

TD08 – Exceptions

Exercice 1 Gestion d'un vecteur de taille dynamique

Soit la classe générique de gestion de vecteur définie par le code suivant :

```
#include <iostream>
#include <algorithm>
#include <vector>

template<typename T>
class CVecteur
{
    std::vector<T> m_vData;

public:
    typedef typename std::vector<T>::iterator
    iterator;
    typedef typename std::vector<T>::const_iterator
    const_iterator;

    CVecteur(void) {}
    void resize(size_t nNewSize) {
        m_vData.resize(nNewSize);
    }

    bool empty() const { return m_vData.empty(); }
    size_t size() const { return m_vData.size(); }

    iterator begin() { return m_vData.begin(); }
    const_iterator begin() const {
        return m_vData.begin();
    }
    iterator end() { return m_vData.end(); }
    const_iterator end() const {
        return m_vData.end();
    }

    T& operator[](size_t nPos) {
        return m_vData[nPos];
    }
    const T& operator[] (size_t nPos) const {
        return m_vData[nPos];
    }


    CVecteur operator + (const CVecteur& vect);
    CVecteur operator - (const CVecteur& vect);
};

template<typename T>
std::ostream& operator<<(std::ostream& out, const
CVecteur<T>& vect)
{
    if(vect.empty())
    {
        out<<"{}";
        return out;
    }


    CVecteur<T>::const_iterator it = vect.begin();
    out<<"{" <<*it++;
    for(;it!=vect.end(); it++)
        out<<" " <<*it;
    out<<"}";
    return out;
}

template<typename T>
CVecteur<T> CVecteur<T>::operator + (const
CVecteur& vect)
{
    CVecteur<T> res(*this);
    if(size() == vect.size())
    {
        iterator it = res.begin();
        const_iterator it2 = vect.begin();
        for(;it != res.end(); it++, it2++)
            *it += *it2;
    }
    return res;
}


template<typename T>
CVecteur<T> CVecteur<T>::operator - (const
CVecteur& vect)
{
    CVecteur<T> res(*this);
    if(size() == vect.size())
    {
        iterator it = res.begin();
        const_iterator it2 = vect.begin();
        for(;it != res.end(); it++, it2++)
            *it -= *it2;
    }
    return res;
}
```

 Dans ce code, déterminez les lignes où des situations exceptionnelles peuvent se produire.

 Pour chacune de ces situations, mettez en place le mécanisme des exceptions.

 On peut coder les situations exceptionnelles par le tableau suivant :

Code	Signification
0	Indice hors limite
1	Paramètre incorrect
2	Les tailles ne correspondent pas
3	?

 Ecrivez un programme de test permettant de valider le mécanisme pour chacune de ces situations



La bibliothèque STL fournit un certain nombre d'objets prévu pour gérer des exceptions. Ils sont déclarés dans l'entête `<stdexcept>` :

Classe	Description
domain_error Class	The class serves as the base class for all exceptions thrown to report a domain error.
invalid_argument Class	The class serves as the base class for all exceptions thrown to report an invalid argument.
length_error Class	The class serves as the base class for all exceptions thrown to report an attempt to generate an object too long to be specified.
logic_error Class	The class serves as the base class for all exceptions thrown to report errors presumably detectable before the program executes, such as violations of logical preconditions.
out_of_range Class	The class serves as the base class for all exceptions thrown to report an argument that is out of its valid range.
overflow_error Class	The class serves as the base class for all exceptions thrown to report an arithmetic overflow.
range_error Class	The class serves as the base class for all exceptions thrown to report a range error.
runtime_error Class	The class serves as the base class for all exceptions thrown to report errors presumably detectable only when the program executes.
underflow_error Class	The class serves as the base class for all exceptions thrown to report an arithmetic underflow.



Modifiez le code pour tirer profit des classes listées dans le tableau ci-dessus.