CS 2110 Homework 1
Bases and Bitwise Operations

Henry Bui, Elizabeth Hong, Huy Nguyen, Bryce Hanna

Spring 2025

Contents

1 Objective

1.1 Purpose

The goal of this assignment is for you to understand bitwise operators, and how to use them to manipulate data (including integers and ASCII codes for characters) programmatically. For this assignment you will be programming in Java, because you should already be familiar with it. The concepts of bitwise operations that you apply here will be used throughout the remainder of this course, including in machine language, assembly language, and C programs.

1.2 Tasks

You will complete the Java methods in the provided files: `BitVector.java`, `Bases.java`, and `Operations.java`. The comments in the provided code files describe what each method should accomplish. The restrictions on which operators you are allowed to use may change between files and sometimes within methods of the same file, so it is in your best interest to read each comment carefully. Before you start this assignment, please read the rest of this document for more detailed instructions and hints.

1.3 Criteria

Your coding grade is based on: 1) the correct output from your methods; 2) not using any banned operators; and 3) not hardcoding a literal result from a method, or any other such workaround. The grade you see on Gradescope is the grade you receive, unless we find that you've hardcoded return values.

You may submit your code to Gradescope as many times as you like until the deadline. We will only grade your last submission. We have also provided a local autograder that you can test your code with. Submit to Gradescope early and often, to help ensure that you do not encounter issues submitting at the last minute.

2 Instructions (for Coding)

1. Make sure all 4 files are in the same folder:

   (a) `BitVector.java`

   (b) `Bases.java`

   (c) `Operations.java`

   (d) `hw1checker.jar`

   Note: An `Examples.java` file is also included which shows and explains examples of two methods similar to those used in your assignment. This is just provided for reference, so there are no tasks to be completed within it.

2. Each file has a set of rules for what you can and cannot use in your solutions. These rules are in place to ensure that you are applying the relevant concepts and critically thinking rather than providing naive solutions that don't properly test your knowledge of binary representations and bitwise operations. The rules for each file are as follows:

   - `BitVector.java`
     - You may not use multiplication, division or modulus in any method.
     - You may not use more than 2 conditionals per method, and you may only use them in select methods. Conditionals are if-statements, if-else statements, or ternary expressions. The else block associated with an if-statement does not count toward this sum.

  - You may not use looping constructs in most methods. Looping constructs include for loops, while loops and do-while loops. See below for exceptions
  - In those methods that allow looping, you may not use more than 2 looping constructs, and they may not be nested.
  - You may not declare any file-level variables.
  - You may not declare any objects.
  - You may not use switch statements.
  - You may not use casting.
  - You may not use any String methods (ex. length, substring).
  - You may not use the unsigned right shift operator (<<<)
  - You may not write any helper methods, or call any method from this or another file to implement any method. Recursive solutions are not permitted.
  - You may only perform addition or subtraction with the number 1.
  - All methods must be done in one line, except for: `onesCount` and `trailingZerosCount`.
  - Looping and conditionals as described above are only allowed for: `onesCount` and `trailingZerosCount`.
- `Bases.java`
  - You cannot use multiplication, division, and modulus operators
  - You cannot use nested loops
  - You cannot declare file-level variables
  - You cannot use switch statements
  - You cannot use the unsigned right shift operator (<<<)
  - You cannot use helper methods, call any other methods, or use recursion.
  - You may not use more than 2 conditionals per method. Conditionals are if-statements, if-else statements, or ternary expressions. The else block associated with an if-statement does not count toward this sum.
  - You may not use more than 2 loops per method. This includes for loops, while loops and do-while loops.
  - The only Java API methods you are allowed to invoke are: `String.length()`, `String.charAt()`
  - You may not invoke the above methods from String literals. Example: `"12345".length()`
  - When concatenating numbers with Strings, you may only do so if the number is a single digit.
- `Operations.java`
  - All of these functions must be completed in ONE line. That means they should be of the form `return [...];`. No partial credit will be awarded for any method that isn't completed in one line.
  - You may not use conditionals.
  - You may not declare any variables.
  - You may not use casting.
  - You may not use the unsigned right shift operator (<<<)
  - You may not write any helper methods, or call any method from this or another file to implement any method. Recursive solutions are not permitted.
  - You may not use addition or subtraction. However, you may use the unary negation operator (-) to negate a number (except in `changeSign`).
  - You may not use multiplication, division or modulus.
  - Basically, the global rules are you can only use logical (&&, ||) and bitwise (&, |, ^, >>, <<, ~) operators, as well as the unary negation (-) operator, all in one line.
  - You may use addition with the number 1 only in `changeSign`.
  - You may not use unary negation in `changeSign`.
  - You may not use bit shifting or the exclusive OR operator (^) in `xor`.

- You may use subtraction in `powerOf2` ONLY.

3. Complete all of the methods in the three java files, in the order that they appear above. Run the autograder and verifier (instructions below) frequently to avoid errors and to ensure that you are only using the allowed operations.

If you get stuck, a helpful file to check out is `Examples.java`. It has detailed explanations for how to complete similar methods.

3 How to run the auto-grader & verifier

1. Make sure that the `hw1checker.jar` file is in the same folder as your `BitVector.java`, `Bases.java`, and `Operations.java` files.
2. Navigate to this folder in your command line.
3. Run the desired command (see below).

3.1 Commands

1. Test all methods and verify that no banned operations are being used (checks all 3 files):

   ```
   java -jar hw1checker.jar
   ```

   Note: Your grade will be dependent on the output of the above command, as it will both test the output of your methods, and verify that you are not using banned operations. If you get stuck though, you can use some of the below commands to help you debug.

