#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