Introduction au C++

Ce TP propose différents exercices permettant de s'habituer à la programmation C++. La durée totale encadrée prévue sur ce TP est de 8h.

Contents

1 Premier programme

1.1 Premiers pas

▷ À l'aide d'un éditeur de texte/code simple tel que Kate, copiez les lignes suivante dans un fichier `main.cpp` (ou `main.cxx`, `main.cc`, etc. suivant vos conventions) que vous devez créer.

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

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

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

  return 0;
}
```

▷ Dans le répertoire de vore fichier source, exécutez la commande suivante (en ligne de commande) pour compiler le code:

```
g++ main.cpp -g -Wall -Wextra -std=c++11 -o exercice1
```

▷ Lancez ensuite l'éxecutable par la commande:

```
./exercice1
```

1.2 Classe

▷ Créez ensuite une struct `Conteneur` contenant un entier dénomé `valeur`. ▷ Faites en sorte que cette struct possède une *fonction membre* (ou *méthode*) appelée `affiche` qui affiche sur la ligne de commande le message *"je contient l'entier N"*, où N est remplacé par la valeur courante de l'entier. Notez que cette fonction membre ne renvoie rien.

Dans la fonction `main()`, la syntaxe d'appelle à cet affichage pourra être par exemple:

```
Conteneur c;
c.valeur=6;
c.affiche();
```

1.3 Boucles

▷ Toujours dans la fonction `main()`, testez le code suivant, et expliquez son fonctionnement.

```
Conteneur tableau[]={4,7,-8,6,2};

for(Conteneur c : tableau)
   c.affiche();
```

2 Compilation et choix d'éditeur

▷ Réalisez le tutorial sur les différentes approches de compilation d'un code C++ (utilisez les données fournies pour ce TP: répertoire `01_tutorial_compilation/`).

▷ Prenez connaissance de la fiche documentaire sur les éditeurs de code et IDE.

▷ Prenez enfin connaissance de la fiche documentaire sur `QtCreator`.

Notez les choses suivantes avant de passer à la suite:

- N'hésitez pas à vous reporter au tutorial de compilation lorsque vous devez compiler un projet en C++ et que vous avez oublié comment faire. Il est désormais de votre reponsabilité de savoir compiler un programme suivant les différentes approches. Les commandes ne seront plus rappelées à chaque fois.
- Si vous n'avez pas d'éditeur favori, n'hésitez pas à tester `QtCreator` et à observer la complétion automatique qu'il propose. Il est désormais de votre responsabilité de prendre en main de manière efficace un éditeur.

3 Classe et héritage

▷ Créez une struct (ou une classe) `personne` qui contient un nom et un prénom (sous forme de `std::string`). On laissera les paramètres (nom et prénom) publiques pour cet exercice.

▷ Assurez vous que votre code compile toujours.

▷ Créez deux constructeurs pour cette structure.

- Un premier constructeur sans paramètre initialisant le nom et prénom à "inconnu".
- Un second constructeur avec deux paramètres permettant de définir directement le nom et prénom.

▷ Assurez vous que votre code compile toujours.

▷ Creez une classe `etudiant` dérivée de celle-ci qui contient en plus une note (un entier). Redéfinissez les deux précédents constructeurs ainsi qu'un troisième prenant en argument le nom, le prénom, et la note. Par défaut, une note non définie sera initialisée à 0.

▷ Assurez-vous que votre code compile.

On considère désormais le code suivant:

```
void denomer(personne p)
{
  std::cout<<p.nom<<" "<<p.prenom<<std::endl;
}
int main()
{
  etudiant A("Franck","Ribery",2);
  etudiant B("Einstein","Albert",18);
  personne C("Huster","Francis");

  std::list<personne> L;
  L.push_back(A);
  L.push_front(B);
  L.push_front(C);

  for(personne p : L)
  denomer(p);

