MEF UNIVERSITY

Computer Engineering

MATH 321 – AUTOMATA THEORY AND FORMAL LANGUAGES

FALL 2020

CODING ASSIGNMENT

Efe Ertekin - 041701006 - ertekinef@mef.edu.tr

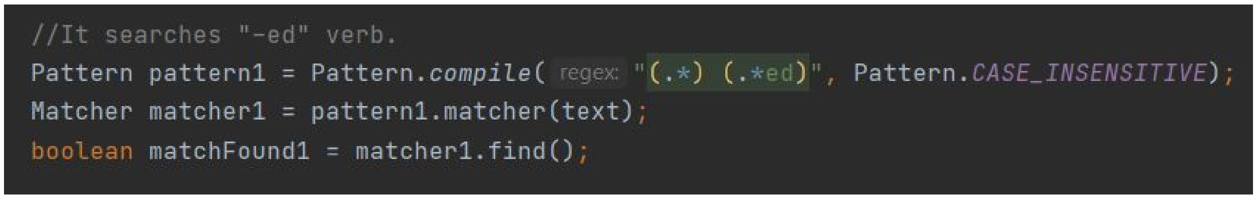
Şafak Barış - 041701007 - barissa@mef.edu.tr

Date : 17.01.2021

EXPLANATION OF PROJECT

In this project, firstly we created files we needed. They are Intermediate.txt and Decrypted.txt. We fill the Intermediate.txt with the given input file Encrypted.txt's lines filtered by alterations. After all alterations applied to the sentences inside of the Encrypted.txt, we insert them into Intermediate.txt. After that insertion, we use Intermediate.txt to write something inside to the Decrypted.txt. We get all lines from Intermediate.txt and apply them to our CFG function. CFG function returns true or false. If a given sentence returns true, then that sentence will be written in the Decrypted.txt.

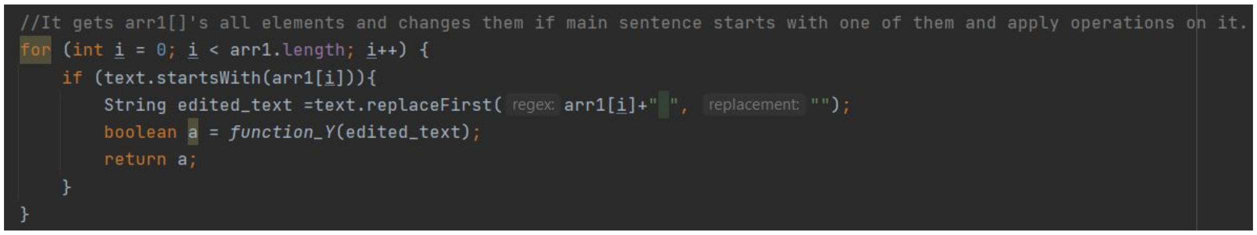
Alterations created with regex and its library. Pattern and Matcher mostly used to make sure our operation works. As it is seen in Figure 1.



```
//It searches "-ed" verb.
Pattern pattern1 = Pattern.compile( regex: "(.*) (.*ed)", Pattern.CASE_INSENSITIVE);
Matcher matcher1 = pattern1.matcher(text);
boolean matchFound1 = matcher1.find();
```

Figure #1: Example for Pattern and Matcher usage

For CFGs we have created functions that call each other to build different sentences. It can cause an infinite number of loops but we had little touches to stop that recursive looping problem. We added IFs and FORs to control that problem. As it is seen in Figure 2.



```
//It gets arr1[]'s all elements and changes them if main sentence starts with one of them and apply operations on it.
for (int i = 0; i < arr1.length; i++) {
    if (text.startsWith(arr1[i])){
        String edited_text =text.replaceFirst( regex: arr1[i]+" ",  replacement: "");
        boolean a = function_Y(edited_text);
        return a;
    }
}
```

Figure #2: Example for restriction to control loops

ALTERATIONS:

Alteration 1 : This alteration changes am-pm and decreases clock time "-1".
For example: 10 pm --- 9 am.

