Computers and Systems Department

Data Structures

XML-Editor Project Details

Member Name	ID Code	Contributions
مصطفى عبدالفضيل محمد ثابت	1701435	- Struct attribute - Class node - Class Tree - Insert function in Tree - Convert to JSON
اسلام مجدي محمد الهادي	1700251	- Spa() - Format (Prettifying) - The frontend (GUI)
عبدالرحمن محمد حسين عبدالقادر	1700759	- Spaces() - Consistency - Similar function in the Tree
مى طارق سعد بدوي	1701510	- Assistant Functions - Print attributes in the Tree - Minifying

• Background

- **XML stands for eXtensible Markup Language.**
- **XML is a markup language much like HTML.**
- **XML was designed to store and transport data.**
- **XML was designed to be self-descriptive.**
- **XML is a W3C Recommendation.**
- **XML simplifies data sharing.**
- **XML simplifies data transport.**
- **XML simplifies platform changes.**
- **XML simplifies data availability.**

• Program specifications

- **We have created a program runs as XML editor that take an input file in XML format and do some operations on it such as checking its consistency, prettifying its shape, minifying its size on disk, and converting its XML extend to JSON extend.**

- **We have built a simple GUI to use it to input the XML file to do operations mentioned above and deliver the output of each operation on it.**

- **We have built a function to check the file consistency.**

- **We have built another function to format(prettifying) the file.**

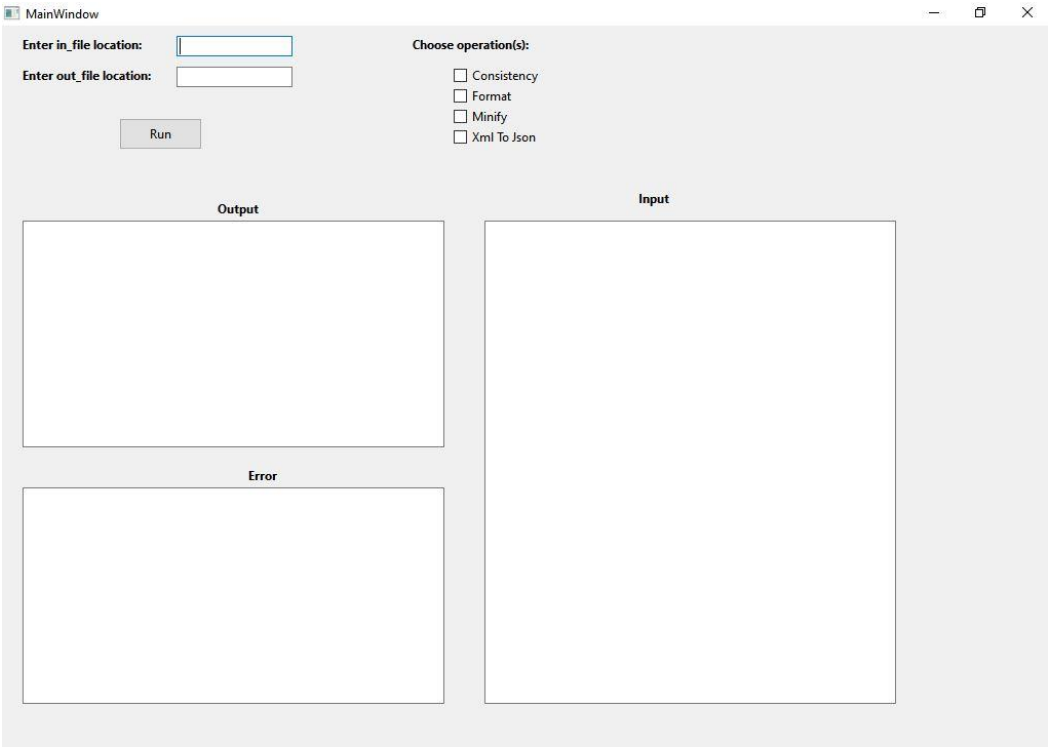
- **We have built a tree data structure to describe the XML file.**

- **We have built a function to convert the file format to JSON format.**

- **We have built another function to minify the file size on disk as much possible.**

• Now in the next section, we will represent further the user program (GUI), data structures and algorithms used, and functions implementation in more details.

