Exercice 1. 1er Programme C++.

A l'aide d'un éditeur de code tel que Kate, emacs, vi, etc (évitez gedit si vous n'avez pas mis en place la complétion de code automatique), copiez les lignes suivantes dans un fichier main.cpp (ou main.cxx, etc. suivant vos conventions).

```
#include <iostream>
#include <string>

int main()
{
  std::string variable="ma_variable";
  variable += " a moi";

  std::cout<<variable<<std::endl;

  return 0;
}
```

- **Compilation, approche numéro 1:**

Compilez ce fichier à l'aide de la ligne de commande suivante:
```
$ g++ main.cpp -g -Wall -Wextra -o exercice1
```

Lancez ensuite la l'executable:
```
$ ./exercice1
```

- **Compilation, approche numéro 2:**

En ligne de commande, dans le répertoire du projet, tapez
```
$ qmake -project
## Observez la création d'un fichier [nom_de_votre_repertoire].pro
$ qmake [nom_de_votre_repertoire].pro
## Observez la création d'un Makefile
$ make
$ ./[nom_de_votre_repertoire]
```

Note: Qmake est un utilitaire permettant de générer des Makefile de manière automatique. Il est particulièrement adapté aux projet utilisant Qt. Il peut faire gagner du temps lors de la création de projets avec de nombreux fichiers.

Lancez également QtCreator à partir du fichier `[nom_de_votre_repertoire].pro`.
`$ qtcreator [nom_de_votre_repertoire].pro`

➔ Assurez-vous que QtCreator charge bien votre projet.
En entrant dans le fichier à éditer: variable suivie d'un point, vous devriez obtenir la complétion automatique des fonctions membres de la classe std::string.

- **Compilation, approche numéro 3:**

Créez le fichier `CMakeLists.txt` suivant:

```
cmake_minimum_required(VERSION 2.6)

project(pgm)


file(
GLOB
source_files
*.cpp
*.hpp
)

SET(CMAKE_BUILD_TYPE Debug)
SET(CMAKE_CXX_FLAGS "-Wall -Wextra" )


add_executable(
   pgm
   ${source_files}
   )

target_link_libraries(pgm "-lm")
```

Puis tapez en ligne de commande:

```
$ cmake .
# Observez la création du nouveau Makefile
$ make
# Observez la compilation et la création de votre exécutable
# La compilation doit cette fois faire apparaître des
pourcentages.
$ ./pgm
```

Note: CMake est un outil permettant, tout comme QMake, de générer automatiquement des Makefile pour des projets importants. CMake est plus générale que QMake et potentiellement plus puissant, il n'est pas lié spécifiquement à Qt.

➔ Encore une fois, QtCreator peut être chargé directement à partir du fichier CMakeLists.txt.

Dans la suite de vos codes, choisissez l'approche que vous préférez ainsi que votre éditeur de code. Il n'est pas obligatoire d'utiliser QtCreator si vous préférez un autre éditeur.
L'utilisation de QMake et/ou CMake est conseillée, cependant, vous devez à tout moment rester capable d'écrire votre propre Makefile pour un projet de taille raisonnable.

- ➔ Créez un programme définissant une référence sur une variable.
- ➔ Observez que le passage d'argument par référence permet de modifier le contenu d'une variable dans une fonction (voir exemples de cours).

Rem. Pour activer les fonctions de c++11 lors de la compilation (list comprehension, etc) on ajoutera l'option `-std=c++0x` lors de la compilation avec gcc (valide pour gcc 4.6).
Pour les versions plus récentes de gcc (portables perso), on pourra utiliser l'option `-std=c++11`.

Exercice 2. Introduction à la STL.

La STL est la librairie standard du C++ qui implémente de nombreux types de données de manière efficace et générique.
Les éléments (conteneurs et algorithmes) remplacent avantageusement de nombreuses implémentations en C.

Règle de programmation : Dans un programme C++, on privilégiera toujours l'utilisation de la STL par rapport à une implémentation manuelle tierce : gain en efficacité, robustesse, généricité, lisibilité de par le caractère standard.

