Data Structure Project – XML Editor

Program:

Course Code: ***CSE323***

Course Name: ***Data Structure***

Submitted to:

Dr : Islam El-maddah

Eng, Amr Mohamed

Ain Shams University
Faculty of Engineering
CSE Department
2021

Student Personal Information for Group Work

Student Names:	Student Codes:
Alaa Ibrahim Mohamed Ibrahim Amer	1700267
Alaa Shaaban Hussien Ali Shatat	1700271
Hussein Mahmoud Fouad El-Sayed	1700462
Nahla Mostafa Abdelkareem	1601583

Table of Contents

1. Background 4

2. Implementation details 4

2.1 The algorithm used in the XML editor 4

2.2 Operations 5

2.2.1 Parsing Function 5

2.2.2 Correction Function: 6

2.2.3 Formatting Function 7

2.2.4 ConvertToJason Function 7

2.2.5 Trim Function 8

2.2.6 Compress Function 8

2.2.7 Save and store in a new file 9

3. Complexity of operations 10

4. Screenshots 10

5. References 13

6. Working files 14

1. Background

XML stands for eXtensible Markup Language, it is similar to HTML language "used in web", XML was designed to store and transport data. XML is a language that defines a set of rules for encoding a document in a format that is both human-readable and machine-readable. [1]

In **XML** there are:

- Start-tag such as <example>.
- End-tag such as </example>.
- Element content, the characters between the open-tag and the end-tag such as <example>**she was able to program her computer**</example>
- Attributes: An *attribute* is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag such as <word lex_id="0">able</word>

XML documents may begin with an *XML declaration* that describes some information about themselves. An example is `<?xml version="1.0" encoding="UTF-8"?>`.

XML identifies data using tags, which are identifiers enclosed in angle brackets, each XML document has a single root element, for example:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

2. Implementation details

- Language used **C#**.

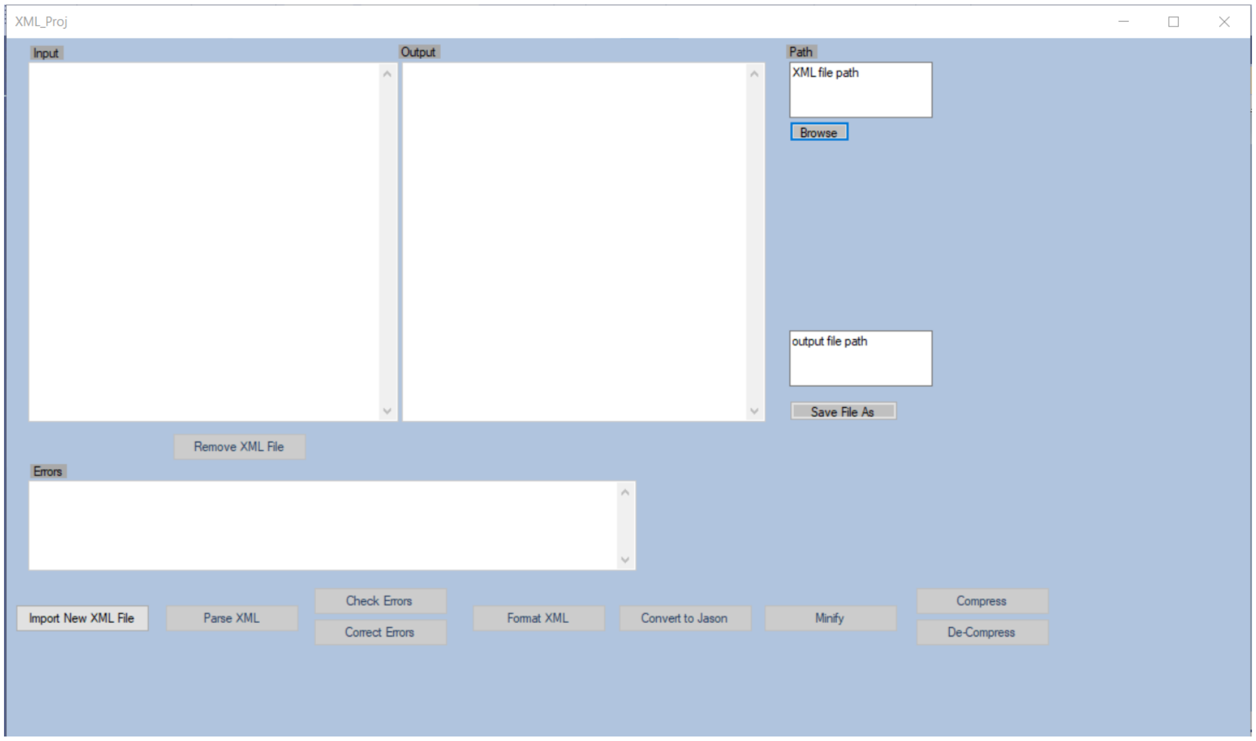
2.1 The algorithm used in the XML editor

At first, in the GUI the user should browse and select a XML file and import it, then can choose what operation will be executed, operations are:

- Parse XML file.
- Check and correct errors.
- Format the XML file.
- Convert the XML to Jason.
- Minify the XML file.
- Compress the data.
- De-compress the data.