Alteration 2 : This alteration changes “There are” to “This is” and also deletes word’s plural “s”.
This alteration changes “This is” to “These are” and also adds word’s plural “s”. And it changes the same way to change past tense like **:** This alteration changes “There were” to “This was” and also deletes word’s plural “s”.

For example :	There are cars.	------	This is car.
	This was car.	------	These were cars.

Alteration 3: In this alteration change “True” to “true” and “true” to “false” . This alteration gives confusion to who reads these sentences.

For example:	This is True.	-----	This is true.
	This is true.	-----	This is false.

Alteration 4: It searches possessive words at the beginning of the sentences. And enters a loop for changes for them. “Your” changes with “My” , “My” changes with “Your”, “Her” changes with “His” , “His” changes with “Her”

For example:	Your phone is the best.	-----	My phone is the best.
	Her phone is the best.	-----	His phone is the best.

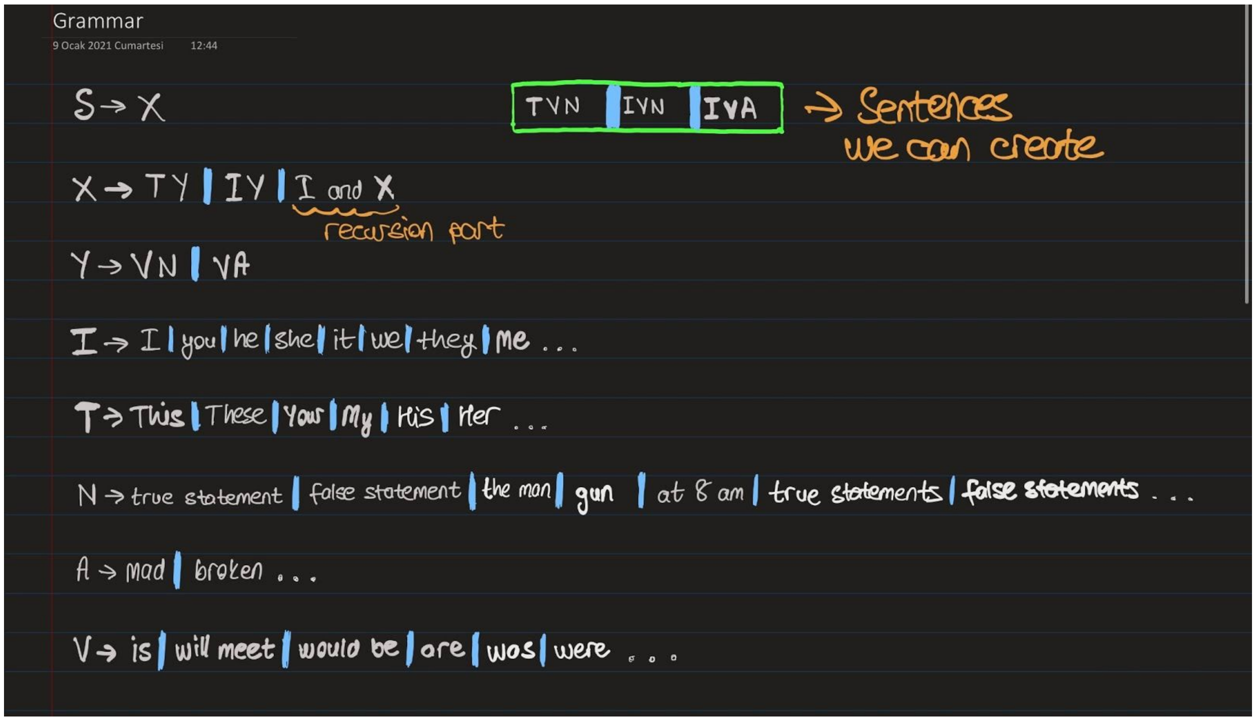
Alteration 5: In this alteration changes "-ed" to "do"; "do" to "-ing"; "-ing" to "-ed".

For example:	I do run.	-----	I am running.
	I passed the exam.	-----	I do pass the exam.
	I am killing the man.	-----	I killed the man.

Alteration 6: If alteration sees “Yes” after the question sentence: “Were You mad? Yes.” it changes like “You would be mad.” and If alteration sees “No” after the question sentence: “ Were you mad? No.” it changes like “You were mad.”

