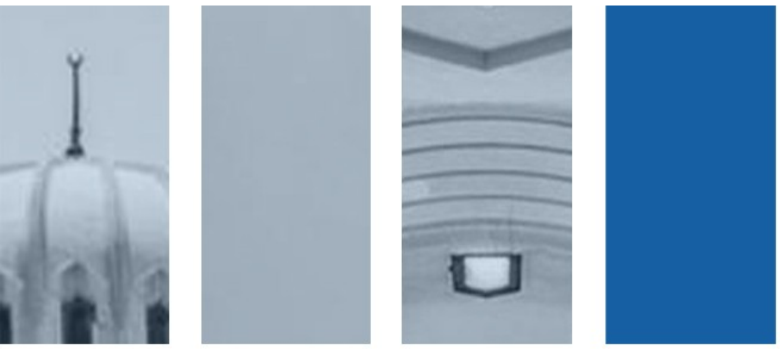


RESEARCH & PROJECT SUBMISSIONS

Program: Computer and Systems

Course Code: CSE323

Course Name: Programming with Data Structures

Examination Committee

Dr. Isalm Ahmed Mahmoud Elmadah

Prof. Hossam Fahmy

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

Student Personal Information for Group Work

Student Names:	Student Codes:
أحمد عبد الحكيم عبد الله	1600122
أحمد علاء محمود	1600133
أحمد حسام كمال	1600057
عبد الله ناصر عبد الله	1600813
طه محمد طه	1600714

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: أحمد عبد الحكيم عبد الله محمد **Date:** **30/5/2020**

Signature/Student Name: أحمد علاء محمود عبد الحميد

Signature/Student Name: أحمد حسام كمال محمود

Signature/Student Name: عبد الله ناصر عبد الله

Signature/Student Name: طه محمد طه

Submission Contents

Github Repository Link: https://github.com/AhmadAbdElHakim/XML-Editor
Video Link: https://www.youtube.com/watch?v=lKnPOPX8FK8

01 Background

First Topic

Tree:

The tree data structure (a data organization, management, and storage format that enables efficient access and modification) is a widely used abstract data type that simulates a hierarchical tree structure, with a root value and sub-trees of children with a parent node, represented as a set of linked nodes.[1]

Trees have many forms, one of which is the XML tree, where XML documents have a hierarchical structure and can conceptually be interpreted as a tree structure. XML documents must contain a root element (one that is the parent of all other elements).[2]

All elements in an XML document can contain sub elements, text and attributes. The tree represented by an XML document starts at the root element and branches to the lowest level of elements.[3]