  return 0;
}
```

▷ Assurez vous que ce code compile (il faut ajouter `#include <list>` afin de pouvoir utiliser une `std::list` qui est une liste doublement chainée). Expliquez le résultat obtenu.

4 Introduction à la STL

4.1 Généralités

La STL est la bibliothèque standard du C++ qui implémente de nombreux types de données de manière efficace et générique. Les éléments (conteneurs et algorithmes) remplacent avantageusement de nombreuses implémentation 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 (tableaux contigus en mémoire, redimensionnables).
- `std::list` :pour les listes de la STL (listes doublement chaînées).
- `std::stack` :pour les piles de la STL.
- `std::queue` :pour les files de la STL.
- `std::set` :pour les ensembles ordonnées de la STL.
- `std::map` :pour les ensembles ordonnées par clés de la STL.

La documentation de la bibliothèque standard est disponible à cette adresse:

http://www.cplusplus.com/reference.

ou encore, une version alternative à cette autre:

http://en.cppreference.com.

On y trouvera en particulier la description précise des différents conteneurs de la STL (méthodes, complexités, exemples d'utilisations, etc). Mais également des informations sur des algorithmes, les string ou encore sur les flux d'entrées/sorties.

La STL est une bibliothèque contenant de nombreuses classes et fonctions, et l'utilisation de celle-ci n'est pas toujours triviale. Arriver à lire, analyser et comprendre la documentation de la STL est donc essentiel. Prenez soin de bien vous reporter à cette documentation dès que vous chercher des informations sur un élément de la bibliotheque standard.
Il est très important que vous arriviez à devenir autonome par rapport à cette documentation, et que vous soyez capable d'y trouver et comprendre les informations que vous y recherchez. N'hésitez pas à contacter un enseignant si vous n'arrivez pas à comprendre ou à vous servir de cette documentation.

4.2 Les vecteurs de la STL

4.2.1 Fonctionnement générale

Les vecteurs (`std::vector`) se comportent comme des tableaux dynamiques qui connaissent leurs propres tailles et sont capables de se redimensionner tout seul lorsqu'on leur ajoute des

éléments.
Un vecteur se défini de cette manière en indiquant le type contenu dans celui-ci entre chevrons[1] <>:

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

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

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

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

  return 0;
}
```

Remarques.

- Pour avoir accès à la définition d'un `std::vector`, il est nécessaire d'écrire `#include <vector>`. L'approche est similaire pour tous les autres conteneurs (ex. `#include <map>` pour une `std::map`, etc).
- Si vous ne souhaitez pas avoir à ré-écrire `std::` devant `vector` à chaque fois, il est possible d'écrire la ligne suivante après les `#include`

  ```
  using std::vector;
  ```

  Vous pouvez alors déclarer directement vos vecteurs sous la forme suivante:

  ```
  vector<int> mon_vecteur;
  vector<double> mon_vecteur2;
  vector<int*> mon_vecteur3;
  ```

  Notez qu'il est également possible de faire de même avec toutes les classes et variables de la STL tel que `using std::cout;`,`using std::endl;`
- A contrario, n'utilisez jamais la commande `using namespace std;`. Cette commande permet d'inclure l'intégralité du *namespace* de la STL dans le *namespace* global ce qui peut engendrer des conflits de noms imprévisibles. Cette commande[2] est considéré comme une mauvaise pratique.

Certaines fonctionnalités de bases d'un `std::vector` sont illustrées dans l'exemple suivant:

[1] On parle de template.
[2] bien que répandue dans de nombreux exemples sur internet

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

//inclusion de la bibliotheque de flux d'affichage (pour std::cout)
#include <iostream>

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

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

  //affectation de valeurs (similaire a un tableau standard)
  mon_vecteur[0]=1;
  mon_vecteur[1]=-2;
  mon_vecteur[2]=5;

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

  //parcours du vecteur (version indicee)
  for(int k=0;k<N;++k)
    std::cout<<mon_vecteur[k]<<" ";
  std::cout<<std::endl;

  //parcours du vecteur (version "range-based loop" (c++11))
  for(int valeur : mon_vecteur)
    std::cout<<valeur<<" ";
  std::cout<<std::endl;

  //parcours du vecteur (version iterateur courte (c++11))
  for(auto it=mon_vecteur.begin(),it_end=mon_vecteur.end();
           it!=it_end;++it)
    std::cout<<*it<<" ";
  std::cout<<std::endl;

  //parcours du vecteur (version iterateur longue)
  std::vector<int>::iterator it=mon_vecteur.begin(),
                             it_end=mon_vecteur.end();
  for(;it!=it_end;++it)
    std::cout<<*it<<" ";
  std::cout<<std::endl;

  //ajout en fin de vecteur avec redimentionnement automatique
  mon_vecteur.push_back(8);
  mon_vecteur.push_back(-1);

  //verification de la nouvelle taille
```

```
  std::cout<<mon_vecteur.size()<<std::endl;//affiche 5

  //on peut copier des vecteurs termes a termes (version courte (c++11))
  auto autre_vecteur=mon_vecteur;