- Save file as function.

When the user choose the operation. According to the user choice the function of this operation will be executed and the required result will be printed and the user can store it, the GUI is shown in the figure below.



2.2 Operations

OOP is used in order to simplify and organize the code.
Classes used:

- XML class: contains the functions used to execute the user choice. Contains of List of Tags and other variables to handle the operations.
- Tag class: contains a tag name, element content , a list of Tag" child tags", because each tag in the XML file may contain a child tag maybe we can call it nested tags, so we need to store the parent tag and Childs for it and contains attributes to store the attribute of each tag.
- Tag Attributes class: contains the attribute name and the attribute value.

2.2.1 Parsing Function

In this function we parse the XML file in order to store the tags and the data in appropriate data structure" as mentioned before", so operations can be handled easily

2.2.1.1 Pseudo Code

```
1. Read the file after passing the path of the uploaded file.
2. Read the file character by character in while loop.
3. Check if the document begins with XML declaration by searching for (<?),
```

If true skip the line.

4. If we find the angle bracket of the opening tag and the next character is not (\),

   If true start storing the tag name and attributes until finding (>).
   Add these values to new tag and add this tag to the stack.
5. Check for element content if there is any store it in the current tag attribute (TagValue)
6. Check for closing tag if any by looking for (<\), then store the current closing tag name and compare it to the top of the stack.

   If they are equal than pop the tag name from the stack.
7. Check for the root if the stack is empty after popping then it is a root tag.
8. Else add the current tag as a child to the top of the stack.

2.2.2 Correction Function:

The algorithm to determine if the start and end-tags balance uses a Stack data structure to keep track of previously read start-tags, Errors are missing end-tag, wrong tag name provided, opening tag with element content the open a new opening-tag without closing the first one.
The algorithm is:

Pseudo code:

1. Through the code we count the number of lines so if we detect an error we can add the number of the line to a string to show the user at the end the lines contain errors in it.
2. Read the input until the beginning of a tag is detected. (i.e. tags begin with <: if the next character is a / (slash), then it is an end-tag; otherwise it is a start-tag).
3. Read the tag's identity. (e.g. both tags <x> and </x> have the same identity: 'x').
4. If the tag was a start-tag, push it onto the Stack.
5. Otherwise, it is an end-tag. In this case, pop the Stack (which contains a previously pushed start-tag identity) and verify that the popped identity is the same as the end-tag just processed. Note**:** if the stack cannot be popped (because it is empty), the input is invalid; the algorithm should handle the error and correct it.
6. If the identities do not match, the XML expression is invalid. the algorithm should handle the error and correct it.
7. If they do match, then no error has been detected (yet!).
8. If there is still input that has not yet been processed, go back to the first step.
9. Otherwise (no more input) then the input is valid **unless** the Stack is not empty. Indicate whether the input is valid (Stack empty) or invalid and the algorithm should handle the error and correct it.

2.2.3 Formatting Function

General description:

This function mainly takes the content of XML file and rewrite it keeping the indentation for each level. So, if tag has children, then the content of the child will be in new line and with tab to make it easier to the user to visualize.

Code:

It contains one for loop to loop the root_tags, and function called formatting_childs take one attribute of type tag.

Pseudo code:

1- Parse the function so each root will be in the root-tags list.
2- For loop on the root-tags list to call the function formatting_childs.
3- In formatting_childs there is two options (if conditions).
4- First if the tag has children loop them and recurrence (formatting_childs) for each child to print it in the string.
5- Taking in consider take tab before each child to keep the indentation for each level.
6- Second if the tag has only tag value, then print this value in the string.

2.2.4 ConvertToJason Function

General description:

This function mainly takes the content of the XML file and rewrite it with the json syntax and structure.

So, if there are two tags have same name, then tag name will be written one time and the two tags content (attributes and value, children if found) will be written in between square brackets [].The content of each tag will be written in between two curly braces {}.Values and attributes will be written between double quotes "".

Code:

It contains one for loop to loop the root_tags.

And function called print take one attribute of type tag.