02 Implementation Details

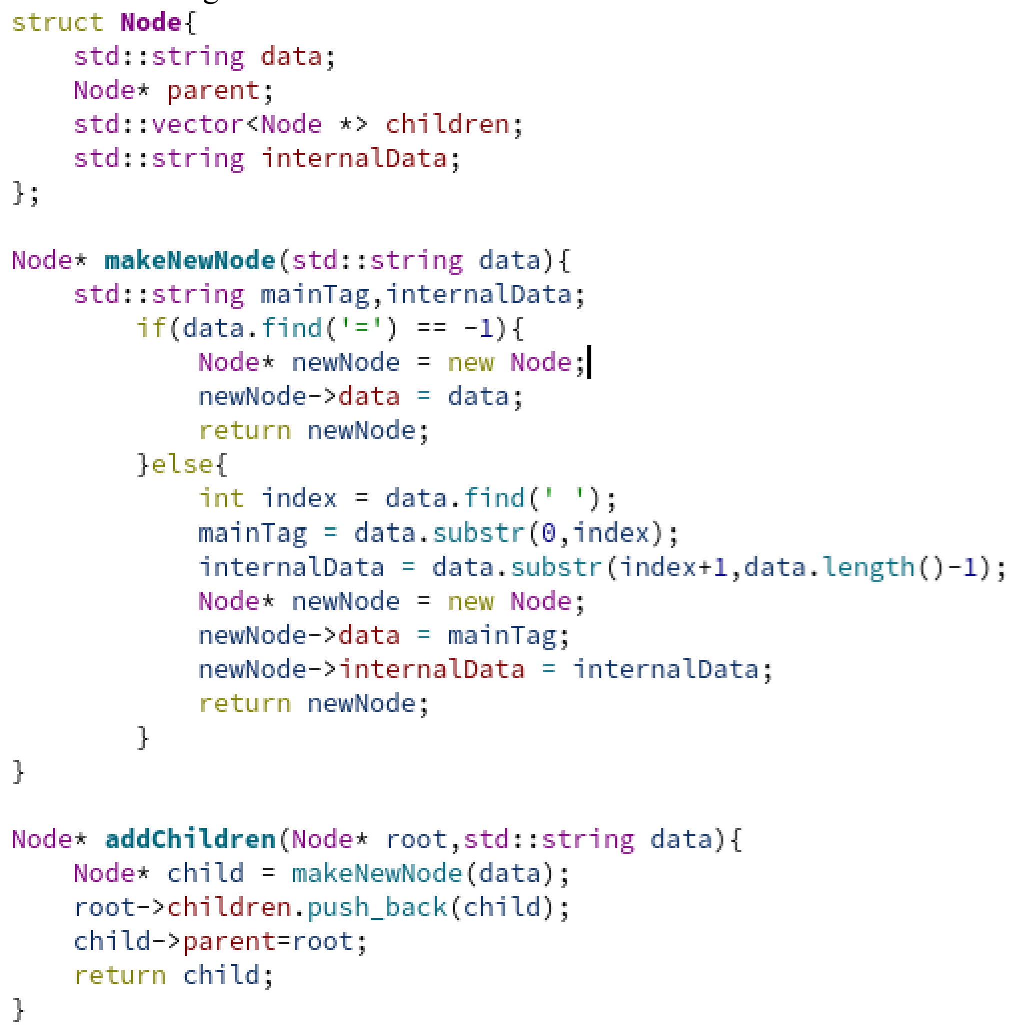
Second Topic

The main idea of this project is to represent the XML file as a general N-ary tree (a rooted tree in which each node has no more than N children).

1) Classes

Node Struct:

Node* makeNewNode(std::string data), and Node* addChildren(Node* root,std::string data) functions are the building blocks for the node class.



```
struct Node{
    std::string data;
    Node* parent;
    std::vector<Node *> children;
    std::string internalData;
};

Node* makeNewNode(std::string data){
    std::string mainTag,internalData;
        if(data.find('=') == -1){
            Node* newNode = new Node;
            newNode->data = data;
            return newNode;
        }else{
            int index = data.find(' ');
            mainTag = data.substr(0,index);
            internalData = data.substr(index+1,data.length()-1);
            Node* newNode = new Node;
            newNode->data = mainTag;
            newNode->internalData = internalData;
            return newNode;
        }
}

Node* addChildren(Node* root,std::string data){
    Node* child = makeNewNode(data);
    root->children.push_back(child);
    child->parent=root;
    return child;
}
```

