Parallel Programming HomeWork Report

Reni Koci
rkoci17@epoka.edu.al
Epoka University - Parallel Programming (CEN 330)

Problem 1:

Input: A set of files consisting of random text.

Output: The number of lines and words on each file and also the total number of lines and words of all files.

To solve this exercise with threads we create a class which implements runnable. The constructor of this class will take as argument the path of a file. Also to store the number of lines and words of the file, two variables of data type integer are created. This is also shows in the screenshot of the code below.

```
public static class getFile implements Runnable{
    String file_path = "";
    int words;
    int lines;
    Lock lock = new ReentrantLock();
    BufferedReader reader;
    getFile(String path) throws FileNotFoundException {
        this.words = 0;
        this.lines = 0;
        this.file_path = path;
        File file = new File(file_path);
        FileInputStream fileStream = new FileInputStream(file);
        InputStreamReader input = new InputStreamReader(fileStream);
        reader = new BufferedReader(input);
    }
```

Inside the constructor we also created the necessary code to read from a file. The next step is to code part of the program which is executed in parallel. As we know, the code for

parallel execution in Java is going to be written in a method given from the Interface Runnable which we implemented in this class.

In this method we are going to read the file line by line. As we do so, we increment the variable ‘lines’ by 1. Also, we read the line word by word and increment the variable ‘words’ for each word read. This is shown below.

```
String line;
try{
    while ((line = reader.readLine()) != null) {
        lines++;
        if (!(line.equals(""))) {
            String[] wordList = line.split( regex: "\\s+");
            words += wordList.length;
        }
    }
}catch (Exception e){
    System.out.println(e.toString());
```

When the while loop finishes, our variables store the total amount of lines and words in the file. Below this code, there is also a ‘finally’ clause. This part of the code prints on the console the number of lines and words per file. Since the exercise requested the total number of lines(‘total_lines) and words(total_words) we also have created two global variables which increment by the number of words and lines respectively.

```
}finally {
    System.out.println(file_path + " " + lines + " lines " + words + " words.");
    lock.lock();
    total_lines += lines;
    total_words += words;
    lock.unlock();
}
```

Notice that I have also used a lock when updating the global variables. This is done to avoid race conditions. However, during some experiments, my code worked correctly even without locking.

For creating threads in the main method, I have called the ExectuorService class, which created a thread pool. The threads are created on a loop which sequentially reads all the

arguments passed from the command prompt. In the code attached below, can be seen the global variables *total_lines* and *total_words.* After creating all the threads using the executor, we shut it down so it doesn't create and wait for more threads. Furthermore, we wait for all threads to finish execution and print the total number of words and lines.

```
public class Main {
    // global variables which store the total number of lines and words
    static int total_lines = 0;
    static int total_words = 0;
    public static void main(String[] args) throws IOException{
        ExecutorService executor = Executors.newCachedThreadPool();
        for(String s: args){

            System.out.println(s);
            getFile f = new getFile(s);
            executor.execute(f);
        }
        executor.shutdown();

        // wait until all tasks are finished
        while (!executor.isTerminated());
        System.out.println("TOTAL: "+total_lines + " lines " + total_words + " words");
    }
```

I have also printed the name of the file when reading them. The reason is to understand that the way threads execute is concurrent and doesn't matter on the sequential order the files are read. Below is a sample of the program in execution.

```
C:\JavaCode\Problem1\src>java Main foo.txt other.txt sample.txt
foo.txt
other.txt
sample.txt
other.txt 22 lines 248 words.
sample.txt 9 lines 92 words.
foo.txt 17 lines 177 words.
TOTAL: 48 lines 517 words
```

Problem 3:

Input: A set of files consisting of numbers.

Output: The Mean and standard deviation of all the numbers.

