CONTENEURS :

- ## Ajouter un élement a un conteneur :

-Vérifier si l'élement est déjà dans la liste

```
void Projet::ajouterTache(const Tache& tache) {
   if (auto it = find(taches_.begin(), taches_.end(), tache); it == taches_.end())
      taches_.push_back(tache);
}
```

- ## Retirer un élement d'un conteneur (L'élement a retirer nous est fournie) :

Trouver l'élement dans le conteneur

Retirer tout les élements egaux a la cible :

```
void Projet::supprimerTache(const Tache& tache) {
   auto it = remove(taches_.begin(), taches_.end(), tache);
   taches_.erase(it, taches_.end());
}
```

Retirer seulement un :

```
void Projet::supprimerTache(const Tache& tache) {
   if (auto it = find(taches_.begin(), taches_.end(), tache); it != taches_.end())
      taches_.erase(it);
```

OU

```
Auto elem = find(taches_.begin(), taches_.end(), tache);
If(elem != taches_.end())
Taches_.erase(elem)
}
```

Erase prends en parametre un ITERATEUR pas l'objet lui meme

- ## Trier un conteneur a l'aide d'un critere :

```
void Projet::trierTaches(function<bool(const Tache&, const Tache&)> f) {
   sort(taches_.begin(), taches_.end(), f);
}

class StringLengthComparator {
public:
   bool operator()(const std::string& s1, const std::string& s2) const {
      return s1.length() > s2.length();
```

```
    }
};
 std::sort(strings.begin(), strings.end(), StringLengthComparator());
```

Trier une liste de string du plus gran nombre d'elements au plus petit

- ## STD::Transform

```
std::transform(v1.begin(), v1.end(), v2.begin(), v3.begin(),
            [](int a, int b) { return a + b; });
```

Permet d'appliquer des operations paralleles sur les elements de deux vecteurs et en mettant le resultats dans un troisieme vecteur

-Exemple :

```
int main() {
    std::vector<int> v1{1, 2, 3, 4, 5};
    std::vector<int> v2{10, 20, 30, 40, 50};
    std::vector<int> v3(v1.size());

    std::transform(v1.begin(), v1.end(), v2.begin(), v3.begin(),
                [](int x, int y) { return x + y; });
```

Output : 11 22 33 44 55

-still works if you change v2 to v1, then the elements will just be added to themselves

- ## Retirer des élements d'apres un critere :

-Erase and remove_if :

```
v.erase(std::remove_if(v.begin(), v.end(), IsEven()), v.end());
```

Fonctionement :

1-Remove_if prend tout les elements satisfaisaont le critere de retrait et les mets a la fin du conteneur et retourne un iterateur vers le premier des elements a retirer

2. Erase s'occupe donc de retirer les elements a retirer a partir de l'iterateur fournie par remove if

Warning :

```
  valeurs.erase(std::remove_if(valeurs.begin(),valeurs.end(),[min, max](double x){return !(min<=x && x<=max);}),valeurs.end());} //Bien vérifier la condition de la lambda
```

- ## STD::Map

- Everytime a map receives a new element, the key is compared to all existing keys in the map.
- The criteria of comparison between the keys can be specified
- To check if a key has a value : if (map.count(key) != 0)

- ## Copier les élements entre les conteneurs :

-std::Copy et std::inserter :

```
{std::vector<int> v {1, 2, 3, 4, 5};
   std::set<int> s;

   std::copy(v.begin(), v.end(), std::inserter(s, s.end()));
}
```

Fonctionement du code :

1. Copy prend en parametre un premier iterateur, dernier iterateur et un output iterator (std::inserter)
2. Std::inserter prend en parametre le conteneur ou on veut mettre les elements et un iterateur de fin.

