CSc 345 — Analysis of Discrete Structures
Fall 2024 (McCann)

http://cs.arizona.edu/classes/cs345/fall24/

Program #1: The VIC (VIC InComplete) Cipher

Due Date: September 19th, 2024, at the beginning of class

Prelude:

The guy wearing the high fade, loose Pompadour haircut at the bar had to be her contact. But she had to be certain. Besides, the protocol could be amusing.

"Excuse me . . . "

He looked up from his phone. His skin was flawless. "Yes?"

"Your phone . . . is that the latest Etaoin model?"

"Etaoin? No, this is a Shrdlu."

"Would you show me how to . . . turn it on?" She couldn't resist the embellishment.

Apparently, neither could he. "It was not designed to respond to *your* touch. However, I have something in my car that you might be able to handle."

"Lead the way."

"Are you familiar with the VIC cipher?"

She was mildly insulted, and annoyed that he had ignored all of her attempts at small–talk on the walk to the underground garage. "Reino Häyhänen's hand cipher that we couldn't break until he defected in '57? From the Nihilist cipher family? Discovered by a Brooklyn paperboy in a fake nickel? That one?"

"We have simplified it for you."

"I'm flattered. But you know . . . "

". . . that you will not use it? I have been briefed on your unusual tradecraft, Agent 85721."

85721's familiarity with the original cipher allowed the contact to explain the simplified version in just a few minutes. "Please repeat it for me."

She sighed. She needed to find out what else they were they telling the new contacts about her. She summarized it for him in one sentence.

He didn't even blink. "Good. The date you are to use is the date the agency was established – 470918. Do you remember your Analysis of Discrete Structures class in college?"

Remember 345? All that documentation? She couldn't forget if she tried. "Let me guess: The phrase is, 'Choose the representation that best supports the operations.' "

"Naturally. For the anagram, use 'HEAT IS ON'. Any questions?"

Time was up, but she had to get *some* kind of reaction out of this guy. It was a matter of pride. “Speaking of heat . . . care to accompany me back to my suite?”

Fezzik would have been proud, but he didn’t even smile. Instead, He passed her a wrapped fortune cookie. “Decode this and you may understand why I must decline.”

She extracted the fortune on the way back to the bar. “Thou shalt not kill -9.” UNIX humor; hilarious. The lucky numbers on the back were more helpful: 62072357857210178. She had it decoded by the time she finished her Manhattan.

“So it’s not so much what they’re *telling* the new contacts . . . ”

Background:

The original VIC cipher is remarkably complex for a hand cipher (one that was designed to be used with just paper and pencil), more complex than I think is necessary for this assignment. I’ve pruned it back to eight steps involving three algorithms and one interesting data structure, enough to retain the feel of the full cipher without being too challenging to implement. The full version is perhaps twice as complex. Keep this in mind as you imagine being a agent in the field who was relying on this cipher for instructions.

Supporting Operations: Before covering the reduced cipher, we will cover the operations necessary to execute the cipher. You will have to create Java methods that perform each of these operations. The method signatures and return types are given with the descriptions.

(a) *No-Carry Addition*: `[ public static String noCarryAddition (String, String) ]`

- When we add 7 and 18 together, we get 25. If you do this by hand, you add 7 and 8 to get 15, write down the 5, and carry the 1 to the next column of digits.
- In no-carry addition (we’ll use $\dotplus$ as the operator symbol), we simply discard the carry. That is, $7 \dotplus 18 = 15$ Another example: $359 + 163$ normally equals 522, but in no-carry addition $359 \dotplus 163 = 412$.
- We are using strings so that we can have values with leading zeroes.

(b) *Chain Addition*: `[ public static String chainAddition (String, int) ]`

- Chain addition lengthens a number by appending digits derived from the number’s original digits.
- Consider 64. By doing a no-carry addition of the first two digits (6 and 4), we create the digit 0, which we append to 64 to form 640. Doing the same with the second and third digits produces 4, with the third and fourth produces 4, etc. We continue this until the resulting number has as many digits as we need it to have (the second argument).
- Another example: Using chain addition to extend 762 to eight digits produces the value 76238513.
- If the number has just one digit, assume zero as the preceding digit to get the process started. Thus, `chainAddition(7,5)` produces 77415. Don’t worry about a second argument of zero, and if the second argument is less than the length of the first, return a prefix of that length. That is, `chainAddition(54321,2)` returns 54.

