 [1]

Data Structure Project – XML Editor

**Program:**

***Course Code: CSE323***

***Course Name: Data Structure***

***Submitted to:***

***Dr : Islam El-maddah***

**Eng, Amr Ahmed**

**Ain Shams University**

**Faculty of Engineering**

**CSE Department**

**2021**

**Student Personal Information for Group Work**

|  |
| --- |
| **Student Codes:** |

1700267

1700271

1700462

1601583

|  |
| --- |
| **Student Names:** |

Alaa Ibrahim Mohamed Ibrahim Amer

Alaa Shaaban Hussien Ali Shatat

Hussein Mahmoud Fouad El-Sayed

Nahla Mostafa Abdelkareem

Table of Contents

[1. Background 4](#_Toc77190579)

[2. Implementation details 4](#_Toc77190580)

[2.1 The algorithm used in the XML editor 4](#_Toc77190581)

[2.2 Operations 5](#_Toc77190582)

[2.2.1 Parsing Function 5](#_Toc77190583)

[2.2.2 Correction Function 5](#_Toc77190584)

[2.2.3 Formatting Function 5](#_Toc77190585)

[2.2.4 ConvertToJason Function 5](#_Toc77190586)

[2.2.5 Trim Function 5](#_Toc77190587)

[2.2.6 Compress Function 5](#_Toc77190588)

[3. Complexity of operations 5](#_Toc77190589)

[4. References 5](#_Toc77190590)

[5. Working files 5](#_Toc77190591)

# Background

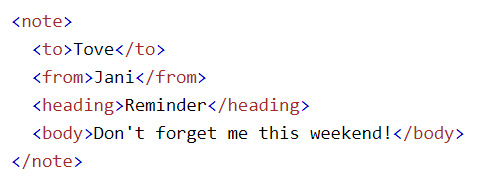
**XML** stands for eXtensible Markup Language, it is similar to HTML language “used in web”, XML was designed to store and transport data. XML is a language that defines a set of rules for encoding a document in a format that is both human-readable and machine-readable. [1]

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](https://en.wikipedia.org/wiki/Attribute%E2%80%93value_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: 

# Implementation details

* Language used C#.

## 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.

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.

## 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.

### Parsing Function

### 2.2.2 Correction Function

### 2.2.3 Formatting Function

### 2.2.4 ConvertToJason Function

### 2.2.5 Trim Function

### 2.2.6 Compress Function

#### 2.2.6.1Reasons 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.

#### 2.2.6.2Lempel–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.

#### 2.2.6.3 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.

#### 2.2.6.4 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.

#### 2.2.6.5. 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

6.4.2. Decompression

In our code, the decompressing code is different from the decompression method in the LZU 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.

# Complexity of operations

# References

|  |  |
| --- | --- |
| [1] | J. Lu, "AN Introduction to XML Query Processing and Keyword Search," in *AN Introduction to XML Query Processing and Keyword Search*. |
|  |  |

# Working files

* Repository link on github:

<https://github.com/AlaaShatat/Data-Structure-project>

* Short video link

# 