- # User program (GUI)



- User inputs

1- **User should enter the path of the XML file in the text box “Enter in_file location”.**
2- **Then, the user should choose his operation from “choose operation(s)” check box (user can choose more than one operation and the operations will be executed in the order of the list”.**
3- **User should enter the path of the XML file in the text box “Enter out_file location”.**
4- **Then, the user should click button “Run” to get the output.**

- Output

The user has two output windows “Output” and “Error”.

1- **“output” window shows the output of any operation of (“Format” or “Minify” or “XML to Json”).**
2- **If the user chooses “Consistency” operation the output will be shown on the two windows “Output” and “Error”.**

• Operations in details

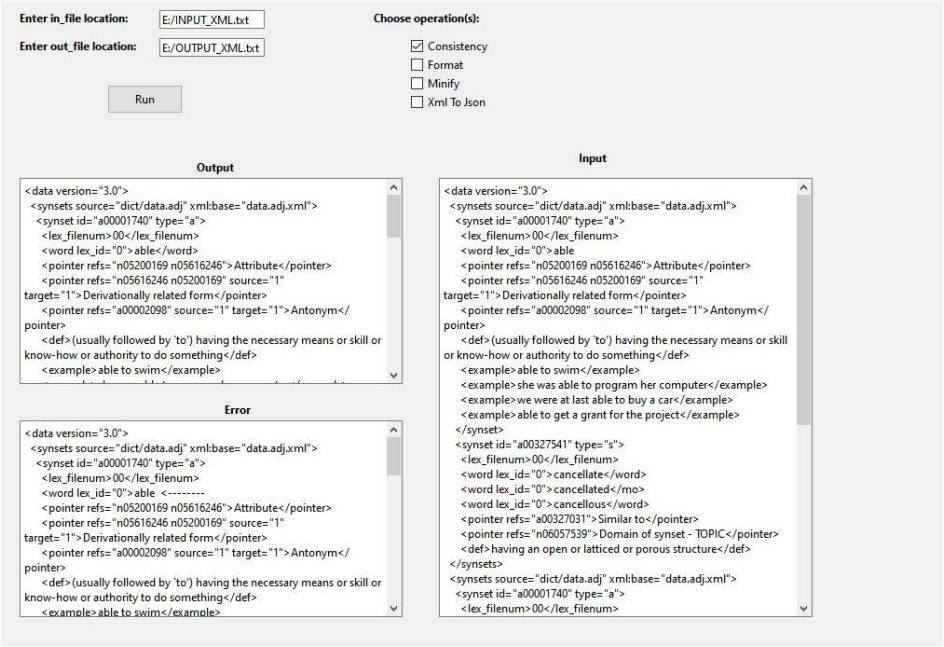
1. Check consistency

- Specifications

- **XML file is consistent if each opening tag is closed by closing tag in the same level.**
- **The program covers three cases of inconsistency:**
  1- **If a leaf closed tag has been missed.**
  2- **If a leaf closed tag has been closed incorrectly.**
  3- **If a non-leaf closed tag has been missed.**

- **The program takes XML file as input and check its consistency, if the file is consistent return the same file but if it is not consistent return two files:**
  1- **an error file contains the input file having arrows point at the locations of the inconsistency errors.**
  2- **A file contains the correctness of the in consistent file.**

- User input specification



- **Enter the input file location and the output file location.**
- **Choose "Consistency" operation.**
- **Press "Run"**

- Data structures and algorithms

- **We used "Consistency()" function to represent this operation that takes input XML file and check its consistency.**