(c) *Digit Permutation*: `[ public static String digitPermutation (String) ]`

- Given a string, `digitPermutation` returns a permutation of the digits 0-9 based on the first 10 characters of the string. If the string is shorter than 10, return null.
- Consider the string BANANALAND. The earliest letter of the alphabet in BANANALAND is 'A', and there are four of them. In left to right order, assign them the digits 0, 1, 2, and 3. The next letter used is 'B'; assign it the next digit (4). The next is 'D'; it gets 5. Continue until all digits are assigned. This creates the permutation "4071826395". The following snapshot sequence might help:

```
B A N A N A L A N D      B A N A N A L A N D      B A N A N A L A N D
  0   1   2   3      ⇒   4 0   1   2   3   5  ⇒   4 0 7 1 8 2 6 3 9 5  ⇒ 4071826395
```

- Here's another way to look at it. If we imagine sorting the string's content, we'd get AAAAB-DLNNN. Now pair up the letters with the digits 0-9: The A's are matched with 0, 1, 2, and 3; B is matched with 4; D with 5; etc. Now imagine 'unsorting' the letters and dragging their assigned digits along with them.
- Check your understanding with this example: The permutation 6704821539 is generated from the string STARTEARLY.

(d) *The Straddling Checkerboard*: `[ public static ArrayList<String> straddlingCheckerboard (String, String) ]`

- A 'straddling checkerboard' is a data structure[1] used to map letters to integers, and appears in ciphers other than VIC. To create it, we pair up a permutation of the digits 0-9 (1st argument) with the 10 characters of an anagram (2nd argument of eight distinct letters and two spaces); return null if either argument has improper content. For example, using the permutation 4071826395 from above and the anagram "a tin shoe", the pairing would look like this:

4	0	7	1	8	2	6	3	9	5
A		T	I	N		S	H	O	E

- There are two digits (0 and 2 in this example) matched to spaces. They are used as labels for the two rows below the anagram. Those rows contain the rest of the upper-case letters of the alphabet, in order. There will be two unused spaces at the end of the last row:

	4	0	7	1	8	2	6	3	9	5
	A		T	I	N		S	H	O	E
0	B	C	D	F	G	J	K	L	M	P
2	Q	R	U	V	W	X	Y	Z		

- Using this straddling checkerboard, we can assign numbers to the letters as follows: For a given letter, concatenate the digit labeling the row (if the row has one) with the digit labeling the column. Thus, 'F' is assigned 01, 'Z' is 23, and 'S' is just 6.
- This 2D structure is good for doing VIC by hand, but isn't as convenient for programming. Instead, `straddlingCheckerboard` will return an `ArrayList` of `String`, with the letters' assignments arranged in alphabetical order. That is, this is what would be returned for the above checkerboard:

(A)	(B)	(C)	(D)	(E)	(F)	(G)		(X)	(Y)	(Z)
0	1	2	3	4	5	6	...	23	24	25
"4"	"04"	"00"	"07"	"5"	"01"	"08"	...	"22"	"26"	"23"

- Notice that we have to return strings; if we returned a numeric type, such as `short`, we'd lose any leading zeros. If there's any problem with the input, return null.

[1] And would also be a great band name!

The 'VIC InComplete' Encoding Algorithm: To encode a message with 'VIC,' we need five pieces of information:

(i) A 5–digit agent ID (e.g., 85721 from the short story)

(ii) A numeric date in the form YYMMDD (e.g., 470918 for Sept 18, 1947)

(iii) A phrase from which the first 10 letters are used, in upper-case (e.g., 'Choose the representation . . . ' gives CHOOSETHER)

(iv) A 10–character anagram of eight unique commonly–used English letters (converted to upper-case if necessary) and two spaces (e.g., HEAT IS ON) — using popular letters here usually results in a shorter encoded message.