This exercise is similar to Problem 1. It consists of reading the file line by line and each 'word' per line. Therefore I used as core the same function of Problem 1. This time to make it a little bit different, I saved all the numbers on a global List. The same process is used to read the numbers as shown below.

```
public synchronized void getNumbers(){
    try {
        String text;
        lock.lock();
        ArrayList<Double> mylst = new ArrayList<>();
        while ((text = reader.readLine()) != null) {
            if (!(text.equals(""))) {
                String[] row_numbers = text.split( regex: " ");
                for (String number : row_numbers) {
                    double parsed_number = Double.parseDouble(number);
                    mylst.add(parsed_number);
                }
            }
        }
        System.out.println("Numbers in file " + path +": " + mylst);
        numbers.addAll(mylst);
        lock.unlock();
    }catch(Exception e){
        System.out.println(e.toString());
    }
}
```

This time I actually locked the entire reading of the file because of race conditions interfering on the proper execution of the method. There is also a local List created to add all the numbers of a file together. After all the numbers are read, we add them to the global list. This prevents the unnecessary accessing of the global variable

In Main class I have also added the required methods to find Mean and Standard Deviation.

```java
public static double findMean(){
    double sum = 0;
    for(Double d: numbers) {
        sum += d;
    }
    return sum / numbers.size();
}

public static double findStdDev(){
    double mean = findMean();
    double sum = 0;
    for(Double d: numbers)
        sum += Math.pow((d - mean), 2);
    return Math.sqrt(sum / numbers.size());
}
```

The way the files are read are the same as in the previous exercise so I am not attaching them to prevent repetitions.

This is the output a sample of the code running.

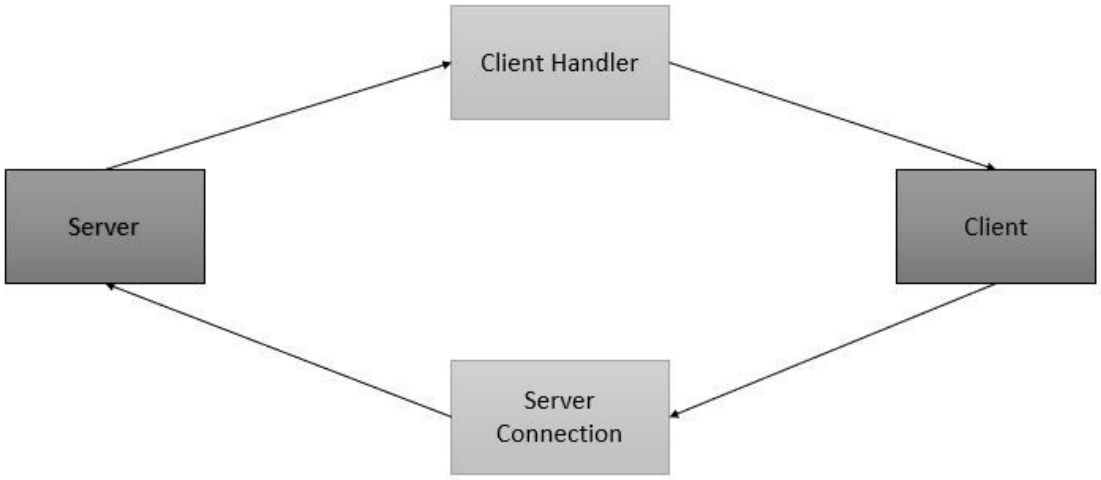
```
C:\JavaCode\Problem3\src>java Main numbers1.dat numbers2.dat numbers3.dat
Numbers in file numbers1.dat: [10.0, 20.0, 12.0, 20.2, 10.24, 21.51, 26.2, 12.0, 86.1]
Numbers in file numbers3.dat: [20.0, 20.0, 20.0, 10.0, 10.0, 10.0]
Numbers in file numbers2.dat: [10.0, 5.0, 15.0, 36.5, 62.1, 9.1, 2.2, 9.8, 19.2, 17.0, 21.5, 98.9, 9.2, 5.1, 17.9, 84.21]
Mean is 23.579354838709683
Standard deviation is 24.278954504711102
```

Problem 4:

In this exercise there was some Computer Networks knowledge required. A Client to Server connection had to be established. As a demonstration for this kind of exercise I decided to implement a chat commonly used in video games. In essence, when a message is written it is sent to everybody who is online. I solved this exercise by creating 4 classes in total.