  //on peut copier des vecteurs termes a termes (version longue)
  std::vector<int> autre_vecteur2; //1-creation de la classe
  autre_vecteur2=mon_vecteur;      //2-copie des elements

  //lorsque l'on quitte la fonction les vecteurs liberent automatiquement
  //  la memoire qu'ils ont alloues.
  return 0;
}
```

Note En pratique, on utilisera `std::vector<Type>` pour remplacer avantageusement une déclaraction sous forme de tableau dynamique de la forme `Type* var=new Type[N];`

4.2.2 Vecteur de string

Dans cet exercice, nous allons réaliser un programme qui permet de stocker des `std::string` et manipuler ceux-ci de manière simple. On rappel que `std::string` est une classe de la STL permettant de manipuler une chaine de caractère en remplacant avantageusement l'utilisation du type char*.

▷ Déclarez un vecteur de nom `mon_vecteur` stockant des `std::string`.
▷ Ajoutez cinq `std::string` dans ce vecteur. Ces `std::string` contiendront 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` contenant *"ma chaine de caractere"*. Dans ce cas, il faut que le vecteur ait une taille d'au moins k+1.

À l'inverse, la syntaxe suivante:

```
mon_vecteur.push_back("ma_chaine_de_caractere");
```

ajoute la `std::string` contenant *"ma chaine de caractere"* en fin de vecteur tout en redimensionnant celui-ci automatiquement.

▷ Affichez la taille de votre vecteur.

▷ Affichez sa capacité (`capacity`). Quelle est la différence avec sa taille ?

▷ Affichez le contenu du vecteur sur une même ligne (chaque mot étant séparé par un espace) de la manière suivante:

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

  - Une seconde fois en utilisant une boucle de type `for(valeur : conteneur)` sur votre vecteur. Quel est l'avantage de cette syntaxe ?
  - Une troisième fois en utilisant explicitement les itérateurs sur votre vecteur. Quel est l'avantage de cette syntaxe ?

▷ Réalisez un échange entre le contenu de la case d'indice 1 avec le contenu de la case d'indice 3 de votre vecteur (vérifiez votre résultat en affichant le vecteur modifié). Notez l'existence de `std::swap`.

▷ Insérez la valeur *"a tous"* après le premier élément dans votre vecteur. Vérifiez votre résultat.

▷ Changez le point d'interrogation final par *"???"*. Vérifiez votre résultat.

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

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

4.3 Passage d'arguments dans une fonction

▷ Créez une fonction `affiche` qui affiche le contenu du vecteur passé en paramètre. Notez qu'ici, on prendra soin de passer le vecteur sous forme de référence constante car il n'a pas à être modifié, ni copié (`std::vector<std::string> const&`).

▷ 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 passez vous le paramètre d'entré, sous quelle forme retournez vous la `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 modifié.

4.4 Les listes

▷ Créez une liste de huit entiers.

▷ Supprimez le troisième élément.

▷ Affichez à nouveau votre liste.

4.5 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 `std::map` qui, pour chaque année, associe une liste d'événements (sous la forme d'une `std::list`). 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.
▷ Faite une fonction qui affiche l'ensemble des dates et des événements associés.

L'ajout 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 est possible d'ajouter une entrée dans une map sous différente forme: `M[cle]=valeur` ou `M.insert(std::make_pair(cle,valeur))`

5 Entrées/sorties (utilisation des flux)

Preambule
▷ Écrivez une phrase et un nombre dans un fichier à l'aide du type `std::ofstream` (voir cours et référence).
▷ Lisez cette même phrase et votre nombre à l'aide du type `std::ifstream` (référence).

Format ppm

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

```
P3
#Ma premiere image
4 3
255
  0   0   0
255 255 255
100 100 100
100 100 180
255   0   0
  0 255   0
  0   0 255
100 180 100
255 255   0
255   0 255
  0 255 255
180 100 100
```

- ▷ 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.
  Notez que l'image doit apparaître avec des pixels clairement délimités, si l'image apparaît floue, il faut désacter l'interpolation des données dans votre outil de visualisation.

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.
La troisième ligne indique l'intensité maximal sur chaque canal.
Une ligne débutant par # est une ligne de commentaire.

6 Classe d'image en 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 (k_x, k_y) est accessible à l'indice $k_x + N_x \times k_y$.

▷ Assurez vous de bien comprendre ces deux structures ainsi que le constructeur 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 (x étant l'indice de ligne variant de haut en bas de l'image, y étant l'indice de colonne variant de gauche à droite de l'image).