2) Helping Functions

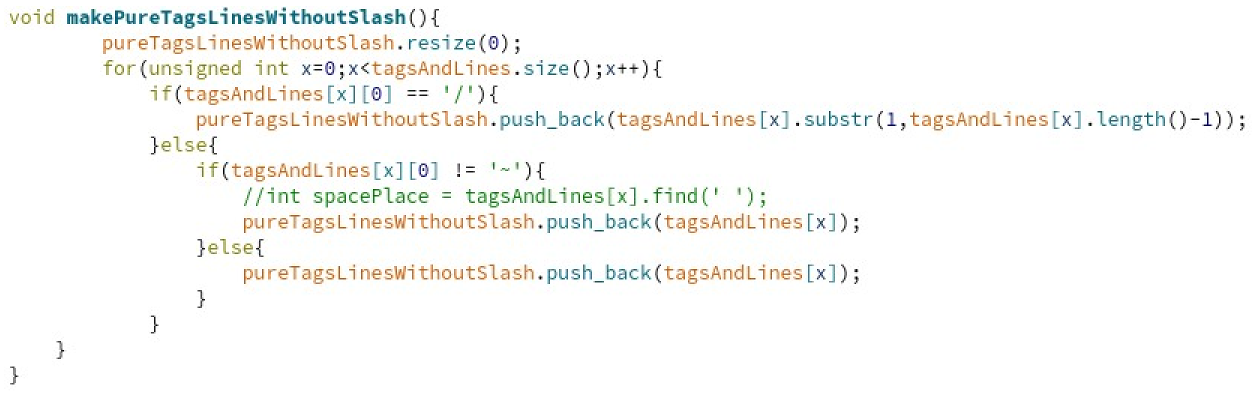
1) getTagsAndLines() function is used to sort lines according to 1-tags only stored in tags vector, 2-tags and sentences stored in tagsAndLines vector.

```
void getTagsAndLines(){
    std::vector<std::string> TandL;
    tags.resize(0);
    tagsAndLines.resize(0);
    for(unsigned long long x=0;x<lines.size();x++){

    int tagCounter = std::count(lines[x].begin(), lines[x].end(), '<');
    int place1 = lines[x].find('<');
    int place2 = lines[x].find('>');

    for(int m=0;m<tagCounter;m++){

        tags.push_back(lines[x].substr(place1+1,place2-place1-1));
        TandL.push_back(lines[x].substr(place1+1,place2-place1-1));

        if(lines[x][place2+1] != '<'){
            int temp = lines[x].find('<',place1+1);

            TandL.push_back("~"+lines[x].substr(place2+1,temp-place2-1));
        }

        int place3 = lines[x].find('<',place1+1);
        int place4 = lines[x].find('>',place2+1);
        place1 = place3;
        place2 = place4;
    }
  }
  for(unsigned long long x=0;x<TandL.size();x++){      //to remove empty lines
    if(! ((TandL[x][0] == '~') && (TandL[x].length() == 1)) ){
        tagsAndLines.push_back(TandL[x]);
    }
  }
}
```

2) makePureTags() function is used to separate line of tags from extra data ex. (ahmed id="1") ---> (ahmed),stored in pureTags vector.

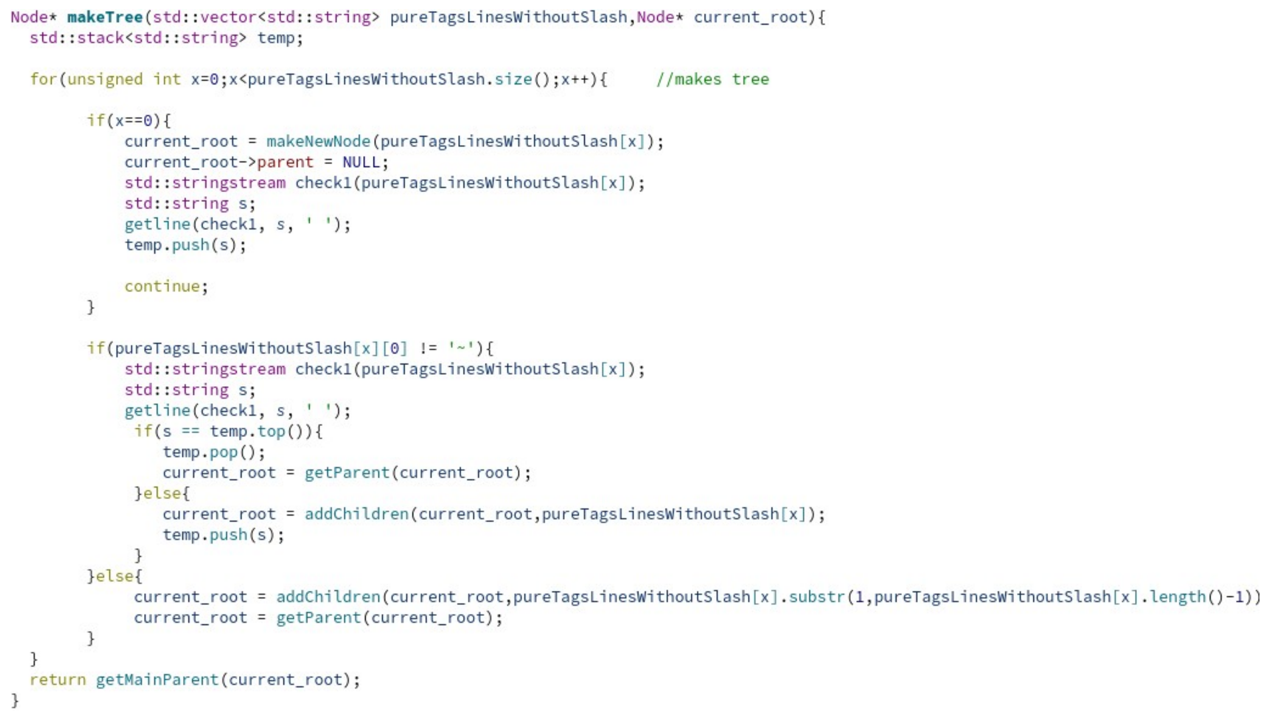
```
void makePureTags(){
    pureTags.resize(0);
  for(unsigned long long x=0;x<tags.size();x++){
    if(!tags[x].empty()){
        if(tags[x].find(' ') != std::string::npos){
            std::string s = tags[x].substr(0,tags[x].find(' '));
            pureTags.push_back(s);
        }else{
            pureTags.push_back(tags[x]);
        }}}}
```