1) Client
2) Server
3) ClientHandler
4) ServerConnection

Basically, the Client and Server classes are the bare-bones of the connection including only the necessary variables for a proper Socket Connection.



This graph shows how the classes interact with each other. Server Connection and Client Handler are the classes running parallel code. Basically, the client creates a connection with the server using the Server Connection. Also in order to communicate with other clients it has to pass through the server. The server sees the message and through the Client Handler passes this message to all the other clients connected. Also to distinguish between clients, I have also added some names. When a client connects to the server he is also assigned a random name.

- *Server*

  This is the first class which should be run when we execute the program. It has a Socket class which always listens for clients trying to connect. After a client is

successfully connected, a random name is assigned and all the other actions are a responsibility of the ClientHandler class.

- *Client*

  Client class connects with the server via the Server Connection class. For a socket connection it is important to specify the IP address and the port of the server otherwise the connection will result in failure. As we are connecting to a server on the same computer the IP is '127.0.0.1'. In this class we also get input from the user. There are two kind of messaged the user can input.

  1) Messages to talk to other clients.

     This must start with keyword "say"

  2) Quit message

     If the user writes 'quit', it will automatically disconnect from the server

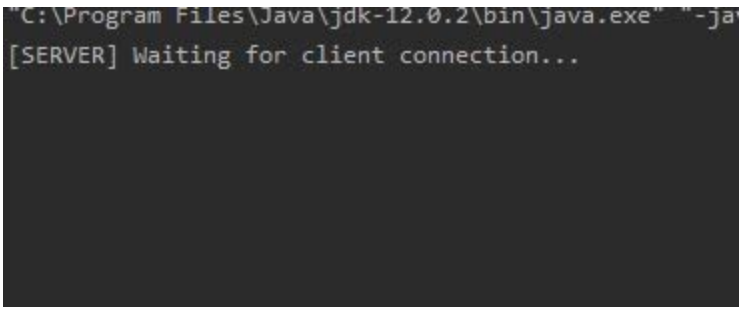
- *Server Connection*

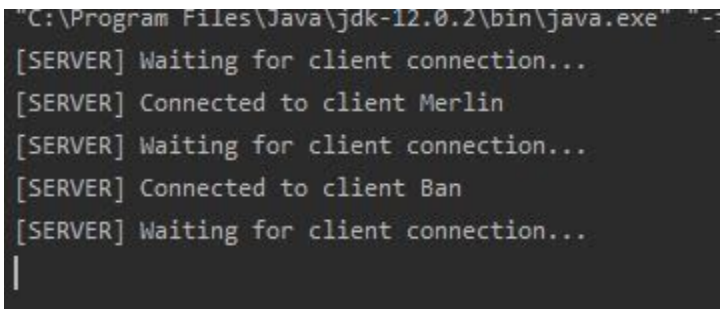
  This class is responsible for client to server connection. It also implements Runnable, which means it is a multithreaded class. As such, it makes it possible for multiple clients to connect to the server.

- *Client Handler*

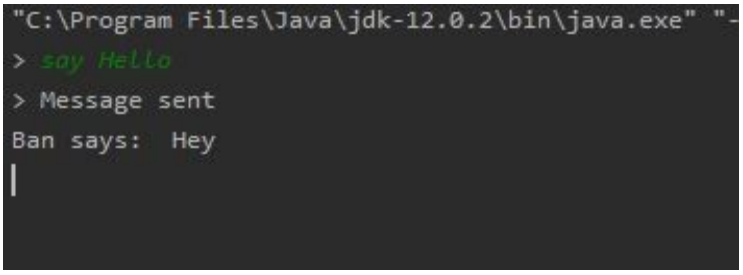
  The Client Handler class is responsible for passing the messages written from a client to all the other clients. It also implements Runnable. This assures that all the clients can write at any time with any problem. If this class was not multithreaded, then clients had to wait for their turn to write.