- ## Copier les élements satisfaisant un critere (copy_if et lambda) :

```
   std::copy_if(valeurs.begin(), valeurs.end(), std::inserter(resultat, resultat.end()), [base](int x){return x%base ==0;});
```

- ## STD::count_if

Compteur, exemple : trouver le nombre de int pairs d'un vecteur :

```
 int count = std::count_if(v.begin(), v.end(), [](int x) { return x % 2 == 0; });
```

- ## STD::transform_reduce :

Permet des opérations comme le produit scalaire :

```
std::vector<int> v1 {1, 2, 3};
   std::vector<int> v2 {4, 5, 6};

   int dot_product = std::transform_reduce(v1.begin(), v1.end(), v2.begin(), 0,std::plus<>{}, std::multiplies<>{});
```

Fonctions:
Find(start, end, element)
Find_if(start,end, condition(lambda))
Remove(start, end, element) (met l'élément à la fin du conteneur)

CLASSES :

- ## Destructeur virtuel :

Permet d'éviter les fuites de mémoires avec les objets dérivés :

```
class Shape {
public:
   virtual ~Shape() {}
};
Shape* s = new Circle(3.0);
delete s;
```

-Aucune fuite de mémoire si le destructeur est virtuel
-Fait en sorte que le destructeur de la classe enfant est appelé avant celui de la classe mere evitant les fuites de mémoires
-Allows abstract classes to be compatible with unique pointers

- ## Héritage et construction :

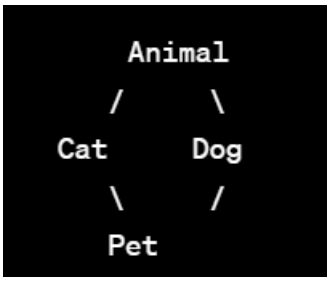
Construction de classe dérivé peut etre fait a l'aide du constructeur de la classe parent :

```
   EtudiantCyclesSup(int matricule, string p, Professeur* s) : Etudiant(matricule,
p){
   superviseur_ = s;
}
```

Use the parent constructor to set the common values between its children

- ## Probleme du diamant et Héritage virtuel :

Probleme du losange :



`Can be solved with virtual heritage :

```
class Animal {
public:
   Animal() {}
   virtual ~Animal() {}
   virtual void speak() {}
};

class Cat : public virtual Animal {
public:
   Cat() {}
   virtual ~Cat() {}
   void speak() override {
      cout << "Meow" << endl;
   }
};

class Dog : public virtual Animal {
public:
   Dog() {}
   virtual ~Dog() {}
   void speak() override {
      cout << "Woof" << endl;
   }
};

class Pet : public Cat, public Dog {
public:
   Pet() {}
   virtual ~Pet() {}
};
```

the inheritance from Animal for both Cat and Dog is declared as virtual. This means that when Pet inherits from both Cat and Dog, there is only one copy of Animal in the inheritance hierarchy.

MVC :

- **Controleur :**
- modifie le modèle suite à certains événements
- modifie la vue suite à certains événements
- vérifie la validité des actions de l'utilisateur
- **Vue :**
-

EXCEPTIONS :

Exemple :

```
struct ErreurN : public domain_error {
   using domain_error::domain_error;
};

struct ErreurP : public domain_error {
   using domain_error::domain_error;
};

struct ErreurDivZero : public domain_error {
   using domain_error::domain_error;
};

double calculerVariance(int n, double p) {
   if (n <= 0)
```

```
        throw ErreurN("n="s + to_string(n) + " n'est pas un
nombre naturel"s);
    if (p < 0 or p > 1)
        throw ErreurP("p="s + to_string(p) + " n'est pas dans
l'intervalle [0, 1]");
    return n * p * (1 - p);
}

double calculerAsymetrie(int n, double p) {
    double variance = calculerVariance(n, p);
    if (variance == 0)
        throw ErreurDivZero("Variance ne peut pas être
nulle.");
    double q = 1 - p;
    return (q - p) / sqrt(variance);
}

int main() {
    try {
        int n = 42;
        double p = 0.0;
        double asym = calculerAsymetrie(n, p);
        cout << "n=" << n << " p=" << p << " var=" << asym <<
endl;
    } catch (ErreurN& e) {
```

```
        cout << "Problème avec nombre d'expériences : " <<
e.what() << endl;
    } catch (ErreurP& e) {
        cout << "Problème avec probabilité : " << e.what() <<
endl;
    } catch (logic_error& e) {
        cout << "Erreur logique : " << e.what() << endl;
    }
}
```