3) makePureTagsLinesWithoutSlash() function is used to make tags without slash,stored in pureTagsLinesWithoutSlash vector, this vector contain openTags without slash or data, closeTag without slash, data start with ~ sign.



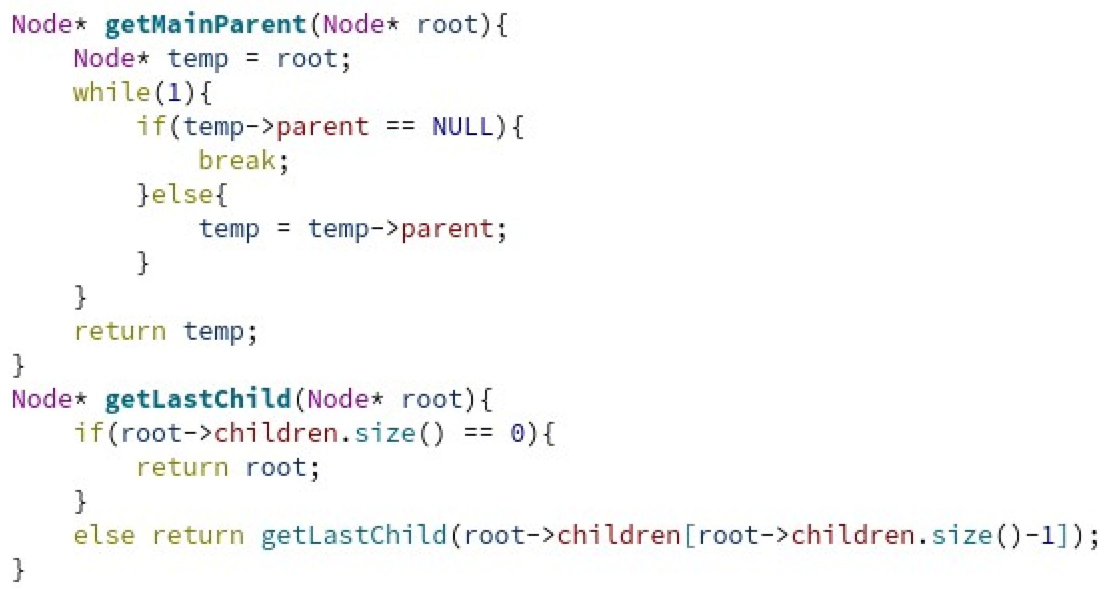
```
void makePureTagsLinesWithoutSlash(){
        pureTagsLinesWithoutSlash.resize(0);
        for(unsigned int x=0;x<tagsAndLines.size();x++){
            if(tagsAndLines[x][0] == '/'){
                pureTagsLinesWithoutSlash.push_back(tagsAndLines[x].substr(1,tagsAndLines[x].length()-1));
            }else{
                if(tagsAndLines[x][0] != '~'){
                    //int spacePlace = tagsAndLines[x].find(' ');
                    pureTagsLinesWithoutSlash.push_back(tagsAndLines[x]);
                }else{
                    pureTagsLinesWithoutSlash.push_back(tagsAndLines[x]);
                }
            }
    }
}
```