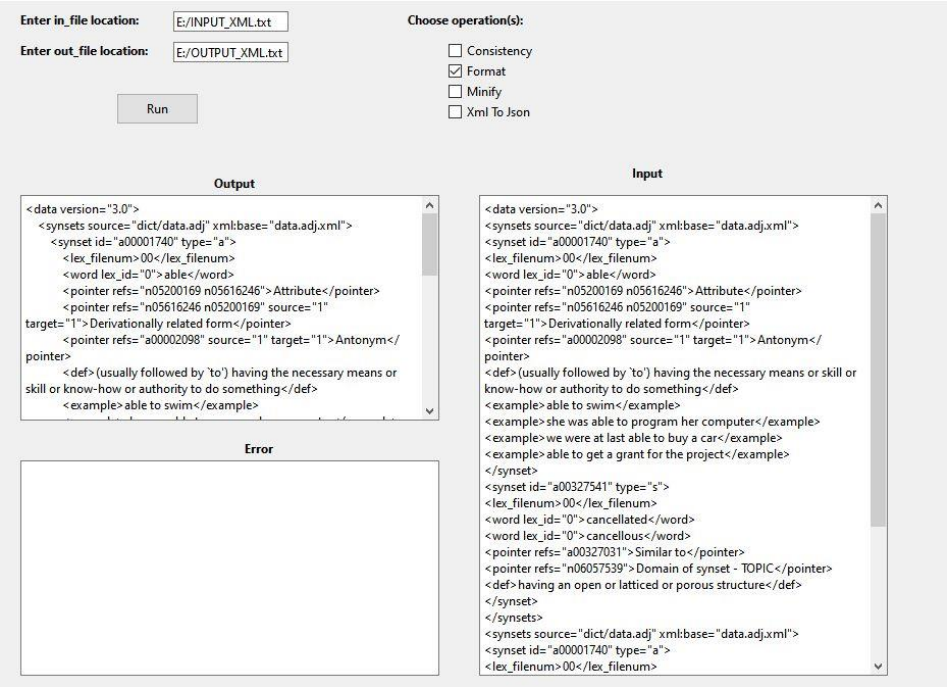
- **If the file is consistent return the same file, if not, then return a file contains arrows points to the errors locations and another file contains the correctness of the file.**
- **Spa(string): return index of first char of the string.**
- **Space1(int): return string of n spaces.**
- **Stack: to push open tags to check consistency.**
- **Queue: to print string on the output file (FIFO)**
- **Consistency():**
    - **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.**
    - **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.**
    - **Print the content of the queue.**

2. Format (Prettifying)

- Specifications

- **XML file is formatted if each tag has been typed in its level and both open tag and closed tag of the same tag has been typed in the same level.**
- **The program takes XML file as input and format it and return formatted file.**

- User input specification



- **Enter the input file location and the output file location.**
- **Choose “Format” operation.**
- **Press “Run”**

- Data structures and algorithms

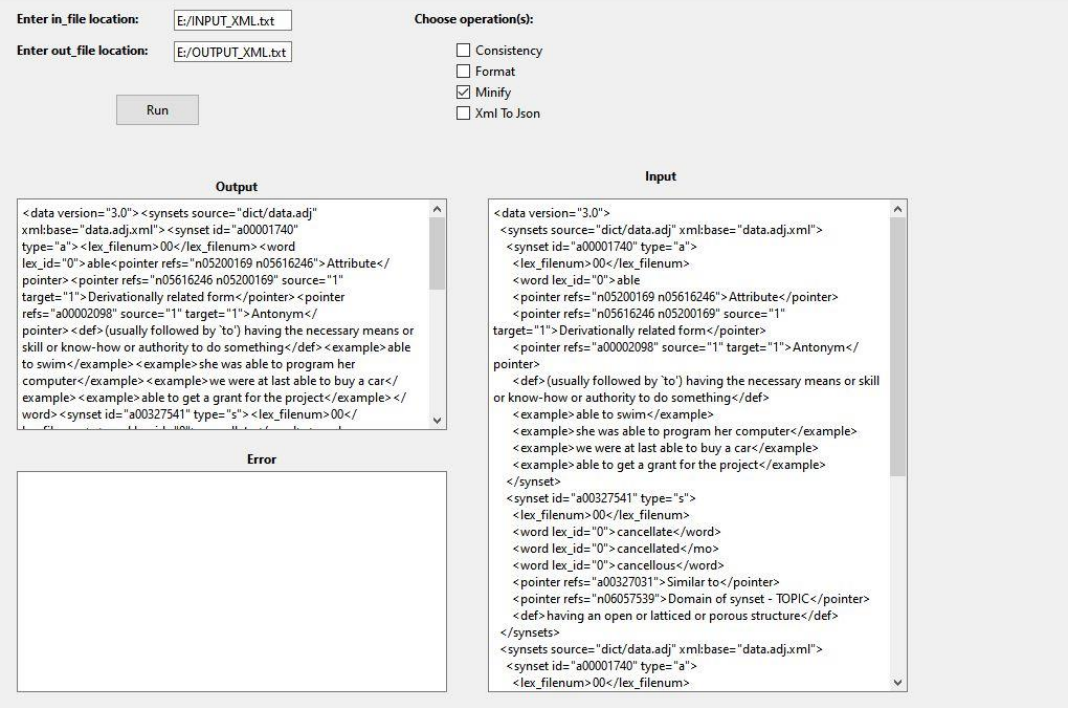
- **We used “Format()” function to represent this operation that takes input XML file and format(Prettifying) it.**
- **Stack: to push open tags to check consistency.**
- **Queue: to print string on the output file (FIFO)**
- **Space(int): return string of n spaces.**
- **Format():**
    - **Use int variable (nu) to send it to “Space(int)”.**
    - **If the input line is opening tag, push in the stack, nu++, push the return of Space(level) and push the open tag in the queue.**
    - **If closed tag, pop the top of the stack, nu--, push the desired spaces returns from “Space()” and the closed tag in the queue.**
    - **If the line is text, push the desired spaces returns from “Space()” and the Text in the queue.**
    - **If the open and closed tag in the same line, push the desired spaces returns from “Space()” and the whole line in the queue.**
    - **Print the whole queue.**

