Assignment 2

CS 1027
Computer Science Fundamentals II

Due date: February 26 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

- Working with doubly linked data structures
- Different number systems such as binary and hexadecimal
- Working with exceptions
- Programming according to specifications

Introduction

Most of us are comfortable with the decimal number system that we've learned about as children and have been using ever since, but not as comfortable when it comes to different number systems such as binary, ternary, octal, and hexadecimal. Although these other number systems may seem daunting at first, they are actually not much different than the decimal system and it is not very difficult to convert numbers between different systems using simple algorithms. We just have to remember that each number system has a maximum digit – this is 9 is our decimal (base 10) system.

For example, the binary (base 2) number system, which is the foundational system for computers, has a maximum digit of 1, i.e. digits can either be 0 or 1. So 10110101 is a valid binary number but 10110102 is not a valid binary number as 2 is not a valid digit in that system.

The ternary (base 3) system has a maximum digit of 2, so 10110102 is a valid number in ternary (even though it is not in binary as we just stated).

Another popular number system in computer science is the hexadecimal (base 16) system. This one is different from the above ones in that it actually contains more digits than decimal. The term hexadecimal essentially means "six" and "ten" because it is base 16. Since our decimal system only has 10 distinct digits, the hexadecimal system requires 6 additional digits beyond the digits 0-9. We use letters A, B, …, F to represent 10, 11, …, 15, respectively.

In this assignment, we will be using doubly-linked data structures to store numbers of any number system. Each digit will be stored in separate node and connected to the next and previous nodes. Figure 1 shows the linked list representing the decimal number 47,152, and Figure 2 illustrates a LinkedNumber object representing this number (read description of class LinkedNumber below). Figures 3 and 4 illustrate LinkedNumber objects storing numbers in other number systems (binary and hexadecimal, respectively).

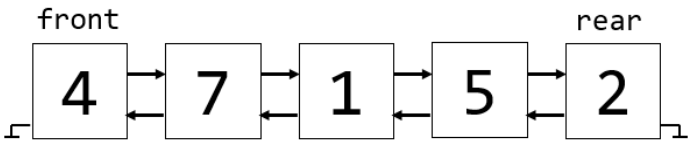


Figure 1. The doubly-linked list representing the number 47,152.

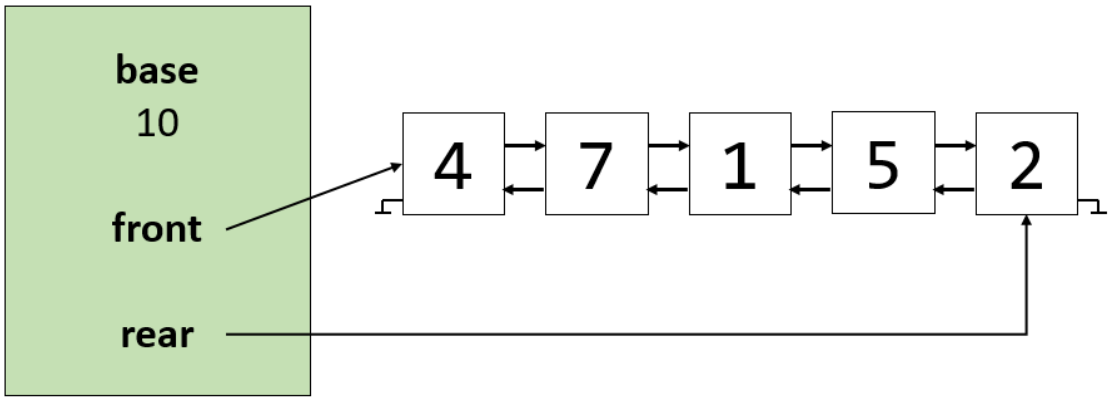


Figure 2. The LinkedNumber object representing the decimal number 47,152.

