Correction du DS 3

Julien Reichert

La correction des questions de cours étant dans le cours, elle ne sera pas donnée ici...

Exercices

Exercice α1 (en OCaml) :

Quitte à récupérer des chaînes de caractères, autant ne pas désérialiser, il y a une tolérance par rapport à la consigne.

Fichier `alpha1.ml` (le `;;` est nécessaire en raison de l'utilisation d'une pseudo-instruction, la commencer par `let _ =` convient aussi) :

```
let sommepossible nstr m =
  let somme = ref 0 in
  let i = ref 0 in
  while !i < String.length nstr && !somme < m do
    if nstr.[!i] <> '-' then (* Rien ne l'a garanti ! *)
      somme := !somme + int_of_char nstr.[!i] - 48;
    incr i
  done; !somme = m;;

if sommepossible Sys.argv.(1) (int_of_string Sys.argv.(2))
  then print_endline "VRAI"
  else print_endline "FAUX"
```

Compilation :

```
ocamlc alpha1.ml -o alpha1
```

Exécution :

```
./alpha1 1234 6
```

Exercice α2 (en C) :

```
struct l_c_t_i;
typedef struct l_c_t_i lcti;
lcti creer_lcti();
void ajout_fin(lcti* m);
void retrait_fin(lcti* m);
bool est_vide_lcti(lcti m);
int* contenu_maillon(lcti m);
lcti* maillon_suivant(lcti* m);

struct trlcti { int taille ; lcti support ; };
typedef struct trlcti itr;
```

```
itr creer_itr ()
{
  itr reponse = { .taille = 0, .support = creer_lcti() };
  return reponse;
}

void append_itr(itr* t, int valeur)
{
  int indice_maillon = t->taille / 64:
  int indice_dans_maillon = t->taille % 64;
  if (indice_dans_maillon == 0) ajout_fin(&(t->support)); // Parenthèses sans doute inutiles, pas vérifié
  lcti* maillon = &(t->support);
  for (int j = 0 ; j < indice_maillon ; j += 1)
  {
    maillon = maillon_suivant(maillon);
  }
  contenu_maillon(*maillon)[indice_dans_maillon] = valeur;
  t->taille += 1;
}

void pop_itr(itr* t, int valeur)
{
  if (t->taille == 0) exit(1);
  int indice_maillon = t->taille / 64:
  int indice_dans_maillon = t->taille % 64;
  if (indice_dans_maillon == 1) retrait_fin(&(t->support));
  t->taille -= 1;
}

int acces_itr(itr t, int i)
{
  if (i < 0 || i >= t.taille) exit(1);
  int indice_maillon = i / 64;
  int indice_dans_maillon = i % 64;
  lcti* maillon = &t.support;
  for (int j = 0 ; j < indice_maillon ; j += 1)
  {
    maillon = maillon_suivant(maillon);
  }
  return contenu_maillon(*maillon)[indice_dans_maillon];
}

void modification_itr(itr t, int i, int valeur)
{
  if (i < 0 || i >= t.taille) exit(1);
  int indice_maillon = i / 64;
  int indice_dans_maillon = i % 64;
  lcti* maillon = &t.support;
  for (int j = 0 ; j < indice_maillon ; j += 1)
  {
    maillon = maillon_suivant(maillon);
  }
  contenu_maillon(*maillon)[indice_dans_maillon] = valeur;
}
```

Exercice β1 :

```
let plus_grand_nombre n =
  if n < 0 then failwith "On voulait un nombre positif !";
  let chiffres = Array.make 10 0 in
  let rec mouline m =
    if m > 0 then
    begin
      chiffres.(m mod 10) <- chiffres.(m mod 10) + 1;
      mouline (m / 10)
    end
  in mouline n;
  let reponse = ref 0 in
  for i = 9 downto 0 do
    for j = 1 to chiffres.(i) do
      reponse := 10 * !reponse + i
    done
  done;
  !reponse;;
```

Exercice β2 :

```
let rec minmax l = match l with
| [] -> failwith "Cas impossible"
| [a] -> (a, a)
| a::q -> let (mini, maxi) = minmax q in (min a mini, max a maxi);;

let amplitudes l =
  let t = Hashtbl.create 8 in
  let fonction (chaine, valeur) =
    match Hashtbl.find_opt t chaine with
    | Some l -> Hashtbl.replace t chaine (valeur::l)
    | None -> Hashtbl.add t chaine [valeur]
  in List.iter fonction l;
  let construction cle valeur =
    let mini, maxi = minmax valeur in
    (cle, maxi - mini) in
  Hashtbl.fold (fun cle valeur accu -> construction cle valeur :: accu) t [];;
```

