```
//
//  FunctionalVector.h
//  CppPractice
//
//  Created by Stewart Bracken on 12/10/13.
//  Copyright (c) 2013 Stewart Bracken. All rights reserved.
//

#ifndef __CppPractice__FunctionalVector__
#define __CppPractice__FunctionalVector__

#include <iostream>
#include <vector>
#include <algorithm> // for copy
#include <iterator> // for ostream_iterator

//Vector wrapper that can 'slice' a vector to specific range offset.
// for example a vector {1,2,3}.slice(1,2) = {2,3} and indexing to 0 returns 2
// instead of 1.
template <class T>
class FunctionalVector : public std::vector<T> {
    size_t i_start, i_length, i_end;
public:
    FunctionalVector();

    FunctionalVector(const FunctionalVector<T>& rhs):
std::vector<T>(static_cast<std::vector<T> >(rhs)),
    i_start(rhs.i_start), i_end(rhs.i_end),
i_length(rhs.i_length){/*std::cout<<"fun_vec copy ctor"<<std::endl;*/}

    //Copy data constructor
    FunctionalVector(const std::vector<T>& other): std::vector<T>(other),
    i_start(0), i_end(other.size()), i_length(other.size()){/*std::cout<<"vec copy
ctor"<<std::endl;*/}

    FunctionalVector<T>& slice(size_t new_start, size_t new_length){
        i_start += new_start;
        i_length = new_length;
        i_end = i_start +i_length;
        return *this;
    }

    //Override vector's bracket operator to place modified indices.
    T& operator[](size_t idx){
        if(idx >= i_length && i_length > 0)
```



```
            idx %= i_length;
        return std::vector<T>::operator[](index_at(idx));
    }
    const T& operator[](size_t idx)const{
        if(idx >= i_length && i_length > 0)
            idx %= i_length;
        return std::vector<T>::operator[](index_at(idx));
    }

    void reset(){i_start=0; i_length=std::vector<T>::size(); i_end = i_length;}


    size_t index_at(size_t norm_idx)const{
        return i_start + norm_idx;
    }

    size_t size()const{
        return i_length;
    }
    friend std::ostream& operator<<(std::ostream& out, const FunctionalVector& me){
        copy(me.begin() + me.i_start, me.begin() + (me.i_end),
std::ostream_iterator<T>(out, ", "));
        return out;
    }
};

#endif /* defined(__CppPractice__FunctionalVector__) */
```