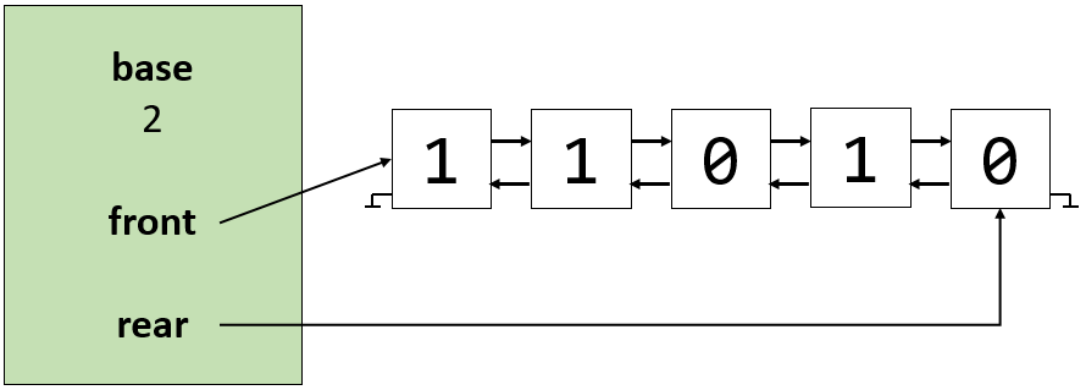


Figure 3. The LinkedNumber object representing the binary number 11010.

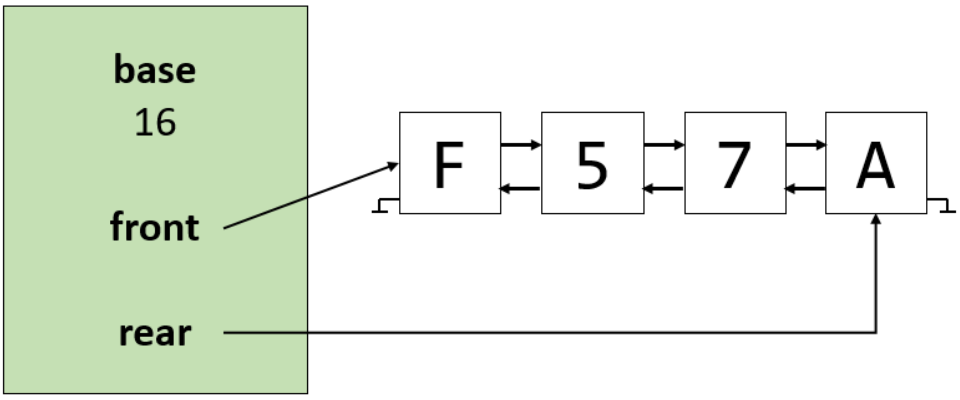


Figure 4. The LinkedNumber object representing the hexadecimal number F57A.

Assignment 2

CS 1027
Computer Science Fundamentals II

Provided files

DLNode.java is a provided file and this represents one of the nodes in the doubly-linked list you will create in the LinkedNumber class.

Digit.java is another provided file which represents one digit of a number. The nodes in the linked lists you create contain Digit objects (not integers). Since the numbers could be of a number system higher than base 10, Digit is used to represent digits that are shown as letters as well as those shown as decimal number digits. Look over this class to see its methods and how it was implemented.

TestLinkedNumber.java is a tester file to **help** check if your java classes are implemented correctly.

Similar tester files will be incorporated into Gradescope's auto-grader. Additional tests will be run that will be hidden from you. **Passing all the tests within the provided files does not necessarily mean that your code is correct in all cases.**

Classes to Implement

For this assignment, you must implement two Java classes: **LinkedNumberException** and **LinkedNumber**. Follow the guidelines for each one below.

In these classes, you may implement more private (helper) methods if you want. However, you may not implement more public methods **except** public static void main(String[] args) for testing purposes (this is allowed and encouraged).

You may **not** add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

You may **not** import any Java libraries such as java.util.Arrays or java.util.LinkedList.

LinkedNumberException.java

This class represents an exception that relates to an invalid condition or operation while working with the LinkedNumber objects. This class must inherit from the RuntimeException class.