Les éléments de la STL sont tous accessibles et reconnaissables par la syntaxe caractéristique `std::` qui précède l'utilisation de ces classes.
On aura ainsi accès aux conteneurs standards, comme par exemple ceux que nous allons voir dans ce TP :

```
std::vector //pour les vecteurs de la STL
std::list //pour les listes de la STL
std::stack //pour les piles de la STL
std::queue //pour les files de la STL
std::map //pour les arbres ordonnés par clé
```

L'ensemble des conteneurs, méthodes et complexité associée sont disponibles ici :
http://www.cplusplus.com/reference/stl/

- **Les vecteurs**

Les vecteurs se comportent comme des tableaux dynamiques qui connaissent leurs propres tailles et sont capables de se redimensionner tout seuls lorsqu'on leur ajoute des éléments.

Un vecteur se définit de cette manière en indiquant le type contenu dans celui-ci entre chevrons < >: (On parle de *template*).

```
//inclusion de la librairie de vecteur de la STL
#include <vector>

int main()
{
    //définition d'un vecteur de nom: mon_vecteur stockant des entiers
    std::vector<int> mon_vecteur;

    //définition d'un vecteur de nom: mon_vecteur2 stockant des doubles
    std::vector<double> mon_vecteur2;

    //définition d'un vecteur de nom: mon_vecteur3 stockant des
    //   pointeurs vers des entiers
    std::vector<int*> mon_vecteur3;


    return 0;
}
```

Les fonctionnalités de bases d'un *std::vector* sont illustrées sur l'exemple suivant :

```cpp
//inclusion de la librairie de vecteurs de la STL
#include <vector>

//inclusion de la librairie de flux d'affichage std::cout<<
#include <iostream>


int main()
{
    //définition d'un vecteur de nom: mon_vecteur stockant des entiers
    std::vector<int> mon_vecteur;

    //redimensionnement de mon_vecteur à une taille de 3
    // il peut alors stocker 3 int
    mon_vecteur.resize(3);

    //affectation de valeurs identique à un tableau standard
    mon_vecteur[0]=1;
    mon_vecteur[1]=-2;
    mon_vecteur[2]=5;

    //récupération de la taille d'un vecteur:
    int N=mon_vecteur.size();//ici N=3

    //parcours du vecteur
    for(int k=0;k<N;++k)
        std::cout<<mon_vecteur[k]<<" "<<std::endl; //affichage des valeurs
                                                   //    du tableau

    //ajout en fin de vecteur avec redimensionnement automatique:
    mon_vecteur.push_back(8); //ici mon_vecteur[3]=8
    mon_vecteur.push_back(-1);//ici mon_vecteur[4]=-1

    //ma nouvelle taille est 5
    std::cout<<mon_vecteur.size()<<std::endl; //affiche 5

    //on a le droit de copier des vecteurs avec l'égalité.
    std::vector<int> autre_vecteur;
    //ici copie des éléments de mon_vecteur dans autre_vecteur
    autre_vecteur=mon_vecteur;

    //en quittant les vecteurs se retaillent à 0.
    // Il n'y pas de gestion mémoire à réaliser explicitement.
    return 0;
}
```

Note : En pratique, en C++ on utilisera *std::vector<type>* pour remplacer avantageusement une déclaration sous forme de tableau dynamique sous la forme *type* var=new type[N]* ;

Dans cet exercice, on va réaliser un programme qui permet de stocker des *std::string* et manipuler ceux-ci de manière simple.
On rappelle que *std::string* est une classe de la STL contenant une chaine de caractère qui remplace avantageusement l'utilisation du type *char**.

➔ Déclarez un vecteur de nom *mon_vecteur* stockant des *std::string.*

➔ Ajoutez 5 *std::string* dans le vecteur qui vaudront respectivement "bonjour", "comment", "allez", "vous", "?".