Etapes d'execution :

1. CalculerAsymetrie est appelé
2. Elle appele CalculeVariance
3. Variance est egale a 0, ErreurDivZero est donc lancé
4. Le main catch l'excpetion avec le bloc logic_error
5. Le bloc de catch affiche un message disant qu'il y a eu une erreur et de quel type
6. Puisque l'excpetion a été attrapé et traité, le code continue son execution des prochains bloc de catch qui ne font rien et l'execution se termine

LAMBDA

Déclaration :

```
std::function<double(double x, double y)> addition() {
    return [](double x, double y) {return x + y; };
}
```

COMPLEXITÉ :

- "Changer la taille ajoute de la complexité"

```
vector<int> v;
int cnt = 0;
for (auto&& e1 : v) {
    cnt += count_if(
        v.begin(),
        v.end(),
        [&](auto e) { return f(e1, e); }
    );
    cout << cnt;
} O(n2) because of count_if
```

-

	vecteur	pile	file	liste
push_back	$O(1)$	$O(1)$	$O(1)$	$O(1)$
pop_back	$O(1)$	$O(1)$	-	$O(1)$
push_front	$O(N)$	-	-	$O(1)$
pop_front	$O(N)$	-	$O(1)$	$O(1)$
tab[i]	$O(1)$	-	-	$O(N)$
insert	$O(N)$	-	-	$O(1)$
erase	$O(N)$	-	-	$O(1)$

-

Big O Notation	Name	Example
O(1)	Constant runtime	Select an item with array index / object key
O(log n)	Logarithmic runtime	Binary search
O(n)	Linear runtime	For loops, Javascript map(), filter(), reduce()
O(n log n)	Linearithmic runtime	Sorting an array with merge sort
O(n^2)	Quadratic runtime	Sorting an array with bubble sort, 2 level nested loops
O(2^n)	Exponential runtime	Recursive calculation of Fibonacci numbers
O(n!)	Factorial runtime	Find all permutations of a given set / string

- The following are examples of computing times in algorithm analysis. To make the difference clearer, let's compare based on the execution time where n = 1000000 and time = 1msec:

Big-Oh	***Description***	***Algorithm***	***Running Time***	***Sample Code Implementation***
O(1)	Constant			return n * (n + 1) / 2
$O(\log_2 n)$	Logarithmic	Binary Search	19.93 microseconds	While *n* > 1 count ← count + 1 n ← n / 2
O(n)	Linear	Sequential Search	1.00 seconds	For i ← 1 to n sum ← sum + i
$O(n \log_2 n)$		Heapsort	19.93 seconds	
$O(n^2)$	Quadratic	Insertion Sort	11.57 days	For *i* ← 1 to *n* For *j* ← *i* to *n* sum ← sum + *j*
$O(n^3)$	Cubic	Floyd's Algorithm	317.10 centuries	
$O(2^n)$	Exponential		Eternity	

Operations on the O-Notation:

- **Rule for Sums**
  - Suppose that $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$.
  - Then, $t(n) = T_1(n) + T_2(n) = O(\max(f(n), g(n)))$.

Binary search = Binary search begins by comparing an element in the middle of the array with the target value. If the target value matches the element, its position in the array is returned. If the target value is less than the element, the search continues in the lower half of the array. If the target value is greater than the element, the search continues in the upper half of the array

Merge sort = Divide the unsorted list into n sublists, each containing one element (a list of one element is considered sorted). Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

Bubble sort = Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.

Permutation = all the ways to order a given set or string

COMPLEXITÉ :

- for loops :
- N'ajoute pas de complexité si la porté d'itération est constante :

```
unordered_map<int,int> v;
// ... des valeurs dans v
int n = v.size();
for (int i : range(42))
```

```
v[f(i)] = i;
```