The class must have no instance variables and must have only a constructor as described here (no other methods should be included):

- public LinkedNumberException (String *msg*): constructor
    - Call super() with the *msg* sent as a parameter to invoke the constructor of its parent class, RuntimeException.

Assignment 2

CS 1027
Computer Science Fundamentals II

LinkedNumber.java

This class is used to represent a positive whole number of any number system (from base 2 to base 16) stored in a doubly linked list such that each digit of the number is stored in a separate node as explained in the Introduction section of this document.

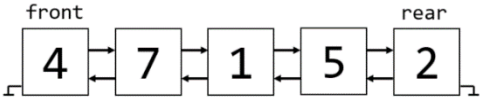
The class must have the following private instance variables:

- private int *base*
- private DLNode<Digit> *front*
- private DLNode<Digit> *rear*

The class must have the following public methods:

- public LinkedNumber (String *num*, int *baseNum*): constructor
  - Assign the value of *baseNum* to the instance variable *base*
  - If *num* is an empty string, i.e. no number is given, then throw a LinkedNumberException with the exact message: "no digits given"
  - Create a doubly linked list, by initializing and correctly connecting a sequence of DLNode objects, that represents the given String *num*. Each node in the linked list must store a Digit object (the Digit class is provided to you) which represents a single digit of the overall number. Hint: You may want to use num.toCharArray() to convert the String *num* into an array of characters **or** you could use charAt(index) to get individual characters from the String. Then initialize a Digit object for each character.
  - The first node of the linked list (storing the first digit of the number) must be referenced by *front* and the last node must be referenced by *rear*.
- public LinkedNumber(int *num*): constructor
  - see description for the first constructor; this one is almost the same
  - In this case, assign the value 10 to instance variable *base*, because this constructor is for numbers that are in the decimal number system.
  - Hint: Convert the int *num* to a String using String.valueOf(*num*) and then you can follow the same hint given above to obtain the individual characters of the String.
- public boolean isValidNumber()
  - Determine whether or not the number stored in this linked list is a valid positive number for the *base* number system specified by instance variable *base*
  - To be valid, every digit must be positive and must be one of the possible digits of the *base* number system. For example, in binary (base 2), only 0 and 1 are valid.
  - Return true if the number is valid, or false otherwise.
- public int getBase()
  - Return the value of *base*.
- public DLNode<Digit> getFront()
  - Return the *front* node.
- public DLNode<Digit> getRear()
  - Return the *rear* node.
- public int getNumDigits()

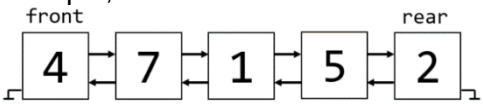
  - Compute and return the number of digits (the number of nodes) in the number
- public String toString()
  - Create and return a String containing all the digits of the number with no spaces or other characters.
  - For example, if the linked list is as follows:



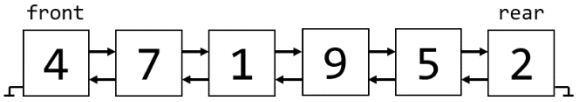
    Then the toString() method must return the String (not int): "47152".