Notez qu'une syntaxe du type:
`mon_vecteur[k]="ma_chaine_de_caractere";`
affecte l'entrée k du vecteur avec la *std::string* valant "*ma_chaine_de_caractere*". (Il faut que le vecteur ait une taille d'au moins k+1).

Alors que la syntaxe:
`mon_vecteur.push_back("ma_chaine_de_caractere");`
ajoute la *std::string* valant "*ma_chaine_de_caractere*" en fin de vecteur tout en redimensionnant celui-ci automatiquement.

➔ Affichez la taille de votre vecteur.
Affichez sa capacité (capacity). Quel est la différence avec sa taille ?
Affichez max_size, à quoi cela correspond ?

➔ Afficher le contenu du vecteur.

- Une première fois en utilisant la notation indexée de tableau en accédant à la valeur avec une syntaxe du type : *mon_vecteur[k]*.

- Une seconde fois en utilisant les itérateurs sur votre vecteur.
  Quel est l'avantage de cette syntaxe ?

➔ Réaliser un échange entre le contenu de la case d'indice 1 et le contenu de la case d'indice 3 de votre vecteur (vérifiez votre résultat en affichant le vecteur). Notez l'existence de std::swap.

➔ Insérez la valeur "a tous" après le premier élément dans votre vecteur. Vérifiez votre résultat (on pourra réaliser cette opération de manière standard).

➔ Changez le point d'interrogation par "???". Vérifiez votre résultat.

➔ Affichez le contenu du vecteur en séparant chaque chaîne par une virgule.

➔ Triez le vecteur en utilisant un algorithme de la STL (inclure l'en-tête algorithm). L'ordre de tri par défaut est celui de la comparaison sur des std::string. Affichez le résultat obtenu.

• Passage d'arguments dans une fonction

➔ Créez une fonction `affiche` qui affiche le contenu du vecteur passé en paramètre. Notez qu'ici, on passera le vecteur sous forme de référence constante car il n'a pas à être modifié, ni copié.

➔ Créez une fonction `concatene` qui concatène l'ensemble des éléments du vecteur dans une seule variable de type std::string. Chaque élément sera espacé d'un 'espace' dans la std::string.
Réfléchissez à la signature de votre fonction, sous quelle forme vous passez le paramètre d'entrée, sous quelle forme retournez vous le std::string?

➔ Créez une fonction `ajoute_virgule` qui ajoute une virgule derrière chaque mot contenu dans le std::vector. Cette fois, la variable de type std::vector passée en paramètre de la fonction doit être modifiée.

• Les listes

➔ Créez une liste de 8 entiers.
➔ Supprimez le troisième élément.
➔ Affichez à nouveau votre liste.

• Les map

Preambule.

➔ Construisez une std::map dont la clé est une chaîne de caractère, et la valeur est un nombre flottant.
Remplissez cette map avec un exemple de type "menu de restaurant" similaire à l'exemple du cours.

Exercice.
Soit une map qui, pour chaque année, associe une liste d'événements. Chaque événement étant stocké dans une std::string. La map sera ordonnée dans l'ordre chronologique, mais les événements associés à une date donnée seront ordonnées suivant leur ordre d'entrée dans la structure.

- ➔ Quel est le type C++ définissant cette map?
- ➔ Construisez une fonction `ajoute_evenement` qui ajoute un événement (date,intitulé) dans votre structure que vous passerez en paramètre.

- ➔ Faites une fonction qui affiche l'ensemble des dates et des événements associés.

