Q1)
Merci Mathieu et Alexis !

```
void Complete(std::vector<int> &a, std::vector<int>&b){
 while(a.size()!=b.size()){
   a.size()>b.size()? b.push_back(0) : a.push_back(0);
 }
}

int main(int argc, char *argv[]) {
 std::vector<int> a;
 std::vector<int> b;
 a.push_back(2);
 a.push_back(2);
 a.push_back(2);
 a.push_back(2);

 Complete(a,b);
 std::cout<<a.size()<<std::endl;
 std::cout<<b.size()<<std::endl;
```

print 4 et 4 donc ok

Q2)

Fonction qui prend en argument une variable de type B, qui retourne une référence de type A.
La fonction s'appelle f1, et est une fonction membre du namespace "Strange".

Q3)
Classe abstraite : Ne peut pas être instancié mais peut être une classe mère.
Spécification C++ : Fonction virtuelle pure (ex : virtual void f() = 0)

Q4)
Chemin d'exécution suivant : (nom de la classe nom de la fonction)
C1F1
C2F1
C1F1
C1F1
C1F1
C1F1
C1F2

C1F2
C3F2
C1F2
C1F2
C3F2
Flemme de justifier, vous pouvez comprendre je vous aime fort.

* QUESTION 5 :
* 1 : internalA sera un pointeur de A, car celui ci est de cardinalité 0 à 1, il peut donc être nul contrairement à une référence
Pas d'objet car création d'un bloc de mémoire de la taille d'un A infinie sinon.
*
* 2 :
* Il faut supprimer la variable internalA actuelle, donc il faut aussi redéfinir son destructeur, qui détruira l'attribut du même nom d'internalA
* Il faut de plus définir une méthode clone renvoyant une instance de a en fonction de son type
* Cela n'est pas nécessaire pour setInternal mais on devra aussi redefinir l'opérateur = et le constructeur par copy

* QUESTION 6 :
* A : Dans la classe B il faudrait rajouter un déréférencement comme ceci return (*pti+j);
* De plus, dans le main B n'est pas initialiser, hors il n'existe pas de constructeur par défaut, donc cela ne peut fonctionner
Si une classe mère n'a pas de constructeur par default, sa classe fille ne peut pas en avoir, comme en JAVA

*
* B : Il n'y a pas le constructeur par copy dans la Class A, il y'a a donc une fuite mémoire lors de l'attribution de valeur,
* Ex : A a = 6;
* a = 8; // Ici le 6 étant alloué dynamiquement il y a fuite mémoire si le opérateur = n'est pas redéfini

Le pointeur pti est delete mais n'est pas à nullptr, dans ce cas la le main s'arrête après 2 lignes donc n'est pas impacté, mais ça pourrait provoquer un comportement inattendu et une fuite de mémoire.

Laura : Le destructeur de A n'est pas virtuel, fuite de mémoire potentielle.
Alexis : Oui, mais la classe fille n'a pas de pointeur, donc, pas de soucis
Laura : yep, d'où le potentielle ^^
Alexis : (oups)

Q7

/**
* Pour a1 on sait qu'il a un constructeur avec un char ou un int en paramètre donc A(int i) ou A(char c)

* Il a aussi un un constructeur A(std::string a, un pointeur random)
* C1 a un constructeur par défaut
* **A a en plus un constructeur qui prend en param une classe C ou C hérite de A**
* C a redéfini son opérateur *; obligatoirement en fonction friend sinon l'opération n'a pas de sens puisque 2,4 ne peut pas être caster du type du paramètre caché.
*
**/

Q8 ça n'y sera pas cordialement la direction.