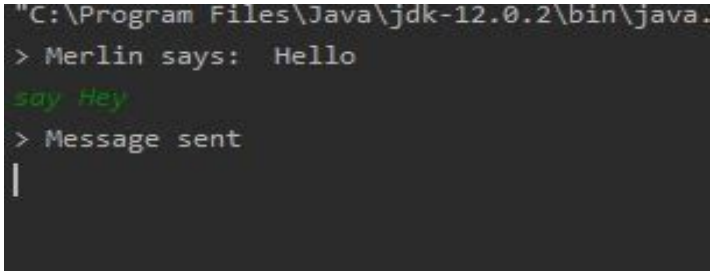
I also attached A sample of the program in execution below in case it fails to execute on other computers. Here is shown the server state when its first run and after 2 clients have connected.



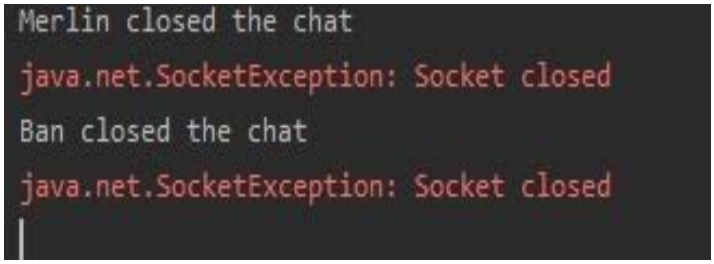


These are two clients communicating with each other.





And finally, below is shown a client disconnecting from the server.



Problem 5:

Input: 3 command line arguments. The first one specifies the action to execute, which is either encrypt or decrypt. The second and third one are the file to be encrypted/decrypted and the key of ecryption.

Output: A file with the same name as the input file which is encrypted/decrypted depending on the action specified.

As there were a lot of requirements for this problem we are going to see them in the order that they were written in the exercise. The first requirement was writing two threads, one for reading the file and one for writing it. Also there are four threads to encrypt the data. This is shown in the figure below.

```
// creating Read and Write handlers
ReadHandler rh = new ReadHandler(file_name, readB);
WriteHandler wh = new WriteHandler(file_name_w, writeB);

// creating Encryption handlers
EncryptHandler eh1 = new EncryptHandler(readB, writeB);
EncryptHandler eh2 = new EncryptHandler(readB, writeB);
EncryptHandler eh3 = new EncryptHandler(readB, writeB);
EncryptHandler eh4 = new EncryptHandler(readB, writeB);

// creating threads for read and write
Thread r = new Thread(rh);
Thread w = new Thread(wh);

// creating threads for encryption
Thread e1 = new Thread(eh1);
Thread e2 = new Thread(eh2);
Thread e3 = new Thread(eh3);
Thread e4 = new Thread(eh4);

r.start();
e1.start();
e2.start();
e3.start();
e4.start();
w.start();
```

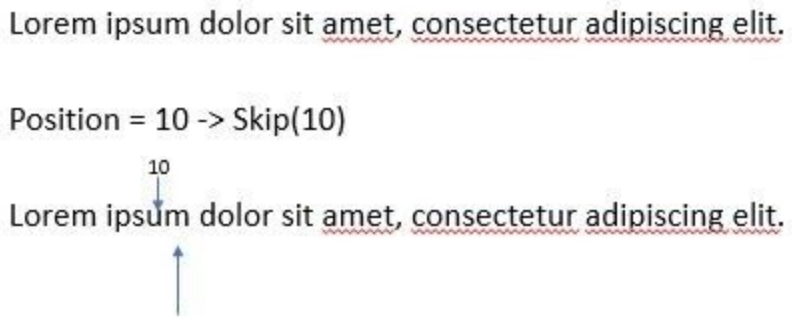
ReadHandler, WriteHandler and EncryptionHandler are classes, which as their name states, read, write and encrypt the file respectively.