L'entrée des événements dans la structure pourra suivre la forme suivante:
```
ajoute_evenement([nom_de_votre_structure],1994,"Creation de CPE");
```

On pourra par exemple considérer les événements suivants:

1994, Création de CPE Lyon
1953, Naissance de Richard Stallman
1953, Sortie de Peter Pan de Walt Disney
1994, Mort d'Ayrton Senna
1953, Naissance de Dorothé
1515, Bataille de Marignan
1953, Naissance de José Bové
1889, Inauguration tour Eiffel
1953, Naissance de Ségolène Royal
1889, Naissance de Edwin Hubble
1953, Naissance de John Malkovich

Aide:
- Il faut séparer les cas où l'on ajoute un événement dans une liste déjà existante, du cas où il faut ajouter une nouvelle liste dans la map.
- Il est possible d'ajouter une entrée dans une map sous différentes formes:

`M[clé]=valeur`
ou
`M.insert(std::make_pair(clé,valeur))`

Exercice 3. Les entrées/sorties (utilisation des flux).

Preambule.

- ➔ Écrivez une phrase et un nombre dans un fichier à l'aide du type `ofstream` (voir cours).
- ➔ Lisez cette même phrase et votre nombre à l'aide du type `ifstream`.

Format ppm.

- ➔ Écrivez le fichier suivant que vous nommerez `im.ppm`:

```
P3
#Ma premiere image
3 3 255
  0   0   0
255 255 255
100 100 100
255   0   0
  0 255   0
  0   0 255
255 255   0
255   0 255
  0 255 255
```

- ➔ Vérifiez que votre fichier est correctement écrit. (renommez le temporairement en .txt pour l'ouvrir plus aisément avec un éditeur de texte).

- ➔ Observez ce fichier avec un outil de visualisation d'images (zoomer sur l'image). On pourra par exemple ouvrir ce fichier avec Gimp.

Le format ppm est un format d'image non compressé simple à manipuler. Il permet d'écrire et de lire très facilement des images sans avoir à utiliser de bibliothèques externes de compression.

- ➔ Concluez sur l'organisation du format d'image ppm.

Notez que les espaces et l'ordonnancement des retours à la ligne est uniquement donné pour aider à la visualisation. On pourrait également retourner à la ligne après chaque valeur.

Aide:
- P3 est une sigle qui signifie que l'image est encodée en ASCII et il s'agit d'une image couleur sur les canaux RGB.
- La seconde ligne indique la dimension de l'image, suivie de l'intensité maximale sur chaque canal.
- Une ligne débutant par # est une ligne de commentaire.

Exercice 4. Une classe d'image (classe C++).

Soit les structures suivantes qui permettent de représenter une image couleur (chaque pixel contient une composante r,g, et b).

```
struct color
{
    //couleur r,g,b
    int r,g,b;

    //constructeurs
    color():r(0),g(0),b(0){}
    color(int r_arg,int g_arg,int b_arg):r(r_arg),g(g_arg),b(b_arg){}
};

struct image
{
    //taille
    int Nx,Ny;
    //donnees
    std::vector<color> data;
};
```

Une image est donc stockée comme un tableau 1D.
Le pixel de coordonnée (kx,ky) est accessible à l'indice kx+Nx ky.

➔ Assurez-vous de bien comprendre ces deux structures, ainsi que les constructeurs de color.

Afin de pouvoir afficher une couleur en utilisant la syntaxe
`color c(255,0,0);`
`std::cout<<c;`
On vous propose l'ajout de la fonction suivante:

```
std::ostream& operator<<(std::ostream& stream,const color& c)
{
    stream<<c.r<<" "<<c.g<<" "<<c.b;
    return stream;
}
```

Note. On définit ainsi l'opérateur << appliqué à un flux sortant en venant placer la composante r,g, et b dans le flux.

Au niveau de la classe image, il est possible d'accéder en écriture et en lecture à une couleur de l'image à l'aide de la syntaxe
`[nom_image].data[kx+Nx*ky];`

Cependant, il serait plus commode de pouvoir adresser directement une couleur à l'aide de la syntaxe:
`[nom_image](kx,ky)`

Pour cela, on peut ajouter les deux méthodes suivantes dans la struct image:

```
void certify_coord(int x,int y) const
{
    if(x<0 || x>=Nx || y<0 || y>=Ny)
    {
        std::cerr<<"Index "<<x<<","<<y<<" hors bornes."<<std::endl;
        exit(1);
    }
}

const color& operator()(int x,int y) const
{
    certify_coord(x,y);
    return data[x+Nx*y];
}
color& operator()(int x,int y)
{
    certify_coord(x,y);
    return data[x+Nx*y];
}
```

Note. Ce sont des définitions de l'opérateur() ayant deux arguments entiers.