- public boolean equals (LinkedNumber *other*)
  - Compare *this* LinkedNumber object and *other* to see if they are considered equivalent. To be considered equivalent, the *base* (number system) of both objects has to be equal **and** the values stored in the nodes in both objects must be the same in the same order.
  - If either linked list contains more nodes than the other, it has to return false, even if the additional node(s) do not affect the overall value of the number. For example, a linked list containing "27" is not considered equal to a linked list containing "027" even though the two values are the same.
- public LinkedNumber convert (int *newBase*)
  - First check if *this* LinkedNumber object represents a valid number. If it's invalid, throw a LinkedNumberException with the message "cannot convert invalid number".
  - If it is valid, then convert the number stored in *this* LinkedNumber object into its equivalent value in the *newBase* number system
  - Create and return a **new** LinkedNumber object that represents the new, converted number
  - See the **Conversions** section below for explanations and examples of converting between different number systems.
  - Hint: it is recommended that you create private helper methods to help with these conversions
- public void addDigit (Digit *digit*, int *position*)
  - First check if the *position* is valid, i.e. between 0 and n+1 (where n+1 represents the number of nodes in the list), inclusive. If it's invalid, throw a LinkedNumberException with the message "invalid position".
  - If it is valid, create a new DLNode storing the given *digit* object and add it into the doubly-linked list at the given *position*, where the *rear* of the list is position 0 and the *front* is at position n.
    - If the value of *position* is 0, add the new digit after *rear*
    - If the value of *position* is n+1, add the new digit before *front*
    - If the value of *position* is between 0 and n+1, add the new digit at the corresponding place within the linked list.
  - Remember to consider all the cases and ensure the list is connected and referenced properly after the insertion of this new node.

- For example, if the initial linked list is as follows:



And then we call addDigit(new Digit('9'), 2), the resulting linked list should be:



- public int removeDigit(int *position*)
  - First check if the *position* is valid, i.e. between 0 and n (where n+1 represents the number of nodes in the list), inclusive. If it's invalid, throw a LinkedNumberException with the message "invalid position".
  - If it is valid, traverse from the *rear* toward the *front* of the list to the node at index *position*. For example, if position is 0, stay at the *rear*. If position is 2, traverse to the node that is 2 nodes to the left of the *rear* node. Obtain the value of that node as part of the whole number (not just the value stored in the node, but the total value it represents within the whole number). For example, in the decimal number 7492, if we call removeDigit(2), it must return the value 400. If we call removeDigit(3), it must return the value 7,000.
  - Remove this node from the linked list and ensure all remaining nodes are connected properly after this removal.
  - Note that the value being returned will be the decimal equivalent of the value regardless of the number system. For example, if we have binary number 1010 and call removeDigit(1), the resulting linked list will represent binary number 100 and the decimal number 2 will be returned (not the binary number 10).

Conversions

Consider 3 cases for converting a number from one number system to another number system:

- From decimal to a non-decimal number system
- From a non-decimal number system to decimal
- From a non-decimal number system to another non-decimal system

Case 1: Decimal to non-decimal

1. Set variable val equal to the whole decimal number
   a. Loop through the linked list from rear to front

   b. Keep track of the position of each node within the overall number, beginning with index 0 as the rear node, index 1 as the node prior to rear, and so on, up to index n for the front node (the overall number has n+1 digits in it)
   c. For each node, multiply its value by (10 raised to the exponent position) based on the explanation of position above
   d. Sum all these products and store the sum in val
2. Convert to newBase using divisions and remainders to determine the new number
   a. Repeatedly divide val by newBase until val is reduced to 0
   b. The remainder from each division gives us the value of the next (going from right to left) digit in the converted number
   c. Any numeric values can be converted to Strings using String.valueOf(int) and any values that are 10-15 must be assigned the corresponding letter (A-F).
   d. Concatenate these digit values into a String and return the String
3. Initialize a new LinkedNumber object with this completed String and the newBase as its parameters

Case 2: Non-decimal to decimal

1. Set variable val equal to the whole number
   a. Loop through the linked list from rear to front
   b. Keep track of the position of each node within the overall number, beginning with index 0 as the rear node, index 1 as the node prior to rear, and so on, up to index n for the front node (the overall number has n+1 digits in it)
   c. For each node, multiply its value by (base raised to the exponent position) based on the explanation of position above
   d. Sum all these products and store the sum in val
2. Initialize a new LinkedNumber object with this variable val and the number 10 as its parameters

Case 3: Non-decimal to non-decimal

1. Use the steps in Case 2 above to convert the current non-decimal number into a decimal LinkedNumber object called temp
2. Use the steps in Case 1 above to convert temp into a new LinkedNumber object in the newBase number system

These algorithms require you to "keep track of the position of each node" as you traverse the linked list from rear to the front. Here is an illustration of what the positions would be for each of the nodes in a linked list representing the decimal number 1,529:

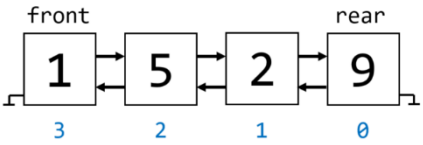
Important: remember these positions are **not** indices we can use to access nodes. This is a linked list, **not** an array, so we must use getPrev() or getNext() to go from one node to the next.