Converting from XML to JSON:

1- Parse the function so each root will be in the root-tags list.
2- For loop on the root-tags list and check if the current tag name is similar to the next, If yes give his attribute (has siblings) value of +1 and the next tag value of +2.
3- Then call the print function if the value of (has siblings) = 1 it will write the tag name and open this practice [, then will check if the tag has attributes or child or

values. If it has children, it will loop also to find if there are any consecutive siblings and recurrence print function for each child.

4- If the value of (has siblings) > 2 it will not write the tag name and print the contents of the tag.

5- If the value of has siblings = 0 it means it has no siblings so it will print the tag name and then all the contents of the tag.

2.2.5 Trim Function

The xml file is read and all new lines and spaces are excluded.

2.2.6 Compress Function

Reasons To Use Data Compression Algorithms:

we need Data Compression mainly because:

- Uncompressed data can take up a lot of space, which is not good for limited hard drive space and internet download speeds.
- While hardware gets better and cheaper, algorithms to reduce data size also helps technology evolve.

there are many techniques for data compressing, but in this project we will use ZLW technique to compress & decompress Xml files.

Lempel–Ziv–Welch (LZW) Algorithm:

LZW algorithm is a very common compression technique. This algorithm is typically used in GIF and optionally in PDF and TIFF. Unix's 'compress' command, among other uses. It is lossless type of compression, meaning no data is lost when compressing. The algorithm is simple to implement and has the potential for very high throughput in hardware implementations.

The Idea relies on reoccurring patterns to save data space. LZW is the foremost technique for general purpose data compression due to its simplicity and versatility.

How does it (LZW) Algorithm work?

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression. Characteristic features of LZW includes,

- When encoding begins the code table is empty. Compression is achieved by using codes 0 (can be started by 256 entries) through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

Implementation:

The idea of the compression algorithm is the following: as the input data is being processed, a dictionary keeps a correspondence between the longest encountered words and a list of code values. The words are replaced by their corresponding codes and so the input file is compressed. Therefore, the efficiency of the algorithm increases as the number of long, repetitive words in the input data increases.

. LZW ENCODING:

PSEUDOCODE:

```
1   Initialize an empty table (list of strings, each string has its numeric counterpart which is the
    index where this string is inserted in the list.
2   P = first input character
3   WHILE not end of input stream
4   C = next input character //if P is not in the string table add it in the table.
5   IF P + C is in the string table
6   P = P + C
7   ELSE
8   output the code for P
9   add P + C to the string table
10  P = C
11  END WHILE
12  output code for P
```

Decompression

In our code, the decompressing code is different from the decompression method in the LZW method, as we reuse the table we filled in in the encoder function and the codes (numbers) corresponding to each encrypted segment within the file, and use them backwards through the table, where each encrypted code is replaced with the corresponding string in the string table.

2.2.7 Save and store in a new file

1. After clicking the “save file as” button.
2. Choose the directory and name the output file In which the output of the operation will be provided in it.
3. Choose the type of the file (.txt, .json, .xml)

