Programming Assignment 1 Part 2

The Collection Class

Objectives: This assignment gives you some experience with designing and writing C++ classes using "big five" (see the textbook), operator overloading, exception, and templates (Part 3).

(5 Points) Create a README based on the standard format

- Submit to Canvas a PDF Version of the README
- Test the Program locally in your Linux Environment
- Starter Code and Autograder is provided for your reference on GitHub
- A `sample_main.cpp` is provided for testing
- One complete, call `make deliverable` and upload the output tar-archive and README to Canvas

---

Download the autograder and starter code with (paste all 3 lines at once):

```
REMOTE=updates git-update-agent --init-with \
https://github.tamu.edu/alex-born/csce221-pa1-p2 \
--dir pa1-p2
```

Problem Description

(35 Points) Implement the Class Collection

The class `Collection` defines a collection as an array that can be automatically resized as necessary using dynamically allocated arrays.

YOU MAY NOT USE `stl::vector` or any similar library

The class `Collection` will store `StressBall` from part 1. The class collection should have the following `private` members:

- `StressBall *array;` - A pointer to dynamically allocated memory
- `int size;` - An integer detailing the number of elements in the collection
- `int capacity;` - An integer detailing the size of the array

Note that `size <= capacity`

The following functions should be implemented:

- `Collection();` - *Default Constructor* - `size` and `capacity` are `0`, `array` is `nullptr`
- `Collection(const int cap);` - `cap` describes the required capacity of the collection. The `array` should be allocated with memory equal to `cap` and `capacity == cap`
- `Collection(const Collection &c);` - *Copy Constructor* - copy the contents of `c` into `this`
- `Collection &operator=(const Collection &c);` - *Copy Assignment* - copy the contents of `c` into `this`. Be sure to not introduce memory leaks.
- `~Collection();` - *Destructor* - deallocate `array`, set `size` and `capacity` to `0`
- `Collection(Collection &&c);` - *Move Constructor* - move the contents of `c` into `this`. Return `c` to the same state as the default constructor.
- `Collection &operator=(Collection &&c);`- *Move Assignment* - move the contents of `c` into `this`. Return `c` to the same state as the default constructor. Be sure to not introduce any memory leaks.
- `void insert_item(const Stress_ball &sb);` - insert `sb` into the collection. If the collection is full, resize by doubling `capacity` and allocating more memory. Be sure to not introduce any memory leaks.
- `bool contains(const Stress_ball &sb) const;` - return `true` if and only if `sb` matches the contents of one or more elements in the collection.
- `Stress_ball remove_any_item();` - remove and return a random stress ball from the collection. Throw an exception if the collection is empty. Ensure there are no gaps in the collection after the element is removed.
- `void remove_this_item(const Stress_ball &sb);` - remove the the first stress ball in the collection that matches `sb`. Throw an exception if the collection is empty or no items match. Ensure there are no gaps in the collection after the element is removed.
- `void make_empty();` - make the collection empty (remove all elements) and deallocate all memory. Be sure not to introduce memory leaks.
- `bool is_empty() const;` - return `true` if the collection is empty.
- `int total_items() const;` - return the the number of items in the collection.
- `int total_items(const Stress_ball_sizes s) const;` - return the number of items in the collection where size is `s`.
- `int total_items(const Stress_ball_colors c) const;` - return the number of items in the collection where color is `c`.
- `void print_items() const;` - print the items in the collection, one item per line in the format `(color, size)`
- `Stress_ball &operator[](const int i);` - overload the operator to return a reference to the `i`th element; where the first element is at `i = 0`
- `const Stress_ball &operator[](const int i) const;` - see `operator[](const int)` above

(45 Points) Additional Support Functions

Implement the following supporting functions *outside* the `Collection` class:

- `std::istream &operator>>(std::istream &is, Collection &c);` - *Stream Removal Operator* - read from input stream `is` and put the elements into the collection. Note: it will be helpful in part 3 for this function to be generic; consider implementing `istream &operator>>(istream &is, StressBall);` and calling that within this function
- `std::ostream &operator<<(std::ostream &os, const Collection &c);` - *Stream Insertion Operator* - for each element in the collection, print the `operator<<` representation into the stream. Insert a newline between each element.
- `Collection make_union(const Collection &c1, const Collection &c2);` - combine the contents of `c1` and `c2` into a new collection without modifying `c1` or `c2`
- `void swap(Collection &c1, Collection &c2);` - using the move constructor and move assignment, swap the contents of `c1` and `c2`. **DO NOT COPY ELEMENTS**
- `void sort_by_size(Collection &c, const Sort_choice sort);` - sort the elements of the collection on one of `bubble_sort`, `insertion_sort`, or `selection_sort`. Use a `switch` statement.

  YOU MAY NOT USE `stl::sort` or any similar library

(0 Points) Additional Notes

You may find the following helper functions to be useful:

- `std::istream& operator>>(std::istream &o, Stress_ball & sb);` - *Stream Removal Stress Ball* - read a color, size pair from the stream and update the `sb` object. This may require the statement `friend std::istream& operator>>(std::istream &o, Stress_ball & sb);` to allow access to the private data members.

If reusing the exact files from PA1-P1, You may need to `const` qualify `operator==` for stress ball:

- `bool operator==(const Stress_ball &sb) const;`