Correction du DS 1

Julien Reichert

La correction des questions de cours étant dans le cours, elle ne sera pas donnée ici. De même, les exercices issus des TD et TP ont leur correction déjà publiée.

Exercices

Exercice 1 :

```
int plus_grand_ecart(int* t, int n)
{
  if (n < 1) exit(1); // Par principe, mais on peut aussi renvoyer 0
  int rep = 0;
  for (int i = 1 ; i < n ; i += 1)
  {
    int ecart = t[i] - t[i-1];
    if (ecart > rep) rep = ecart;
  }
  return ecart;
}
```

Exercice 2 :

```
int plus_petit_ecart(int* t, int n)
{
  if (n < 1 || t[n-1] == t[0]) exit(1);
  int rep = -1; // Initialisation possible si on est prudent
  for (int i = 1 ; i < n ; i += 1)
  {
    int ecart = t[i] - t[i-1];
    if (ecart > 0 && (rep == -1 || ecart < rep)) rep = ecart;
  }
  return ecart;
}
```

Exercice 3 :

```
int nombre_superieurs(int* t1, int n1, int* t2, int n2) // Ordre arbitraire
{
  int n = n1 < n2 ? n1 : n2;
  int rep = 0;
  for (int i = 0 ; i < n ; i += 1)
  {
    if (t1[i] > t2[i]) rep += 1;
  }
  return rep;
}
```

Exercice 4 :

```
char* identifiant(char* identite)
{
  int nombre_mots = 1; int indice = 1; int taillenom = 0;
  while (identite[indice] != '\0')
  {
    if (lettre(identite[indice]) && !lettre(identite[indice-1]))
    {
      nombre_mots += 1; taillenom = 0;
    }
    else if (lettre(identite[indice]))
    {
      taillenom += 1; // Vaudra le nombre de lettres du dernier mot à la fin
    }
    indice += 1;
  }
  if (nombre_mots > 2)
  {
    char* rep = malloc((nombre_mots + 1) * sizeof(char));
    rep[0] = identite[0];
    int indecr = 1; int indice = 1;
    while (identite[indice] != '\0')
    {
      if (lettre(identite[indice]) && !lettre(identite[indice-1]))
      {
        rep[indecr] = identite[indice];
        indecr += 1;
      }
      indice += 1;
    }
    rep[indecr] = '\0';
    return rep;
  }
  else
  {
    char* rep = malloc((taillenom + 2) * sizeof(char));
    rep[0] = identite[0];
    int indecr = -1; int indice = 1;
    while (identite[indice] != '\0')
    {
      if (lettre(identite[indice]) && !lettre(identite[indice-1]))
      {
        indecr = 1;
      }
      else if (indecr == 1 && lettre(identite[indice]))
      {
        rep[indecr] = identite[indice];
        indecr += 1
      }
      indice += 1;
    }
    rep[indecr] = '\0';
    return rep;
  }
}
```

On aura écrit au préalable les deux fonctions suivantes :

```
bool lettre(char c)
{
  return 'A' <= c && c <= 'Z' || 'a' <= c && c <= 'z';
}

char minus(char c)
{
  return 'A' <= c && c <= 'Z' ? (char) ((int) c + 32) : c;
}
```

Exercice 5 :

```
bool augmentation(int** tab, int n)
{
  for (int i = 1 ; i < n ; i += 1)
  {
    for (int j = 0 ; j < i ; j += 1)
    {
      if (tab[j][0] >= tab[i][0] && tab[j][1] >= tab[i][1]) return false;
    }
  }
  return true;
}
```

Exercice 6 :

```
double** elargir(double** mat, int n, int m)
{
  double** rep = malloc((2*n-1) * sizeof(double*));
  for (int i = 0 ; i < 2*n-1 ; i += 1) rep[i] = malloc((2*m-1) * sizeof(double));
  for (int i = 0 ; i < n ; i += 1)
  {
    for (int j = 0 ; j < m ; j += 1)
    {
      rep[2*i][2*j] = mat[i][j];
    }
    for (int j = 1 ; j < m ; j += 1)
    {
      rep[2*i][2*j-1] = (rep[2*i][2*j-2] + rep[2*i][2*j]) / 2;
    }
  }
  for (int i = 1 ; i < n ; i += 1)
  {
    for (int j = 0 ; j < m ; j += 1)
    {
      rep[2*i-1][2*j] = (rep[2*i-2][2*j] + rep[2*i][2*j]) / 2;
    }
    for (int j = 1 ; j < m ; j += 1)
    {
      rep[2*i-1][2*j-1] = (rep[2*i-1][2*j-2] + rep[2*i-1][2*j]) / 2;
    }
  }
  return rep;
}
```

Problème

Question P1 :

Le nombre qui s'écrit $\overline{42}^b$ vaut en particulier $4b + 2$, ce qui n'est pas le cas de 36 qui est un multiple de 4.

Question P2 :

Il suffit de convertir depuis la base b et de comparer au nombre en question, en ajoutant l'obligation que les chiffres représentés soient valides dans la base b.

En pratique, la méthode de Horner permettrait de gérer aussi le cas où le chiffre des unités aurait été mis à la fin.

```
int convertir(int* tab, int taille, int b)
{
  int rep = 0;
  int bpi = 1;
  for (int i = 0 ; i < taille ; i += 1)
  {
    rep += tab[i] * bpi;
    bpi *= b;
  }
  return rep;
}

int maxi(int* tab, int taille)
{
  int rep = tab[0];
  for (int i = 0 ; i < taille ; i += 1)
  {
    if (tab[i] > rep) rep = tab[i];
  }
  return rep;
}

bool exprimable_b(int* tab, int taille, int b, int n)
{
  return convertir(tab, taille, b) == n && maxi(tab, taille) < b;
}
```

Question P3 :

Version avec dichotomie. L'essentiel est de comprendre qu'à même représentation, si la base augmente, le résultat augmente.

Attention, cas particulier : s'il n'y a qu'un chiffre non nul et que c'est celui des unités, la valeur est constante dans toute base et la boucle risque d'être infinie.

```
bool verif_possible(int* tab, int taille)
{
  for (int indice = 1 ; indice < n ; indice += 1)
  {
    if (tab[indice] > 0) return true;
  }
  return false;
}
```

```
bool exprimable(int* tab, int taille, int n)
{
  int bmini = maxi(tab, taille) + 1;
  if (bmini < 2) bmini = 2;
  int test = convertir(tab, taille, bmini);
  if (test > n) return false;
  if (!verif_possible(tab, taille)) return tab[0] == n;
  while (test < n)
  {
    bmini *= 2;
    test = convertir(tab, taille, bmini);
  }
  if (test == n) return true;
  bmaxi = bmini;
  bmini = bmini / 2 + 1;
  while (bmini <= bmaxi)
  {
    btest = (bmini + bmaxi) / 2;
    test = convertir(tab, taille, btest);
    if (test == n) return true;
    if (test < n) bmini = btest + 1;
    else bmaxi = btest - 1;
  }
  return false;
}
```