TP 2 - Utilisation du conteneur `std::vector`

Objectifs :

- Utilisation de la classe **`vector`** de la bibliothèque standard
- Constructeurs / destructeurs
- Tirage aléatoire

1 Création d'une classe modélisant un point en 2D

1. En utilisant l'explorateur de fichiers, créer un dossier **`T:\Info3\TP2`**. Lancer votre IDE, et créer un nouveau projet de type "Console" avec le nom **`tp2`**.

2. Créer deux nouveaux fichiers source, et les nommer **`point.h`** et **`point.cpp`**.

3. Dans le fichier **`point.h`**, donner la déclaration d'une classe **`Point`**, composée de :
   - deux attributs privés x et y, de type **`double`**;
   - deux constructeurs, l'un sans arguments, initialisant le point à (0,0), l'autre avec deux arguments permettant de donner une coordonnée au point;
   - une méthode **`void Print()`**, affichant les coordonnées du point dans le flot **`cout`**, suivi d'un saut de ligne.

   Le fichier **`point.cpp`** devra ressembler à ceci :

```
// point.cpp
#include "point.h"
#include <iostream>
using namespace std;

Point::Point()
{
    x = y = 0;
}
Point::Point( double x0, double y0 )
{
    x = x0; y = y0;
}
void Point::Print()
{
    std::cout << "(" << x << ","<< y << ")\n";
}
```

4. Vérifier le fonctionnement avec, par exemple, le programme suivant :

```
// main.cpp
#include "point.h"
int main()
{
    Point pt1;
    Point pt2( 4, 5);
    pt1.Print();
    pt2.Print();
}
```

5. Ajouter à la classe **`Point`** une méthode **`double Distance(Point pt)`**. Elle soit renvoyer la distance entre le point courant auquel la méthode est appliquée (x,y) et le point transmis en argument (pt.x, pt.y). La formule à utiliser est la formule de Pythagore, qui est fournie dans la bibliothèque standard avec la signature suivante [1] :

```
double hypot( double a, double b );
```

   Cette fonction renvoie l'hypothénuse du triangle rectangle de dimensions (a,b). Par exemple : **`h = hypot( 1, 1 )`** renverra 1,414 dans h.

6. Vérifiez votre méthode en ajoutant la ligne suivante dans main() :
   `cout << "distance 1-2=" << pt1.Distance( pt2 ) << endl;`
   Cette méthode devra évidemment être bijective, c'est à dire qu'on doit avoir le même résultat avec :
   `cout << "distance 2-1=" << pt2.Distance( pt1 ) << endl;`

2 Création d'une classe contenant des points

Vous allez créer une classe permettant de regrouper une liste de points en utilisant le conteneur **`vector`**, fourni par la bibliothèque standard du C++. Ce

1. il faudra ajouter en tête du fichier où cette fonction est utilisée la ligne **`#include <cmath>`**, ainsi que **`using namespace std;`**

conteneur simplifie la gestion d'un ensemble d'élements par rapport aux tableaux hérités du C. La gestion mémoire est automatisée, on a plus besoin de spécifier la taille à la création, et on peut ajouter des éléments sans se préoccuper de la gestion de la mémoire.

Par rapport à la simple utilisation d'un tel conteneur, le fait de l'encapsuler dans une classe permet d'y associer des méthodes en facilitant l'usage.

1. Créer deux nouveaux fichiers, qui vous nommerez **`liste_points.h`** et **`liste_points.cpp`** (pensez à inclure un commentaire en haut de chaque fichier pour l'identifier).

2. Dans le fichier **`liste_points.h`**, déclarez une classe **`ListePoints`**, qui comprendra comme seul attribut l'élement suivant :
`std::vector<Point> vp;` (Il faudra ajouter en haut de ce fichier l'inclusion du fichier **`vector`**)

3. Déclarez (en "public") les 4 méthodes suivantes :
   - **`int GetNb();`** renvoie le nombre de points enregistrés dans la liste ;
   - **`void Print();`** pour l'affichage de la liste des points ;
   - **`void Add( Point pt );`** pour ajouter un point **`pt`** dans la liste ;
   - **`Point Get(int i);`** renvoie le point n° "i".

   Utilisation du vector
   - Pour connaître le nombre d'éléments contenus, on utilise la méthode **`size()`** de la classe **`vector`**.
   - Pour l'accès aux éléments du tableau, on utilise pour obtenir l'élément n° "i" la méthode **`at()`**, par example : **`vp.at(i)`**.
     Attention, pas de mécanisme de protection intégré : si on essaye d'accéder à un élément inexistant, on aura une erreur à l'exécution. Il faut donc un mécanisme de protection, qui **vérifie** que l'élément démandé existe (via la méthode **`GetNb()`**).
   - Pour ajouter un élément au tableau, on utilise la méthode **`push_back()`** de la classe **`vector`**. Par exemple, avec un vecteur d'entiers **`vint`**, l'expression **`vint.push_back(3);`** ajoute un nouvel élément dans le conteneur et lui donne la valeur "3".

