Assignment 1 – A 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. Also you will learn how to use the command line interface (CLI) and makefiles.*

- (5 points) Create a file, called README (Template provided as LyX file. You are free to use any software but stick to the format given in the templated file):
  - Submit to eCampus an electronic version of the file README by September 10th.
  - Test the C++ programs on your Computer Science Linux machine.
  - The assignment will be graded focusing on: program design, correctness.
  - You will be given a .zip file containing `Collection.cpp, Stress_ball.cpp, Collection.h, Stress_ball.h, main.cpp, makefile, Stress_ball1.data` and `Stress_ball2.data` files. Implement the methods given in `Collection.h` and `Stress_ball.h` and complete `Collection.cpp` and `Stress_ball.cpp` respectively.
  - Use `main.cpp` to test your implementation. When your program works correctly, upload only **`Collection.cpp, Stress_ball.cpp`** and **`makefile`** to Mimir Classroom by September 10th where your program will be tested against TA's test cases. **Do not upload any other file. Do not use main() function in `Collection.cpp or Stress_ball.cpp`. Do not remove the header files included in `Collection.cpp or Stress_ball.cpp`.**
  -

Problem Description – Part 2 (100 pts)

1. (35 points) The class `Collection` defines a collection as an array that can be automatically resized as necessary using dynamically allocated arrays.

   You are not allowed to use the STL class vector.

   (a) There is a class `Collection` which uses the class `Stress_ball`. The collection holds stress balls of different colors and sizes (see Part 1) and can contain many stress balls of the same color and size. The class `Collection` should have three private members:

   `Stress_ball *array; //pointer to dynamically allocated memory`

   `int size; //logical size of array - the number of elements (Stress_balls) in use`

   `int capacity; //physical size of array`

   Note that `size` <= `capacity`.

   (b) These are the following functions defined for a collection:

   - constructor with no arguments, `size` and `capacity` are `0`, and `array` is `nullptr`.
   - constructor with one argument which is the required size of the collection
   - copy constructor – makes a copy of a collection
   - copy assignment – overwrites an exiting collection by another collection
   - destructor – destroys a collection (deallocates allocated memory, set to zero `size` and `capacity`)
   - move constructor – efficiently creates a new collection from an existing one
   - move assignment – efficiently copies a collection during an assignment

- insert a stress ball to the collection:
  `void insert_item(const Stress_ball& sb);`
  If the collection is full, increase the array by doubling its size. Use the private helper function `resize()` to complete this task. The function `resize()` should double the size of the array and correctly copy elements from the old array to a new array.
- check if a stress ball of a given color and size is in the collection; return true if it is there and false otherwise:
  `bool contains(const Stress_ball& sb) const;`
- remove and return a random stress ball (you have no control which stress ball is selected):
  `Stress_ball remove_any_item();`
  Do not decrease the size of the array (Do not change capacity). Also, be sure that there are no gaps between elements of the array. Throw an exception if the collection is already empty.
- remove a stress ball with a specific color and size from the collection:
  `void remove_this_item(const Stress_ball& sb);`
  Do not decrease the size of the array. Also, be sure that there are no gaps between elements of the array. Throw an exception if the collection is already empty.
- make the collection empty (deallocate allocated memory, set to zero `size` and `capacity`):
  `void make_empty();`
- check if the collection is empty; return true if it is empty and false otherwise:
  `bool is_empty() const;`
- return the total number of stress balls in the collection:
  `int total_items() const;`
- return the number of stress balls of the same size in the collection:
  `int total_items(Stress_ball_sizes s) const;`
- return the number of stress balls of the same color in the collection:
  `int total_items(Stress_ball_colors t) const;`
- print all the stress balls in the collection (print color and size of a stress ball, see the class `Stress_ball`):
  `void print_items() const;`
  The format has to be :
  ```
  (color, size)
  (color, size)
  .
  .
  .
  (color, size).
  ```
  For example, output should look like,
  ```
  (red, small)
  (blue, medium)
  (yellow, large)
  (blue, large)
  .
  .
  .(green, medium).
  ```

(c) To directly access a stress ball in a collection, overload `operator[]`. It will access a stress ball in `array` at position `i` where `i` starts from `0` through `size-1`:
`Stress_ball& operator[](int i);`

(d) To directly access a stress ball in a `const` collection, overload `operator[]`. It will have the exact same body as the above overload, but the function header should read:
`const Stress_ball& operator[](int i) const;`

The C++ program must be submitted to Mimir Classroom and the README file must be submitted to eCampus by September 10th. You should test all the implemented functions/operators of this class.

2. (45 points) Add these functions for manipulating collections. They are *not* part of the class `Collection`.

   - input operator (reading from a file):
     `istream& operator>>(istream& is, Collection& c);`
     reads from the `istream is` pairs in this format: color size (no parentheses or colons, use space to separate them). As colors use strings (you can use STL class `string` here): red, blue, yellow, green, and as sizes use strings: small, medium, large. Data is read from an input file in `main.cpp` and it is passed to `istream is`. Sample input files are also provided.
   - output operator:
     `ostream& operator <<(ostream& os, const Collection& c);`
     prints to the `ostream os` all the collection items in format: (color, size), each in one line. Use `cout` for output.
   - a union operation that combines the contents of two collections into a third collection (the contents of `c1` and `c2` are not changed):
     `Collection make_union(const Collection& c1, const Collection& c2);`
   - a swap operation that swaps two collections:
     `void swap(Collection& c1, Collection& c2);`
     Use the move constructor and move assignment to do this. Do not copy the collection elements.
   - A sort function that sorts the collection with respect to the size of its elements (`small` < `medium` < `large`):
     `void sort_by_size(Collection& c, Sort_choice sort);`
     Then elements will be sorted with respect to their size in `array` (we do not sort them with respect to color). You need to implement 3 different sorting algorithms (do not use the STL `sort`): bubble sort, insertion sort, and selection sort. Here is the enum class `Sort_choice`:
     `enum class Sort_choice { bubble_sort, insertion_sort, selection_sort };`
     Use `switch` statement to choose the required one. You need to test all three sort functions.

3. (15 points) You are given a skeleton `makefile`. Complete it and upload the same. "`test`" is the name of executable file that is used in the `makefile`. Please do not change this. Your files (`Collection.cpp`, `Stress_ball.cpp` and `makefile`) will be tested against a `main.cpp` on Mimir. Hence, do not upload your `main.cpp`.

   The C++ program must be submitted to Mimir Classroom and the README file must be submitted to eCampus by September 10th. You should test all the implemented functions/operators of this class.