Project 1: Bit Vectors

Software languages are tuned for data representations of integers, floating point numbers, even strings of data. However, while fundamentally the underlying hardware uses bit vectors, it is harder to represent this data in software languages. To build computing hardware emulation flows, we must be able to design data types to represent groups of bits.

Description

A bitset or bit array is a simple data structure representing a set of bits, each of which may be 0 (false) or 1 (true). Bitsets have many uses in computing and appear in several algorithms. The C++ standard library has a bitset class where the number of bits or *size* (N) must be known at **compile time**. Our class is similar but the size of the bitset can be specified at **runtime**. An individual bit in the set is addressed by an *index* in the range 0 to (N – 1).

The underlying implementation of a bitset is as a dynamic array of integer types, that is, one of the fixed width types. The size of the underlying array should be large enough to hold N bits. For example, suppose you used an array of type `u_int8_t` and the size of the bitset is 12 (N = 12), then the size of the array would need to be 12.

Specification

Define and implement a class `Bitset` that should support:

- default construction of a valid bitset of size 8, with all bits set to 0
- construction of a valid bitset of size N, where N is of type `intmax_t`, with all bits set to 0; the bitset is invalid if N <= 0.
- construction of a valid bitset initialized with a string of the form `00010000`. If the string contains any character other than `0` or `1`, the bitset is invalid.
- a method to return the size of the bitset
- a method to determine if the bitset is valid
- a method to set the nth bit to 1, and if n is not in [0, N-1] then the bitset becomes invalid
- a method to reset the nth bit to 0, and if n is not in [0, N-1] then the bitset becomes invalid
- a method to toggle the nth bit (1 to 0 or 0 to 1), and if n not in [0, N-1] then the bitset becomes invalid
- a method to check if the nth bit is set (1) by returning a bool value of true if set and false if not, and if n is not in [0, N-1] then the bitset becomes invalid and false is returned
- a method to return the bitset as a std::string of characters `0` and `1` . This string should represent the bitset digits from left-to-right with the most significant bit first.

Hint: valid or invalid means that there should be a private member variable that tells if this instance of the class is valid or not. This variable should be set according to corresponding behavior above.

Note: the class is not Copy-Constructible or Copy-Assignable. This means you will not be able to copy or assign one bitset instance to another, and as a consequence, you will not be able to pass or return a bitset by value from a function.

The outline of the class is defined in the starter code inside the file `bitset.hpp` as:

```
class Bitset{
public:
  // TODO COMMENT
  Bitset();
  // TODO COMMENT
  Bitset(intmax_t size);
  // TODO COMMENT
  Bitset(const std::string & value);
  // TODO COMMENT
  ~Bitset();

  Bitset(const Bitset & ) = delete;
  Bitset & operator=(const Bitset &) = delete;
  // TODO COMMENT
  intmax_t size() const;
  // TODO COMMENT
  bool good() const;
  // TODO COMMENT
  void set(intmax_t index);
  // TODO COMMENT
  void reset(intmax_t index);
  // TODO COMMENT
  void toggle(intmax_t index);
  // TODO COMMENT
  bool test(intmax_t index);
  // TODO COMMENT
  std::string asString() const;
private:
  // TODO
};
```

Do not modify the public portion of the class definition. To ensure you understand memory management, it is **required** that you manually perform allocation and deallocation of memory as part of your implementation, that is, do not use `std::vector` or any other container in your implementation. Your implementation should not leak memory or have invalid read/writes.

To complete this project:

- In `bitset.cpp` : Define the **internal members and methods (marked TODO)** and **implement all methods**.
- In `bitset.hpp` : Add appropriate comment blocks describing each method (marked TODO COMMENT).
- In `bitset_test.cpp` : Write tests using the Catch testing framework. The included `CMakeLists.txt` file sets up everything for you.

Testing Your Code

It is strongly recommended that you aggressively test your code.

Before you submit your project, remove CMakeCache.txt and recompile and run the executable to mimic how the graders will run your code - make sure that what you're about to submit is what you want graded. **If you submit a project that does not compile, it will receive a grade of 0.** To recompile the project, open a terminal in the same working directory that has your top-level code and run the following commands:

- `rm CMakeCache.txt`
- `cmake .`
- `make`
- `./bitset_test`

Ensure that all tests pass, and that there are enough tests for all of the specified functionality.

Project Submission

Generate a zip file for submission via Canvas by running this command in the top-level directory:

- `make submission`

These are some of the questions we are going to take into account as when we grade the code:

- Does it compile?
- Does it pass the test cases you created?
- Does it pass the test cases we create?

When we test your projects, we will run the same commands as above to check the output.

Bonus

While we didn't specify what type of fixed-width integers to use for the dynamic bitset array, using any type of integer (`u_int8_t` , `u_int16_t`, etc…) to represent one bit would be waste of memory. It makes more sense to actually make each of these bits correspond to a bit in the bitset.
For example, if we are going to use `u_int8_t` and the size of the bitset is 12, then we will be needing an array of size 2 (2 * 8 [size of the `u_int8_t`] = 16 > 12). So we would be using all the 8 bits of one element and only 4 for bits of the other.

Doing so would involve a lot Boolean algebra manipulation (bitwise and, or, xor), logical shifting (>> or <<), and (mod % and div /) operation. Please **DO NOT** attempt this part unless you actually finish the project the regular way first and you are **absolutely comfortable** with Boolean algebra.

Implementing the project either way should have no effect on the test bench output at all. We will check this part by manually viewing the code.

NOTE: Before you attempt this part, make sure you have committed (git commit) and pushed (git push) your working code for the project, so you have a working commit to return to if you cannot finish this part.