➔ Ajoutez une méthode `resize` qui prend en argument une taille Nx et Ny et redimensionne la taille de la variable data et met à jour les paramètres d'une image.

➔ Assurez-vous que le code suivant compile et fonctionne:

```
image im;
im.resize(3,2);
im(0,0)=color(255,0,0);
im(1,0)=color(0,255,0);
im(2,0)=color(0,0,255);
im(0,1)=color(255,255,0);
im(1,1)=color(0,255,255);
im(2,1)=color(255,0,255);

std::cout<<im(0,1)<<std::endl;
```

➔ Créez une fonction `save_image` qui prend en paramètre un nom de fichier et une image, et l'enregistre dans le fichier désigné au format ppm.

➔ Assurez-vous que vous puissiez visualiser (avec Gimp par exemple) des images que vous créez vous même dans le code.

Aide:
- Il est possible d'extraire à tout moment le pointeur de type (char *) à partir d'un std::string en appelant la méthode `c_str()` appliquée sur la string.

Exercice 5. Classes, fichiers et documentation Doxygen.

➔ Téléchargez les classes d'images et de couleurs. Celles-ci remplacent celles que vous aviez codé directement dans la fonction main. Elles sont similaires à celles développées précédemment mais cette fois, on sépare la signature et le corps des fonctions.

Dans un code de grande envergure, il est important de bien séparer vos classes en différents fichiers et de séparer les en-têtes du corps des fonctions.

➔ Observez la syntaxe `[nom_classe]::` devant chaque fonction membre dans les fichiers .cpp. Cette syntaxe permet de différencier les fonctions membres (méthodes) des simples définitions de fonctions.

Notez également le style des commentaires. Ceux-ci sont compatibles avec l'outil Doxygen qui permet de réaliser une documentation standard d'un code.
Pour appeler Doxygen sur votre code, appelez depuis la ligne de commande les instructions suivantes:

```
$ doxygen -s -g config
$ doxygen config
$ firefox html/index.html
```

- Vous devez obtenir la création d'une documentation automatique extraite à partir du code.

Exercice 6. Chargement et édition d'une image (flux et vecteurs).

Préambule.

➔ Récupérez une image de votre choix. Avec Gimp, sauvez la au format ppm (mode ASCII). (Observez que la taille de l'image devient très importante de par l'absence de compression.)

➔ Observez le fichier ppm à l'aide d'un éditeur de texte.

Exercice.

➔ Codez une fonction `load_image` qui prend en argument un nom de fichier (sous forme de const std::string&) et retourne une struct image contenant les données de l'image.

Si vous n'êtes pas à l'aise, vous pouvez coder la fonction dans le fichier main.cpp. Dans le cas où vous le souhaitez, vous pouvez coder cette fonction dans un fichier à part (ex. image_ppm.cpp/hpp), ou bien encore directement dans le fichier image.cpp/hpp.

La lecture sera réalisée à l'aide des flux C++. Essayez de faire en sorte que votre programme s'arrête en indiquant une erreur si le fichier lu n'est pas conforme à celui attendu (ex. Fichier de type autre que P3, dimensions incorrectes, etc).

Aide:
- Pensez à la fonction std::getline qui lit une ligne complète d'un fichier
- Dans la version la plus simple du programme, on pourra supposer qu'il y a toujours une ligne de commentaire en seconde position, et qu'il n'y en a pas ailleurs.
- Il est possible de vérifier qu'une std::string soit égale à une certaine valeur en utilisant l'opérateur ==.
Par exemple v=="Bonjour", retourne vrai si la variable v contient "Bonjour", et faux sinon.

