EPITA SPE promo 2011
Programmation
Épreuve machine

Marwan Burelle*

16 mai 2008

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 dûe au non respect des consignes explicites ne sera pas contestable, en particulier pour les problèmes de compilation de votre code.

Dans le répertoire `sujet` *vous trouverez un sous-répertoire* `Skel` *vous devez copier le* **contenu** *de ce répertoire dans votre répertoire de 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* `questionX.c` *ou* `questionX.ml`. *Seuls les fichiers de question peuvent être modifiés, les autres seront remplacés lors de la correction par les originaux, toute modification sera perdue* **et** *sanctionnée (perte de point.)*

Les commandes pour copier le contenu de `Skel` *dans votre répertoire sont les suivantes :*

```
# On commence par aller dans votre home
> cd
# et on copie
> cp sujet/Skel/* rendu/
# Enfin on verifie que ca marche
> cd rendu
> make clean  # au cas ou ...
> make depend # IMPORTANT a faire apres chaque make clean
> make        # il ne devrait pas y avoir d'erreur ici ...
```

Le fichier `Makefile` *permet d'engendrer un petit programme de test pour chaque question. La méthode de compilation pour la correction sera celle décrite dans ce fichier, par conséquent vous n'aurez les points à la question X que si la commande "*`make questionX`*" réussit et que le résultat est correct.*

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

Il y a 23 points et 8 questions dans cet examen, tout point au dessus 20 est considéré comme point bonus.

*Burelle@kh405.net

Programmation OCaml : Listes

Question 1 (2)

On se propose de coder la fonction suivante :

```
val minpair : ('a * 'a) list -> 'a list
```

`minpair [(e1,f1); ...; (en,fn)]` renvoie la liste des minimums de chaque couple dans la liste. Exemple :

```
minpair [ (5,7); (5,8); (6,3); (4,0); ] = [ 5; 5; 3; 0; ]
```

Question 2 (2)

On se propose de coder la fonction suivante :

```
val dbllist : 'a list -> ('a*'a) list
```

`dbllist [a1; ...; an]` renvoit `[(a1, a1); ...; (an, an)]` *càd* elle renvoie la liste des couples formés de chaque élément de la liste doublé. Exemple :

```
dbllist [ 3; 4; 0; ] = [ (3,3); (4,4); (0,0); ]
```

Programmation OCaml : Arbres

Question 3 (3)

Soit le type des arbres binaires valués aux feuilles (et seulement aux feuilles) suivant :

```
type 'a bintree = Leaf of 'a | Node of 'a bintree * 'a bintree
```

On se propose de coder la fonction suivante :

```
val rightFirst : 'a list -> 'a bintree -> 'a list
```

`rightFirst [] a = l` construit récursivement la liste des valeurs sur les feuilles à l'aide d'un parcours profondeur. La liste renvoyée contient les valeurs des feuilles droites en premier. Le premier paramètre sert d'accumulateur. Exemples :

```
t = Node(Leaf(4),Leaf(0))
rightFirst [] t = [ 0; 4; ]

t = Node(Node(Leaf(6),Leaf(3)),
         Node(Leaf(4),Leaf(0)))
rightFirst [] t = [ 0; 4; 3; 6; ]
```

Question 4 (3)

Soit la définition de type pour des expressions booléens avec variables :

```
type expr =
    Bool of bool
  | Not of expr
  | Or of expr * expr
  | And of expr * expr
  | Impl of expr * expr
  | Eq of expr * expr
  | Var of string
```

Exemple : l'expression booléen ($\texttt{true} \vee \texttt{x}$) se traduit

```
Or(Bool(true),Var("x"))
```

On rappelle les règles de base d'évaluation des expressions booléennes :

$$\frac{}{\texttt{true} \to \texttt{true}} \qquad \frac{}{\texttt{false} \to \texttt{false}}$$

$$\frac{e \to \texttt{true}}{\neg e \to \texttt{false}}\ \texttt{Not(e)} \qquad \frac{e \to \texttt{false}}{\neg e \to \texttt{true}}\ \texttt{Not(e)}$$

$$\frac{e_1 \to \texttt{true}}{e_1 \vee e_2 \to \texttt{true}}\ \texttt{Or(e1,e2)} \qquad \frac{e_2 \to \texttt{true}}{e_1 \vee e_2 \to \texttt{true}}\ \texttt{Or(e1,e2)} \qquad \frac{e_1 \to \texttt{false} \quad e_2 \to \texttt{false}}{e_1 \vee e_2 \to \texttt{false}}\ \texttt{Or(e1,e2)}$$

$$\frac{e_1 \to \texttt{false}}{e_1 \wedge e_2 \to \texttt{false}}\ \texttt{And(e1,e2)} \qquad \frac{e_2 \to \texttt{false}}{e_1 \wedge e_2 \to \texttt{false}}\ \texttt{And(e1,e2)} \qquad \frac{e_1 \to \texttt{true} \quad e_2 \to \texttt{true}}{e_1 \wedge e_2 \to \texttt{true}}\ \texttt{And(e1,e2)}$$

On rappelle les équivalences suivantes :

$$e_1 \Rightarrow e_2 \equiv \neg e_1 \vee e_2$$

$$e_1 \Leftrightarrow e_2 \equiv (e_1 \Rightarrow e_2) \wedge (e_2 \Rightarrow e_1)$$

Enfin on utilisera les tables de hashage d'OCaml comme environnement pour les variables. On rappelle la seule opération sur les tables de hashage utile pour cette exercice :

```
val Hashtbl.find: ('a,'b) Hashtbl.t -> 'a -> 'b
```

`Hashtbl.find env k` cherche la valeur associée à la clef `k` dans la table `env`. On considérera que toutes les variables disposent d'une entrée dans l'environnement.

