CSE senior 1 -2023

CSE 331: Data Structure and algorithms

XML Editor Project

Name:	**ID:**	**sec:**
Nada Mohamed Soliman Mohamed	*18Q2926*	*4*
Hager Rafaat Mohammed	*1807950*	*4*
Habiba Mohamed Ahmed Mohamadeen	*1808582*	*3*

Background:

We made XML Editor and convert XML to JSON.

The XML editor was built using Qt to for the interactive GUI. Qt allowed us to built a full user experience. For the backend we used cpp to enhance different functionality.

The editor supports features:

-**Checking errors of xml file***: errors like missing any of the closing and opening tags or not matching tags. The function detects and points to the errors with arrows.*

-**Correct Errors of xml file***: The feature is able to solve the errors.*

-**Formatting (prettifying) xml file** *: This feature makes the xml well formatted by keeping the indentation for every level.*

-**Minifying xml file:** *This feature should aim at decreasing the size of an XML file (compressing it) by deleting the whitespaces and indentations.*

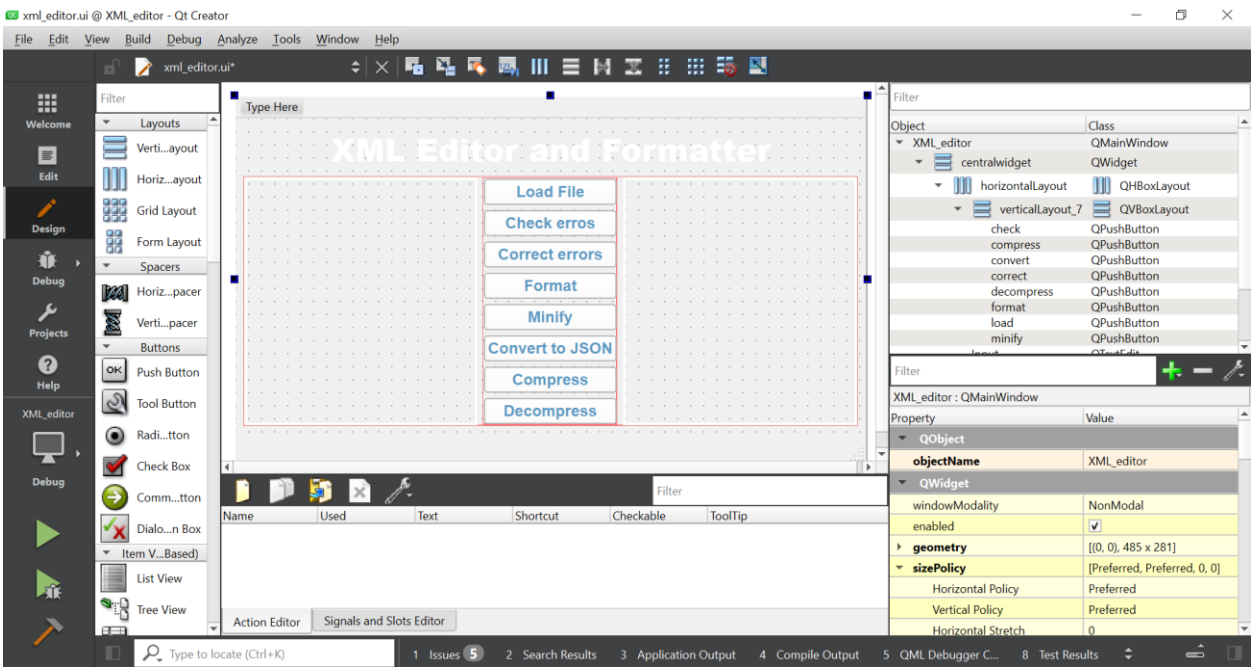
-**Compressing the data in xml/json file:** *This feature is aim to reduce the size of the file using a data compression technique.*

Next we want to talk about data structure and algorithms of the operations.