   On Windows and Mac, you can also double click the `hw1checker.jar` in your file explorer to test and verify all 3 files. The results will be placed in a new file called `gradeLog.txt`. Any errors with compilation, infinite loops, or other runtime errors will be placed in a new file called `errorLog.txt`.

2. Test & verify all methods in a single file. Useful for when you just want to look at one file at a time. For example, using `Bases.java`:

   ```
   java -jar hw1checker.jar -g Bases.java
   ```

3. Test all methods in a single file without running verifier. This means that this will only run the unit tests, and will not check for the use of banned operations. Useful for when you just want to try and get something that works. For example, using `Bases.java`:

   ```
   java -jar hw1checker.jar -t Bases.java
   ```

4. Verify all methods in a single file without running tests. For example, using `Bases.java`:

   ```
   java -jar hw1checker.jar -v Bases.java
   ```

5. Any combination of files can also be graded, tested, or verified at the same time. For example grading, `Bases.java` and `Operations.java` simultaneously:

   ```
   java -jar hw1checker.jar -g Bases.java Operations.java
   ```

3.2 Note for M1 Mac Users

When running the autograder, you may see some `UnsatisfiedLinkError`s above the autograder output with the message starting *"cannot open shared object file"*. Ignore them, they will not affect your score.

4 Rubric

The grade the autograder gives you should be the same as the grade you get (unless you intentionally hardcode just the solutions or try to hack the autograder).

5 Deliverables

1. Please upload the following 3 files to the "Homework 1: Bases and Bitwise Operations" assignment on Gradescope:
   (a) `BitVector.java`
   (b) `Bases.java`
   (c) `Operations.java`

6 Hints

6.1 Printing numbers in different bases

Remember that all numbers are stored in your computer as binary. When you perform operations such as `System.out.println()`, the computer does the translation into another base for you. All you need to do is tell the computer how you are representing your numbers, and how to interpret them.

For example, you can specify 16 in decimal, octal, or hexadecimal like so:

```
System.out.println(16);   // decimal (base 10), the default
System.out.println(020);  // octal (base 8), precede the number with a zero
System.out.println(0x10); // hexadecimal (base 16), precede the number with a "0x" (zero x)
```

You can also tell Java to print out your number in different bases using a method called `printf`. `printf` is the GRANDFATHER of all printing functions! When we get to C programming, you will be using it a lot. It is useful if you would like to write your own tester as well.

`printf` takes a variable number of arguments, the first of which is a format string. After the format string come the parameters. The formatting for the numbers is controlled by the format string.

6.1.1 Example

```
System.out.printf("In decimal: %d", 16);
System.out.printf("In octal: %o", 16);
System.out.printf("In hexadecimal: %x", 16);
```

The `%d`, `%o`, or `%x` get replaced by the parameter passed in. printf does not support printing the number out in binary.

For more information about `printf` read `http://en.wikipedia.org/wiki/Printf`.

6.2 Multiplication and division

You may find that there are times in which you need to use division or multiplication, but are not allowed to. Recall from lecture that you can use bitshifting to multiply or divide by powers of 2; this concept isn't found in the book, but is in the lecture slides.

7 Rules and Regulations

1. Please read the assignment in its entirety before asking questions.
2. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
3. If you find any problems with the assignment, please report them to the TA team. Announcements will be posted if the assignment changes.
4. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency please reach out to your instructor and the head TAs **IN ADVANCE** of the due date with documentation (i.e. note from the dean, doctor's note, etc).
5. You are responsible for ensuring that what you turned in is what you meant to turn in. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Gradescope. Email submissions will not be accepted.
6. See the syllabus for information regarding late submissions; any penalties associated with unexcused late submissions are non-negotiable.

7.1 Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work. Homework assignments will be examined using cheat detection programs to find evidence of unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student. If you supply a copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any platform which would allow other parties to it (public repositories, pastebin, etc). If you would like to use version control, use a private repository on github.gatech.edu

Homework collaboration is limited to high-level collaboration. Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment.

High-level collaboration means that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code, or providing other students any part of your code.

Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

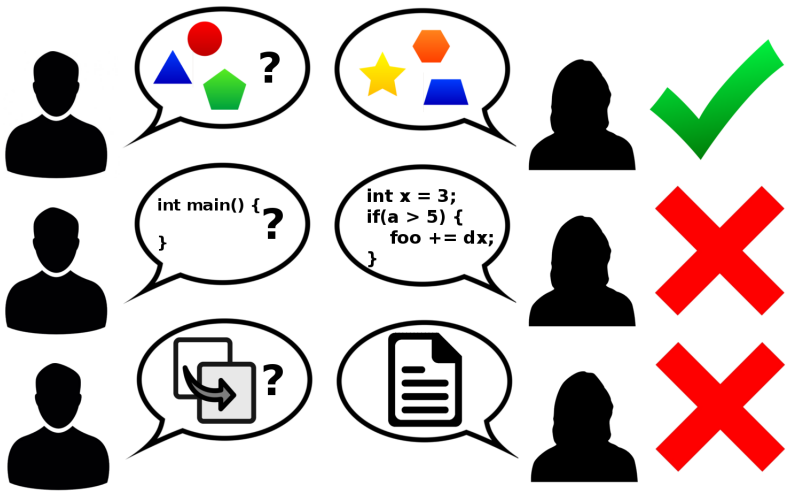

Figure 1: Collaboration rules, explained colorfully