Programming Exercise 02 – Quinary

Authors: Brian, Nicolas, Tomas, Ishuma, Sang Yoon
Topics: Branching, Switch, Iteration, Printf

Problem Description

Typically, we expect the numbers we use to be represented in the decimal numeral system, base-10. However, this is not the only way we can represent numbers. Computers will often store data in the binary numeral system, base-2. When tallying, people will often use the unary numeral system, base-1. For this assignment, we will be exploring the Quinary numeral system, base-5. To represent a decimal number, which we will call n, in Quinary, we can perform the following algorithm:

- Begin with an empty string to hold our answer
- Compute n modulo 5 and append it to the front of our answer
- Divide n by 5 (what kind of division should occur?)
- Repeat the above two steps until n is zero

To demonstrate, let's apply the algorithm to n = 97 and see what happens.

1. 97 % 5 is 2, so our partial answer is "2"
2. 97 / 5 is 19
3. 19 % 5 is 4, so our partial answer is "42"
4. 19 / 5 is 3
5. 3 % 5 is 3, so our partial answer is "342"
6. 3 / 5 is 0
7. We reached 0. The final answer is "342"

We can verify that the final answer is correct because

$$3 \cdot 5^2 + 4 \cdot 5^1 + 2 \cdot 5^0 = 75 + 20 + 2 = 97$$

Notice how this conversion algorithm repeats the same two steps over and over (mod and division). This means we can implement it with iteration a.k.a. loops!

Your assignment will consist of 2 main parts:

- Start with some arbitrary number in decimal and create the Quinary string representation
- Print out information about the Quinary representation

Solution Description

The Quinary Converter

1. Create a class called `Quinary`
2. Create the `main` method
3. Inside, create these variables:
   a. Create an `int` called `initialNum` and assign it any positive integer from 1-1000

b. Create a `String` called `answer` and assign it to the empty string, "". This will hold our Quinary representation as we build it
c. Create an `int` called `zeroCount` and assign it the value 0. This will count the number of zeroes in our answer string
d. Create an `int` called `oneCount` and assign it the value 0. This will count the number of ones in our answer string
e. Create an `int` called `twoCount` and assign it the value 0. This will count the number of twos in our answer string
f. Create an `int` called `threeCount` and assign it the value 0. This will count the number of threes in our answer string
g. Create an `int` called `fourCount` and assign it the value 0. This will count the number of fours in our answer string
h. Create an `int` called `currentNum` and assign it the value `initialNum`

4. Create a `while` loop that terminates when `currentNum` is equal to 0
5. Within the `while` loop, create a local `int` variable called `digit` and assign to it `currentNum` % 5
6. Use `if-else if` statements to do the following:
   a. If `digit` is a 0, increment `zeroCount`
   b. If `digit` is a 1, increment `oneCount`
   c. If `digit` is a 2, increment `twoCount`
   d. If `digit` is a 3, increment `threeCount`
   e. If `digit` is a 4, increment `fourCount`
7. Concatenate digit to the front of `answer`
8. Re-assign `currentNum` to `currentNum` / 5
9. After the while loop, use three `printf` statements to output the following on separate lines:
   a. `Decimal representation: <initialNum>`
   b. `Quinary representation: <answer>`
   c. `<zeroCount> zero(s), <oneCount> one(s), <twoCount> two(s), <threeCount> three(s), <fourCount> four(s)`
   - **Note: The < > are there to show that values of specific variables will be used. They should not be in the actual print statement. Do not include the bullet points either.**
10. Create an `int` variable `digitSum` and assign it to the sum of the digits in the Quinary representation
    a. One way of evaluating this sum is 0 * `zeroCount` + 1 * `oneCount` + 2 * `twoCount` + 3 * `threeCount` + 4 * `fourCount`
11. Using a `switch` statement, print one of the following in its own line according to the modulo value of `digitSum`:
    a. If `digitSum` % 5 is 0:
       i. `The Quinary digits sum to a multiple of 5!`
    b. If `digitSum` % 5 is 1:
       i. `The Quinary digits almost summed to a multiple 5!`
    c. If `digitSum` % 5 is 4:
       i. `So close!`
    d. If `digitSum` % 5 is any other value:
       i. `Nope!`

12. Finally, using a ternary expression (`?:`) and `printf`, print one of the following on its own line depending on whether zero is the most used digit:
    a. If `zeroCount` is larger than `oneCount`, `twoCount`, `threeCount`, and `fourCount`:
        i. `Zero is the most used digit.`
    b. Otherwise:
        i. `Zero is not the most used digit.`

HINT: To help debug your code, try inserting print statements in places where variables are changed.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Quinary.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

To prevent trivialization of the assignment, you may not import any classes for this assignment.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items.
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **<u>No inappropriate language is to be used</u>, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.