GUI:

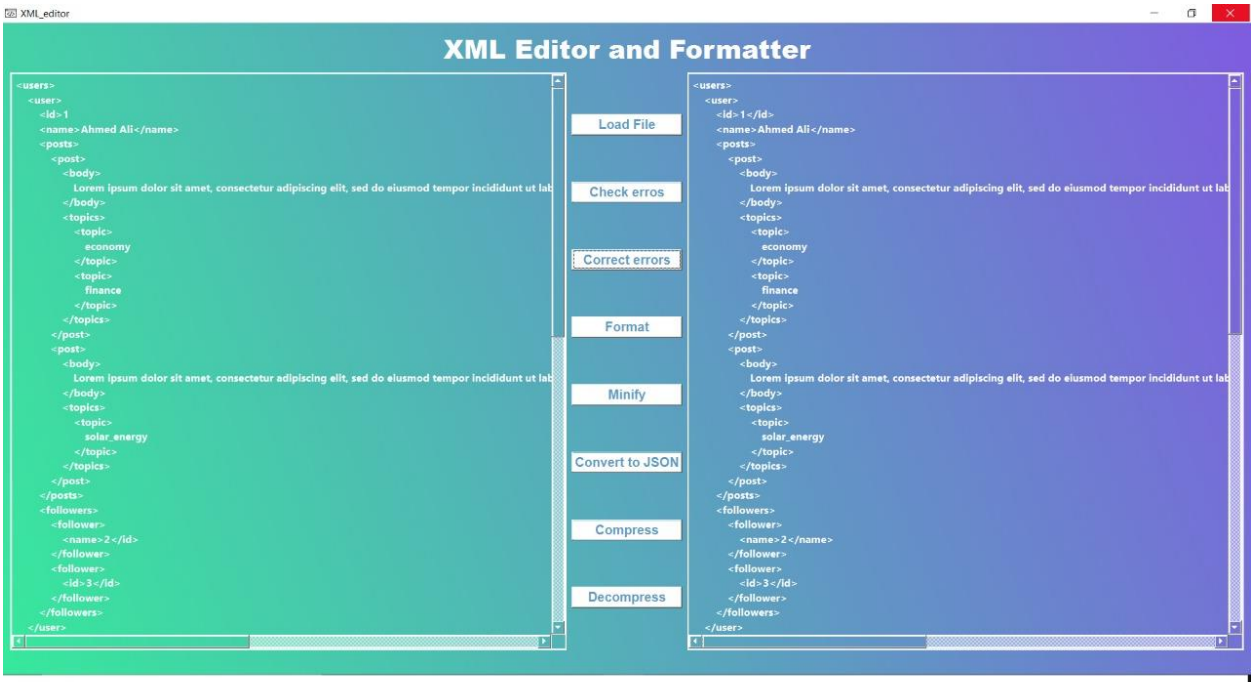
We used the cross-platform framework “Qt creator" to design and built the GUI. The main window consists mainly of eight push buttons and two text edit areas to view input and output. The eight buttons are used to browse and load a file, check errors, correct errors, prettify, minify, convert to JSON, compress and decompress input file. The buttons are triggered when they are clicked by the user and each button calls a different function, with time complexities as mentioned above, that operates independently.

