Correction du DS 2

Julien Reichert

Questions de cours

Question de cours A1 : Pour prouver la terminaison d'une boucle conditionnelle, on utilise en général un **variant**, c'est-à-dire une expression dépendant des variables utilisées dans le programme, qui s'évalue en un entier positif et qui décroît strictement à chaque tour de boucle. L'idée est de montrer l'impossibilité que le nombre de tours soit infini. Bien entendu, le corps de boucle doit terminer lui aussi.

Question de cours A2 : Pour calculer la complexité d'une fonction récursive, on fait apparaître une relation de dépendance entre la complexité d'un appel et la complexité des appels que celui-ci déclenche. Cela amène en général à la résolution mathématique d'une relation de récurrence.

Question de cours A3 : Les flottants sont représentés selon un principe proche de l'écriture scientifique, mais en binaire. Un bit matérialise le signe, un certain nombre (11 sur 64) la puissance de deux, et le reste donne des précisions sur la partie après la virgule.

Questions de cours en OCaml

Question de cours B1 :

```
let dixieme s = if String.length s > 10 then s.[9] else failwith "Trop court";;
```

Question de cours B2 :

```
type listent = V | C of int * listent;;
```

Question de cours B3 :

```
let o1 = { a = 10 ; b = "décembre" ; c = 2022 };;
```

Le mot-clé à ajouter est `mutable` devant le nom de l'attribut lors de la création du type. Alors on écrit :

```
o1.b <- "Décembre";;
```

Questions de cours en C

Question de cours C1 :

Le type `string` n'existe pas. Le type `float` existe bien, mais on ne s'en servira pas, en conformité avec le programme et l'intérêt de ne pas se limiter à 32 bits.

Question de cours C2 :

Les opérateurs booléens usuels sont `&&`, `||` et `!`.

Question de cours C3 :

```
for (int i = 0 ; i < n ; i += 1) t1[i] = t2[i];
```

Exercices en OCaml

Exercice 1 : Il s'agit d'une fonction de composition de deux fonctions (premier argument « rond » deuxième argument). À la manière de ce qu'on fait en mathématiques, la fonction en deuxième argument attend un certain type et en renvoie un autre (ou le même), qui est attendu par la fonction en premier argument, celle-ci renvoyant un type potentiellement encore différent. La composée prend en argument quelque chose du premier type évoqué et renvoie quelque chose du dernier type évoqué, et c'est cette composée que la fonction `c` renvoie, d'où le type de cette fonction `('a -> 'b) -> ('c -> 'a) -> 'c -> 'b`, l'ordre des lettres étant l'ordre d'apparition.

Remarque : toute proposition de réponse correcte à l'ordre près des types variables sera acceptée. Des parenthèses finales ne sont pas nécessaires mais seront acceptées aussi.

On peut utiliser cette fonction ainsi :

```
let c = fun g f -> (fun x -> g (f x));;
let f = int_of_string;;
let g = char_of_int;;
let h = c g f;;
h "106";; (* 'j', le caractère à la position 106 de la table ASCII *)
```

Exercice 2 :

```
let conjonction_colonnes mat =
  let n = Array.length mat.(0) in
  let res = Array.make n true in
  for i = 0 to Array.length - 1 do
    for j = 0 to n - 1 do
      if not mat.(i).(j) then res.(j) <- false
    done
  done; res;;
```

Exercice 3 :

```
let plsp tab p =
  let imax = ref 0 in
  let lmax = ref 0 in
  let l = ref 0 in
  for i = 0 to Array.length tab - 1 do
    if p tab.(i) then
    (
      incr l;
      if !l > !lmax then
      (
        lmax := !l;
        imax := i - !l + 1
      )
    )
    else l := 0
  done; !imax, !lmax;;
```

Exercices en C

Exercice 4 :

```
int pgcd(int a, int b)
{
  int absa = a < 0 ? -a : a; // Afin de présenter l'opérateur ternaire
  int absb = b < 0 ? -b : b;
  int i = a < b ? a : b;
  while (a % i != 0 || b % i != 0) i-= 1;
  return i;
}
```

Exercice 5 : On verra cette année une version en temps linéaire...

```
int mediane (int t[], int taille)
{
  for (int i = 0 ; i < taille ; i += 1)
  {
    int superieurs = 0;
    int inferieurs = 0;
    for (int j = 0 ; j < taille ; j += 1)
    {
      if (t[i] > t[j]) inferieurs += 1;
      else if (t[i] < t[j]) superieurs += 1;
    }
    if (inferieurs <= taille / 2 && superieurs <= taille / 2) return t[i];
  }
}
```

Exercice 6 : La fonction `f` prend en argument deux entiers, le premier noté `n` et le deuxième noté `tailleres`, et elle renvoie un pointeur pour lequel elle a réservé de la mémoire pour écrire ce qui sera la représentation en binaire (en consommant certes quatre octets pour chaque bit de `n` fourni...), sous l'hypothèse que cette représentation soit sur au plus `tailleres` bits (sinon on n'écrit que la fin de `n`).

