Assignment 4.

Bitwise operations and generic programming

Purpose of the exercise

This exercise will help you do the following:

- Demonstrate understanding of object-oriented programming techniques.
- Develop understanding of functionality provided by a type `std::bitset` from STL.
- Practice use of bitwise operators and related conversion rules.

Requirements

The Standard Template Library of C++ offers a type std::bitset to represent a collection of `N` bits in an efficient way, that is storing them packed with multiple bits in each byte. Bits can be manipulated individually through member functions such as:

- `set(size_t pos, bool value = true)` - sets a bit at a position `pos` to a given `value`;
- `reset(size_t pos)` - clears a bit at a position `pos` by setting its value to `false`;
- `flip(size_t pos)` - toggles a bit at a position `pos` by negating it.

Your task is to get familiar with the documentation describing `std::bitset<N>`, and implement your own version of this class template that offers similar functionality. Include only member functions that are used by the test driver; make sure that their declarations match the ones from `std::bitset<N>` in terms of parameter types, count and return values. You are expected to develop the interface (*bitset.h*) and the implementation (*bitset.hpp*) that behave as expected when used by the test driver. If you compile the code with a pre-processor macro definition `USE_STL_BITSET` you can build and test the driver with the STL implementation of `std::bitset<N>`. STL implementation runs only the first 13 tests. Your implementation has to follow additional constraints and offer **dynamic memory allocation**; it must complete all tests.

To successfully implement the assignment, before writing code review bitwise operators. Pay attention to data type promotions and conversions that happen during evaluation.

While implementing the assignment, you must follow these rules:

1. You need to define only the functionality used by the test driver; no other functionality is required even though `std::bitset<N>` exposes more functions and their overloads.
2. An instance of your class must have exactly `sizeof(void*)` size; the data must be stored in a dynamic array of the smallest possible size sufficient to contain `N` bits (counted from `0` to `N-1`); take note that `std::bitset<N>` does not adhere to this constraint as it may store data in a data member of a static array type.
3. Your implementation must be cross-platform: it must work properly even on a platform with a non-standard number of bits in a byte (use the `CHAR_BIT` macro).
4. You **must** use bitwise operators for setting, clearing, flipping and testing individual bits in bytes; performing arithmetic with other operations (i.e. addition, multiplication) may be penalized in manual code review.
5. You must split the interface and the implementation of the class template into separate files, **.h* and **.hpp* headers respectively. You may need to include **.hpp* at the end of **.h*.
6. You must not use any header files besides the ones already included in the provided code.
7. Pay attention to encapsulation, `const`-correctness, avoid shallow copy and memory leaks.

Requested files

Without any comments you can expect files to be around the following sizes:

- *bitset.h* - 50 lines.
- *bitset.hpp* - 150 lines.

No *checklist* or comments are required; it is a good practice to include them, but the automated grading tests will not penalize you for the lack of comments, at least in this assignment.

At this level of the course, it is expected that the style of your code is elegant, easy to read and has a desired quality of being self-explanatory and easy to understand. The automated grading tests will not penalize you for low quality of the code working properly, at least in this assignment.

Submitting the deliverables

You have to upload requested files to Moodle - DigiPen (Singapore) online learning management system, where they will be automatically evaluated.

To submit your solution, open your preferred web browser and navigate to the Moodle course page (pay attention to the section name suffix at the end of the course name). In the course page find a link to the Virtual Programming Lab activity that you are submitting.

In the **Description** tab of the activity you can find a due date, and a list of requested files. When you switch to the **Submission** tab you should see the controls for uploading or typing exactly the files that are required. Upon clicking the *Submit* button the page will validate submitted files and report any errors. If the submission was successful, the page will display a message that the submission has been *Saved*. You should press the *Continue* button to see the *Submission view* page with the results of the evaluation and the grade.

If you received an *A* grade, congratulations! If not, before the due date you can still review and update your solution and resubmit again. Apart from exceptional circumstances, all grades after the due date are final, and students who did not submit their work will receive a grade *F*.