```cpp
#include <iostream>
#include <string>
#include <deque>
#if 1 //CREATE A REAL STL EXAMPLE
	#include <map>
	#include <stack>
	#include <vector>
	namespace ft = std;
#else
	#include <map.hpp>
	#include <stack.hpp>
	#include <vector.hpp>
#endif

#include <stdlib.h>

#define MAX_RAM 4294967296
#define BUFFER_SIZE 4096
struct Buffer
{
	int idx;
	char buff[BUFFER_SIZE];
};


#define COUNT (MAX_RAM / (int)sizeof(Buffer))

template<typename T>
class MutantStack : public ft::stack<T>
{
public:
	MutantStack() {}
	MutantStack(const MutantStack<T>& src) { *this = src; }
	MutantStack<T>& operator=(const MutantStack<T>& rhs)
	{
		this->c = rhs.c;
		return *this;
	}
	~MutantStack() {}

	typedef typename ft::stack<T>::container_type::iterator iterator;

	iterator begin() { return this->c.begin(); }
	iterator end() { return this->c.end(); }
};

int main(int argc, char** argv) {
	if (argc != 2)
	{
		std::cerr << "Usage: ./test seed" << std::endl;
		std::cerr << "Provide a seed please" << std::endl;
		std::cerr << "Count value:" << COUNT << std::endl;
		return 1;
	}
	const int seed = atoi(argv[1]);
	srand(seed);

	ft::vector<std::string> vector_str;
	ft::vector<int> vector_int;
	ft::stack<int> stack_int;
	ft::vector<Buffer> vector_buffer;
	ft::stack<Buffer, std::deque<Buffer> > stack_deq_buffer;
	ft::map<int, int> map_int;
	for (int i = 0; i < COUNT; i++)
	{
		vector_buffer.push_back(Buffer());
	}

	for (int i = 0; i < COUNT; i++)
	{
		const int idx = rand() % COUNT;
		vector_buffer[idx].idx = 5;
	}
	ft::vector<Buffer>().swap(vector_buffer);

	try
	{
		for (int i = 0; i < COUNT; i++)
		{
			const int idx = rand() % COUNT;
			vector_buffer.at(idx);
			std::cerr << "Error: THIS VECTOR SHOULD BE EMPTY!!" <<std::endl;
		}
	}
	catch(const std::exception& e)
	{
		//NORMAL ! :P
	}

	for (int i = 0; i < COUNT; ++i)
	{
		map_int.insert(ft::make_pair(rand(), rand()));
	}

	int sum = 0;
	for (int i = 0; i < 10000; i++)
	{
		int access = rand();
		sum += map_int[access];
	}
	std::cout << "should be constant with the same seed: " << sum << std::endl;

	{
		ft::map<int, int> copy = map_int;
	}
	MutantStack<char> iterable_stack;
	for (char letter = 'a'; letter <= 'z'; letter++)
		iterable_stack.push(letter);
	for (MutantStack<char>::iterator it = iterable_stack.begin(); it !=
iterable_stack.end(); it++)
	{
		std::cout << *it;
	}
	std::cout << std::endl;
	return (0);
}
```