned to 8
data[8] =
djrxzzge

fastcopy(data,copy,8)
```

```
copy[8] =
djrxzzge
> ./question12 0 12
size aligned to 16
data[16] =
djrxzzge
cfpbrbyq

fastcopy(data,copy,16)
copy[16] =
djrxzzge
cfpbrbyq
> ./question12 0 17
size aligned to 24
data[24] =
djrxzzge
cfpbrbyq
zoadhcqa

fastcopy(data,copy,24)
copy[24] =
djrxzzge
cfpbrbyq
zoadhcqa
```

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) appellée `rendu` qui déclenche un rendu comme si vous quittiez (demande de mot de passe et envoie du rendu.)

Que faire si le mot de passe ne marche pas ?

- S'assurer que la fenêtre a bien le *focus* (mettre le pointeur de la souris dessus.)
- Il n'y a pas d'écho du mot passe lorsque vous le tapez et c'est normal : taper le correctement, puis appuyiez sur *return*. Si vous avez des doutes sur ce que vous avez taper, vous pouvez utiliser la touche *backspace*.
- Enfin, si vous avez un message d'erreur, si votre mot de passe n'est pas accepté, si . . . vous pouvez redemarrez la machine, vous connectez et de nouveau reprendre la procédure de rendu au départ (quitter, si le temps n'est pas fini, et rendre.)

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 `assert false`, même si elle n'est plus dans la fonction, elle sera problamenent exécutée quand même.)

Exemple 1:
À titre d'exemple, si l'on vous demande la fonction OCaml suivante :

```
val identity: 'a -> 'a
```

`identity x` renvoi `x`.
Vous trouverez dans le fichier de question correspondant le code :

```
let identity _ =
  (* FIX ME *)
  assert false
```

Que vous devrez remplacer par :

```
let identity x = x
```

Pour la partie C, le principe est le même. En plus, les paramètres n'étant pas utilisés dans le code il y a quelques lignes *pour faire comme-ci.* Un petit exemple vaut mieux qu'un long discourts :

Exemple 2:
Si l'on vous demande d'écrire la fonction C suivante :

```
void *identity(void *x);
```

`identity(x)` renvoie le pointeur `x`.
Vous trouverez dans le fichier de question correspondant le code :

```
void *identity(void *x)
{
  x = x;
  /* FIX ME */
  abort();
  return NULL;
}
```

Que vous devrez remplacer par :

```
void *identity(void *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 3:
Le binaire produit pour la question 3 fournit l'aide suivante :

```
Question 3:
./question03 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.ml`.)

Bien évidement, les même remarques sont valables pour les questions en C.

Listes des fichiers à rendre

À rendre sans modifications

Ces fichiers **doivent être** dans votre répertoire de rendu (directement sans autre répertoire), mais **ne** doivent **pas** avoir été modifié par rapport à leur version originale :

Makefile
base_test.c
base_test.h
cheaters.h
question01.mli
question02.mli
question03.mli
question04.mli
question05.mli
question06.h
question07.h
question08.h
question09.h
question10.h
question11.h
question12.h
test_frame.ml
test_q01.ml
test_q02.ml
test_q03.ml
test_q04.ml
test_q05.ml
test_q06.c
test_q07.c
test_q08.c
test_q09.c
test_q10.c
test_q11.c
test_q12.c

Fichiers de réponses

Ces fichiers **doivent être** dans votre répertoire de rendu (directement sans autre répertoire) et peuvent être modifiés (si vous répondez à la question bien sûr.)

question01.ml
question02.ml
question03.ml
question04.ml
question05.ml
question06.c
question07.c
question08.c
question09.c
question10.c
question11.c
question12.c