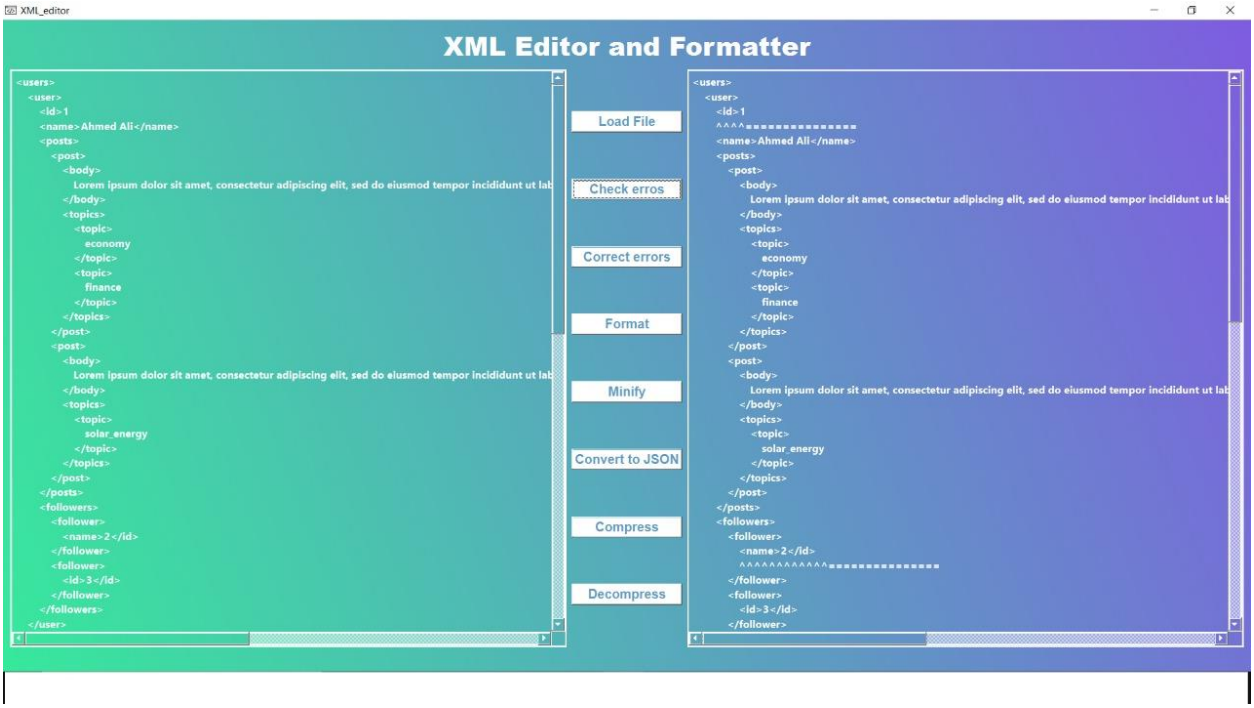
The main window takes all codes and functions (format, minify, …) and links them together to produce a systematic, well-organized document. When a file is loaded, its content is displayed on one of the text edit areas and, when a button is triggered to do some function, the output is displayed on the other text edit area. This configuration allows the user to view the input and output files simultaneously.



*Aside from the formatting and editing functions that are called when push buttons are triggered, some functions are used to display graphics and set the size of the window (e.g. **resizeEvent**). Such functions have a complexity of O(1) and run only once throughout the running time of the program.*

Consistency of xml file:





- *XML file is consistent if each opening tag is closed by closing tag in the same level.*

- *The program take xml string and detects the errors on it and points arrows on the errors (using check errors) and correct errors (using correct errors). If the file is consistent return the same file but if it is not consistent return the file with arrows point to the errors (check error) or corrected file (correct error).*
- *The program covers three cases of inconsistency:*

1- If root/leaf closed tag has been missed.

2- If closed tag has been closed incorrectly.

Consistency Algorithm:

❖ functions used:

(1) int countSpaces(string s) : it takes a string "s" as parameter, and It is used to return index of first character in string.

(2) string space(int n) : it takes integer "n" as parameter which represents number of tabs and returns a string with "n" tabs .

(3) string space1(int n): it takes integer "n" as parameter which represents number of spaces and returns a string with "n" spaces .

(4) void putErrorArrows(queue<string>& error, int index, int lineLength): it takes error and line length and the index of first character of error and returns the arrow to point to the error(we use it in checkErrors)

(5) void consistency(string FileLocIn, string FileOutCorrect, string FileOutError, bool returnerror): it takes the file we want to check/correct error in and take the checked/corrected file (path) to create the file in the desired path and the bool parameter is used to choose I want a checked file (true) or corrected one(false)

❖ Explanation on how consistency function works:

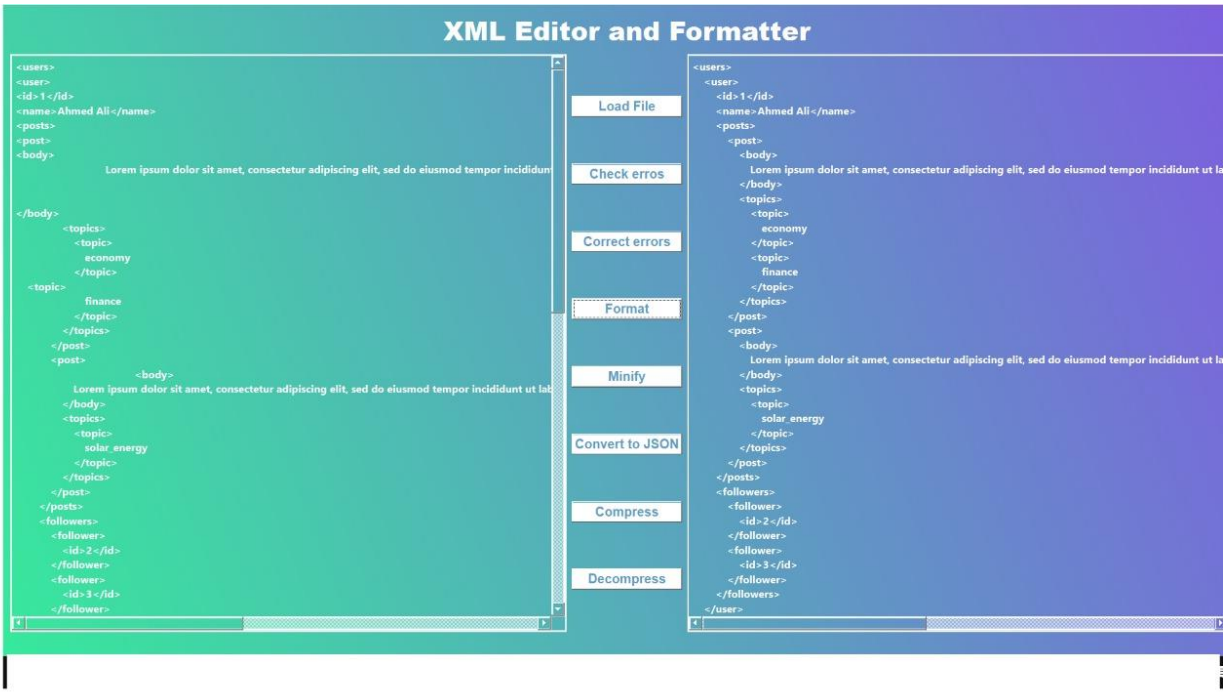
✓ *-Consistency() function takes input XML file and check its consistency.*

- ✓ *-If the file is consistent return the same file in both operations (checking for errors / correcting errors), if not (in case of checking errors) returns the file with arrows but in case of correcting returns the corrected file.*
- ✓ *-We used stack to push open tags to check consistency*
- ✓ *-We used queue to print string on the output file*
- ✓ *If the input line is open tag, then push the tag without the attributes to the stack and with attributes to the queue.*
- ✓ *If the input line is closed tag, check it matches any open tag in the stack.*
- ✓ *If matches, pop that matches with it and push the closing in the queue.*
- ✓ *-If not, then it is error and push the matching closed tag of the top of the stack in the queue and pop the stack.*
- ✓ *-If the line input is opening tag with text, then check the matching closing tag in the end of the line. o If matching push in the queue, if not, then it is error and push the opening and correct closing tag in the queue.*
- ✓ *-If the line is text push in the queue. o Print the content of the queue.*

Time Complexity:

(1) coutSpaces() : Big O(N)
(2) space() : O(N)
(3) space1(): O(N)
(4) putErrorArrows: O(1)
(5) Consistency(): O(N^3) after we calculated in code
(6) For loop used to print output: O(N)

Formatting:



- *The program take xml string and convert it to formatted xml file.*
- *The xml file is formatted if indentation of open tag level and closed tag level are the same.*
- *If there is an empty line in xml it will eliminate it.*

Formatting Algorithm:

❖ Functions used:

(1) int countspaces(string s) : it is the same one that is used in consistency , it takes a string "s" as parameter, and It is used to return index of first character in string.

(2) string tab(int n) : it takes integer "n" as parameter which represents number of tabs and returns a string with "n" tabs .

(3)void format(string FileLocIn, string FileLocOut) : it is the function that is responsible for formatting (prettifying) the xml.

❖ Variables used:

(1) We use a stack <string> c : to push

(2) We use a queue<string>formatted : to push formatted string to be printed in the xml.

(3) Integer x : to be sent to tab function.

(4) String line: the input xml file is read line by line and put in it.

❖ Explanation on how format function works:

-The xml file is read line by line until it is finished.

-if the stack is empty push string line in the queue starting from the first character in it, and push string line in stack but after removing spaces in it (only if it has).

-if the first character in string line is not open tag "text input" push in the queue string of x+1 spaces followed by the input text line.

-if "<" and ">" not in the same line push in the queue string of x+1 spaces followed by the input line.

-if closed tag exist push string of x+1 spaces followed by closed tag and string and x—and pop the top of the stack.