```
[nom_image].data[ky+Ny*kx];
```

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

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

▷ Ajoutez une méthode `resize` à la classe `image` qui prend en argument une taille Nx et Ny et redimensionne la taille de la variable data et met à jour les paramètre 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`.

▷ Créez une image ayant un dégradé linéaire du noir au blanc de haut en bas. Assurez-vous que vous puissiez visualiser (avec `Gimp` par exemple) des images que vous créez vous même dans le code.

▷ Créez une autre image formée d'un cercle rouge sur un fond dégradé du rouge vers le bleu de gauche à droite.

7 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, les signatures et les implémentations des fonctions sont séparées.

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 compatible 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.

8 Chargement et édition d'une image (flux et vecteurs)

Preambule

▷ 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 part 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 `std::string const&`) 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"`, renvoie vrai si la variable v contient *"Bonjour"*, et faux sinon.

Figure 1: Exemple de traitement spécifique sur les pixels rouges.

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

1. Charger une image
2. Illuminez spécifiquement tous les pixels de couleurs rouges (donnez leur une autre couleur, voir par exemple Figure 1).
3. Sauvez l'image du résultat dans un autre fichier.

9 Tracé de courbes paramétriques et listes

Préambule.

▷ Construire une classe `vec2` contenant un `float` x, et un `float` y.

▷ Définissez la méthode `to_string` tel que son en-tête soit la suivante:

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

Et le corps de la fonction soit:

```
std::string to_string(vec2 const& v,std::string const& 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` tel 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 dix `vec2` et former un demi cercle tel qu'illustré à la Figure 2.

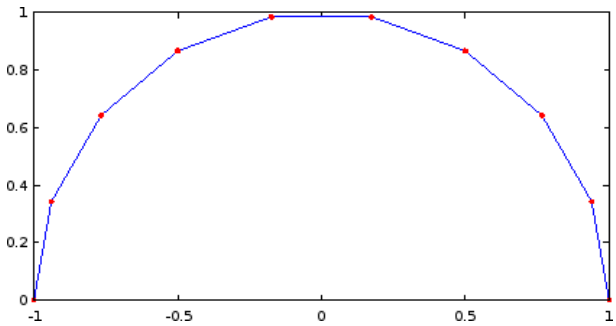

Figure 2: Demi-cercle formé par dix sommets discrets.

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

Observez qu'il s'agit d'une fonction générique. Elle présuppose que la classe `DATA` peut être itéré 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 fichier d'en-tête. Il 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 3). Cette courbe pourra être plus finement échantillonnée. Faite en sorte de ne pas avoir de sommets dupliqués.

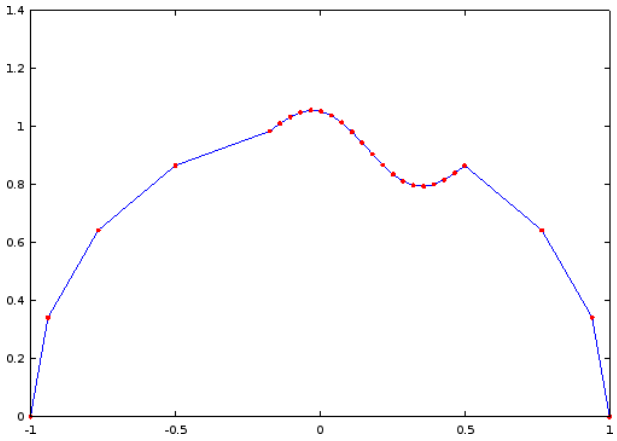

Figure 3: Ajout d'une partie intérmédiaire sur la courbe.

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 sommet est limité, mais on pourrait supposer un ensemble de donnée beaucoup plus volumineux que l'on ne souhaiterait pas dupliquer.

Aide.

- Le morceau de courbe ajouté est une sinusoïde suivant l'axe de y. Les coordonnées x varient linéairement entre celles des échantillons 4 et 6. De même, la hauteur en y est linéairement adapté pour s'adapter à ces même 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 *reviennent* pas sur ceux-ci et qu'il n'y ait pas de duplication de sommets avec le 6ème échantillon.

10 Histogramme des mots d'un texte et 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'occurrence rencontrée.

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 sort d'afficher le résultat final dans l'ordre du nombre décroissant 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
```

11 Polynomes creux (classes et 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. Quatres coefficients suffisent à définir ce polynôme.

On souhaite éviter de stocker les valeurs inutiles tout en gardant une bonne efficacité d'évaluation d'un coefficient donnée. 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.