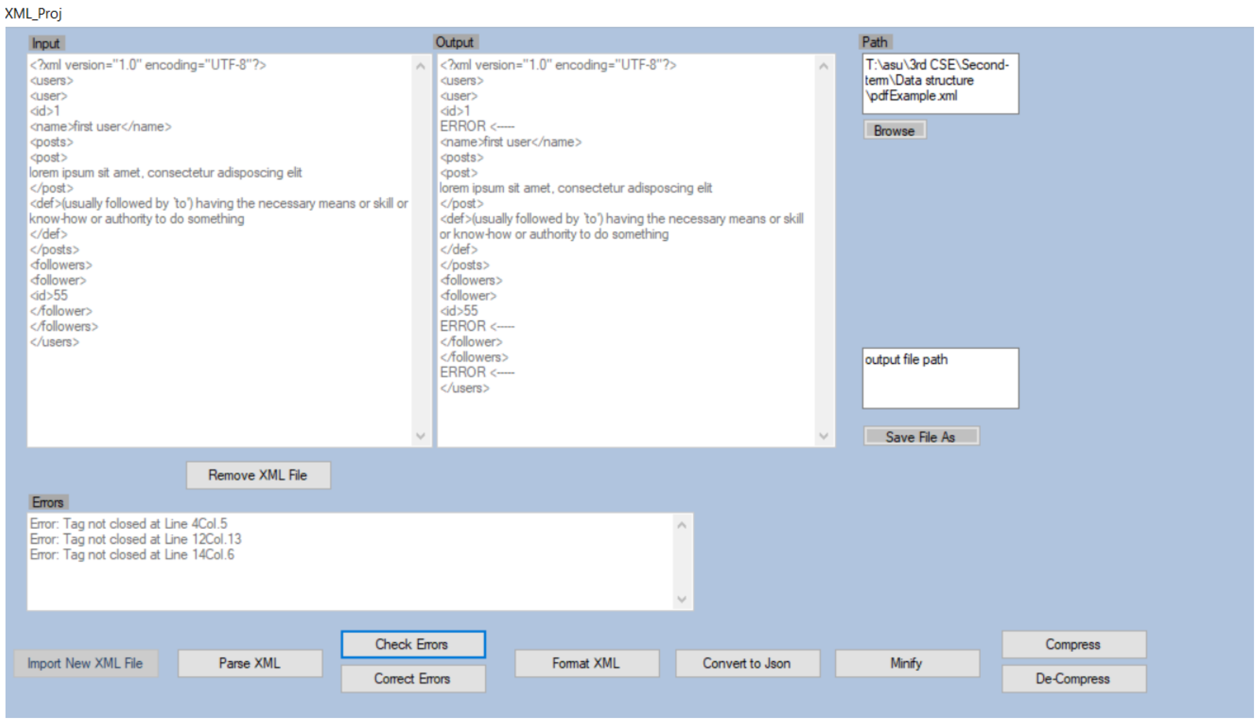
3. Complexity of operations

- *N is the total number of characters in XML file*
- *K is the total number of tags in XML file*

Function	Complexity
void Parse_XML()	O(N)
string correction()	O(N)
string FormatXML()	O(K)
string ConvertToJson()	O(K)
string Trim()	O(N)
string Compress()	O(N)
string Decompress()	O(N)
void `storeOutput(string filepath_, string sotred)`	O(N)

4. Screenshots

Check errors


XML_Proj

Input
```
<?xml version="1.0" encoding="UTF-8"?>
<users>
<user>
<id>1
<name>first user</name>
<posts>
<post>
lorem ipsum sit amet, consectetur adisposcing elit
</post>
<def>(usually followed by 'to') having the necessary means or skill or
know-how or authority to do something
</def>
</posts>
<followers>
<follower>
<id>55
</follower>
</followers>
</users>
```

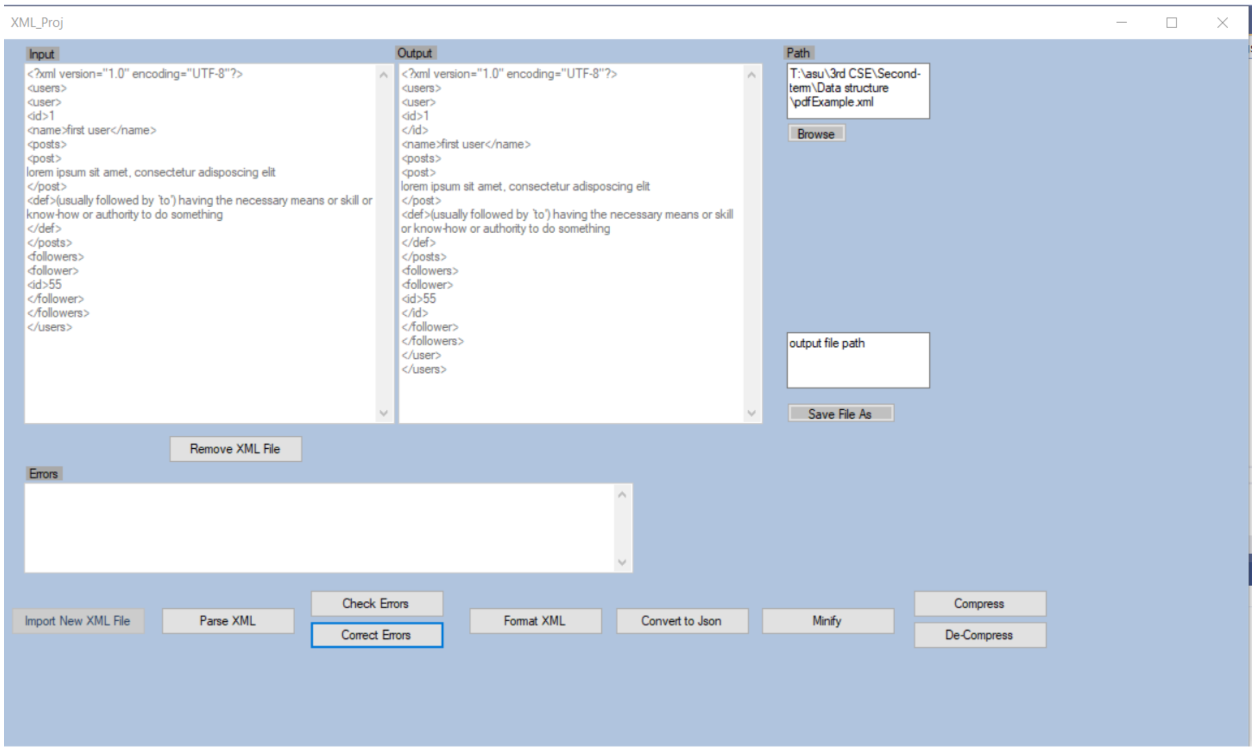
Output
```
<?xml version="1.0" encoding="UTF-8"?>
<users>
<user>
<id>1
ERROR <----
<name>first user</name>
<posts>
<post>
lorem ipsum sit amet, consectetur adisposcing elit
</post>
<def>(usually followed by 'to') having the necessary means or skill
or know-how or authority to do something
</def>
</posts>
<followers>
<follower>
<id>55
ERROR <----
</follower>
</followers>
ERROR <----
</users>
```