Écrire la fonction d'évaluation d'expression booléennes :

```
val eval : (string, bool) Hashtbl.t -> expr -> bool
```

`eval env e` renvoie vraie si et seulement si l'expression `e` est vraie pour les valeurs des variables définits dans `env`. Elle renvoie faux sinon.

Le programme lit des expressions booléens sur l'entrée standard, quelques exemples sont fournits :

```
example-expr
example-expr01
example-expr03
example-expr2
```

Programmation C

Question 5 (2)

On se propose de coder la fonction suivante :

```
void mystrnfill(char *s, char c, size_t n);
```

`mystrnfill(s,c,n)` remplit le buffer `s` depuis la *fin* (caractère `0`) de `s` à hauteur de la taille `n` avec le caractère `c`. Soit le buffer suivant :

```
char s[10] = {'a','a','a',0,0,0,0,0,0,0};
```

alors après l'appel :

```
mystrnfill(s,'b',10);
```

le buffer contient :

```
s = {'a','a','a','b','b','b','b','b','b',0};
```

Ce qui donne l'interraction avec le programme de test :

```
>./question5 10 b aaa
aaa
aaabbbbbb
```

Si le buffer contient déjà `n` caractères, il n'est pas modifié. De plus, aucune allocation n'est nécessaire, le buffer est réputé déjà alloué avec suffisament d'espace.

Question 6 (3)

Soit la définition de liste chaînée de caractères en C :

```
typedef struct s_list *t_list;
struct s_list {
  t_list next;
  char content;
};
```

On se propose de coder la fonction suivante :

```
t_list  add_fun(t_list l, char c);
```

`add_fun(l,c)` ajoute en tête de la liste `l` le caractère `c`.

Programmation C : Images Arborescentes Récursives

Soit le type suivant :

```
typedef struct s_image *t_image;

struct s_image {
  int white;
  t_image tl, tr, bl, br;
};
```

définissant des images sous forme d'arbres. Soit un pointeur `t_image img` : l'image est noire si le pointeur vaut `NULL`, sinon si `img->white` est vrai (différent de 0) l'image est blanche, sinon l'image est la composition des quatres images `img->tl` (haut gauche), `img->tr` (haut droit), `img->bl` (bas gauche) et `img->br` (bas droit).

Question 7 (3)

On se propose de coder la fonction suivante :

```
int is_white(t_image img);
```

`is_white(img)` renvoie vrai (différent de 0) si l'image est complètement blanche, c'est a dire :

- `img->white` est vrai, **ou**
- l'image n'est composée que d'image blanche c'est à dire : `img->white` est faux mais `img->tl`, `img->tr`, `img->bl` et `img->br` sont toutes les quatres blanches (récursivement.)

Programmation C : Graphes dynamiques

Soit la définition des graphes dynamiques reprennant la définition du TD d'algo :

```
typedef struct s_graphe *t_graphe;
typedef struct s_adj    *t_adj;
typedef struct s_som    *t_listsom;

struct s_som {
  int som;
  t_adj succ;
  t_listsom suiv;
};

struct s_adj {
  t_listsom vsom;
  t_adj suiv;
};

struct s_graphe {
  size_t ordre;
  size_t orient;
  t_listsom lsom;
};
```

La seule différence est que l'on néglige le nombre de lien (on ne travail qu'avec des 1-graphes), on n'a pas de coût et l'on utilise pas (donc on ne la définit pas) la liste des prédécesseurs.

Question 8 (5)

On se propose de coder la fonction suivante :

```
int cherche_som(t_graphe g, int src, int dst, int **pere);
```

`cherche_som(g,src,dst,pere)` qui cherche s'il existe un chemin entre le sommet `src` et le sommet `dst` à l'aide d'un parcours en profondeur (en utilisant l'ordre des sommets du graphe.) La fonction doit allouer et remplir le tableau de père passé en paramètre.

Le squelette fournit réalise déjà l'allocation du tableau de père et propose une sous-fonction pour la partie récursive de l'algorithme.

Le graphe est chargé à l'aide d'un fichier à fournir en paramètre. En exemple de fichier peut être construit à l'aide du programme `make_graphe` fournit. Ce programme prend en paramètre le nom du ficher de sauvegarde du graphe et génère le graphe orienté suivant (à vous de faire le dessin) :

```
Ordre: 7
Transitions:
    6 -> 7
    5 -> 6
    5 -> 3
    4 -> 7
    3 -> 4
```

```
2 -> 3
1 -> 5
1 -> 2
```

Remarques sur les programmes de test

La commande `make questionX` produira un binaire `questionX`. 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 succinte s'il sont appelés avec l'option `-help`.

Exemple : le binaire produit pour la question 4 fournit l'aide suivante :

```
Prog TP: question 4
question4 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` font tous référence à l'initialisation d'un générateur de nombre aléatoire. Dans le cas présent la commande `./question4 X Y` va initialiser le générateur de nombre aléatoire avec X et l'utiliser pour engendrer une liste de taille Y. Pour les mêmes valeurs de X et de Y on obtiendra la même liste (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 jetter un oeil sur le fichier contenant les tests de la question X (`test-qX.ml`.)

Listes des fichiers à ne pas modifier

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 original :

Makefile
question1.ml
question2.mli
question3.mli
question4.mli
question5.h
question6.h
question7.h
question8.h
test-q1.ml
test-q2.ml
test-q3.ml
test-q4.ml
test-q5.c
test-q6.c
test-q7.c
test-q8.c

Listes des fichiers modifiables

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.)

question1.ml
question2.ml
question3.ml
question4.ml
question5.c
question6.c
question7.c
question8.c