➔ Une fois que votre chargeur d'image fonctionne, réaliser les opérations suivant:

1. Charger une image
2. Illuminez spécifiquement tous les pixels de couleur rouge. (donnez leur une autre couleur).
3. Sauvez l'image du résultat dans un autre fichier.

Exemple de traitement spécifique sur les pixels rouges.

Exercice 7. Tracé de courbe paramétrique (utilisation des std::list).

Préambule.

➔ Construire une classe vec2 contenant un flottant x, et un flottant y.

➔ Définissez la méthode `to_string` telle que son en-tête soit le suivant:

```
std::string to_string(const vec2& v,const std::string& separator=" ");
```

Et le corps de la fonction soit:

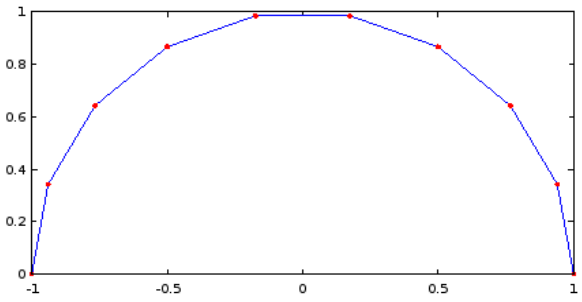
```
std::string to_string(const vec2& v,const std::string& separator)
{
    std::stringstream stream;
    stream<<v.x<<separator<<v.y;

    return stream.str();
}
```

L'appel à la fonction `to_string` sur la classe `vec2` permet d'exporter une `std::string` telle que le séparateur entre la coordonnée x et coordonnée y soit paramétrable.
L'utilisation de separator=" " dans l'en-tête de la fonction permet de passer des paramètres par défaut. Il n'est ainsi pas obligatoire de spécifier le séparateur, on pourra appeler ainsi la méthode `to_string` simplement par `to_string(vec2(x,y));` si on souhaite le séparateur par défaut.

Exercice.

➔ Construisez une liste de `vec2`.

La liste doit contenir 10 `vec2` et former un demi cercle tel qu'illustré à la figure suivante.



➔ Utilisez la fonction `export_matlab` fournie qui permet d'exporter les positions vers un format compatible avec Matlab.

Observez qu'il s'agit d'une fonction générique. Elle présuppose que la classe DATA peut être itérée avec des itérateurs, et que l'on puisse appeler la fonction `to_string()` sur les éléments contenus

dans le conteneur.
Notez également que tout le corps de la fonction est contenu dans le fichier d'en tête. En effet, contrairement aux fonctions et classes standards, les templates doivent être écrits directement dans les fichiers d'en-tête. Ils ne peuvent pas bénéficier de la compilation séparée.

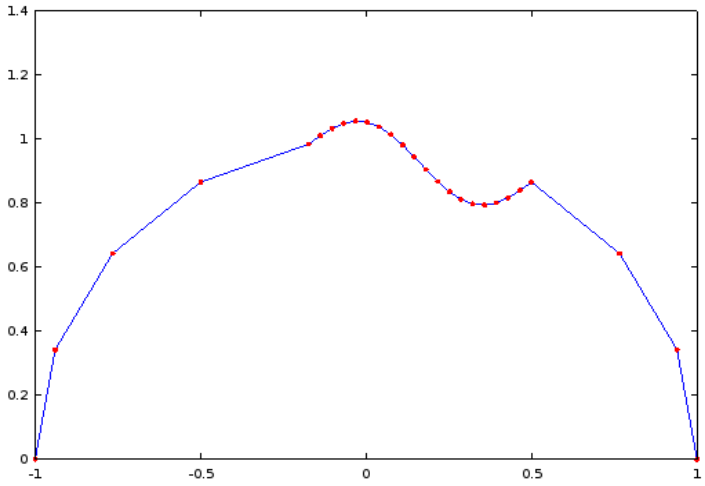
- ➔ Observez le fichier obtenu. On supposera que votre fichier se nomme data.m
- ➔ Lancez Matlab ou Octave, et tapez la commande suivante:

```
data;
plot(DATA(1,:),DATA(2,:),'.');
(ou plot(DATA(1,:),DATA(2,:),'-'); )
axis equal;
```

Vérifiez que cette commande permette bien d'afficher votre demi cercle.