Path
T:\asu\3rd CSE\Second-term\Data structure\pdfExample.xml

output file path

Errors
```
Error: Tag not closed at Line 4Col.5
Error: Tag not closed at Line 12Col.13
Error: Tag not closed at Line 14Col.6
```

Correct errors

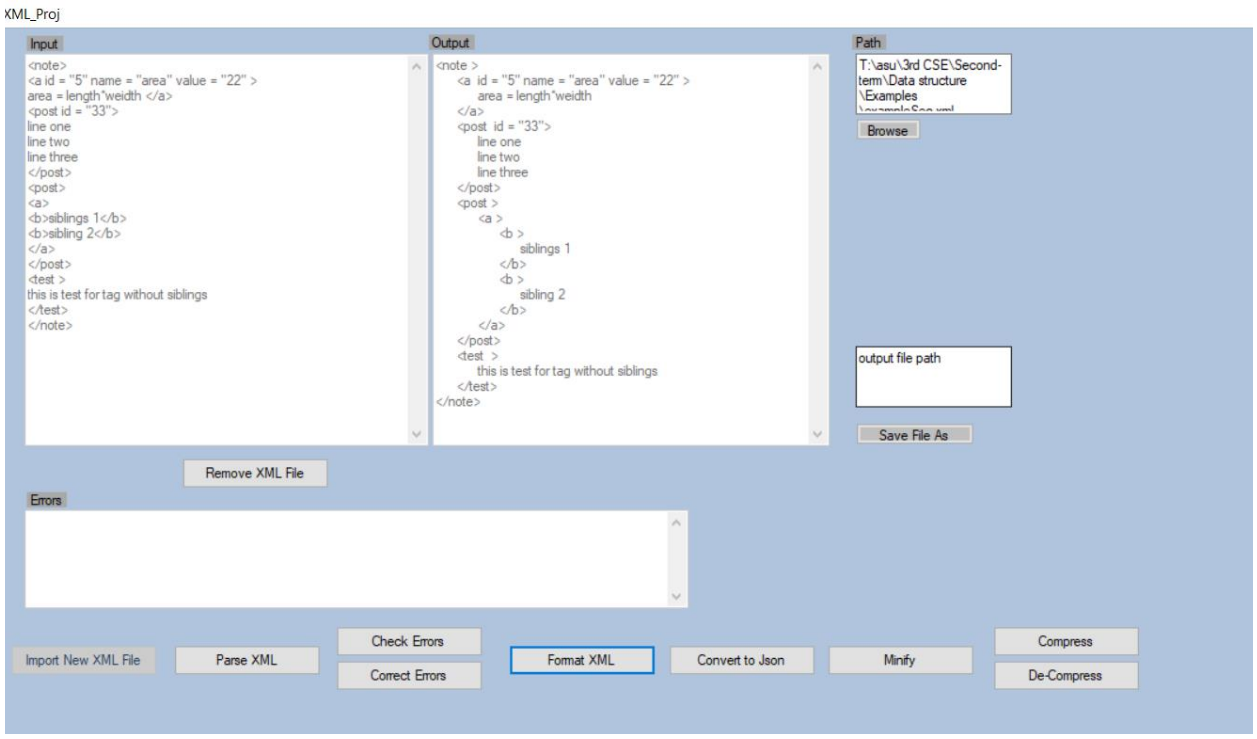


```
<?xml version="1.0" encoding="UTF-8"?>
<users>
<user>
<id>1
<name>first user</name>
<posts>
<post>
lorem ipsum sit amet, consectetur adisposcing elit
</post>
<def>(usually followed by 'to') having the necessary means or skill or
know-how or authority to do something
</def>
</posts>
<followers>
<follower>
<id>55
</follower>
</followers>
</users>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
<user>
<id>1
</id>
<name>first user</name>
<posts>
<post>
lorem ipsum sit amet, consectetur adisposcing elit
</post>
<def>(usually followed by 'to') having the necessary means or skill
or know-how or authority to do something
</def>
</posts>
<followers>
<follower>
<id>55
</id>
</follower>
</followers>
</user>
</users>
```

```
T:\asu\3rd CSE\Second-
term\Data structure
\pdfExample.xml
```

