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, and templates. Also you will learn how to use the command line interface (CLI) and makefiles.*

- (5 points) Create a text file, called README, in which you provide:
  - Your First Name, Last Name, UIN, Section Number, User Name, E-mail address
  - State the Aggie Honor statement:

    I certify that I have listed all the sources that I used to develop the solutions and code to the submitted work.

    On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

    Your Name Date
  - List any resources used such as webpages (provide URL). Do not mention the textbook and discussions with the Instructor, TA, or Peer Teachers.
  - List any known problems with the assignment you are turning in. For example, if you know your code does not run correctly, state that. This should be a short explanation.
  - Provide a short description or pseudocode.
  - Write how you tested your program(s) for correctness and how you used exceptions for the error handling.
  - Submit to eCampus an electronic version of the file README along with your C++ code for the problems listed below and a hard copy to your TA.
  - The assignment will be graded focusing on: program design, correctness.
  - Your programs will be tested on a CSE Linux machine.
  - Provide all tests done to verify correctness of your program in the report. Give an explanation why have you selected such tests.
  - Your program will be tested on a TA's input file.

Problem Description – Part 1 (100 pts)

1. Write a C++ program to implement a collection for organizing data and for performing operations on this data. A collection contains items in no particular order and it might have duplicate items. It is probably the simplest way of organizing data. In our particular case you have a collection of stress balls of different colors and sizes and there is no specific arrangement or organization of these items.

2. (25 points) Write a class `Stress_ball` which represents a stress ball.

   (a) The class default constructor creates a stress ball with a randomly selected color and size, every time the constructor is called. Use only the following colors: `red`, `blue`, `yellow`, and `green`, and sizes: `small`, `medium`, and `large`. Apply `enum class Stress_ball_colors` to define colors and `Stress_ball_sizes` for sizes.

   (b) The class parameterized constructor creates a stress ball with a given color and size:
   `Stress_ball(Stress_ball_colors c, Stress_ball_sizes s)`

   (c) The function `get_color()` returns the color of a stress ball using the enum class `Stress_ball_colors`

   (d) The function `get_size()` returns the size of a stress ball using the enum class `Stress_ball_sizes`

   (e) The operator `==(const Stress_ball& sb)` returns `true` if `sb`'s color and size are the same as color and size of the stress ball `this` (object calling the operator).

   (f) Outside of this class: overload `operator<<(std::ostream& o, const Stress_ball& sb);` to print a stress ball as a pair of color and size in this form: (color, size). Example: `(red, small)`

The class Stress_ball should be presented to your TA or PT during the labs and submitted to eCampus by January 22nd. You should test all the implemented functions/operators of this class.

3. (35 points) The class `Collection` defines a collection as an array that can be automatically expanded as necessary using dynamically allocated arrays. *You are not allowed to use the STL class vector.*

   (a) Write a C++ class `Collection` which uses the class `Stress_ball`. The collection holds stress balls of different colors and sizes (see above) 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 in use
   int capacity; //physical size of array = the number of allocated elements
   ```

   Note that `size` <= `capacity`.

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

   - constructor with no arguments. The default size of the collection is 8.
   - constructor with one argument which is the required size of the collection
   - copy constructor – allows to make a copy of the collection
   - copy assignment – allows to overwrite an exiting collection by another collection
   - destructor – allows to destroy a collection
   - move constructor – allows to efficiently create a new collection from an existing one
   - move assignment – allows to efficiently copy 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. 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.
- make the collection empty:
  `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;`

(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 class Collection should be presented to your TA or PT during the labs by January 29. You should test all the implemented functions/operators of this class.

4. (20 points) Add these functions for manipulating collections. They are not part of the class `Collection`.

   - a copy operation that makes a copy of a collection:
     `Collection make_copy(const Collection& c);`
   - a union operation that combines the contents of two collections into a third collection (the contents of `c1` and `c2` are not changed):
     `Collection 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);`
     Then elements will be sorted with respect to their size in `array` (but we do not sort them with respect to color).

5. (15 points) Write a program for testing collections where you will test the class `Collection` implemented based on an array.

   (a) Call the file `test_stress_ball_collection.cpp`. It should contain the `main()` function. Use makefile to compile your program.

   (b) Create two collection objects. Fill them with stress balls: read stress balls from a user input file. Use between 10 and 15 stress balls.

(c) Sort and print out their contents: print out the number of stress balls of a given color and the total amount of stress balls in the first and the second collection.

(d) Create a third collection which is the union of the two objects from (b). Print its elements. Print the number of stress balls of the same color and size. Print the total number of stress balls.

(e) Swap two collections in (b). Print its elements to check if they correctly swapped.

(f) Sort the first collection in (b). Print its elements to check if it is correctly sorted.

The class Collection should be presented to your TA or PT during the labs by February 5th. You should test all the implemented functions/operators.

Problem Description – Part 2 (100 pts)

due February 14

1. (50 points) Write a templated version of the class `Collection` with the template parameters: `Object`, `Feature1`, `Feature2`.

   (a) Test it with the class `Stress_ball` as `Object`, `Stress_ball_sizes` as `Feature1`, and `Stress_ball_colors` as `Feature2`.

   (b) Perform the same operations: `make_copy`, `union`, `swap`, and `sort_by_size` for testing.

   (c) Modify the test file to do this.

2. (30 points) Write a class `Jeans` similar to `Stress_ball` with the same class members: color and size. Feel free to use your own colors and sizes.

   - Apply the `Collection` functions to `Jeans` objects
   - Provide similar testing cases for the templated `Collection` with `Jeans` objects.

3. (15 points) Write about the generic programming using templates based on this assignment part 2.