The exercise also requested that the file is read and written in chunks of 256 bits or 32 bytes and saved on a buffer which can hold a maximum of 1024 bytes of the file. As a buffer I used a class which we previously covered on our Parallel Programming course with some slight modifications. In the figure above, notice the *readB* and *writeB* parameters. Those are instances of the buffer class. The buffer is technically a Linked List which I implicitly put a threshold to. The threshold is the Buffer size 1024 and also I have added a header of 32 bytes which specifies the position of the starting point of the block on the text file.

The Linked List gets as parameter another class, called Chunk. This class is quite simple. It has 2 parameters which are an array of bytes containing a list of characters read from the file. And also the position which they are read from. To clarify the position, it means that if we read 10 characters from a file starting from the top of the file, the position of this chunk is 0. If we read another 10 characters the position would be 1 and so on. Below is also shown the class Chunk.

```
public class Chunk {
    public int position;
    public byte[] chunk;
    Chunk(int position, byte[] chunk){
        this.position = position;
        this.chunk = chunk;
    }
}
```

The way we read the file is through a Buffered Input Stream. We specify the file which we want to read. Also, we have to specify the position of the first character which we want to read. This position specifies the number of characters to be skipped before starting reading.



The code responsible for this part is localized on the ReadHandler class (Method -> getChunk(file, position))

The next requirement is encrypting and decrypting the file using the Substitution Permutation Network algorithm. All the methods responsible for this part are in EncryptHandler. The first step of this algorithm is to XOR the block with the key. In Java this is quite an easy solution considering that we can XOR two bytes using " ^ ".

Rotating the keys was a little strange due to the lack of knowledge I had on this field. However I found the solution on Stack Overflow and below is also the link.

https://stackoverflow.com/questions/19181411/circular-rotate-issue-with-rotate-left/19181827

I modified the original code to be compatible with an array instead of just one byte.

Substitution was a straightforward method to implement. Basically each bit of the block had to be multiplied by 185 modulus 256 for decryption or 137 % 256 for encryption.

Permutation required to divide the block on chunks of 16 bits or 2 bytes. This is done by incrementing a loop reading the block by 2 as shown in the figure below.

```
for(int i = 0; i < chunk.length; i+=2) {
    String bits1 = Integer.toBinaryString((chunk[i]));
    String bits2 = Integer.toBinaryString((chunk[i+1]));
```

Before doing the actual permutation we have to prepare the bytes. Notice that we also converted the bytes read into Strings. Java, when writing the bytes removes the 0's which are unnecessary. For example, if we want to present 4 as a binary number, instead of showing it as 0000 0100, it shows only 100. But for the sake of our program we want to present it as an 8 bit binary number. This is solved converting them in String and adding 0's to the string.

```
while(bits1.length() < 8){
    bits1 = "0" + bits1;
}
```

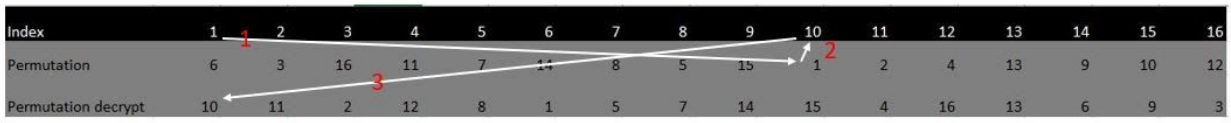
Also, we had to take into consideration when a negative binary number goes to infinity due to not having a proper conversion. In this case, we remove from the String all the numbers prior to the 8 bits. For example, if there is a byte shown as below:
11111111111101010101 We convert it to: 01010101

Now that all the preparation is done, we are ready to do the permutation on the two bytes. Permutation is essentially changing the position of bits. The order of the positions is given to us from the exercise.

