Class Vector avec Template

Objectif : Le but de cet exercice est de comprendre le fonctionnement interne de la classe vector du C++ en implémentant une version simplifiée de cette classe.
Instructions :

- ➔ Créez une classe template nommée vector qui simule le fonctionnement de la classe vector prédéfinie du langage C++.

- ➔ La classe vector doit contenir les fonctionnalités de base suivantes :
  - ◆ **Initialisation** du vecteur avec une taille spécifiée.
  - ◆ **Accès aux éléments** du vecteur à l'aide de l'opérateur [].
  - ◆ **Ajout d'éléments à la fin** du vecteur avec la méthode push_back.
  - ◆ **Suppression de l'élément de fin** du vecteur avec la méthode pop_back.
  - ◆ **Redimensionnement du vecteur** avec la méthode resize.
  - ◆ **Affichage des éléments** du vecteur avec une méthode dédiée.

- ➔ Utilisez une approche de gestion dynamique de la mémoire pour allouer et libérer de l'espace pour le stockage des éléments du vecteur.

- ➔ Assurez-vous de fournir des constructeurs, un destructeur, et un opérateur d'assignation pour votre classe

- ➔ Testez votre implémentation en créant plusieurs instances de la classe vector, en ajoutant et en supprimant des éléments, et en affichant les éléments de chaque instance.

code source

```
#include <iostream>

using namespace std;

template<typename T>
class vector {
   int taille;
   T* elem;
   int space;

public:
   vector() : taille{0}, elem{nullptr}, space{0} {}

   vector(int s) : taille{s}, elem{new T[s]}, space{s} {
      for (int i = 0; i < taille; ++i) elem[i] = T{};
   }

   // Constructeur de copie
   vector(const vector& other) : taille{other.taille}, elem{new T[other.taille]},
space{other.space} {
      for (int i = 0; i < taille; ++i) elem[i] = other.elem[i];
   }

   T& operator[](int n) { return elem[n]; } // Accès aux éléments

   const T& operator[](int n) const { return elem[n]; } // Accès aux éléments (const)

   int size() const { return taille; } // Taille du vecteur

   int capacity() const { return space; } // Espace alloué

   void resize(int newsize) {
      reserve(newsize);
      for (int i = taille; i < newsize; ++i) elem[i] = T{};
      taille = newsize;
   }

   void push_back(const T& val) {
      if (space == 0)
         reserve(8);
      else if (taille == space)
         reserve(2 * space);
      elem[taille] = val;
      ++taille;
   }
```

```
    void pop_back() {
        if (taille > 0) {
            --taille;
        }
    }

    void reserve(int newalloc) {
        if (newalloc <= space) return;
        T* p = new T[newalloc];
        for (int i = 0; i < taille; ++i) p[i] = elem[i];
        delete[] elem;
        elem = p;
        space = newalloc;
    }

    void display() const {
        for (int i = 0; i < taille; ++i) {
            cout << elem[i] << " ";
        }
        cout << endl;
    }

    // Opérateur d'assignation
    vector& operator=(const vector& other) {
        if (this != &other) {
            delete[] elem;
            taille = other.taille;
            elem = new T[other.taille];
            for (int i = 0; i < taille; ++i) elem[i] = other.elem[i];
        }
        return *this;
    }

    ~vector() {
        delete[] elem;
    }
};
int main() {
    vector<int> tab(5);
    tab.push_back(10);
    tab.push_back(20);
    cout << "Size: " << tab.size() << endl;
    cout << "Capacity: " << tab.capacity() << endl;
    cout << "Elements: ";
    tab.display(); // Utilisation de la méthode display pour afficher les éléments
    return 0;
}
```