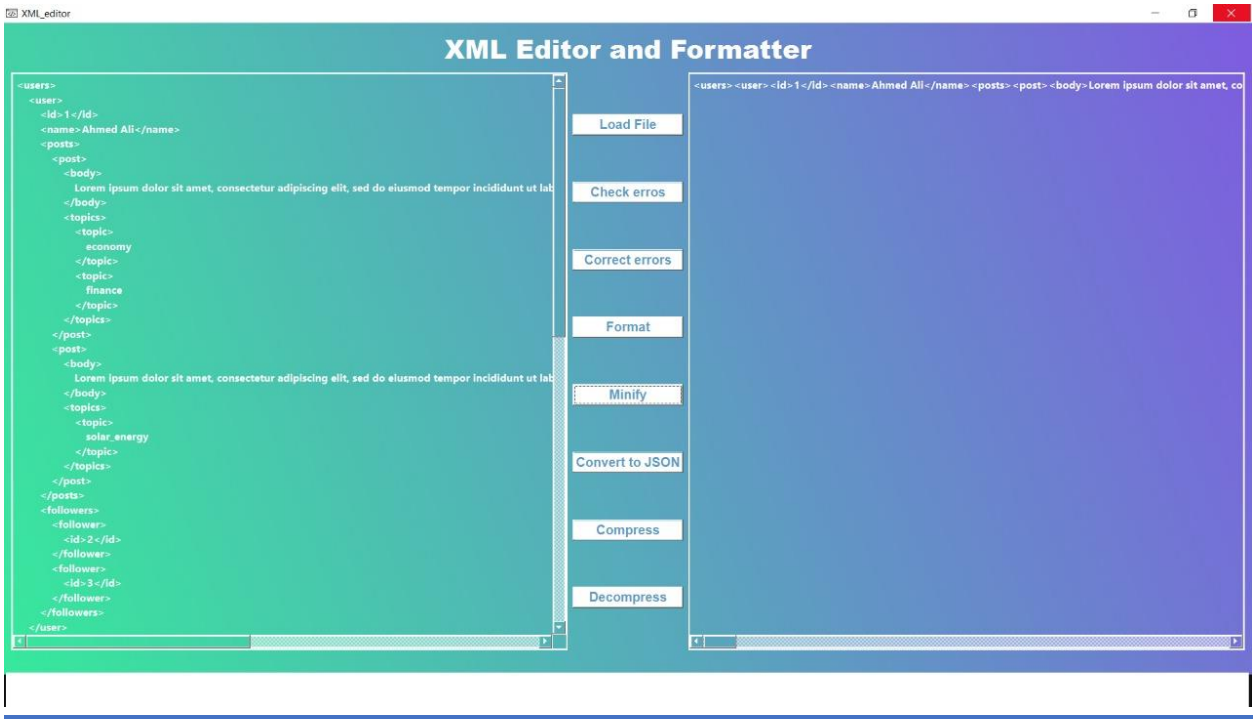
-if the input is open tag x++, push string of x+1 spaces followed by string line in the queue , and push input line in stack.

-for loop for printing queue.

Time Complexity:

(1) coutspaces() : Big O(N)
(2) tab () : O(N)
(3) format(): O(N^3) after we calculated in code
(4) For loop used to print output: O(N)

Minify:



- *The program takes a string input xml and minify it.*
- *As the xml file has a large size so we need to minify it by removing white spaces and indentations and by putting the lines of the file line by line behind each other.*

Minify function Algorithm:

-The function is void (has no return type) and takes two parameters.

-Reading input string line by line until it is finished.

-Eliminating white spaces before "<".

-Eliminating white spaces before values.

-Eliminating white spaces after opening tag.

-Eliminating white spaces before closing tag.

-All stored in string minimum.

-Write minimum into minified file.

