Creative Software Design, Assignment 11-2

Deadline: 2024-11-12 23:59 (No score for late submission)

- Submit your homework by uploading your zip file to the LMS assignment section. Below is an example.

```
13178_Assignment1-1_2024123456.zip
├─ 1.cc
├─ 2.cc
├─ 3.cc
└─ ...
```

- Your zip file name should follow this format:
  13178_Assignment[Assignment-number]_[Student-ID].zip
  - Ex. 13178_Assignment1-1_2024123456.zip
- Source files should be named as **<filename>.cc** *or* **<filename>.cpp**
- **You must submit your solution in the zip file before the deadline.**

1. **Implement a MyStack class template by inheriting from the Container class to create a stack data structure.**

   A. **The `Container` class template is provided as follows:**

```
template <typename T>
class Container {
public:
    Container() {}
    virtual void push(T value) = 0;
    virtual T pop() = 0;
    virtual bool isEmpty() = 0;
    virtual ~Container() {}
};
```

   1. **Data Storage:**
      i. Store data in a dynamic array within `MyStack`.
      ii. **Do not use** STL containers or external libraries for data storage
   2. **Constructor and Destructor:**
      i. `MyStack` should have a constructor to set the maximum size of the stack.
      ii. Implement a destructor to handle any dynamically allocated memory.
   3. **Member Functions:**
      i. `push(T value)`: Adds an item to the top of the stack.
      ii. `pop()`: Removes and returns the top item from the stack.
      iii. `isEmpty()`: Returns `true` if the stack is empty; otherwise, returns `false`.
   4. **Overflow and Underflow Handling:**
      i. **Overflow**: If `MyStack` reaches its maximum capacity when `push()` is called, double the size of the dynamic array to accommodate additional elements.
      ii. **Underflow**: If `MyStack` is empty and `pop()` is called, it should safely do nothing to prevent underflow.

B. **Example of the `main()` function**

```
int main() {
    Container<int>* myStack = new MyStack<int>(5);
    Container<double>* myStack_2 = new MyStack<double>(5);

    for (int i = 0; i < 10; i++) {
        myStack->push(i * 10);
        myStack_2->push(i * 10 + 0.5);
    }

    for (int i = 0; i < 10; i++) {
        int num = myStack->pop();
        cout << num << " ";
    }

    cout << endl;

    for (int i = 0; i < 10; i++) {
        double num_2 = myStack_2->pop();
        cout << num_2 << " ";
    }

    cout << endl;

    delete myStack;
    delete myStack_2;

    return 0;
}
```

C. Example output of your program (Bold text indicates user input):

```
90 80 70 60 50 40 30 20 10 0
90.5 80.5 70.5 60.5 50.5 40.5 30.5 20.5 10.5 0.5
```

D. Submission file: one C++ source file (File name: 1.**cc** or 1.**cpp**)

2. **Implement the MyFind function using a function template to locate an element within a specified range.**

A. **`MyFind` Function:**

1. The `MyFind` function finds the position of an element within a range.
2. **Template Arguments:**
   i. `InputIt`: The input iterator type for the elements to examine.
   ii. `T`: The data type of the elements to compare.
3. **Input Arguments:**
   i. `first, last`: Input iterators representing the beginning and end of the sequence to search.
   ii. `val`: The value to search for in the range.
4. **Return Value:**
   i. Returns an iterator to the first element in the range that matches `val`.
   ii. If no element matches, the function returns `last`.

B. **Restrictions**

1. **Do not** use the library `find` function.

C. **Example of the `main()` function:**

```
int main() {
    int myInts[] = { 10, 20, 30, 40 };
    int* p;
    vector<int> myVector(myInts, myInts + 4);
    vector<int>::iterator it;

    p = MyFind(myInts, myInts + 4, 50);
    if (p != myInts + 4)
        cout << "Element found in myInts: " << *p << '\n';
    else
        cout << "Element not found in myInts\n";

    it = MyFind(myVector.begin(), myVector.end(), 30);
    if (it != myVector.end())
        cout << "Element found in myVector: " << *it << '\n';
    else
        cout << "Element not found in myVector\n";

    return 0;
}
```

D. Example output of your program (Bold text indicates user input):

```
Element not found in myInts
Element found in myVector: 30
```

E. Submission file: one C++ source file (File name: 2.**cc** or 2.**cpp**)

3. **Implement MyCopy and MyMerge functions using function templates to replicate the functionality of copying and merging sequences.**

   A. **`MyCopy` Function:**

      1. The `MyCopy` function copies elements from a specified range to a destination sequence.
      2. **Template Arguments:**
         i. `InputIt`: The input iterator type of elements to copy.
         ii. `OutputIt`: The output iterator type where the elements are stored.
      3. **Input Arguments:**
         i. `first, last`: Input iterators specifying the beginning and end of the range to be copied.
         ii. `result`: Output iterator pointing to the starting position in the destination sequence where the copied elements will be stored.
      4. **Return Value:**
         i. `void`

   B. **`MyMerge` Function:**

      1. The `MyMerge` function combines elements from two sorted ranges into a new, single sorted range.
      2. **Template Arguments:**
         i. `InputIt1`: The input iterator type for the first range of elements.
         ii. `InputIt2`: The input iterator type for the second range of elements.
         iii. `OutputIt`: The output iterator type where the merged elements are stored.
      3. **Input Arguments:**
         i. `first_1, last_1`: Input iterators defining the beginning and end of the first sorted sequence.
         ii. `first_2, last_2`: Input iterators defining the beginning and end of the second sorted sequence.
         iii. `result`: Output iterator specifying the starting position for the combined sequence
      4. **Return Value:**
         i. `void`

C. **Restrictions**

1. **Do not** use the library `copy` or `merge` functions.

D. **Example of the `main()` function:**

```
int main() {
   int first[] = { 5, 10, 15, 20, 25 };
   int second[] = { 10, 20, 30, 40, 50 };
   vector<int> myVector(5);
   vector<int> v(10);

   MyCopy(first, first + 5, myVector.begin());
   cout << "myVector contains:";
   for (vector<int>::iterator it = myVector.begin(); it != myVector.end(); ++it)
      cout << ' ' << *it;
   cout << endl;

   MyMerge(first, first + 5, second, second + 5, v.begin());
   cout << "the resulting vector contains:";
   for (vector<int>::iterator iter = v.begin(); iter != v.end(); ++iter)
      cout << ' ' << *iter;
   cout << '\n';

   return 0;
}
```

E. Example output of your program (Bold text indicates user input):

```
myVector contains: 5 10 15 20 25
the resulting vector contains: 5 10 10 15 20 20 25 30 40 50
```

F. Submission file: one C++ source file (File name: 3.**cc** or 3.**cpp**)