4. Dans le fichier **`liste_points.cpp`**, écrivez la définition des 4 méthodes. Quelques indications :
   - La méthode **`Print()`** devra d'abord afficher [2] :
     `La liste contient XX points`
     Elle devra ensuite parcourir le tableau (boucle "for"), et afficher chaque point, en lui appliquant sa méthode d'affichage : **`vp.at(i).Print();`**
   - La méthode **`Get()`** devra vérifier que l'indice demandé existe, et si oui, renvoyer cet élement (via un return). A défaut, on affiche un message d'erreur, et on quitte (appel de la fonction **`exit(1)`**, précédé d'un message d'erreur pour l'utilisateur).

5. Tester le programme avec les lignes suivantes dans **`main()`** :

```
int main()
{
     ListePoints listp;
     Point pt1;
     Point pt2( 4, 5 );
     listp.Add( pt1 );
     listp.Add( pt2 );
     listp.Print();
}
```

6. Vérifier la cohérence de la classe en essayant d'enlever un point qui n'existe pas :

```
int main()
{
     ListePoints listp;
     Point pt( 4, 5 );
     for( int i=0; i<15; i++ )
          listp.Add( pt );
     cout << "Nb Points=" << listp.GetNb() << endl;
     Point pt2 = listp.Get( 15 );
}
```

Ce programme devra afficher le message d'erreur prévu, étant donné qu'il essaie d'accéder au 16ième point d'une liste qui en compte 15.

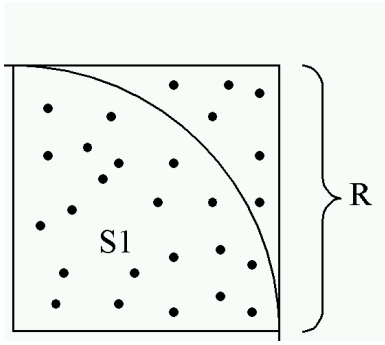
2. Avec XX le nombre réel de points...

3 Utilisation de ces classes pour le calcul de π

Vous aller utiliser les classes créées pour un programme qui fera un calcul de la valeur de π par une méthode de Monte Carlo (méthode statistique).

3.1 Principe du calcul

La surface d'un cercle est donnée par $S = \pi R^2$



Si on ne considère qu'un quart de cercle, la surface est $S1 = \pi R^2/4$

En plaçant un grand nombre de points n aléatoirement dans un carré de coté R et de surface R^2, il y aura un nombre n_1 de points qui vont "tomber" a l'intérieur du cercle ($n_1 < n$). Quand n va tendre vers l'infini, les points vont remplir toute la surface, et on pourra dire alors que le rapport des points dans le cercle (n_1) sur le nombre total de points (n) sera égal au rapport de la surface du quart de cercle sur la surface du carré (R^2). Soit :

$$\frac{n_1}{n} = \frac{S1}{R^2}$$

On peut en déduire : $S1 = n_1 \cdot R^2/n$

D'ou l'expression : $\pi = 4 * n_1/n$, pour $n \to \infty$

3.2 Implémentation informatique

Ecrire un programme (fonction **`main()`**) en trois parties successives :

— Etape 1 : générer une liste de points contenant 1000 points de coordonnées aléatoires.

Pour le tirage aléatoire, la bibliothèque standard fournit la fonction **`rand()`** (déclarée dans le fichier d'en-tête **`cstdlib`**) qui renvoie une valeur entière aléatoire comprise entre 0 et **`RAND_MAX`**, constante symbolique qui est définie dans **`cstdlib`** (ne **pas** la re-définir). Cette valeur correspondra donc au rayon du cercle R, et on pourra déterminer si un point donné est à l'intérieur du cercle, si la distance entre lui et le point à l'origine **`pt0`** (qu'il faudra créer) est inférieure à **`RAND_MAX`**.
Si la liste est désignée par une variable **`lp`**, alors, vous pourrez ajouter à la liste un point de coordonnées aléatoires avec les lignes suivantes :

```
Point pt( rand(), rand() );
lp.Add( pt );
```

On peut aussi éviter la création d'une variable intermédiaire, et utiliser le constructeur comme argument de la méthode Add() :

```
lp.Add( Point( rand(), rand() ) );
```

— Etape 2 : parcourir la liste produite (boucle **`for`**), et compter ceux qui sont à l'intérieur du cercle (n_1) en calculant pour chacun des points la distance au point à l'origine **`pt0`**, et en incrémentant le compteur si celle-ci est inférieure à **`RAND_MAX`**.

— Etape 3 : Faire le calcul de π à partir de n et n_1, et l'afficher.

Une fois ceci fonctionnel, vous ferez les améliorations suivantes :

— Ajoutez au début les instructions permettant de choisir le nombre de points de la liste (utilisation des flots cin / cout).

— Englober le programme dans une structure **`do ... while();`** dont on ne sortira qu'en tapant 0 pour n. Attention à ne pas provoquer d'erreur d'exécution dans ce cas (ne pas faire le calcul de π !).

— Calculer l'erreur avec la valeur réelle ($\frac{v_{calcul} - v_{reelle}}{v_{reelle}}$) en prenant comme valeur réelle la valeur donnée par le système, disponible via la constante **`M_PI`**.

— Relever plusieurs valeurs d'erreurs pour n=1k, 10k, 100k, 1M, 10M, ... Conclure sur la performance de l'algorithme.