Problème

Arithmétique des chaînes de caractères

Question P1 : Un nombre à un chiffre n'en demeure pas moins un entier représenté comme les autres, et les caractères prennent moins de place que les entiers dans la mémoire, d'où ce choix (qui est de toute manière sous-optimal, il faut bien le reconnaître). La possibilité de déterminer la fin de la chaîne facilite aussi le traitement.

Question P2 : (On peut aussi renverser la chaîne et utiliser `atoi`...)

```
int atoi_rev(char *s)
{
  int reponse = 0;
  int dpi = 1; // dix puissance i
  for (int i = 0 ; s[i] != '\0' ; i += 1)
  {
    reponse += dpi * ((int) s[i] - 48); // 48 : position de 0, on caste pour respecter le programme
    dpi *= 10;
  }
  return reponse;
}
```

Question P3 : Il faut réserver une case de plus au cas où une retenue se propagerait au point d'ajouter un chiffre au résultat. Si `i` est plus grand que la taille, on peut gérer aussi...

```
char *retenue(char *s, int i)
{
  int taille = strlen(s);
  int allocation = i > taille ? i+1 : taille+2;
  char *reponse = malloc(allocation);
  for (int j = 0 ; j < taille+1 ; j += 1) reponse[j] = s[j]; // '\0' inclus si taille > i
  for (int j = taille+1 ; j <= i ; j += 1) reponse[j] = '0';
  reponse[i] += 1; // là, je n'ai plus envie de respecter le programme
  int j = i;
  while (j < taille && reponse[j] > '9')
  {
    reponse[j] = '0';
    j += 1;
    reponse[j] += 1;
  }
  if (j >= taille)
  {
    reponse[j] = '1'; // cas particulier où on a écrasé le '\0', redondance si i >= taille
    reponse[j+1] = '\0'; // on assure le coup
  }
  return reponse;
}
```

Question P4 :

```
char *addition(char *s, char *t)
{
  int tailles = strlen(s);
  int taillet = strlen(t);
  int taille = tailles > taillet ? tailles : taillet;
  char *reponse = malloc(taille+2);
  int ret = 0;
  for (int i = 0 ; i < taille ; i += 1)
  {
    int som = ret;
    if (i < tailles) som += s[i]-48;
    if (i < taillet) som += t[i]-48;
    if (som > 10)
    {
      som -= 10;
      ret = 1;
    }
    else ret = 0;
    reponse[i] = som + 48;
  }
  if (ret == 1)
  {
    reponse[taille] = '1';
    reponse[taille+1] = '\0';
  }
  else reponse[taille] = '\0';
  return reponse;
}
```

Question P5 :

```
char *multiplication(char *s, char *t)
{
  int tailles = strlen(s);
  int taillet = strlen(t);
  int taille = tailles + taillet;
  char *reponse = malloc(taille+1); // On ne débordera pas.
  for (int i = 0 ; i < taille ; i += 1) reponse[i] = '0';
  reponse[taille] = '\0';
  for (int i = 0 ; i < tailles ; i += 1)
  {
    for (int j = 0 ; j < taillet ; j += 1)
    {
      int valeur = (s[i] - 48)*(t[j] - 48);
      int unites = valeur % 10;
      int ret = valeur / 10;
      reponse[i+j] += unites;
      if (reponse[i+j] > '9')
      {
        reponse[i+j] -= 10;
        ret += 1;
      }
      int pos = i+j+1;
      while (ret != 0)
      {
        reponse[pos] += ret;
        if (reponse[pos] > '9')
        {
          reponse[pos] -= 10;
          ret = 1;
        }
        else ret = 0;
      }
    }
  }
  if (reponse[taille-1] == '0') reponse[taille-1] = '\0';
  // Cela ne peut arriver qu'une fois si la chaîne ne commence pas par un zéro
  return reponse;
}
```

Arithmétique des tableaux d'entiers

Question P6 : Les calculs se feront paquet par paquet, et on ne peut pas se permettre d'avoir de multiplications qui causent des dépassements d'entiers.

