
**Faculty of Engineering, Ain Shams University**

**CSE331s - Data Structures and Algorithms**

# <<<<XML EDITOR PROJECT>>>>


| Name | ID |
|---|---|
| Saher Attia Farid | 1901612 |
| Hazem Hamdy Abd El-Sattar | 1900465 |
| Mohamed Emad Mostafa | 1900304 |
| Mohamed Ashraf Ibrahim | 1901607 |
| Ahmed Reda Kamal | 2001177 |

# Introduction:

This is XML editor that allows user to load, edit, save XML files and many other features with XML files. This XML editor can detect errors, show the number of errors, point to places of errors, and fix them to make the XML file consistent. Also, user can format XML file, minify it, convert it into JSON file, compress it to reduce its size ,and able to decompress it, user can generate graph visualization for social network represented by xml file, and get some important analysis from it.

## Features:

- **Browsing XML Files**
- **Checking the XML consistency**
- **Detecting & Correcting Error in consistency in XML File**
- **Format XML Files**
- **Minifying (Reducing file size by removing extra spaces).**
- **Compressing XML/JSON File**
- **Decompressing XML/JSON File**
- **Converting XML File to JSON File**
- **Graph representation**
- **Social Network Analysis**

# 1.XML Consistency:

- **Checking the XML consistency**

```
bool consistency_checker(string opent, string closedt, stack <string>& s);
bool check_consistency(vector <string> xml_vector);
```

checking consistency process is done using these two functions.

First function is a helper function that push or pop tags from the stack and return false if it faced error during this process.

Second function use the first function for each line in xml file saved in the form of **vector <string>** to check if the total file is consistent or not.

- **Detecting Errors in consistency in XML File**

```
struct err_data
{
    string err_type;
    int err_loc;
};
```

```
bool error_detector(string opent, string closedt, stack <string>& s, string& error_type);
vector <err_data> detect_error(vector <string> xml_vector);
```

Error detecting process is done using these two functions.

First function is a helper function that check if there is an error and if there is an error it detects the error type (missing open tag or missing closed tag or not matching tags) and put it in error_type variable.

The second function use the first one for each line to detect if there is error or not and get the error type if there is an error as it iterates on the xml line by line, then it stores the error type and location in a **vector<err_data>** then return it at the end of the file

Note: error type detection process is done by the following.

If the closed tag is not matching the top of the stack then there are two option that there is missing open tag or missing closed tag.

If the current closed tag exist in the stack but not the top then the type of error is missing closed tag = top of the stack

If the current closed tag doesn't exist in the stack, then the type of error is missing open tag = the current closed tag

- **Correcting Error in consistency in XML File**

```
vector <string> error_corrector(vector <string> xml_vector, vector <err_data> error_vector);
```

The correction process is done by this function that takes the vector <err_data> result from error detection process and the original vector containing xml data the return corrected xml vector by adding the missing open or closed tags to the original vector.

Note: error correction process may not provide a hundred  percent correct xml file as it only solve errors in consistency like missing open tag or missing closed tag or not matching tags. But it will provide you with hundred percent consistent xml.

| | **Time complexity** | **Space complexity** |
|---|---|---|
| **Check consistency** | **O(n)** | **O(n)** |
| **Detect errors** | **O(n)** | **O(n)** |
| **Correct errors** | **O(n)** | **O(1)** |

# 2.XML Formatting:

```
#include"Helpers.h"

void format(vector <string> xml_vector);
```

the formatting process is done by the above function.

First the xml file is read then stored in format of vector <string> that is used in all the previous functions. in this process all leading or ending spaces got deleted for each line in xml file then the line is stored as a string in the xml_vector.

For format function it takes that vector and print it in a new file “formatted_file”. The algorithm used is simple as whenever there is open tag it print it the increase the number of spaces and whenever there is closed tag it decrease the number of spaces first then print the closed tag

**Example for input and output files:**

```
<users>
  <user>
                      <id>1</id>
      <name>Ahmed Ali</name>
   <posts>
          <post>
    <body>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
            </body>
                                              <topics>
                <topic>
                    economy
     </topic>
                <topic>
                    finance
                                          </topic>
            </topics>
                       </post>
     <post>
            <body>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
   </body>
```

```
<users>
   <user>
      <id>1</id>
      <name>Ahmed Ali</name>
      <posts>
         <post>
            <body>
               Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
            </body>
            <topics>
               <topic>
                  economy
               </topic>
               <topic>
                  finance
               </topic>
            </topics>
         </post>
         <post>
            <body>
               Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
            </body>
```

| | Time complexity | Space complexity |
|---|---|---|
| **XML Format** | **O(n)** | **O(1)** |

## 3.Minifying:

this process is done by function **Minify** that exist in format file

the function takes xml vector representing xml and return a one string without any new lines or spaces

in creating xml vector process leading and ending spaces got deleted for each line and then stored.so he Minify function role is to provide one string from this vector

<users><user><id>1</id><name>Ahmed Ali</name><posts><post><body>saher dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic>economy</topic><topic>finance</topic></topics></post><post><body>saher attia dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic>economy</topic></topics></post></posts><followers><follower><id>2</id></follower><follower><id>3</id></follower></followers></user><user><id>2</id><name>Yasser Ahmed</name><posts><post><body>saher attia dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic>economy</topic></topics></post></posts><followers><follower><id>1</id></follower></followers></user><user><id>3</id><name>Mohamed Sherif</name><posts><post><body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</body><topics><topic>sports</topic></topics></post></posts><followers><follower><id>1</id></follower></followers></user></users>

This is an example for the xml file after being minified all he file become one line.

| | Time complexity | Space complexity |
|---|---|---|
| **XML Format** | **O(n)** | **O(1)** |

# XmlTree construction:

xml tree is a tree of nodes. Each node has name, value and children as the following:

```
class Node {
public:
    std::string name;
    string value;
    std::vector<Node*> children;
};
```

Then the tree class that has a root node and some functions as build xml tree and print xml tree functions as following:

```
class Tree
{
private:
    Node* Root;

    void _print_xml(Node* node);

public:
    void bulid_xml_tree(vector <string> xml_vector);
    void print_xml();
    Node* getRoot();
};
```

Process of building xml tree is done using stack as build function take xml vector as an input an then start to build tree as following:

1. First Create an empty stack to store the elements and their relationships.
2. Iterate over the xml vector and extract individual tokens
3. For each token, do the following:
    - If the token is an opening tag, put it as child for the current node(at the top of the stack) and then push it onto the stack to make it the current node
    - If the token is an attribute, store it with the current element on the top of the stack.
    - If the token is closing tag, pop the top element from the stack.

4. Repeat steps 3 until all tokens have been processed.

Converting xml files into xml tree is very useful and make it easier to deal with xml data and so can use it in other operations like converting xml to json and build a graph representing the data to make it easier to make some analysis.

| | **Time complexity** | **Space complexity** |
|---|---|---|
| **XML Format** | **O(n)** | **O(n)** |

Note: The space complexity of this process is O(n), where n is the maximum depth of the XML tree.

Time complexity is O(n) where n is number of lines in xml file or number of tokens in xml file.