Exercice β3 :

```
let supplementaire l1 l2 =
  let t1 = Hashtbl.create 8 in
  let t2 = Hashtbl.create 8 in
  let bump t element =
    match Hashtbl.find_opt t element with
    | Some n -> Hashtbl.replace t element (n+1)
    | None -> Hashtbl.add t element 1
  in List.iter (bump t1) l1; List.iter (bump t2) l2;
  let reponse = ref (List.hd l1) in
  Hashtbl.iter (fun cle valeur -> if Hashtbl.find_opt t2 cle <> (Some valeur) then reponse := cle) t1;
  !reponse;;
```

Exercice γ1 :

```
int plus_longue_croissante(char** tab, int taille)
{
  int reponse = -1;
  int taillemax = -1; // Une chaîne vide doit faire mieux
  for (int i = 0 ; i < taille ; i += 1)
  {
    int indice = 0;
    bool croissant = true;
    while (croissant && tab[i][indice] != '\0')
    {
      if (indice == 0 || tab[i][indice] >= tab[i][indice-1]) indice += 1;
      else croissant = false;
    }
    if (croissant && indice > taillemax)
    {
      reponse = i;
      taillemax = indice;
    }
  }
  return reponse;
}
```

Exercice γ2 :

```
void echange(int* tab, int i, int j)
{
  if (i != j)
  {
    int buff = tab[i];
    tab[i] = tab[j];
    tab[j] = buff;
  }
}

void tri_denombrement(int* tab, int taille, int nombre)
{
  int* positions = malloc(nombre * sizeof(int)); // pour nombre-1, ce sera toujours égal à i
  for (int i = 0 ; i < nombre ; i += 1) positions[i] = 0;
  for (int i = 0 ; i < taille ; i += 1)
  {
    int valeur = tab[i];
    for (int j = nombre-1 ; j > valeur ; j -= 1)
    {
      echange(tab, positions[j], positions[j-1]);
      positions[j] += 1;
    }
    positions[valeur] += 1;
  }
  free(positions);
}
```

Problème 1

Question P1.1 :

```
let serialise_liste_entiers liste =
  let rec mouline l = match l with
  | [] -> "]" (* Atteint seulement si liste est vide *)
  | [a] -> string_of_int a ^ "]"
  | a::q -> string_of_int a ^ "; " ^ mouline q
  in "[" ^ mouline liste;;
```

Question P1.2 :

```
int taille_serialisation(int* tab, int taille)
{
  char buff[12];
  int rep = 1; // Pour le '\0'
  for (int i = 0 ; i < taille ; i += 1)
  {
    sprintf(buff, "%d", tab[i]);
    rep += strlen(buff) + 2; // "; " ou les crochets comptabilisés à la place
  }
  return rep;
}

char* serialise_tableau_entiers(int* tab, int taille)
{
  int taille_totale = taille_serialisation(tab, taille);
  char* reponse = malloc(taille_totale * sizeof(int));
  reponse[0] = '[';
  reponse[1] = '\0';
  for (int i = 0 ; i < taille ; i += 1)
  {
    char buff[12];
    sprintf(buff, "%d", tab[i]);
    strcat(reponse, buff);
    if (i < taille - 1) strcat(reponse, "; ");
    else strcat(reponse, "]");
  }
  return reponse;
}
```

Question P1.3 :

En OCaml :

```
let deserialise_liste_entiers chaine =
  if chaine = "[]" then [] else
  let rec mouline buffer indice =
     if chaine.[indice] = ']' then [buffer]
     else if chaine.[indice] = ';' then buffer::mouline 0 (indice + 2)
     else mouline (10 * buffer + int_of_char chaine.[indice] - 48) (indice + 1)
  in mouline 0 1;;
```

En C :

```
int taille_deserialisation(char* chaine)
{
  if (chaine[1] == ']') return 0;
  int rep = 1;
  for (int i = 0 ; chaine[i] != '\0' ; i += 1)
  {
    if (chaine[i] == ';') rep += 1;
  }
  return rep;
}

int* deserialise_tableau_entiers(char* chaine)
{
  int taille_totale = taille_deserialisation(chaine);
  if (taille_totale == 0) exit(1);
  int* reponse = malloc((taille_totale+1) * sizeof(int));
  reponse[0] = 0;
  char buff[12];
  int position_buff = 0;
  for (int i = 1 ; chaine[i] != '\0' ; i += 1)
  {
    if ('0' <= chaine[i] && chaine[i] <= '9')
    {
      buff[position_buff] = chaine[i];
      position_buff += 1;
    }
    else
    {
      buff[position_buff] = '\0';
      reponse[reponse[0]+1] = atoi(buff);
      position_buff = 0;
      reponse[0] += 1;
    }
    if (chaine[i] == ';') i += 1;
  }
  return reponse;
}
```