4) Node* **makeTree**(std::vector<std::string> pureTagsLinesWithoutSlash, Node* current_root) function is the building block of the conversion to JSON, it has the pureTagsLinesWithoutSlash vector of strings, and a pointer to a node as arguments when called. It used to make the tree, and returns the main node.



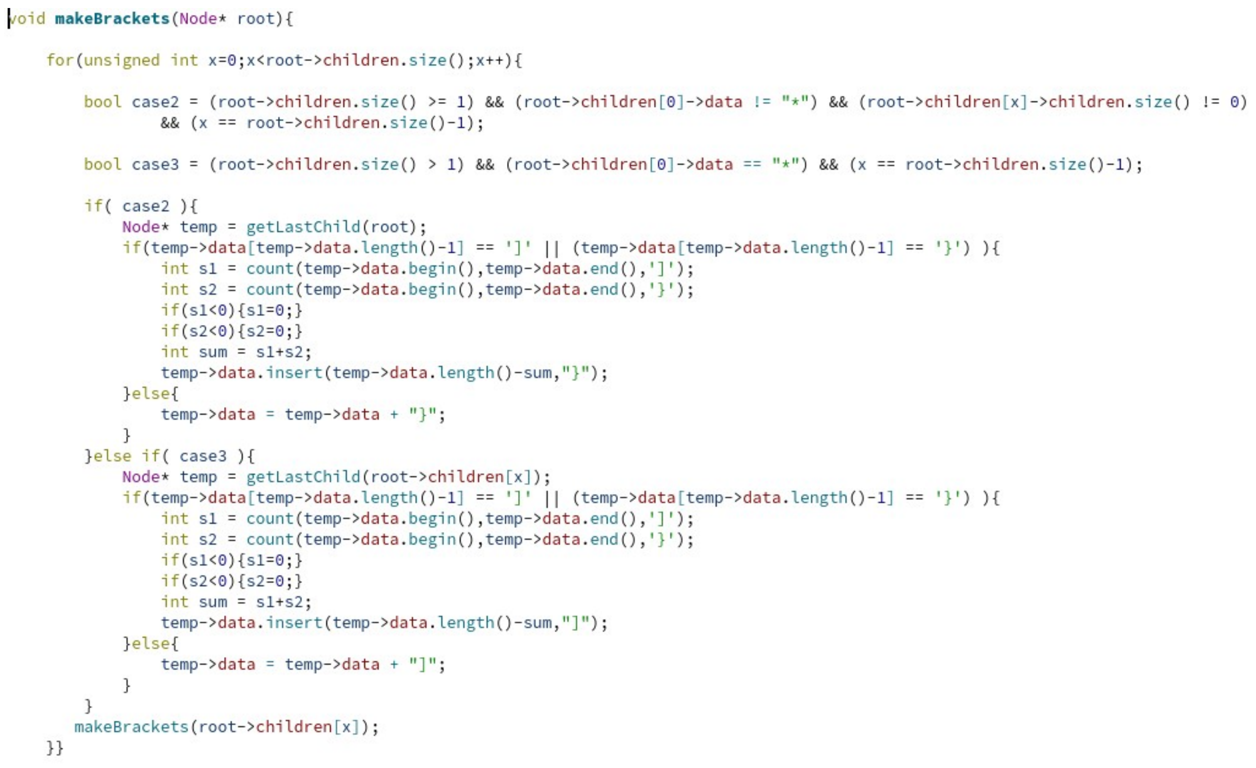
```
Node* makeTree(std::vector<std::string> pureTagsLinesWithoutSlash,Node* current_root){
  std::stack<std::string> temp;

  for(unsigned int x=0;x<pureTagsLinesWithoutSlash.size();x++){     //makes tree

        if(x==0){
            current_root = makeNewNode(pureTagsLinesWithoutSlash[x]);
            current_root->parent = NULL;
            std::stringstream check1(pureTagsLinesWithoutSlash[x]);
            std::string s;
            getline(check1, s, ' ');
            temp.push(s);

            continue;
        }

        if(pureTagsLinesWithoutSlash[x][0] != '~'){
            std::stringstream check1(pureTagsLinesWithoutSlash[x]);
            std::string s;
            getline(check1, s, ' ');
             if(s == temp.top()){
                temp.pop();
                current_root = getParent(current_root);
             }else{
                current_root = addChildren(current_root,pureTagsLinesWithoutSlash[x]);
                temp.push(s);
             }
        }else{
             current_root = addChildren(current_root,pureTagsLinesWithoutSlash[x].substr(1,pureTagsLinesWithoutSlash[x].length()-1))
             current_root = getParent(current_root);
        }
  }
  return getMainParent(current_root);
}
```

