ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours d'Information, Calcul et Communication, sections MA et PH
Chappelier J.-C.

Information, Calcul et Communication (SMA/SPH) :
Correction de l'Examen II

21 novembre 2019

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (9h15 – 11h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.
   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée, **MAIS** ne mélangez pas les réponses d'exercices différents !
   Ne joignez aucune feuilles supplémentaires ; **seul ce document sera corrigé**.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. L'examen comporte trois exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 54). Toutes les questions comptent pour la note finale.

Question 1 — Spectre et filtre [sur 13 points]

On s'intéresse ici à représenter, en C++, la notion de spectre et de filtre passe-bas idéal.

Question 1.1 – Spectre [sur 4 points]

Quel est le spectre du signal

$$X(t) = 4\,\sin(6\,\pi\,t + \frac{\pi}{2}) + 2\,\sin(2\,\pi\,t + \frac{\pi}{3}) + 5\,\sin(12\,\pi\,t + \frac{\pi}{4}) \quad ?$$

Répondez à cette question en remplissant le code C++ suivant (en supposant que le type `Spectre` existe ; voir juste ci-dessous) :

```
Spectre spectre_de_X( { { 4, 3 }, { 2, 1 }, { 5, 6 } } );
```

ou (inversion amplitude–fréquence)

```
Spectre spectre_de_X( { { 3, 4 }, { 1, 2 }, { 6, 5 } } );
```

ou n'importe quelle permutation au niveau 1 de l'un des deux précédents, comme p.ex. :

```
Spectre spectre_de_X( { { 2, 1 }, { 4, 3 }, { 5, 6 } } );
```

Définissez alors le ou les type(s) nécessaire(s) à une telle représentation (y compris, donc, le type `Spectre` lui-même). Ce type sera utilisé dans la suite de la question.

```
struct Point {
  double a; // amplitude
  double f; // frequence
};

typedef vector<Point> Spectre;
```

ou aussi, tout simplement :

```
typedef vector<array<double, 2>> Spectre;
```

Question 1.2 – Bande passante [sur 4.5 points]

Définissez une fonction qui prend un `Spectre` tel que défini plus haut et retourne sa bande passante :

```
double bande_passante(Spectre const& s)
{
  double bp(0.0);
  for (auto p : s) {
    if (p.f > bp) bp = p.f; // ou bp = max(p.f, bp);
  }
  return bp;
}
```

Note : l'inégalité peut être large.

Question 1.3 – Filtre passe-bas idéal [sur 4.5 points]

Définissez une fonction qui prend un `Spectre` et une fréquence f_c et retourne le spectre filtré par un filtre passe-bas idéal de fréquence de coupure f_c :

```
Spectre filtre_pbi(Spectre const& s, double f_c)
{
  Spectre filtre;
  for (auto p : s) {
    if (p.f <= f_c) filtre.push_back(s);
  }
  return filtre;
}
```

Note : l'inégalité peut très bien être stricte ici.

Question 2 — Un chien et ses maîtres [sur 18 points]

On s'intéresse ici à résoudre à l'aide d'un programme le problème mathématique suivant :

> Un chien est aux pieds de son maître lorsqu'il aperçoit sa maîtresse à 200 m devant eux et fonce vers elle. Arrivé à sa maîtresse, le chien fait demi-tour et revient en courant vers son maître. Et ainsi de suite. Le maître marche vers sa femme (la maîtresse du chien) à vitesse constante de 0.7 m/s, la femme vient vers son mari à vitesse constante de 0.8 m/s et l'on suppose que le chien court tout le temps, sans s'arrêter, à vitesse moyenne de 2.5 m/s.
>
> Quelle est la distance parcourue par le chien lorsque son maître et sa maîtresse se rencontrent ?

Question 2.1 – Déplacements [sur 4.5 points]

Pour traiter ce problème, commencez par définir un type `Moving` représentant une entité qui peut se déplacer (maîtresse, maître ou chien). Une telle entité est représentée par :

- sa position (ici en 1D : uniquement suivant le segment de droite joignant la position de départ du maître à celle de la maîtresse) ;
- sa vitesse ;
- la distance totale parcourue (nous allons aussi calculer les distances parcourues par le maître et la maîtresse).

On aurait ainsi dans le `main()` :

```
Moving master  ({   0.0    ,  0.7, 0.0 });
Moving mistress({ 200.0    , -0.8, 0.0 });
Moving dog     ({ master.x,  2.5, 0.0 });
```

Définition du type `Moving` :

```
struct Moving {
  double x;
  double v;
  double tot_dist;
};
```

Définissez ensuite une fonction `move()` qui prend un `Moving` et un « pas de temps » (c.-à-d. un délais, un écart de temps ; en secondes).

Cette fonction modifie l'entité reçue et la fait avancer de la distance correspondant au « pas de temps » reçu :

- elle ajoute à sa position le produit de sa vitesse par le « pas de temps » ($\Delta x = v\,\Delta t$) ;
- et ajoute à sa distance totale la valeur absolue (fonction `abs()`) du déplacement ci-dessus (produit de sa vitesse par le « pas de temps »).