Format

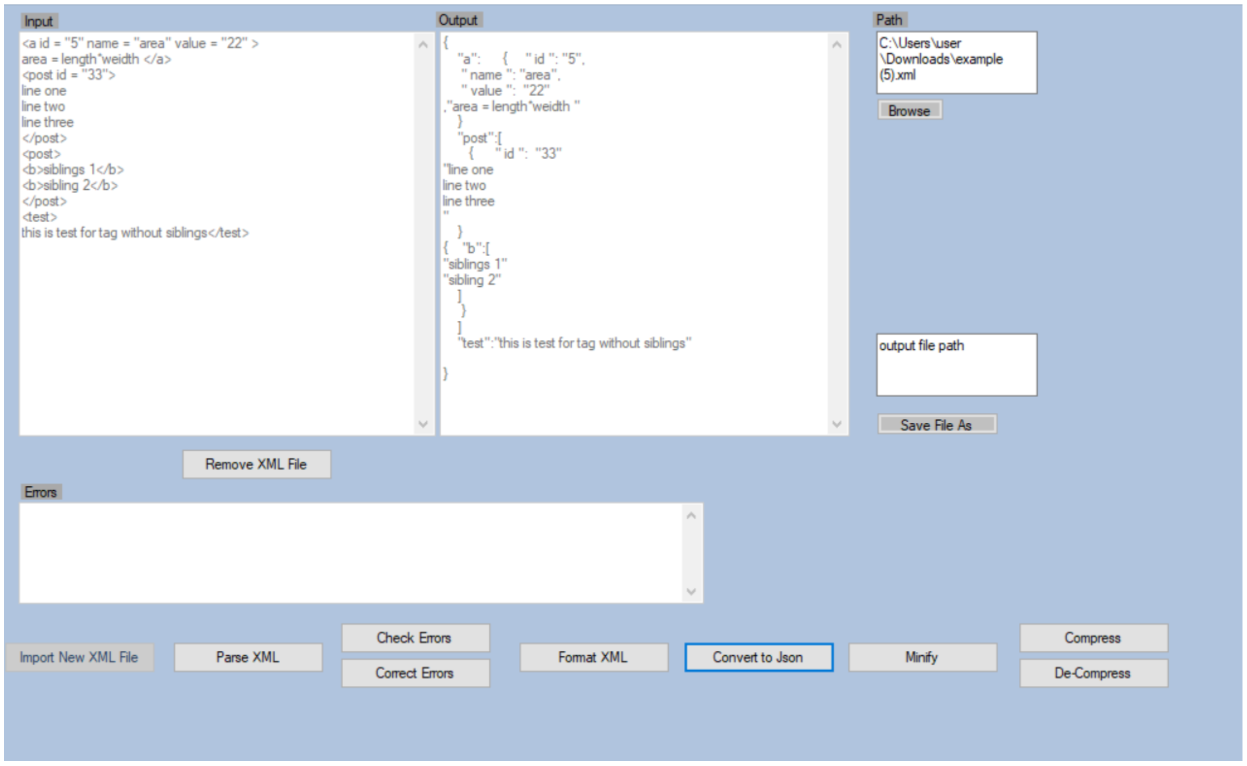


```
<note>
<a id = "5" name = "area" value = "22" >
area = length*weidth </a>
<post id = "33">
line one
line two
line three
</post>
<post>
<a>
<b>siblings 1</b>
<b>sibling 2</b>
</a>
</post>
<test >
this is test for tag without siblings
</test>
</note>
```

```
<note >
    <a id = "5" name = "area" value = "22" >
        area = length*weidth
    </a>
    <post  id = "33">
        line one
        line two
        line three
    </post>
    <post >
        <a >
            <b >
                siblings 1
            </b>
            <b >
                sibling 2
            </b>
        </a>
    </post>
    <test  >
        this is test for tag without siblings
    </test>
</note>
```

```
T:\asu\3rd CSE\Second-
term\Data structure
\Examples
```