5) Node* **getMainParent**(Node* root), and Node* **getLastChild**(Node* root) functions are used to get the main parent and the last child.



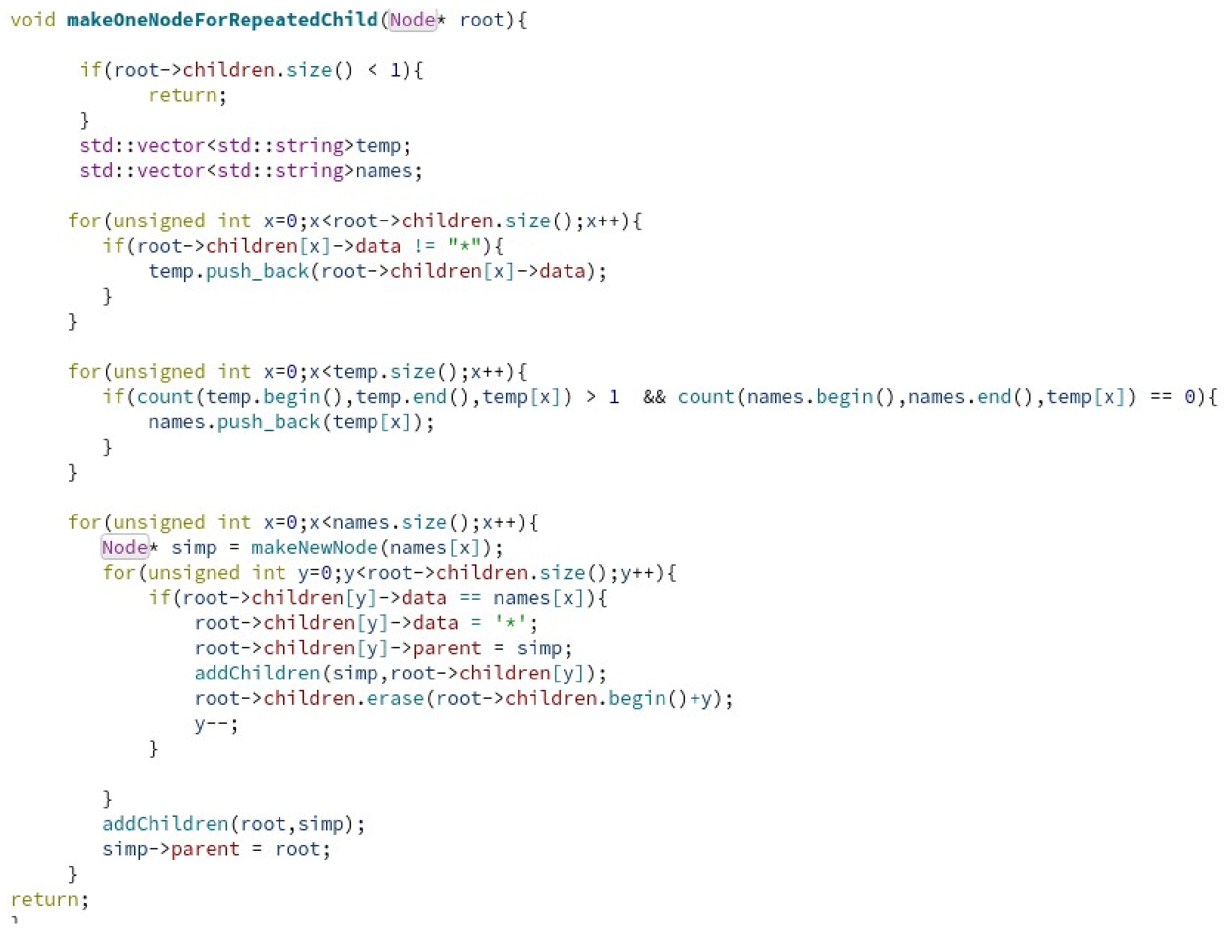
```cpp
Node* getMainParent(Node* root){
    Node* temp = root;
    while(1){
        if(temp->parent == NULL){
            break;
        }else{
            temp = temp->parent;
        }
    }
    return temp;
}
Node* getLastChild(Node* root){
    if(root->children.size() == 0){
        return root;
    }
    else return getLastChild(root->children[root->children.size()-1]);
}
```

