```cpp
#include <vector>
#include <iostream>
#include <memory>
#include <exception>

using std::vector;
using std::exception;
using std::cout;
using std::endl;
using std::shared_ptr;

class BadInput : std::exception
{
    public:
        explicit BadInput() = default;
};


template <class T>
std::vector<T> slice(std::vector<T> vec, int start, int step, int stop)
{
    if(start<0 || start>=vec.size() || stop<0 || stop>vec.size() ||
step<=0){
        throw BadInput();
    }
    if(start >= stop){
        vector<T> empty;
        return empty;
    }
    vector<T> slice;
    for(int index=start; index<stop; index+=step){
        try
        {
            slice.push_back(vec.at(index));
        }
        catch(const std::exception& e)
        {
            std::cerr << e.what() << '\n';
        }
    }
    return slice;
}

/**
 * Changed the pointers in values to smart pointers in order to not reach
double memory deallocation
 * Declared the default constructor and destructor
 */
class A {
public:
    std::vector<shared_ptr<int>> values;
    void add(int x) { values.push_back(std::make_shared<int>(x)); }
    ~A() = default;
    A() = default;
};
```

```
int main() {
    A a, sliced;
    a.add(0); a.add(1); a.add(2); a.add(3); a.add(4); a.add(5);
    sliced.values = slice(a.values, 1, 1, 4);
    *(sliced.values[0]) = 800;
    std::cout << *(a.values[1]) << std::endl;
    return 0;
}
```