Convert to json



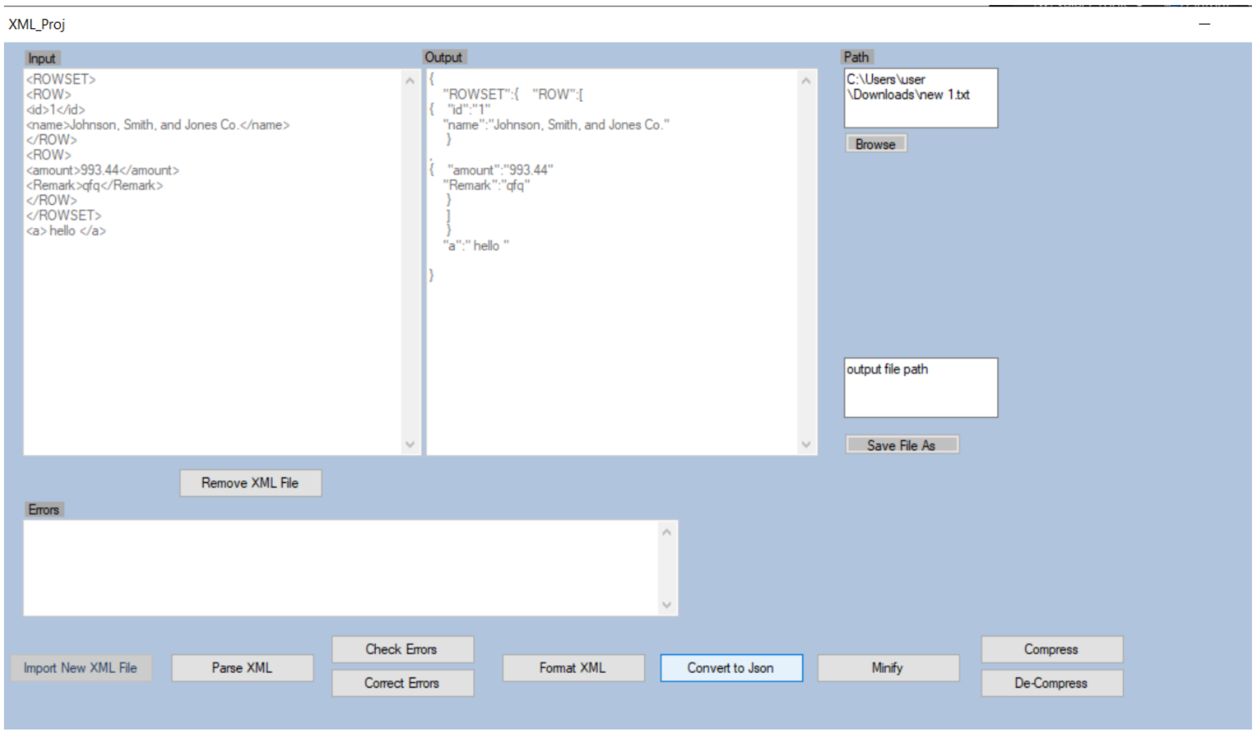
```
<a id = "5" name = "area" value = "22" >
area = length*weidth </a>
<post id = "33">
line one
line two
line three
</post>
<post>
<b>siblings 1</b>
<b>sibling 2</b>
</post>
<test>
this is test for tag without siblings</test>
```

```
{
   "a":    {    " id ": "5",
    " name ": "area",
    " value ":  "22"
,"area = length*weidth "
   }
   "post":[
     {    " id ":  "33"
"line one
line two
line three
"
   }
{   "b":[
"siblings 1"
"sibling 2"
   ]
    }
   ]
   "test":"this is test for tag without siblings"

}
```

```
C:\Users\user
\Downloads\example
(5).xml
```

```
output file path
```

2nd example



```
<ROWSET>
<ROW>
<id>1</id>
<name>Johnson, Smith, and Jones Co.</name>
</ROW>
<ROW>
<amount>993.44</amount>
<Remark>qfq</Remark>
</ROW>
</ROWSET>
<a> hello </a>
```