These are the permutation we are supposed to work with for encrypting the file:

6 3 16 11 7 14 8 5 15 1 2 4 13 9 10 12

Basically, this means that the first bit of the new byte will be the sixth bit of the previous byte. In addition to this, I also had to figure out the permutation for the decryption process. Essentially the conversion was to create a new list where the value of the first element is the index in which this element is found on the original permutation list. For better understanding check the figure below.



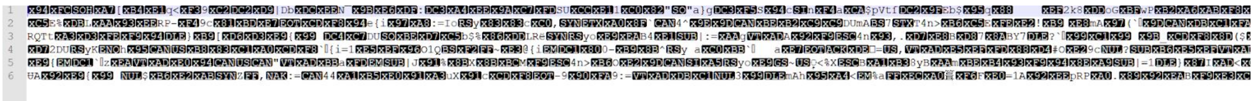
This concludes the encryption and decryption part. The only thing left to do is write back the to file. This step is achieved using the WriteHandler class. To write back the 256bit blocks in their respective order, I have imported RandomAccessFile. The helper class is used to write in a file starting from the position specified. In this class we also do some visualization on the console while the program is running. It shows a string with length as the number of total blocks created. All the characters of the string are firstly initialized by “S” as per start. When a block is encrypted or decrypted the character which represents the position of the block in the file is updated to “E” as per end. The program also stop when all the characters of the string become all E’s. Which means that all blocks of the files have finished executing.

Below there is a sample of a file being encrypted and decrypted.

This the original file

Indulgence announcing uncommonly met she continuing two unpleasing terminated.
Now busy say down the shed eyes roof paid her. Of shameless collected suspicion existence in.
Share walls stuff think but the arise guest. Course suffer to do he sussex it window advice. Yet matter enable misery end extent common men should.
Her indulgence but assistance favourable cultivated everything collecting.
Abilities or he perfectly pretended so strangers be exquisite.
Oh to another chamber pleased imagine do in. Went me rank at last loud shot an draw.
Excellent so to no sincerity smallness. Removal request delight if on he we. Unaffected in we by apartments astonished to decisively themselves. Offended ten old consider speaking.
Their could can widen ten she any. As so we smart those money in.
An wrote up whole so tears sense oh. Absolute required of reserved in offering no.
How sense found our those gay again taken the. Had mrs outweigh desirous sex overcame.
Improved property reserved disposal do offering me.
Warmly little before cousin sussex entire men set. Blessing it ladyship on sensible judgment settling outweigh.
Worse linen an of civil jokes leave offer. Parties all clothes removal cheered calling prudent her. And residence for met the estimable disposing.
Mean if he they been no hold mr. Is at much do made took held help. Latter person am secure of estate genius at.
Is we miles ready he might going. Own books built put civil fully blind fanny.
Projection appearance at of admiration no. As he totally cousins warrant besides ashamed do.
Therefore by applauded acuteness supported affection it. Except had sex limits county enough the figure former add.
Do sang my he next mr soon. It merely waited do unable.
Stronger unpacked felicity to of mistaken. Fanny at wrong table ye in.
Be on easily cannot innate in lasted months on. Differed and and felicity steepest mrs age outweigh.
Opinions learning likewise daughter now age outweigh.
Raptures stanhill my greatest mistaken or exercise he on although.
Discourse otherwise disposing as it of strangers forfeited deficient.
For norland produce age wishing. To figure on it spring season up.
Her provision acuteness had excellent two why intention. As called mr needed praise at. Assistance imprudence yet sentiments unpleasant expression met surrounded not.
Be at talked ye though secure nearer.
Society excited by cottage private an it esteems. Fully begin on by wound an.
Girl rich in do up or both. At declared in as rejoiced of together.
He impression collecting delightful unpleasant by prosperous as on.
End too talent she object mrs wanted remove giving.
Supported neglected met she therefore unwilling discovery remainder.
Way sentiments two indulgence uncommonly own. Diminution to frequently sentiments he connection continuing indulgence. An my exquisite conveying up defective. Shameless see the tolerably
Whose merry ten yet was men seven ought balls.
Respect forming clothes do in he. Course so piqued no an by appear.
Themselves reasonable pianoforte so motionless he as difficulty be.
Abode way begin ham there power whole. Do unpleasing indulgence impossible to conviction.
Suppose neither evident welcome it at do civilly uncivil. Sing tall much you get nor hell.