(v) The message to be encoded, again converted to upper–case only (e.g., Run Away! $\longrightarrow$ RUNAWAY)

Here are the eight steps of our reduced version of VIC. They need to be performed in this order, as the later steps build on the earlier ones. As we present the steps, we'll also perform them as a running example (ha!) to encode the RUNAWAY message.

1. <u>Add the ID to the first five digits of the date, using no–carry addition.</u>
   - See the *No-Carry Addition* operation description.
   - **Running Example:** 85721 $\dotplus$ 47091 = 22712

2. <u>Expand the 5-digit result of (1) to 10 digits, using chain addition.</u>
   - See the *Chain Addition* operation description.
   - **Running Example:** 22712 $\Rightarrow$ 2271249836

3. <u>Use the phrase to create a digit permutation.</u>
   - See the *Digit Permutation* operation description.
   - **Running Example:** $\begin{array}{cccccccccc} \text{C}&\text{H}&\text{O}&\text{O}&\text{S}&\text{E}&\text{T}&\text{H}&\text{E}&\text{R}\\ 0&&&&&1&&&2& \end{array} \Rightarrow \begin{array}{cccccccccc} \text{C}&\text{H}&\text{O}&\text{O}&\text{S}&\text{E}&\text{T}&\text{H}&\text{E}&\text{R}\\ 0&3&5&6&&1&&4&2& \end{array} \Rightarrow$ 0356819427

4. <u>Add the results of (2) and (3) using no–carry addition.</u>
   - Again, see the *No–Carry Addition* operation description.
   - **Running Example:** 2271249836 $\dotplus$ 0356819427 = 2527058253

5. <u>Use the result from (4) to create a digit permutation.</u>
   - Unlike step (3), this time we're using a sequence of digits instead of a sequence of letters as the basis for the permutation. This doesn't change anything; digits are characters in ASCII, too.
   - **Running Example:** $\begin{array}{cccccccccc} 2&5&2&7&0&5&8&2&5&3\\ 1&&2&&0&&&3&& \end{array} \Rightarrow \begin{array}{cccccccccc} 2&5&2&7&0&5&8&2&5&3\\ 1&5&2&&0&6&&3&7&4 \end{array} \Rightarrow$ 1528069374

6. <u>Build a straddling checkerboard from (5)'s result and the anagram.</u>
   - See the *No–Carry Addition* operation description, and remember that the funky 2D table isn't what your `straddlingCheckerboard()` method will be returning. I'm just using that representation here because it's compact.
   - **Running Example:** $\begin{array}{ccccccccccc} &1&5&2&8&0&6&9&3&7&4\\ &\text{H}&\text{E}&\text{A}&\text{T}&&\text{I}&\text{S}&&\text{O}&\text{N}\\ 0&\text{B}&\text{C}&\text{D}&\text{F}&\text{G}&\text{J}&\text{K}&\text{L}&\text{M}&\text{P}\\ 3&\text{Q}&\text{R}&\text{U}&\text{V}&\text{W}&\text{X}&\text{Y}&\text{Z}&& \end{array}$

7. <u>Use the straddling checkerboard to encode the message.</u>

   - **Running Example:** $\begin{array}{ccccccc} \text{R} & \text{U} & \text{N} & \text{A} & \text{W} & \text{A} & \text{Y} \\ 35 & 32 & 4 & 2 & 30 & 2 & 39 \end{array} \Rightarrow 35324230239$

8. <u>Insert the ID into the message</u>

   - If you're running a spy agency, you have more than just one agent in the field. When you receive a message, how do you know which agent sent it? You need to know the agent ID to decode the message. The solution is to bury the ID inside of the encoded message. But where? We used the first five digits of the date in step (1); we use the last digit here: The ID is inserted after the n-th digit of the encoded message, where n is the last digit of the date, and the first digit of the encoded message is digit #1. If the encoding doesn't have enough positions, we'll just append the ID to the encoding.
   - **Running Example:** 47091**8** ⇒ **35324230**$\boxed{85721}$239 ⇒ 3532423085721239, and we're done.