Question P1.4 :

```
let serialise_foo f =
  Printf.sprintf "{a = %d ; b = %f}" f.a f.b;;

let deserialise_foo chaine =
  let abuff = ref "" in let bbuff = ref "" in
  let traitement_a = ref true in
  for i = 0 to String.length chaine - 1 do
    if chaine.[i] = 'a' then traitement_a := true
    else if chaine.[i] = 'b' then traitement_a := false
    else if chaine.[i] = '.' then bbuff := !bbuff ^ "."
    else if '0' <= chaine.[i] && chaine.[i] <= '9' then
      if !traitement_a then abuff := !abuff ^ (String.make 1 chaine.[i])
      else bbuff := !bbuff ^ (String.make 1 chaine.[i])
  done; { a = int_of_string !abuff ; b = float_of_string !bbuff };;
```

Question P1.5 :

```
char* serialise_foo (struct foo bar)
{
  int taille_serialisation = 18;
  char buffa[12];
  sprintf(buffa, "%d", bar.a);
  char buffb[64]; // En théorie 25, mais cela ne coûte globalement rien
  sprintf(buffb, "%f", bar.b);
  char* reponse = malloc((taille_serialisation + strlen(buffa) + strlen(buffb)) * sizeof(char));
  strcpy(reponse, "{ .a = "); // Du coup pas besoin de '\0' au début si on n'utilise pas strcat
  strcat(reponse, buffa);
  strcat(reponse, " , .b = ");
  strcat(reponse, buffb);
  strcat(reponse, " }");
  return reponse;
}

struct foo deserialise_foo(char* chaine)
{
  struct foo reponse;
  char buffa[12];
  int position_buffa = 0;
  char buffb[64];
  int position_buffb = 0;
  bool traitement_a = true;
  for (int i = 0 ; chaine[i] != '\0' ; i += 1)
  {
    if (chaine[i] == 'a') traitement_a = true;
    else if (chaine[i] == 'b') traitement_a = false;
    else if (chaine[i] == '.' && (chaine[i+1] != 'a' && chaine[i+1] != 'b'))
    {
      buffb[position_buffb] = '.';
      position_buffb += 1;
    }
    else if ('0' <= chaine[i] && chaine[i] <= '9')
    {
      if (traitement_a)
      {
        buffa[position_buffa] = chaine[i];
        position_buffa += 1;
      }
      else
      {
        buffb[position_buffb] = chaine[i];
        position_buffb += 1;
      }
    }
  }
  buffa[position_buffa] = '\0';
  buffb[position_buffb] = '\0';
  reponse.a = atoi(buffa);
  reponse.b = atof(buffb);
  return reponse;
}
```

Problème 2

Question P2.1 :

```
(nn.reste nn).tete
```

Question P2.2 :

```
let n123 = { tete = 1 ; reste = fun fl -> { tete = (fl.tete mod 3) + 1 ; reste = fl.reste }};;
```

Question P2.3 :

(On peut envisager de créer une fonction principale qui vérifie une fois pour toutes que l'entier est positif.)

```
let rec flux_to_list fl n = match n with
| 0 -> []
| _ -> fl.tete :: (flux_to_list (fl.reste fl) (n-1));;
```

Question P2.4 :

```
int* flux_to_array(flux fl, int taille)
{
  if (taille <= 0) exit(1);
  int* reponse = malloc(taille * sizeof(int));
  milc* maillon = tete_flux(fl);
  for (int i = 0 ; i < taille ; i += 1)
  {
    reponse[i] = acceder_maillon(maillon);
    maillon = suivant_flux(maillon);
  }
  return reponse;
}
```

Question P2.5 :

```
void retire(flux* fl)
{
  if (est_vide_flux(*fl)) return;
  milc* maillon = tete_flux(*fl);
  bool retirer = true;
  while (!est_dernier(maillon))
  {
    milc* m2 = suivant_flux(maillon);
    if (retirer) retirer_flux(fl, maillon);
    retirer = !retirer;
    maillon = m2;
  }
  if (retirer) retirer_flux(fl, maillon);
}
```

Question P2.6 :

```
void fusionne(flux fl)
{
  if (est_vide_flux(fl)) return;
  milc* maillon = tete_flux(fl);
  while (!est_dernier(suivant_flux(maillon)))
  {
    milc* m2 = suivant_flux(maillon);
    if (fusionner)
    {
      int n1 = acceder_maillon(maillon);
      int n2 = acceder_maillon(m2);
      modifier_maillon(maillon, n1+n2);
      maillon = suivant_flux(m2);
      retirer_flux(&fl, m2);
    }
  }
}
```

Question P2.7 :

```
void rassemble(flux fl)
{
  if (est_vide_flux(fl)) return;
  milc* maillon = tete_flux(fl);
  int somme = 0;
  while (!est_dernier(maillon))
  {
    int n = acceder_maillon(maillon);
    somme += n;
    modifier_maillon(maillon, somme);
    maillon = suivant_flux(maillon);
    }
  }
}
```

Question P2.8 :

Les éléments du flux après `retire` sur la version en C de `nn` deviennent les premiers entiers naturels impairs dans l'ordre. Après `rassemble`, on aura alors les sommes de ces entiers jusqu'à chacun dans l'ordre, ce qui correspond aux carrés des premiers entiers naturels non nuls dans l'ordre.