CFGs:

In this part of the project, we wanted to focus on creating grammar that can build a sentence as well as a recursive sentence. To make it easier to handle, we designed this grammar according to the Chomsky Normal Form. So, we can easily follow our splitted sentences. For instance, when we initiate X, we should split our sentence. One side of that sentence should go for T and the other one should go to Y. With the help of Chomsky, we could create this splitting easier.



Grammar

9 Ocak 2021 Cumartesi 12:44

S → X $\boxed{\text{TVN} \mid \text{IYN} \mid \text{IVA}}$ → Sentences we can create

X → TY | IY | $\underbrace{\text{I and}}$ X recursion part

Y → VN | VA

I → I | you | he | she | it | we | they | me ...

T → This | These | Your | My | His | Her ...

N → true statement | false statement | the man | gun | at 8 am | true statements | false statements ...

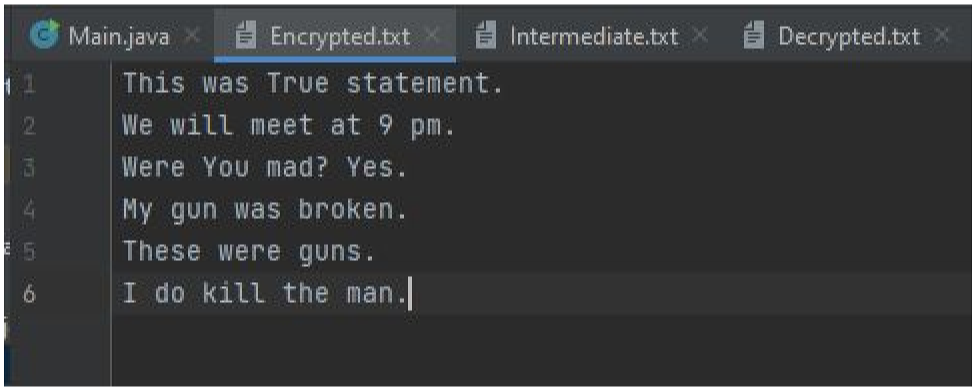
A → mad | broken ...

V → is | will meet | would be | are | was | were ...

Figure #1: Grammar for CFG

EXECUTION SAMPLES

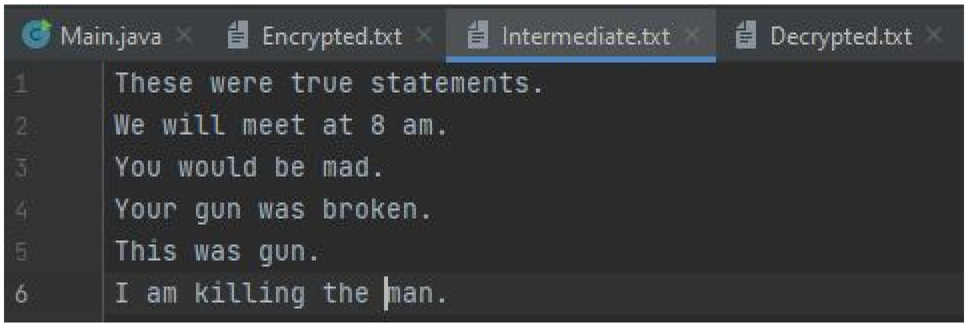
This is an input file. We get encrypted messages with this file. This is the source for our alterations.



```
This was True statement.
We will meet at 9 pm.
Were You mad? Yes.
My gun was broken.
These were guns.
I do kill the man.
```

Figure #2: Encrypted.txt (Input file)