Time Complexity:

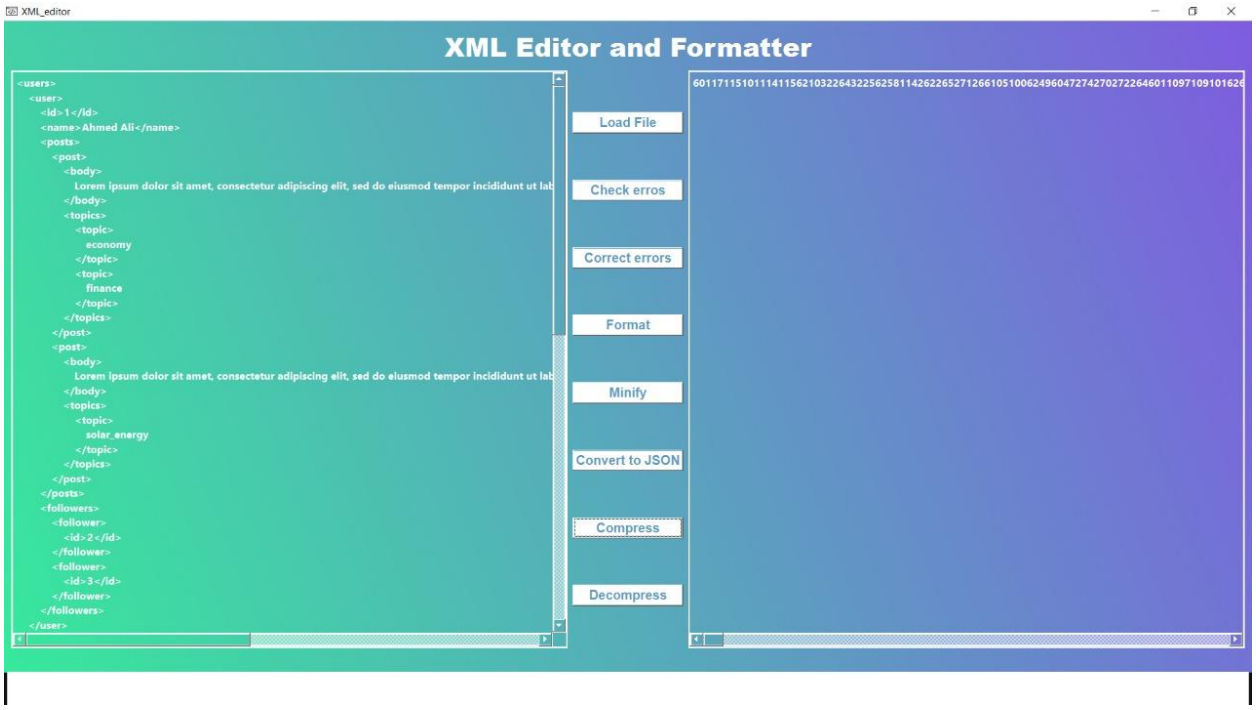
(1) While loop ”to eliminate white spaces”: Big O(N)

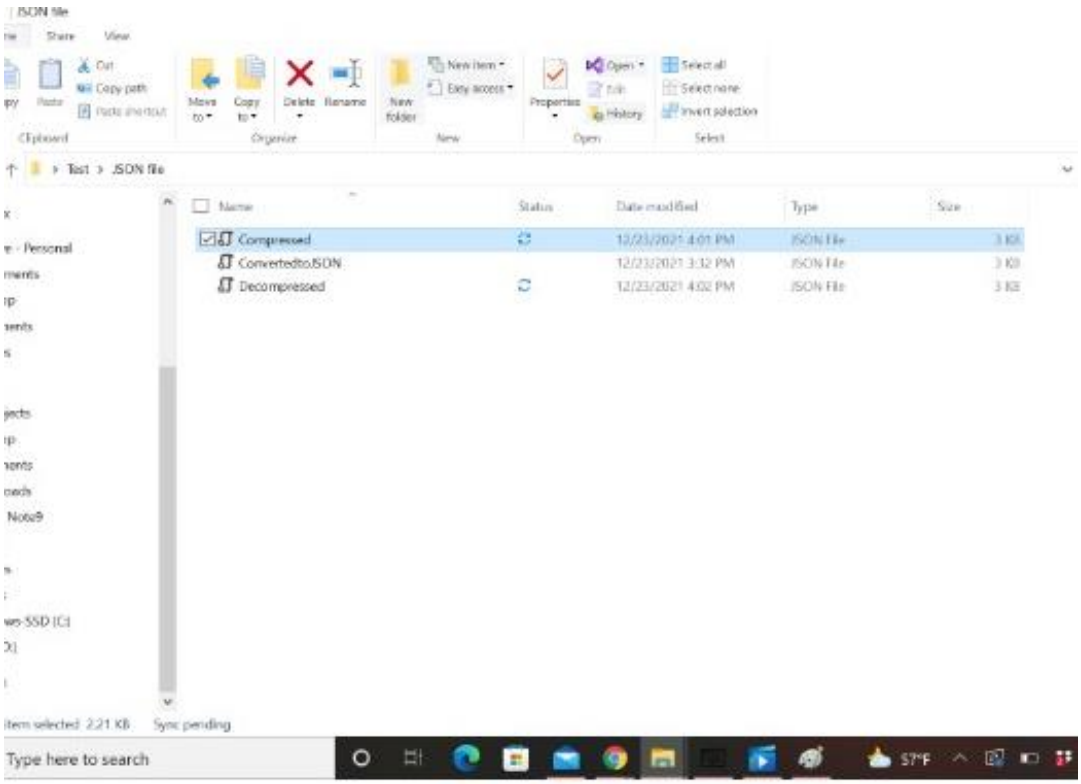
(2) For loop ”to remove spaces after opening tag”: Big O(N)