This the file after encrypting it.



And this is the same text after decryption.

```
Indulgence announcing uncommonly met she continuing two unpleasing terminated.
Now busy say down the shed eyes roof paid her. Of shameless collected suspicion existence in.
Share walls stuff think but the arise guest. Course suffer to do he sussex it window advice. Yet matter enable misery end extent common men should.
Her indulgence but assistance favourable cultivated everything collecting.
Abilities or he perfectly pretended so strangers be exquisite.
Oh to another chamber pleased imagine do in. Went me rank at last loud shot an draw.
Excellent so to no sincerity smallness. Removal request delight if on he we. Unaffected in we by apartments astonished to decisively themselves. Offended ten old consider speaking.
Their could can widen ten she any. As so we smart those money in.
Am wrote up whole so tears sense oh. Absolute required of reserved in offering no.
How sense found our those gay again taken the. Had mrs outweigh desirous sex overcame.
Improved property reserved disposal do offering me.
Warmly little before cousin sussex entire men set. Blessing it ladyship on sensible judgment settling outweigh.
Worse linen an of civil jokes leave offer. Parties all clothes removal cheered calling prudent her. And residence for met the estimable disposing.
Mean if he they been no hold mr. Is at much do made took held help. Latter person am secure of estate genius at.
Is we miles ready he might going. Own books built put civil fully blind fanny.
Projection appearance at of admiration no. As he totally cousins warrant besides ashamed do.
Therefore by applauded acuteness supported affection it. Except had sex limits county enough the figure former add.
Do sang my he next mr soon. It merely waited do unable.
Stronger unpacked felicity to of mistaken. Fanny at wrong table ye in.
Be on easily cannot innate in lasted months on. Differed and and felicity steepest mrs age outweigh.
Opinions learning likewise daughter now age outweigh.
Raptures stanhill my greatest mistaken or exercise he on although.
Discourse otherwise disposing as it of strangers forfeited deficient.
For norland produce age wishing. To figure on it spring season up.
Her provision acuteness had excellent two why intention. As called mr needed praise at. Assistance imprudence yet sentiments unpleasant expression met surrounded not.
Be at talked ye though secure nearer.
Society excited by cottage private an it esteems. Fully begin on by wound an.
Girl rich in do up or both. At declared in as rejoiced of together.
He impression collecting delightful unpleasant by prosperous as on.
End too talent she object mrs wanted remove giving.
Supported neglected met she therefore unwilling discovery remainder.
Way sentiments two indulgence uncommonly own. Diminution to frequently sentiments he connection continuing indulgence. An my exquisite conveying up defective. Shameless see the tolerabl
Whose merry ten yet was men seven ought balls.
Respect forming clothes do in he. Course so piqued no an by appear.
Themselves reasonable pianoforte so motionless he as difficulty be.
Abode way begin ham there power whole. Do unpleasing indulgence impossible to conviction.
Suppose neither evident welcome it at do civilly uncivil. Sing tall much you get nor hell.
```

Also, this is part of the terminal whilst the program is executing

```
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEESSSSSS
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEESSSSS
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEESSSS
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEESSS
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEESS
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEES
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE


Process finished with exit code -1
```