3. Minifying the file size on disk

1- Specifications

- **XML file formatted input has a large size on the disk of the computer, so we need to minify this file as much possible using some techniques.**
- **The technique followed is to print each line in the large file in another line but all lines behind each other and so we have could remove the indentations and white spaces.**

- User input specification



- **Enter the input file location and the output file location.**
- **Choose "Minify" operation.**
- **Press "Run"**

- Data structures and algorithms

- **Assistant functions:**
  - **FindFirstAngle(string): return the index of first angle bracket.**
  - **Num_Of_Angle(string): return the number of angle brackets.**
  - **Tokens(string): return a vector contains the string splitted**
- **Stack**
- **Queue**
- **Struct "attribute":**

  "attribute" represents the attribute of the tag in the following data members.

- **Data members:**
  - **Name: its name**
  - **Value: its value**
  - **Level: its level in the tree**

- **Class "node":**

  "node" represents each tag in node object.

  - **Data members:**
    - **String tag: represent the whole tag with its attributes.**
    - **String name: represent the tag without its attributes.**
    - **String JsonName: represent the name of the tag in JSON format.**
    - **Bool JsonMark: set true if the tag is repeated in the same level.**
    - **Vector<attribute> attrs: represent the tag attributes.**
    - **Int level: represent the node level.**
    - **Node* parent: represent the parent of the current node.**
    - **Vector<node*> childs: represent the childs of the current node.**
  - **Methods members:**
    - **Constructors: node(), node(string) to set name.**
    - **Set_tag(string), get_tag().**
    - **Set_name(string), get_name().**
    - **Set_JsonName(string), get_JsonName().**
    - **Set_Mark(bool), get_Mark().**
    - **Set_attributes(vector<string>), get_attributes().**
    - **Set_level(int), get_level().**
    - **Set_parent(node*), get_parent().**
    - **Set_childs(vector<node*>), get_childs().**

- **Class "Tree":**

  "Tree" represents the tags nodes in tree structure.

