System and Device Programming

The Hard Life of C++ Developers

Exercise 1

Stacks are containers specifically designed to operate in a last-in first-out (LIFO) context, where elements are inserted and extracted only from one end of the container. In particular, you expect to add a new element at the top so that it will be the first to be retrieved. The STL library already has an implementation for this kind of container.
Try to implement your own Stack template container class following the **partial** implementation provided as follow:

```
template<typename T> class StackClass;

template <typename T> class StackClass
{
  public:
    StackClass();

    ~StackClass();

    void push(const T &val);

    T pop();

    bool empty() const;

    int getSize() const;

    T* getDataContainer();

    vector<T> getStackAsVector();

private:
    T * _dataContainer;
    int _size;

}
```

The functionalities are:

- Copy constructor and assignment operator.
- Move constructor and assignment operator.
- Dynamic container size: allocating only the necessary amount of memory when a push operation is asked or reducing the size when a pop is done.

- `void push(const T &val)`: inserts the new element with value val at the top of the stack.
- `T pop()`: deletes the element at the top of the stack and returns it.
- `vector<T> getStackAsVector()`: returns the stack content as a `std::vector` container. It respects the actual items' order (first element of the vector is the last entered one).
- `T* getDataContainer()`: return a raw pointer to the internal data.
- Reverse function to invert the order of the stack.
- All feeling needed operators (such as '+'), including `friend` operators for streaming.

Make the methods exception-aware when necessary.
Try to develop some small example to test your container.

Exercise 2

Modify the container defined in the previous exercise to manage first-in first-out (FIFO), in order to support the implementation of a simple queue system for a postal office. The system provides a menu to the user to do one of the following operations:

1. Request a new number for a client.
2. Serve next client.
3. Serve all & close the office.

If the user chose to request a new number, it has to provide what kind of client is at the front door:

1. Priority client.
2. Postal services client.
3. Money services client.

If a priority client is coming, it must be put the front of the queue if no other priority client is already at the top of the queue. In this latter case, the priority client is append before the first non-priority client in the list. If a postal client is coming, it must be placed in the queue by respecting the following rule: without considering priority clients, every three money clients one postal client must be served. Eventually, a money client is ever append at the end of the queue.
Whenever the user chose to serve the next client, the client at the top of the list is served (removed from the list) and the system display its number and what kind of client has been served.
To conclude the program, the "serve all" option should be chose: it iterates a serve next client operation until the list is empty.