```
#ifndef CODEREVIEWTASK_MYVECTOR_HPP
#define CODEREVIEWTASK_MYVECTOR_HPP

#include <vector>
#include <string>
#include <algorithm>
#include <stdexcept>

/*
 * MyVector stores a collection of objects with their names.
 *
 * For each object T, MyVector stores T`s name as std::string.
 * Several objects can have similar name.
 * operator[](const std::string& name) should return the first object
 * with the given name.
 *
 * Your task is to find as many mistakes and drawbacks in this code
 * (according to the presentation) as you can.
 * Annotate these mistakes with comments.
 *
 * Once you have found all the mistakes, rewrite the code
 * so it would not change its original purpose
 * and it would contain no mistakes.
 * Try to make the code more efficient without premature optimization.
 *
 * You can change MyVector interface completely, but there are several rules:
 * 1) you should correctly and fully implement copy-on-write idiom.
 * 2) std::pair<const T&, const std::string&> operator[](int index) const must take constant time
at worst.
 * 3) const T& operator[](const std::string& name) const should be present.
 * 4) both operator[] should have non-const version.
 * 5) your implementation should provide all the member types of std::vector.
 * 6) your implementation should provide the following functions:
 *    1) begin(), cbegin(), end(), cend()
 *    2) empty(), size()
 *    3) reserve(), clear()
 */

template <typename T>
class MyVector : public std::vector<T>
{
public:
    MyVector()
    {
```

```
        m_ref_ptr = new size_t(1);
        m_names = new std::vector<std::string>();
    }

    MyVector(const MyVector& other)
        : std::vector<T>(other),
          m_ref_ptr(other.m_ref_ptr),
          m_names(other.m_names)
    {
        (*m_ref_ptr)++;
    }

    ~MyVector()
    {
        if (--*m_ref_ptr == 0)
        {
            delete m_ref_ptr;
            delete m_names;
        }
    }

    void push_back(const T& obj, const std::string& name)
    {
        copy_names();

        std::vector<T>::push_back(obj);
        m_names->push_back(name);
    }

    std::pair<const T&, const std::string&> operator[](int index) const
    {
        if (index >= std::vector<T>::size())
        {
            throw new std::out_of_range("Index is out of range");
        }

        return std::pair<const T&, const std::string&>(std::vector<T>::operator[](index),
(*m_names)[index]);
    }

    const T& operator[](const std::string& name) const
    {
        std::vector<std::string>::const_iterator iter = std::find(m_names->begin(), m_names->end(),
name);
```

```
        if (iter == m_names->end())
        {
            throw new std::invalid_argument(name + " is not found in the MyVector");
        }

        return std::vector<T>::operator[](iter - m_names->begin());
    }

private:
    void copy_names()
    {
        if (*m_ref_ptr == 1)
        {
            return;
        }

        size_t* temp_ref_ptr = new size_t(1);
        std::vector<std::string>* temp_names = new std::vector<std::string>(*m_names);

        (*m_ref_ptr)--;
        m_ref_ptr = temp_ref_ptr;
        m_names = temp_names;
    }

private:
    // Use copy-on-write idiom for efficiency (not a premature optimization)
    std::vector<std::string>* m_names;
    size_t* m_ref_ptr;
};


#endif //CODEREVIEWTASK_MYVECTOR_HPP
```