  - **Data members:**
    - **Node* root: represent the root of the tree.**
  - **Methods members**
    - **Constructor: Tree().**
    - **Get_root().**
    - **Empty(): return true if empty tree and false if not.**
    - **Similar(vector<node*>):**
      - **This function takes a vector of any Childs's node and return a 2D vector.**
      - **Each row vector contains a child if this child does not have similars.**
      - **if repeated, the row will contain the similar childs.**
    - **Insert(string):**

- **This function takes the XML file location and trace it to insert all nodes of the tree.**
- **If the tree is empty, it sets the root tag as the root node of the tree and set the parent of this node to (NULL) and if the following tag is opening tag, then it is a child of the root and set this node to the child's vector of the root and set the parent of this node.**
- **but if not, there is three possible cases for the following tag "Text" or "closing tag" or "opening".**
- **If "Text", then it is a leaf node and set the "Text" as child of the current node and set its parent.**
- **If "closing tag", then this level has been ended.**
- **If "opening tag", then set this node as child to the current parent and set its parent.**

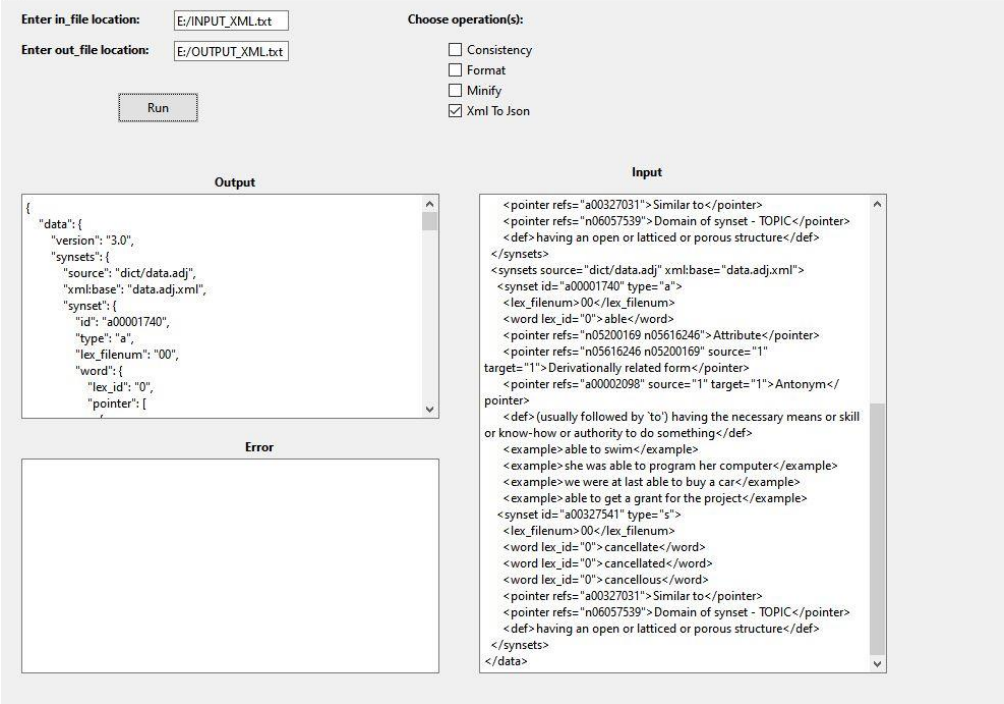
- **Print_attributes(node*)**
- **Print_minify(node*):**
  - **This function prints the minified file all lines behind each other without indentations, white spaces, or next lines.**
  - **It is a recursive function that firstly takes the root of the tree after inserting the XML file in it using "insert" function.**
  - **Print the tag of the root.**
  - **Print_minify() its childs recursively.**
  - **Print the closing tag of current node.**
- **Print_Json(node*): we will describe this function in the next section.**

4. Convert XML format to JSON Format

- Specifications

- **We need to convert the input XML file from its format (XML format) to another known format called Json format**

- User input specification



- **Enter the input file location and the output file location.**
- **Choose “Xml To json” operation.**
- **Press “Run”**

- Data structures and algorithms

- **Stack**
- **Queue**
- **Struct “attribute”**
- **Class “node”**
- **Class “Tree”:**
    - **Print_Json(node*):**
        - **This function takes the root of the tree file that need to be converted to JSON format.**
        - **print Json_Name of the node in its node level followed by ‘{‘ and after printing print ‘}’.**
        - **Print its attributes and Store its childs in 2D vector using “similar()” function.**
        - **Recursively print_Json() of the childs in the 2D vector.**
        - **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.**

5. More than one operation