Self-Test #1: You are agent 73003. Using a date of May 8, 1945, the phrase "Don't do drugs," and the anagram A ROSE TIN, encode the message "Send money." The solution is given at the end of this document.

The 'VIC InComplete' Decoding Algorithm: To decode a message the was encoded with 'VIC,' we need four pieces of information:

(i) A numeric date in the form YYMMDD

(ii) A phrase from which the first 10 letters are used (converted to upper-case when necessary)

(iii) A 10-character anagram of eight unique letters and two spaces (again, converted to upper-case).

(iv) The message to be decoded (all digits).

Now that we know how to encode, the decoding algorithm is almost trivial. Assume that the date, phrase, and anagram are the same as in the encoding running example, above, and that the encoded message is 815052098572156920365.

1. <u>Extract the ID from the message.</u>

   - This is the reverse of step 8 from the encoding algorithm.
   - **Running Example:** The message is 815052098572156920365 and the last digit of the date is 8. Skipping the first eight digits and taking the next five gives us the agent ID (85721). Pulling them out leaves just the encoded message, 8150520956920365.

2. <u>Perform steps 1 - 6 from the encoding algorithm.</u>

   - See above! The end result should be the same straddling checkerboard, because we're using the same starting information.

3. <u>Match the message digits to letters using the straddling checkerboard.</u>

   - In our straddling checkerboard, the digits 0 and 3 matched to the spaces in the anagram, and thus became prefixes for the letters in the last two rows. This makes parsing the message easy: The next digit represents a letter by itself, unless it's a 0 or a 3, when the next two digits are used.
   - **Running Example:** 8150520956920365 ⇒ 8 1 5 05 2 09 5 6 9 2 03 6 5 ⇒ THECAKEISALIE

Parsing the letters into words isn't part of 'VIC' (or of VIC); the spies had to do that themselves.

Self-Test #2: Because the running example uses the same info as does the prelude story, you should be able to decode Agent 85721's fortune cookie message. Try it! The solution is given at the end of this document, but, really, try it before you look.

Assignment: Write **two** complete, well–documented, and suitably object–oriented Java 16 programs named `EncryptVIC.java` and `DecryptVIC.java` that encrypt and decrypt, respectively, messages using the 'VIC InComplete' algorithm described above. Details of their inputs and outputs are provided in the next two sections.

You are also to create and submit a file named `VICOperations.java` whose class contains the four VIC operation methods detailed earlier: `noCarryAddition()`, `chainAddition()`, `digitPermutation()`, and `straddlingCheckerboard()`. Both `EncryptVIC.java` and `DecryptVIC.java` should access these methods from the `VICOperations` class.

You may create additional methods and even additional classes if you wish. Just be sure to submit all of them!

Input: Both programs accept a command–line argument that is the file name (optionally prefixed with the path to the file) of the file containing the data required to execute the algorithm. For example:

```
$ java EncryptVIC instructions.enc
$ java DecryptVIC ../data/secretmessage.dec
```

(I recommend using those file extensions to help you remember which of your testing files are for encryption and which are for decryption.)

`EncryptVIC.java` expects data files of five lines, one piece of algorithm data per line. The data is not case–sensitive. For example, here's the data file content for the running example of this handout:

```
85721
470918
Choose the representation that best supports the operations
heat is on
Run away!
```

`DecryptVIC.java` expects just four lines, also one piece of data per line, also not case–sensitive. This file content corresponds to the second self–test:

```
470918
choose THE representation THAT best SUPPORTS the OPERATIONS
hEaT iS oN
91592357857210178
```

Because this assignment is challenging enough without having to worry a lot about reading the data from the files, we are supplying the `readVICdata()` method for `EncryptVIC`. The method source code is linked to the class web page under the link to this assignment handout. It should be fairly easy to create the corresponding method for `DecryptVIC`. We expect that your version for `DecryptVIC` will also perform similar input sanity–checking of the file data. That is, when invalid input is encountered, your programs are to exit with a non–zero code value, as demonstrated in `readVICdata()`. You should expect us to test your error–handling, thus, so should you.