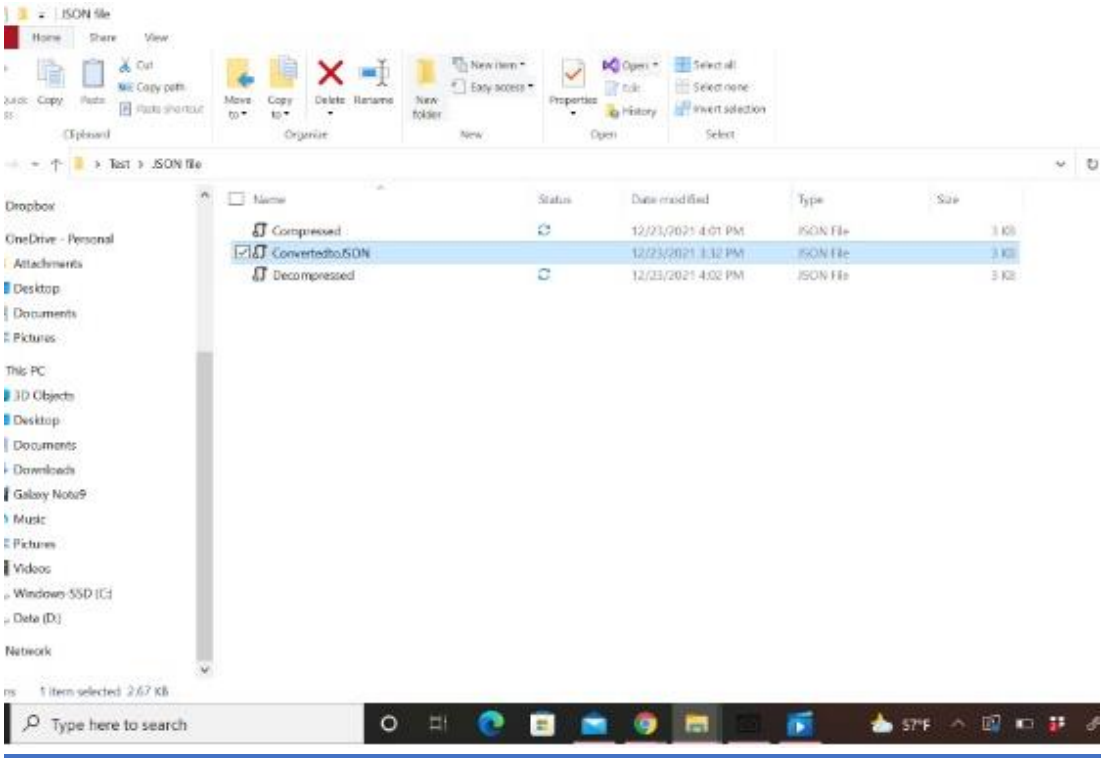
(3) For loop “to remove spaces before closing tags”: Big O(N)

(4) So total: O(N)

Compression:







- *The program takes a string input xml and compress it.*
- *As the xml file has a large size so we need to compress it by removing white spaces and indentations and convert the characters to numbers.*
- *We made a function decompress to decompress the compressed file and return the original xml again.*

Algorithm:

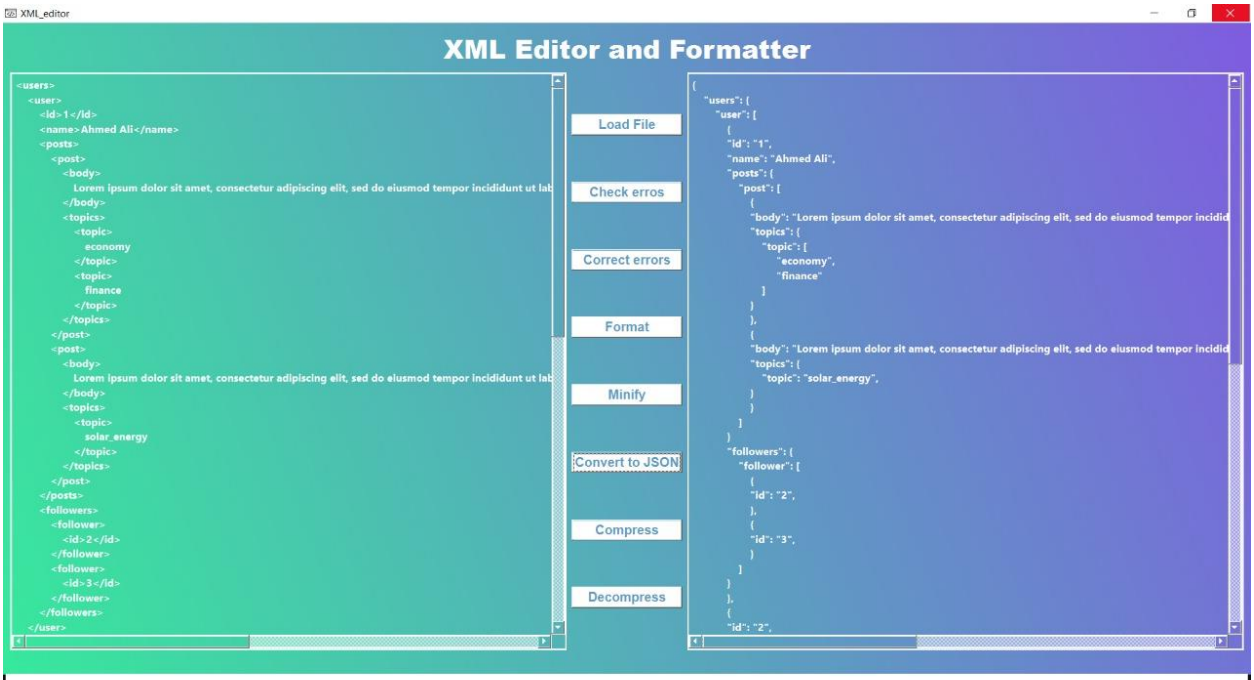
-for loop to create map of ascii char to ascii decimal value with time complexity O(N)

- for loop is used for adding new indices in the table by increasing the numbers compressed characters with time complexity O(N)

-print output file

-decompress function is used to reverse the operation and return the original file using two for loops

Converting to json:



-Requirement: converting file.xml to file.json.

-we used stack and queue and Tree and node in the implementation of the function.

- ***Print_Json(node*):***

- takes the root of the tree file that we want to convert to json.

- ***Similar() function:***

Used to print attributes and store its children in vector.

- ***Algorithm:***

If the row of this 2D vector has one element repeat step 1.

-If the row has more than one element, then this tag has similars, then print Json_name of this node followed by '[' and print_Json() recursively of all nodes in the row separated by ',' Then print ']'.

- If the node have no similars, then go step 1.

- If the node is leaf node, print its Json_Name, then go to next step.

-If the node is "Text" node, then print " "#text": " its Json_Name if this node have attributes, if is not have attributes print Json_Name only.

➢ *Time Complexity:*

-int FindFirstAngle(string s): O(n)

-int Num_Of_Angle(string s): O(n)

-vector<string> tokens(string s): O(n)

-node::node(string t): O(1)

-void node::set_attributes(vector<string> toks): O(n)

-string tab_json(int n): O(n)

-vector<vector<node>> Tree::similar(vector<node*> same): O(n^2)*

-void Tree::insert(string FileLoc): O(n)

-void Tree::print_attributes(node n): O(n)*

-void Tree::print_Json(node rootptr): recursive*

T(n)=1T(n/2)+O(n^2)