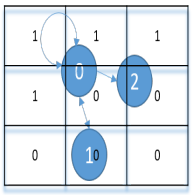

Data Structures with C++:

Programming Assignment 1 (37 Points): C++ Review, ADT's, & Online Profiles

Assignment Introduction:
There are two parts to assignment 1. Part 0 is self explanatory. Part 1 is to provide you with an opportunity to brush up on your C++ skills and to give you time to install the tools needed to complete programming assignments in this course.

Part 1 (35 points) Implement an Interface:
In part 1 you will implement an interface provided in this handout for a simple Abstract Data Type (ADT) called a **Container**. The interface has two private member variables, a stack allocated generic array and an integer size. The ADT must pass your own confidence tests submitted with your implementation in a file named **A01.cpp**. Refer to the incomplete example confidence tests provided in **driver.cpp**. **Be advised that assignments that do not compile and run will not be graded.**

Refer to the lecture slides, course-textbook, and supplemental reading regarding C++ classes and template programming. Implement the generic **Container** type using C++'s **templating** mechanism. Use the **inclusion method** for template programming by placing the member template definitions and the Container template declaration in a single file called **Container.hpp**. The **Container** type will have the functionality as provided in the interface shown on pages 2 & 3 of this handout. Implement only the `empty`,`add_item`,`get_item`,`remove_item` methods.

A **Container** object is instantiated with a datatype and a value argument. For example, the declaration for a ten element container of std::string elements is **Container<std::string,10>**. The value argument, 10 in the previous example, must be a constant expression. To access the instance type of a **Container** from a **Container** object, a **value_type** alias is provided in the implementation.

To support modern C++'s range based for looping mechanism, the **begin** and **end** methods are provided. For more details about how to implement the interface refer to the example usage and sample output on page 4 of this handout. Much of the code is provided for convenience to assist with the first assignment.

The **Container** interface is described below. Place the interface and implementation in a single file named **Container.hpp**:

```
//
// Name: Your First Name & Last Name
// SSID: Student ID Number
// Assignment #: 1
// Submission Date: 3/9/21
//
#ifndef _CONTAINER_HPP
#define _CONTAINER_HPP
#include <iostream>
#include <string>

template <class T, int N>
class Container
{
  public:
    using value_type = T;
    void add_item(T item);     //output container full, if add_item cannot add
    T get_item(int index);     //throw a string if index out of bounds
    void remove_item(T item); //remove first occurrence of item
    bool empty();              //check of Container is empty
    void clear();              //clear all contents, assign value_type
    constexpr int size();      //return current number of elements in container
    T* begin();
    T* end();

  private:
    int _size = 0;
    T container[N];
};//Container interface
```

```
  /* Sample Template member function definitions*/

  template <class T, int N>
  constexpr int Container<T, N>::size(){return _size;}

  template <class T, int N>
  void Container<T, N>::clear(){
        for (int i = 0; i < _size; i++)
                container[i] = T(); //or value_type();
        _size = 0;
  }

  template <class T, int N>
  T* Container<T, N>::begin(){ return _size ? &container[0] : nullptr;}

  template <class T, int N>
  T* Container<T, N>::end(){return begin() + _size;}

  //
  // TODO: implement add_item,get_item,remove_item, empty
  //
#endif
```

Example tests of the Container type. Place in a file named A01.cpp

```
#include <iostream>
#include <string>
#include "Container.hpp"

int main(){
  std::cout << "\n********************Test 1********************\n";
  std::cout << "Testing Container<std::string,5>\n";
  Container<std::string,5> container_of_strings;

  std::cout << "Testing Container<std::string,5>::add_items() {Green, Red, Black}\n";
  container_of_strings.add_item("Green");
  container_of_strings.add_item("Red");
  container_of_strings.add_item("Black");

  std::cout << "Testing Container<std::string,5>::range-based-for-loop()\n";
  for (auto& str : container_of_strings)
    std::cout << str << " ";

  std::cout << "\nTesting Container<std::string,5>::remove_item(Red)\n";
  container_of_strings.remove_item("Red");

  std::cout << "Testing Container<std::string,5>::range-based-for-loop()\n";
  for (auto& str : container_of_strings)
    std::cout << str << " ";

  std::cout << "\nTesting Container<std::string,5>::size()\n";
  std::cout << "container_of_strings.size()=" << container_of_strings.size() << "\n";
```

```
  std::cout << "\n********************Test 2********************\n";
  std::cout << "Testing Container<int,10>::add_item(){0,2,4,...,64,81}\n";
  Container<int,10> container_of_ints;
  for (int i = 0; i < 10; i++)
    container_of_ints.add_item(i * i);

  std::cout << "Testing Container<int,10>::range-based-for-loop\n";
  for (auto num : container_of_ints)
    std::cout << num << " ";

  std::cout << "\nTesting Container<int,10>::remove_item(16)\n";
  container_of_ints.remove_item(16);

  std::cout << "Testing Container<int,10>::range-based-for-loop\n";
  for (auto num : container_of_ints)
    std::cout << num << " ";

  std::cout << "\nTesting Container<int,10>::clear()\n";
  container_of_ints.clear();

  std::cout << "Testing Container<int,10>::empty()\n";
  std::cout << "container_of_ints.empty() is " << (container_of_ints.empty() ? "True" : "

  std::cout << "\n*****Test completed, enter any key to exit*******\n";
  char s; std::cin >> s;
  return 0;
}
```

Example output from tests:

```
********************Test 1********************
Testing Container<std::string,5>
Testing Container<std::string,5>::add_items() {Green, Red, Black}
Testing Container<std::string,5>::range-based-for-loop()
Green Red Black
Testing Container<std::string,5>::remove_item(Red)
Testing Container<std::string,5>::range-based-for-loop()
Green Black
Testing Container<std::string,5>::size()
container_of_strings.size()=2

********************Test 2********************
Testing Container<int,10>::add_item(){0,2,4,...,64,81}
Testing Container<int,10>::range-based-for-loop
0 1 4 9 16 25 36 49 64 81
Testing Container<int,10>::remove_item(16)
Testing Container<int,10>::range-based-for-loop
0 1 4 9 25 36 49 64 81
Testing Container<int,10>::clear()
Testing Container<int,10>::empty()
container_of_ints.empty() is True

*****Test completed, enter any key to exit*******
```