Ce document n'étant pas (normalement) prévu à destination des élèves, veuillez m'excuser des éventuelles fautes d'orthographe, de l'indentation, et quelques raccourcis (e.g., variables globales).

Encore une fois, le fait d'utiliser des .jpeg est volontaire.

Includes and co.

```
//============================================================================
/// \name       allPermutationWithSTL.cpp
/// \author     Saplanque
/// \version        1.0 (2019.2020)
/// \brief      Algorithme combinatoire récursif.
/// Les balises des commentaires sont ici grossierement du Doxygen, quelques
/// bonus des TD&Ps sont lies à doxygen (/// equiv a /**  **/ plus habituelles
/// pour du Doxygen. Cela permet facilement ensuite de generer une documentation
/// en HTML ou Latex.
//============================================================================
#include <bits/stdc++.h>
#include <cmath>
using namespace std;
constexpr int codeASCIIde_a = 97;
constexpr int codeASCIIde_A = 65;
constexpr int nombreDeLettres = 26;
constexpr int tailleMinNomVille = 4;
constexpr int tailleMaxNomVille = 12;
constexpr int grainePourLeRand = 1;
constexpr int nombreDeVilles = 4;
constexpr int nombreCombinaisons = 24;
constexpr int tailleCoteCarte = 100;
```

On se permet quelques variables globales

```
/// Variable globale
std::array<pair<vector<string>, int>, nombreCombinaisons> toutesLesTourneesEtLeurDistanceTotale;
int indiceCombinaison;
/// Declaration des methodes implementees apres le main().
void
toutesLesPermutations (
  std::vector<std::string> &vecVilles,
  int indexDebut,
  int indexDernier,
  std::map<std::string, std::tuple<int, int, int> > &maMapNomsVillesEtCoordonnees,
  std::vector<std::vector<int> > &DIST);
int
calculTotalDistanceTournee (
  std::vector<std::string> &vecteurDeNomsDeVille,
  std::vector<std::vector<int> > &DIST,
  std::map<std::string, std::tuple<int, int, int> > &maMapNomsVillesEtCoordonnees);
bool
compareTwoPairs (pair<vector<string>, int> &tournee1,
       pair<vector<string>, int> &tournee2);
```

Début du main() avec la génération des données liées aux villes et leur affichage

```
int main ()
{
int nbLettresNomVille; indiceCombinaison = 0;
std::vector<std::string> vecteurDeNomsDeVille;
std::map<std::string, std::tuple<int, int, int> > maMapNomsVillesEtCoordonnees;
/// \brief On fixe la graine (a changer pour obtenir des scenarios/instances
///  differents comme avec time(NULL)). (Changer la constante pour cela).
srand (grainePourLeRand);
/// \brief On genere les villes et leurs données
```

```
for (auto i = 0; i < nombreDeVilles; i++)
  {
    std::string monStringTemp;
    int ASCIItempLettre;
    /// \brief On genere le nombre de lettre de la ville i, attention au '+1' dans le modulo
    nbLettresNomVille = tailleMinNomVille
    + rand () % (tailleMaxNomVille - tailleMinNomVille + 1);
    /// \brief On ajoute la majuscule pour la premiere lettre. On utilise push_bach
    /// qui existe aussi (peu connu) pour les std::string.
    /// Ici pas de +1 pour le modulo.
    ASCIItempLettre = codeASCIIde_A + rand () % (nombreDeLettres);
    monStringTemp.push_back (char (ASCIItempLettre));
    /// \brief On demarre a 1 du fait qu on deja genere la lettre majuscule
    for (auto j = 1; j < nbLettresNomVille; j++)
    {
    ASCIItempLettre = codeASCIIde_a + rand () % (nombreDeLettres);
    monStringTemp.push_back (char (ASCIItempLettre));
  }
    /// \brief On genere les coordonnees X et Y de la ville sur la carte de cote tailleCoteCarte
    int tempX, tempY;
    tempX = rand () % (tailleCoteCarte + 1);
    tempY = rand () % (tailleCoteCarte + 1);
    /// \brief On cree un tuple temporaire comportant l index de la ville i, X et Y.
    /// L index de la ville servira pour la matrice des distances DIST.
    /// auto fait du bien car on evite :
    /// std::tuple<int, int, int> tempTuple_indexVille_X_Y = make_tuple(i, tempX, tempY);
    /// enfin pas si on vient d ecrire ce commentaire...
    auto tempTuple_indexVille_X_Y = std::make_tuple (i, tempX, tempY);
    /// \brief On rentre notre tuple dans maMapNomsVillesEtCoordonnees
    maMapNomsVillesEtCoordonnees.insert (
    maMapNomsVillesEtCoordonnees.begin (),
    std::pair<std::string, std::tuple<int, int, int> > (
        monStringTemp, tempTuple_indexVille_X_Y));
    /// \brief On remplit notre vecteur de nom de ville (qui ne sont finalement que
    /// les cles pour la std::map.
    vecteurDeNomsDeVille.push_back (monStringTemp);
  }
```

```
/// \brief On affiche les noms et le tuple de chaque ville.
/// Cette partie montre comment se servir des tuples
/// en C++, chose qui est un peu surprenante avec du C++ :
/// l utilisation des get<i> avec i l index de l element
/// du tuple.
for (auto it = maMapNomsVillesEtCoordonnees.begin ();
    it != maMapNomsVillesEtCoordonnees.end (); ++it)
  {
    std::cout << it->first << " " << get<0> (it->second) << " "
    << get<1> (it->second) << " " << get<2> (it->second) << std::endl;
  }
```