Définition de la fonction `move()` :

```
void move(Moving& obj, double dt)
{
  const double dx(obj.v * dt);
  obj.x += dx;
  obj.tot_dist += abs(dx);
}
```

Question 2.2 – Égalité de positions [sur 3 points]

Comme il est déconseillé de tester directement l'égalité de deux `double`, et que deux êtres humains, encore plus avec un chien, sont rarement exactement au même point, nous vous demandons d'écrire ici une fonction `together()` qui teste si la position de deux `Moving` sont égales à une précision donnée près (c.-à-d. si la valeur absolue de leur différence est inférieure à cette précision).

```
bool together(Moving const& a, Moving const& b, double precision)
{
  return abs(a.x - b.x) < precision ;
}
```

NOTE : ici l'inégalité peut être stricte ou large ; la donnée ne le précisait pas.

Question 2.3 – Affichage [sur 2.5 points]

Ecrire une fonction `display()` qui prend en arguments une chaîne de caractères et un `Moving` et qui affiche le texte reçu, puis les caractéristiques de l'entité en mouvement, p.ex. comme :

```
Maîtresse : distance parcourue = 106.48 m (position = 93.52)
```

```
void display(string_view txt, Moving const& a)
{
  cout << txt << " : ";
  cout << "distance parcourue = " << a.tot_dist << " m ";
  cout << "(position =" << a.x << ")" << endl;
}
```

Remarque : `string_view` est totalement optionnel ; `string` ou `string const&` vont tout aussi bien au niveau de ce cours.

Question 2.4 – Résolution du problème [sur 8 points]

Pour finir, on souhaite compléter le `main()` suivant de sorte à résoudre le problème évoqué au départ *en faisant faire au programme la simulation complète du déplacement* :

```
int main() {
  Moving master  ({   0.0   ,  0.7, 0.0 });
  Moving mistress({ 200.0   , -0.8, 0.0 });
  Moving dog     ({ master.x,  2.5, 0.0 });

  double t(0.0);
  constexpr double dt(0.1);   // pas de temps (en secondes)
  constexpr double prec(0.2); // précision de position

  // A COMPLÉTER ICI

  cout << "Temps total : " << t << " s." << endl;
  display("Maître",    master  );
  display("Maîtresse", mistress);
  display("Chien",     dog     );

  return 0;
}
```

Pour éviter des problèmes de précision :

- on utilisera la valeur de `prec` pour évaluer si le chien a rejoint son maître ou sa maîtresse, mais **deux** fois la valeur de `prec` pour savoir si le maître et la maîtresse sont ensembles ;
- le chien ne fait demi-tour (inversion de sa vitesse) que s'il est près de son maître/de sa maîtresse **et** que sa vitesse est dans le bon sens (positive quand il va vers sa maîtresse et négative lorsqu'il va vers son maître).

Juste pour fixer les idées, le `main()` complété produirait alors la sortie suivante :

```
Temps total : 133.1 s.
Maître : distance parcourue = 93.17 m (position = 93.17)
Maîtresse : distance parcourue = 106.48 m (position = 93.52)
Chien : distance parcourue = 332.75 m (position = 93.25)
```

(il n'y a pas d'affichage pendant la simulation des déplacements).

Ecrivez ici le code nécessaire pour la partie « *// A COMPLÉTER ICI* » du `main()` ci-contre :

```
while (not together(master, mistress, 2*prec)) {
    move(master  , dt);
    move(mistress, dt);
    move(dog     , dt);

    if (   ((dog.v > 0) and together(dog, mistress, prec))
        or ((dog.v < 0) and together(dog, master  , prec))) {
      dog.v = - dog.v;
    }

    t += dt;
}
```

Question 3 — Coder et décoder [sur 23 points]

On s'intéresse ici au codage de lettres par des mots de codes. On souhaite p.ex. représenter le fait que

- la lettre 'A' est codée par « `100` »,
- la lettre 'B' est codée par « `010` »,
- la lettre 'C' est codée par « `011` »,
- la lettre 'D' est codée par « `11` »,
- la lettre 'E' est codée par « `00` »,
- et qu'aucune autre lettre n'est codée.

On décide par convention que les lettres à encoder sont toutes consécutives dans l'alphabet.

Un tel code sera alors représenté par :

- la première lettre codée par ce code ('A' dans l'exemple ci-dessus) ;
- et un ensemble de chaînes de caractères, représentant les différents mots de code pour chacune des lettres dans l'ordre alphabétique.

Par exemple, le code de l'exemple ci-dessus sera représenté par :

```
{ 'A',
```

```
  { "100", "010", "011", "11", "00" }
}
```

Question 3.1 – Type `Code` [sur 2 points]

Définissez ici le type `Code` correspondant à la description précédente.

```
struct code {
  char first;
  vector<string> codewords;
};
```

Question 3.2 – Code sans préfixe [sur 5.5 points]

On suppose à partir de maintenant qu'il *existe* une fonction `is_prefix()` qui prend deux chaînes de caractères et renvoie « `true` » si la première est préfixe de la suivante, et renvoie « `false` » sinon. Par exemple :

`is_prefix("01", "0110010")` renvoie `true`, et

`is_prefix("01", "1001")` renvoie `false`.