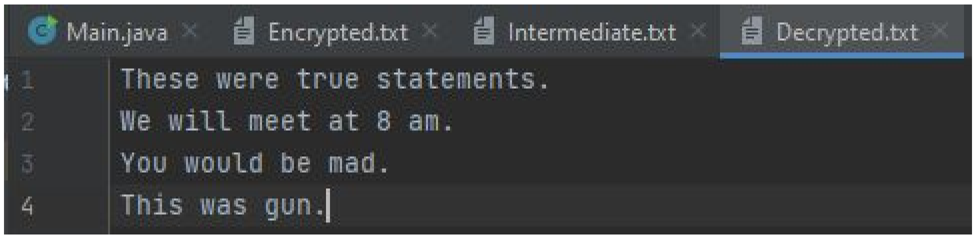
This is an after alteration file. Which means, we gets Encrypted.txt as an input for this file and then we apply our alterations to Encrypted.txt. Results that come from operations will be written in this file which is Intermediate.txt.



```
These were true statements.
We will meet at 8 am.
You would be mad.
Your gun was broken.
This was gun.
I am killing the man.
```

Figure #3: Intermediate.txt

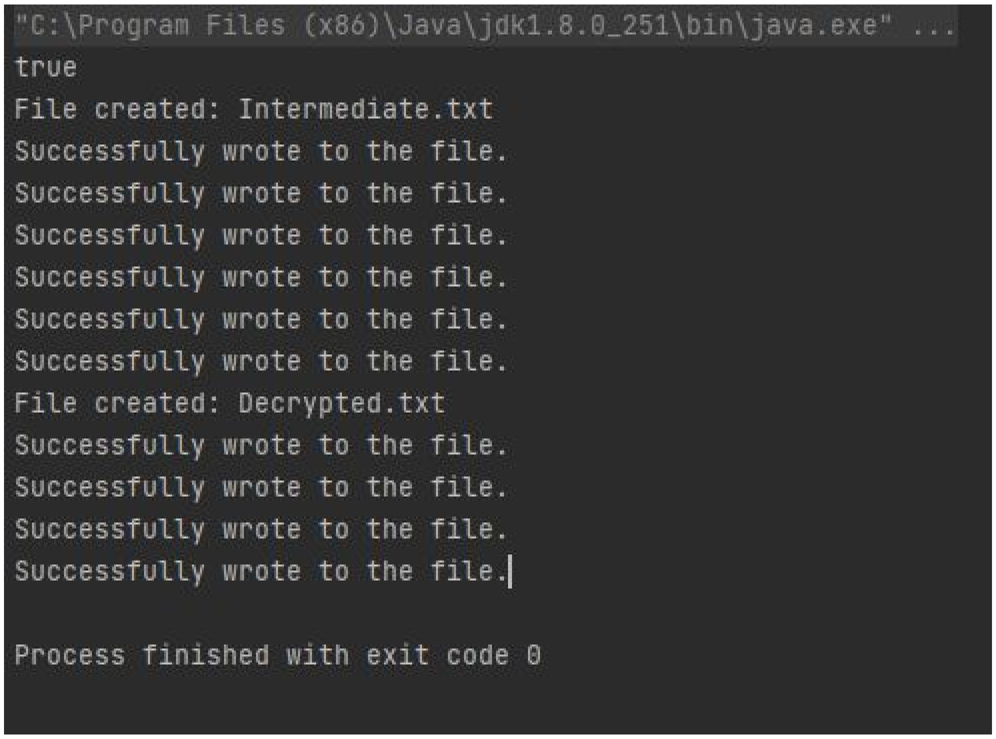
This file is created for the CFG part of the project. We get all lines from Intermediate.txt to apply CFG functions. If we get “true” as output from that line, we insert that line to this file. So, this basically checks whether we can create that sentence with our CFG or not.



```
These were true statements.
We will meet at 8 am.
You would be mad.
This was gun.
```

Figure #4: Decrypted.txt

These are the outputs after our project is stopped. First output for recursive sentence check. We gave a recursive sentence to check if our CFG can create a recursive sentence or not. Other outputs are File creation and write operations to created files.



```
"C:\Program Files (x86)\Java\jdk1.8.0_251\bin\java.exe" ...
true
File created: Intermediate.txt
Successfully wrote to the file.
Successfully wrote to the file.
Successfully wrote to the file.
Successfully wrote to the file.
Successfully wrote to the file.
Successfully wrote to the file.
File created: Decrypted.txt
Successfully wrote to the file.
Successfully wrote to the file.
Successfully wrote to the file.
Successfully wrote to the file.

Process finished with exit code 0
```

Figure #5: Output