Calcul de la matrice des distances

```
/// \brief On calcule les distances grace a la methode de cmath
/// Pour les eleves utilisant des <array> il sera facile d eviter
/// de recalculer les distances symetriques. Ici, c est bien plus
/// prise de tête.
int indexVilleDepart, indexVilleDArrivee;
for (auto itVilleDepart = maMapNomsVillesEtCoordonnees.begin ();
    itVilleDepart != maMapNomsVillesEtCoordonnees.end (); ++itVilleDepart)
  {
    indexVilleDepart = get<0> (itVilleDepart->second);
    for (auto itVilleDArrivee = maMapNomsVillesEtCoordonnees.begin ();
    itVilleDArrivee != maMapNomsVillesEtCoordonnees.end ();
    ++itVilleDArrivee)
  {
    indexVilleDArrivee = get<0> (itVilleDArrivee->second);
    if ((get<0> (itVilleDepart->second))
        != (get<0> (itVilleDArrivee->second)))
      {
        DIST[indexVilleDepart][indexVilleDArrivee] = hypot (
        get<1> (itVilleDepart->second)
            - get<1> (itVilleDArrivee->second),
        get<2> (itVilleDepart->second)
            - get<2> (itVilleDArrivee->second));
      }
  }
  }
```

Quelques méthodes d'affichage.

```
/// On se rassure en affichant toute la matrice (on peut montrer ce resultat aux etudiants).
for (int i = 0; i < nombreDeVilles; i++)
  {
    for (int j = 0; j < nombreDeVilles; j++)
  {
    std::cout << "\t" << DIST[i][j];
  }
    cout << endl;
  }
/// On lance la methode toutesLesPermutations definie ci apres.
toutesLesPermutations (vecteurDeNomsDeVille, 0,
           vecteurDeNomsDeVille.size () - 1,
           maMapNomsVillesEtCoordonnees, DIST);
cout << " Avant le tri " << endl;
for (int numCombi = 0; numCombi < nombreCombinaisons; numCombi++)
  {
    for (auto itVille =
    toutesLesTourneesEtLeurDistanceTotale[numCombi].first.begin ();
    itVille
        != toutesLesTourneesEtLeurDistanceTotale[numCombi].first.end ();
    itVille++)
  {
    cout << "\t" << *itVille;
  }
    cout <<  " DIST = " << toutesLesTourneesEtLeurDistanceTotale[numCombi].second << endl;
  }
```

Tri sur les distances totales, affichage du résultat et fin du main

```
std::sort (toutesLesTourneesEtLeurDistanceTotale.begin (),
       toutesLesTourneesEtLeurDistanceTotale.end (), compareTwoPairs);
cout << " Apres le tri " << endl;
for (int numCombi = 0; numCombi < nombreCombinaisons; numCombi++)
  {
    for (auto itVille =
    toutesLesTourneesEtLeurDistanceTotale[numCombi].first.begin ();
    itVille
        != toutesLesTourneesEtLeurDistanceTotale[numCombi].first.end ();
    itVille++)
  {
    cout << "\t" << *itVille;
  }
    cout <<  " DIST = " << toutesLesTourneesEtLeurDistanceTotale[numCombi].second << endl;
  }
std::cout << " Fin." << std::endl;
return 0;
}
```