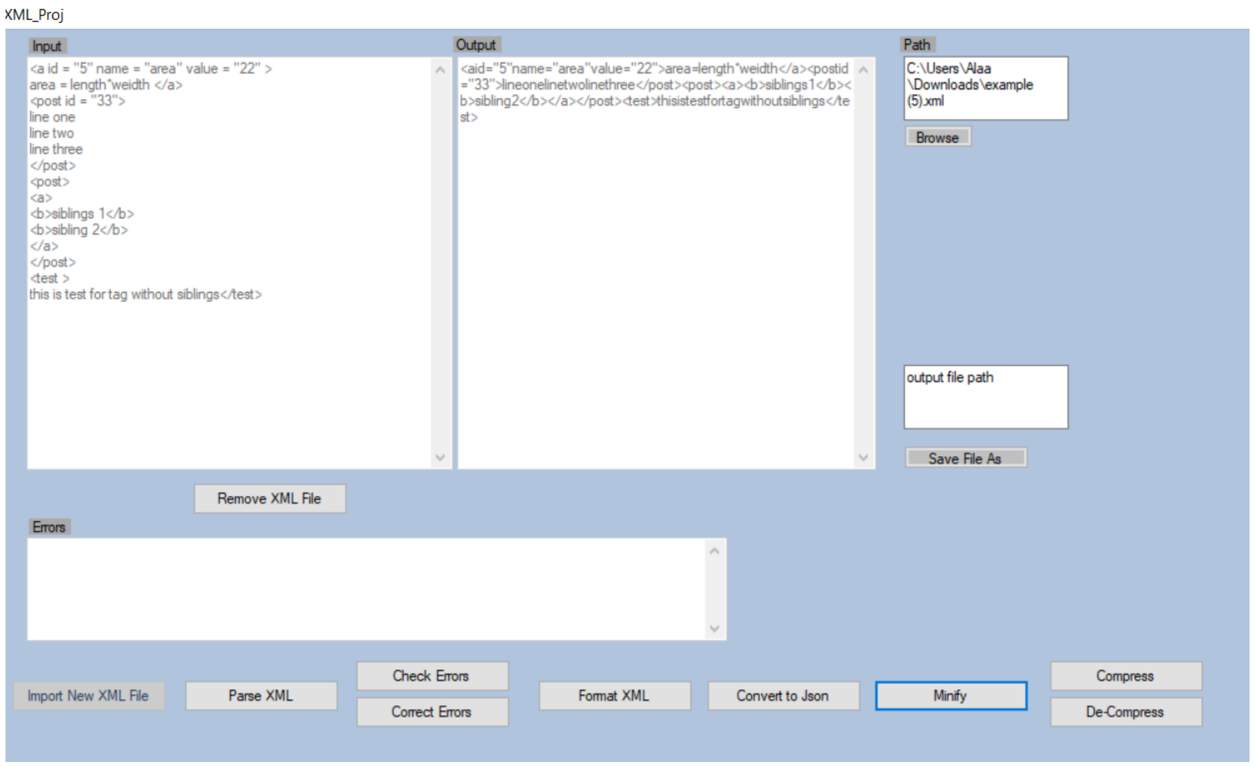
```
{
   "ROWSET":{   "ROW":[
{   "id":"1"
   "name":"Johnson, Smith, and Jones Co."
    }
,
{   "amount":"993.44"
   "Remark":"qfq"
    }
    ]
    }
   "a":" hello "

}
```

```
C:\Users\user
\Downloads\new 1.txt
```

```
output file path
```

Indentations are not supported, because the algorithm used depends on storing the values in the structure as mentioned before" in parsing", then rewrite it from the stored.

Minify



```
<a id = "5" name = "area" value = "22" >
area = length*weidth </a>
<post id = "33">
line one
line two
line three
</post>
<post>
<a>
<b>siblings 1</b>
<b>sibling 2</b>
</a>
</post>
<test >
this is test for tag without siblings</test>
```

```
<aid="5"name="area"value="22">area=length*weidth</a><postid
="33">lineonelinetwolinethree</post><post><a><b>siblings1</b><
b>sibling2</b></a></post><test>thisistestfortagwithoutsiblings</te
st>
```

C:\Users\Alaa
\Downloads\example
(5).xml

output file path

Compress and decompress will be implemented in appropriate way in the video, so we can show the difference in the size of the file after compressing.

5. References

1. Lu, Jiaheng, An Introduction to XML Query Processing and Keyword Search.
2. LZW (Lempel–Ziv–Welch) Compression technique - GeeksforGeeks.
3. Wikipedia.
4. We searched also to get more information in https://www.w3schools.com/xml/xml_whatis.asp
5. https://www.oxygenxml.com/doc/versions/23.1/ug-editor/topics/convert-XML-to-JSON-x-tools.html
6. https://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html?fbclid=IwAR3uSeH97adRW4ggQ9-yWJAAYMTjh7aFJPSfFDc7mVM-pR-OvhdWUmcNFvQ

6. Working files

- Repository link on github:
  https://github.com/AlaaShatat/Data-Structure-project
- Short video link
  https://drive.google.com/drive/folders/1sm73_dtK6MLdGTQj51qjSwVI7YBtTAph?usp=sharing