- ➔ Ajouter désormais une courbe intermédiaire localement suivant la forme d'une oscillation sinusoïdale entre l'échantillon 4, et l'échantillon 6 (voir figure suivante).

Cette courbe pourra être plus finement échantillonnée.
Faites en sorte de ne pas avoir de sommets dupliqués.



Notez l'intérêt des listes chaînées qui permet d'ajouter localement des détails locaux sans avoir à modifier le reste de la courbe.
Ici, le nombre de sommets est limité, mais on pourrait supposer un ensemble de données beaucoup plus volumineux que l'on ne souhaiterait pas dupliquer.

Aide:

- Le morceau de courbe ajouté est une sinusoïde suivant l'axe des y. Les coordonnées x varient linéairement entre celles des échantillons 4 et 6. De même, la hauteur en y est linéairement adaptée pour s'adapter à ces mêmes échantillons.

On rappelle que l'on peut interpoler linéairement entre deux valeurs x1 et x2 suivant la formule x(t)=(1-t) x1+t x2 = x1+t(x2-x1),
avec t un paramètre d'interpolation variant entre 0 et 1.
- Il sera nécessaire de supprimer les 3 échantillons (4,5,6) afin que la courbe ne "revienne" pas sur ceux-ci et qu'il n'y ait pas de duplication de sommets avec le 6ème échantillon.

Exercice 8. Histogramme des mots d'un texte (utilisation des std::map)

Choisissez un fichier contenant un texte suffisamment long.

➔ Lisez votre texte mot à mot et créez un histogramme des mots que vous rencontrez.
On utilisera une `std::map` dont la clé sera le mot lu, et la valeur correspondra au nombre d'occurrences rencontrées.

Améliorations possibles:

- Afin d'avoir une mesure plus précise, on tentera d'éliminer les caractères de ponctuations (points, virgules, etc), et de ne pas prendre en compte l'effet des majuscules/minuscules.

- Faites en sorte d'afficher le résultat final dans l'ordre du nombre croissant d'occurrences (du plus faible nombre d'occurrence au plus grand).

- Réalisez le même histogramme en utilisant cette fois un `std::vector` comme conteneur. A chaque ajout, il est alors nécessaire de parcourir le vector afin d'identifier si un mot y est déjà référencé.
  Comparez les temps d'exécutions entre l'approche à base de `std::map` et l'approche à base de `std::vector`.

  Note. On pourra utiliser la syntaxe suivante pour connaitre le temps d'execution.

```
#include <boost/timer.hpp> //use of boost library
...
boost::timer t; //timer class
t.restart(); //set-up the timer to 0
...
t.elapsed(); //elapsed time
```

Exercice 9. Polynômes creux (classes et std::map)

Un polynôme creux est un polynôme possédant beaucoup de coefficients valant 0. Il n'est alors pas intéressant de stocker les nombreux coefficients inutiles.

Par exemple:
$P(x)=1,2\ x^{624}+1,1\ x^{248}+0,6\ x^{56}+1$
est un polynôme creux. 4 coefficients suffisent à définir ce polynôme.

On souhaite éviter de stocker les valeurs inutiles tout en gardant une bonne efficacité de recherche d'un coefficient donné du polynôme. On utilise pour cela une `std::map`.
La clé de la map sera l'exposant, et la valeur sera le coefficient.
De cette manière, il est possible d'ajouter/supprimer efficacement tout coefficient, et dans le même temps, d'avoir une évaluation rapide pour la valeur d'un exposant particulier.

➔ Implémentez cette classe et faites en sorte de pouvoir évaluer la valeur d'un polynôme pour un x donné, ainsi que additionner, soustraire et multiplier vos polynômes entre eux.