Output: The output of `EncryptVIC.java` is to be a single line with the encrypted message (the sequence of digits), just like this:

```
$ java EncryptVIC filename.enc
123456789012345
```

Similarly, the output of `DecryptVIC.java` is also a single line containing the decrypted message:

```
$ java DecryptVIC filename.dec
THATSALLFOLKS
```

As shown, you should not attempt to add spaces to decrypted messages. Just leave them without punctuation.

Hand In: On the due date, submit your well-documented program files using the `turnin` facility on lectura. The submission folder is `cs345p1`. (Need help with turnin? Instructions are available from the class web page. For that matter, the documentation we expect to see is summarized in the Programming Style handout you received, and documentation examples can be found from the class web page.) Name your source files `EncryptVIC.java`, `DecryptVIC.java`, and `VICOperations.java`, as shown above, so that we don't have to guess which files to compile. I recommend that you test–submit an early version of your assignment well in advance of the due date, just to be sure that you have figured out how to use `turnin` successfully.

Want to Learn More?

- The real VIC cipher was designed for the Cyrillic alphabet. These sites describe English versions:
  - `http://www.quadibloc.com/crypto/pp1324.htm`
  - `http://everything2.com/title/VIC+cipher`

  Changes made for this assignment include replacing no–carry subtraction with no–carry addition in the first step, cutting the phrase in half, and eliminating a lot of steps. The result isn't a very secure cipher, so don't use this algorithm to try to hide anything from the NSA.

- What's special about the date of 470918? The National Security Act of 1947 authorized the creation of the Central Intelligence Agency (CIA) on September 18 of that year.
  - `http://en.wikipedia.org/wiki/National_Security_Act_of_1947`

- 'ETAOIN SHRDLU' is a meaningless phrase that lists many of the most popular letters in English prose in order by decreasing frequency of use. Keys of linotype typesetting machines were ordered by that sequence. 'HEAT IS ON' is an anagram of the first eight of those letters.

Other Requirements and Hints:

- **Start early!** There are LOTS of little things that need to be done to complete this assignment. You will almost certainly not be able to complete all of them if you wait until the night before the due date to get started.

- Make sure that you understand our 'VIC InComplete' algorithm before you start writing the program; you can't write a program to solve a problem you don't understand.

- Don't try to write the whole assignment at once; start small. For example, consider coding (and testing!) `noCarryAddition()` first; it's fairly straight–forward. Then move on.

- Plan on checking your program's work by hand. Create your own data and messages for encryption and decryption, and see if your programs get the same results that you do.

- Feel free to exchange data and encrypted messages (but never assignment code!) with your classmates. Why? If you only test with your own encryption and decryption routines, any logical errors within your VIC operation implementations are likely to appear in both programs. Without independent verification, it may appear that your logically–flawed code is correct.

Answer to Self-Test #1: (1) $73003 \dotplus 45050 = 18053$ (2) $18053 \Rightarrow 1805398582$
(3) DONTDODRUG $\Rightarrow$ 0548162793 (4) $1805398582 \dotplus 0548162793 = 1343450275$
(5) $1343450275 \Rightarrow 1354670298$

(6)

	1	3	5	4	6	7	0	2	9	8
	A		R	O	S	E		T	I	N
3	B	C	D	F	G	H	J	K	L	M
0	P	Q	U	V	W	X	Y	Z		

(7) SENDMONEY $\Rightarrow$ 678353848700 (8) 67835384730038700

Answer to Self-Test #2: 62072357857210178 $\Rightarrow$ 62072357 $\boxed{85721}$ 0178 $\Rightarrow$ 620723570178 $\Rightarrow$ 6 2 07 2 35 7 01 7 8 $\Rightarrow$ JBNBSPCPU (Couldn't resist peeking, could you? I've shifted the letters to hide the real message. Replace each letter with the previous letter of the alphabet to get the true message.)