Procédure récursive de permutations.

```
/// \brief Methode recursive generant l ensemble des mots possibles
/// avec les caracteres du mots villes donne en parametre.
///
/// Aide : le premier affichage correspondera a la liste telle quelle.
/// EN effet, on va pousser le i jusqu a qu il soit egal a indexDernier
/// et ceci en faisant des fausses permutations puisque i == indexDebut
/// des le debut du for.
void
toutesLesPermutations (
  std::vector<std::string> &vecVilles,
  int indexDebut,
  int indexDernier,
  std::map<std::string, std::tuple<int, int, int> > &maMapNomsVillesEtCoordonnees,
  std::vector<std::vector<int> > &DIST)
{
/// Si on a fait toutes les permutations on affiche.
if (indexDebut == indexDernier)
  {
    /// On affiche toutes les villes pour la permutation courante.
    for (const std::string &uneVille : vecVilles)
  cout << uneVille << " \t\t";
    int distanceTotaleTournee = calculTotalDistanceTournee (
    vecVilles, DIST, maMapNomsVillesEtCoordonnees);
    toutesLesTourneesEtLeurDistanceTotale[indiceCombinaison] =
    std::make_pair (vecVilles, distanceTotaleTournee);
 //   std::array<pair<vector<string>, int>, nombreDeVilles> toutesLesTourneesEtLeurDistanceTotale;
    indiceCombinaison++;
  }
else /// On doit encore permuter avant d afficher
  {
    /// On fait les permutations par recursion
    for (int i = indexDebut; i <= indexDernier; i++)
  {
    swap (vecVilles[indexDebut], vecVilles[i]);
    /// Appel Recursif
    toutesLesPermutations (vecVilles, indexDebut + 1, indexDernier,
              maMapNomsVillesEtCoordonnees, DIST);
    /// On revient a l état precedent.
    swap (vecVilles[indexDebut], vecVilles[i]);
  }
  }
}
```

Procédure `calculTotalDistanceTournee`

```
/// \brief Methode qui va calculer la distance totale d une tournee. Ainsi,
/// la combinaison des noms de ville (vecteurDeNomsDeVille) représente une tournee
/// il s agit alors d aller chercher la distance entre deux villes consecutives du vector
/// et le faire de facon a boucler.
int
calculTotalDistanceTournee (
  std::vector<std::string> &vecteurDeNomsDeVille,
  std::vector<std::vector<int> > &DIST,
  std::map<std::string, std::tuple<int, int, int> > &maMapNomsVillesEtCoordonnees)
{
/// On prend deux indices temporaires pour chaque couple dont on doit trouver la distance
/// qui les separe.
int indiceVilleDepart, indiceVilleArrivee;
int distanceTotale = 0, distanceEntreDeuxVilles;
/// On a besoin itVecVillesSuivante pour la ville "suivante" d une paire de villes consecutive
/// dans la tournee.
auto itVecVillesSuivante = vecteurDeNomsDeVille.begin ();
itVecVillesSuivante++;
for (auto itVecVilles = vecteurDeNomsDeVille.begin ();
    itVecVilles != vecteurDeNomsDeVille.end (); itVecVilles++)
  {
    /// Si l iterateur suivant de celui pointe par itVecVille est end(), il faut passer à la premiere ville
    /// pour fermer la bouche (Si la tournee est ADTZ, on ferme la bouche avec la distance "retour" ZA, sinon,
    /// on peut prendre la ville qui est effectivement la suivante.
    if (itVecVillesSuivante != --vecteurDeNomsDeVille.end ())
  {
    /// La ville suivante existe bien on peut pointer dessus
    itVecVillesSuivante = itVecVilles;
    itVecVillesSuivante++;
  }
    else
  {
    /// La ville suivante existe bien on peut pointer dessus
    itVecVillesSuivante = vecteurDeNomsDeVille.begin ();
  }
```

```
    /// On cherche l indice des villes dans la map
    indiceVilleDepart = get<0> (
    (maMapNomsVillesEtCoordonnees.find (*itVecVilles))->second);
    indiceVilleArrivee = get<0> (
    (maMapNomsVillesEtCoordonnees.find (*itVecVillesSuivante))->second);
    /// On peut récuperer la distance entre les deux villes et la sommer avec le total.
    distanceEntreDeuxVilles = DIST[indiceVilleDepart][indiceVilleArrivee];
    distanceTotale += distanceEntreDeuxVilles;
    /// Je laisse l'affichage, on peut montrer rapidement le resultat de ce code (les cout).clean
    cout << "\n\t\t\t\t Info sur la tourneee : ";
    for (uint i = 0; i < vecteurDeNomsDeVille.size (); i++)
  {
    cout << vecteurDeNomsDeVille[i] << "\t";
  }
    cout << endl;
    cout << " Pour les villes : " << *itVecVilles << " et "
    << *itVecVillesSuivante << endl;
    cout << "\t\tindiceDep " << indiceVilleDepart << " indiceArr "
    << indiceVilleArrivee << " distance " << distanceEntreDeuxVilles
    << " distanceTotale " << distanceTotale << endl;
  }
return distanceTotale;
}
```

Et enfin le foncteur compareTwoPairs qui permet d'utiliser l'algorithme sort afin de trouver la solution optimale.

```
/// \brief La methode suivante remplace un foncteur,
bool
compareTwoPairs (pair<vector<string>, int> &tournee1,
       pair<vector<string>, int> &tournee2)
{
return tournee1.second < ville2.tournee2;
}
```