Question P7 : Attention, cette fois-ci le paquet le plus petit sera à droite, occasionnant des décalages. Par précaution on crée un paquet de plus, quitte à le supprimer à la fin.

```
let acces tab pos = if pos >= 0 then tab.(pos) else 0;;

let dptu = 2147483648;; (* 2 puissance 31 *)

let vraie_diveucl a b =
  if a >= 0 then (a / b, a mod b)
  else (a / b - 1, a mod b + b);;
```

```
let somme t1 t2 =
  let n1 = Array.length t1 in
  let n2 = Array.length t2 in
  let n = 1 + max n1 n2 in
  let reponse = Array.make n 0 in
  let i1 = ref (n1 - 1) in
  let i2 = ref (n2 - 1) in
  let i = ref (n - 1) in
  let retenue = ref 0 in
  while !i1 >= 0 || !i2 >= 0 do
    let contenu = acces t1 !i1 + acces t2 !i2 + !retenue in
    let (q, r) = vraie_diveucl contenu dptu in
    reponse.(!i) <- r;
    retenue := q; (* de -2 à 1, valeurs négatives possibles si !i1 ou !i2 est nul *)
    decr i; decr i1; decr i2
  done;
  reponse.(!i) <- retenue;
  let j = ref 0 in while reponse.(!j) = 0 then incr j done;
  Array.sub reponse !j (n-!j);;
```

Question P8 : On reprend le principe de la P5, la gestion n'est pas forcément plus difficile dans l'absolu vu qu'on a exclu les nombres négatifs. On réutilise `dptu` défini avant.

```
let produit t1 t2 =
  let n1 = Array.length t1 in
  let n2 = Array.length t2 in
  let n = n1+n2 in
  let reponse = Array.make n 0 in
  for i1 = 0 to n1-1 do
    for i2 = 0 to n2-1 do
      let contenu = t1.(n1-1-i1) * t2.(n2-1-i2) + reponse.(n-1-i1-i2) in
      let q = contenu / dptu in
      let r = contenu mod dptu in
      reponse.(n-1-i1-i2) <- r;
      reponse.(n-1-i1-i2-1) <- reponse.(n-1-i1-i2-1) + q;
      let j = ref (n-1-i1-i2-1) in
      while reponse.(!j) > dptu do (* pas besoin de vérifier, j ne peut pas valoir -1 *)
        reponse.(!j) <- reponse.(!j) - dptu; (* impossible de déborder plus d'une fois *)
        decr j;
        reponse.(!j) <- reponse.(!j) + 1
      done
    done
  done;
  if reponse.(0) = 0 then Array.sub reponse 1 (n-1) else reponse;; (* la vraie taille est n ou n-1 *)
```

Question P9 : Il ne faut déjà pas tomber dans le piège d'oublier les négatifs, puis savoir comment les gérer est encore une autre affaire...

```
let pas_de_retenue t =
  let reponse = ref true in
  let j = ref 1 in
  while !reponse && !j < Array.length t do
    if t.(!j) <> 0 then reponse := false;
    incr j
  done; !reponse;;
```

```
let imprime_tableau t =
  let n = Array.length t in (* supposé non nul *)
  if t.(0) > 0 || pas_de_retenue t then
  (
    print_int t.(0);
    for i = 1 to n-1 do Printf.printf "%09d" t.(i) done
  )
  else
  (
    print_int (t.(0) + 1);
    for i = 1 to n-2 do Printf.printf "%09d" (999999999 - t.(i)) done;
    Printf.printf "%09d" (1000000000 - t.(n-1))
  );
  print_newline ();;
```

Question P10 : On ne peut pas envisager de calculer la valeur exacte du nombre représenté, qui causerait des débordements. Le problème est que chaque puissance de deux peut influencer le chiffre des unités. Une possibilité est de créer une fonction qui permet de construire à partir d'un tableau représentant un entier d'après la première méthode un tableau représentant le même entier d'après la seconde méthode (attention à la taille qui peut différer à force), en se servant de l'arithmétique des tableaux d'entiers adaptée à la nouvelle base et en faisant des sommes de puissances de deux, sommes calculées idéalement avec la méthode de Horner. Il faudra prendre bien garde à recycler la mémoire dans ce cas.