Assignment 2

CS 1027
Computer Science Fundamentals II

Examples of Conversions

Decimal number 1,529 converted to hexadecimal



1. Set val = 1,529
    a. Node at position 0 has value 9 → $9 \times 10^0 = 9$
    b. Node at position 1 has value 2 → $2 \times 10^1 = 20$
    c. Node at position 2 has value 5 → $5 \times 10^2 = 500$
    d. Node at position 3 has value 1 → $1 \times 10^3 = 1{,}000$
    e. Sum these results to get 1,529
2. Divide by 16 repeatedly and use remainders
    a. 1,529 / 16 = 95 with remainder 9
    b. 95 / 16 = 5 with remainder 15 (which is denoted by F)
    c. 5 / 16 = 0 with remainder 5
    d. The remainders of these calculations, in reverse order (bottom to top), is the hexadecimal number that is equivalent to decimal 1,529. This number is: **5F9**.

Binary number 110101 converted to decimal

1. Determine the position of each node and calculate $2^{position} \times value$
    a. Node at position 0 has value 1 → $1 \times 2^0 = 1$
    b. Node at position 1 has value 0 → $0 \times 2^1 = 0$
    c. Node at position 2 has value 1 → $1 \times 2^2 = 4$
    d. Node at position 3 has value 0 → $0 \times 2^3 = 0$
    e. Node at position 4 has value 1 → $1 \times 2^4 = 16$
    f. Node at position 5 has value 1 → $1 \times 2^5 = 32$
2. Sum these products together to get the decimal value: **53**.

Hexadecimal number 3F7B converted to octal (base 8)

1. Use the Case 2 steps to convert the hexadecimal number 3F7B to decimal
    a. Node at position 0 has value B (11) → $11 \times 16^0 = 11$
    b. Node at position 1 has value 7 → $7 \times 16^1 = 112$
    c. Node at position 2 has value F (15) → $15 \times 16^2 = 3{,}840$
    d. Node at position 3 has value 3 → $3 \times 16^3 = 12{,}288$
    e. Sum these products together to get the decimal value: **16,251**.

2. Use the Case 1 steps to convert the decimal number 16,251 to octal (base 8), dividing by 8 repeatedly and using the remainders as the digits in the octal number.
    a. 16,251 / 8 = 2,031 with remainder 3
    b. 2,031 / 8 = 253 with remainder 7
    c. 253 / 8 = 31 with remainder 5
    d. 31 / 8 = 3 with remainder 7
    e. 3 / 8 = 0 with remainder 3
    f. The remainders of these calculations, in reverse order (bottom to top), is the octal number that is equivalent to decimal 16,251 (and hexadecimal 3F7B). This number is: **37 573**.

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produce compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty.
- Including a "package" line at the top of a file will receive a penalty.

Remember **you must do** all the work on your own. **Do not copy** or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Monday, February 26 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Assignment 2

CS 1027
Computer Science Fundamentals II

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own tester class to do this) to ensure that your code works for a variety of cases. We are providing you with some tests but we may use additional tests that you haven't seen for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code as many times as you wish, however, re-submissions after the assignment deadline will receive a late penalty.

Files to submit

- LinkedNumberException.java
- LinkedNumber.java

Grading Criteria

Total Marks: [20]

Functional Specifications:

[1] LinkedNumberException.java

[3] LinkedNumber.java

[13] Passing Tests (some additional, hidden tests will be run on Gradescope)

Non-Functional Specifications:

[1] Meaningful variable names, private instance variables

[1] Code readability and indentation

[1] Code comments