Définissez ici une fonction `is_prefix_free()` permettant de déterminer si un `Code` tel que défini précédemment est un code « sans préfixe ». Vu la définition du type `Code` et le fait que nous n'avons par encore vu comment représenter des arbres, utilisez ici simplement un algorithme en $\mathcal{O}(n^2)$, où n est le nombre de mots de code.

```
bool is_prefix_free(code const& c)
{
  const size_t last(c.codewords.size() - 1);
  for (size_t i(0); i < last; ++i) {
    for (size_t j(i+1); j < c.codewords.size(); ++j) {
      if (is_prefix(c.codewords[i], c.codewords[j])) return false;
      if (is_prefix(c.codewords[j], c.codewords[i])) return false;
    }
  }
  return true;
}
```

Notes : il y a plusieurs façons d'écrire les boucles, mais il faut faire attention à bien comparer toutes les paires (fait ici avec deux `if`, mais on peut aussi faire commencer `j` au début) et de ne pas comparer un mot de code à lui-même (car il sera préfixe de lui-même !).

Question 3.3 – Encodage [sur 6.5 points]

Définissez ici une fonction `encode()` qui reçoit une chaîne de caractères et un code et qui retourne la chaîne encodée correspondante. Par exemple, avec le code précédent

```
Code c({ 'A',
         { "100", "010", "011", "11", "00" } });
```

l'appel

```
encode("ECDAB", c)
```

retournera la chaîne `"0001111100010"`.

Si la chaîne à encoder contient des caractères que le code ne connaît pas, la fonction `encode()` retournera simplement la chaîne vide. Par exemple, toujours avec le code précédent, l'appel

```
encode("EF", c)
```

retournera la chaîne vide car la lettre 'F' n'est pas dans le code fourni.

Remarque : on supposera ici qu'un `char` peut s'utiliser directement comme un `size_t`.

```
string encode(string_view s, code const& c)
{
  string output;
  for (auto x : s) {
    if (x - c.first < c.codewords.size())
      output += c.codewords[x - c.first];
    else
      return ""s;
  }
  return output;
}
```

Remarques :

- `string_view` est totalement optionnel ; `string` ou `string const&` vont tout aussi bien au niveau de ce cours.
- l'hypothèse « *un* `char` *peut s'utiliser directement comme un* `size_t` » inclut également le fait que si `x < c.first` alors `x - c.first >= c.codewords.size()` ; mais c'est mieux de penser à faire ce test (`x >= c.first`) en soi.

Question 3.4 – Décodage [sur 9 points]

Définissez ici (et sur la page suivante) une fonction `decode()` qui reçoit une chaîne de caractères et un code et qui retourne la chaîne décodée correspondante. Par exemple, avec le code précédent

```
Code c({ 'A',
         { "100", "010", "011", "11", "00" } });
```

l'appel

```
decode("0001111100010", c)
```

retournera la chaîne `"ECDAB"`.

Si la chaîne reçue est impossible à décoder (arrive sur une sous-séquence qui n'est pas un mot de code), la fonction `decode()` ajoutera simplement le message « `- ERROR: cannot decode further:` » à la fin de la chaîne déjà décodée, puis ajoutera le morceau de chaîne reçue qu'il reste à décoder (celui à partir duquel s'est produite l'erreur, donc). Par exemple, toujours avec le code précédent, l'appel

```
decode("10011101011", c)
```

retournera la chaîne `"AD - ERROR: cannot decode further: 101011"`, car `10011` a pu être correctement décodé (en `AD`), mais `101` ne correspond à aucun début de mot de code.

Vu que nous n'avons par encore vu comment représenter des arbres, utilisez ici pour décoder simplement un parcours de tous les mots de codes.

Indications :

- n'oubliez pas `is_prefix()` présentée en Question 3.2 ;
- on supposera ici qu'un `size_t` peut s'utiliser *directement* comme un `char` (et qu'ajouter `1` à `'A'` donne `'B'`).

```
string decode(string_view s, code const& c)
{
  string output;

  if (is_prefix_free(c)) {
    size_t incr(0);
    for (size_t i(0); i < s.size(); i += incr) {
      incr = 0;
      for (size_t j(0); j < c.codewords.size(); ++j) {
        if (is_prefix(c.codewords[j], s.substr(i, c.codewords[j].size()))) {
          incr = c.codewords[j].size();
          output += c.first + j;
        }
      }
      if (incr == 0) {
        output += " - ERROR: cannot decode further: ";
        output += s.substr(i);
        return output;
      }
    }

  } else { // non prefix-free
    output = "ERROR: cannot decode: non prefix-free code";
    // ou n'importe quoi d'autre qui fait sens...
  }

  return output;
}
```

Remarque : `string_view` est totalement optionnel ; `string` ou `string const&` vont tout aussi bien au niveau de ce cours.