6) **makeBrackets**(Node* root) function is used to make brackets, as one of the steps to convert the XML to JSON.



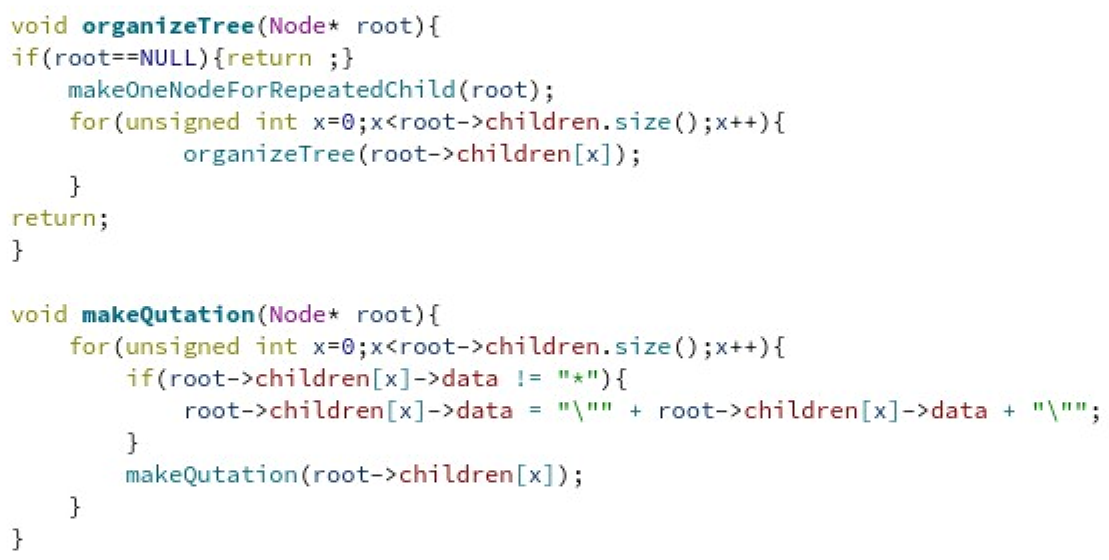
```cpp
void makeBrackets(Node* root){

    for(unsigned int x=0;x<root->children.size();x++){

        bool case2 = (root->children.size() >= 1) && (root->children[0]->data != "*") && (root->children[x]->children.size() != 0)
                && (x == root->children.size()-1);

        bool case3 = (root->children.size() > 1) && (root->children[0]->data == "*") && (x == root->children.size()-1);

        if( case2 ){
            Node* temp = getLastChild(root);
            if(temp->data[temp->data.length()-1] == ']' || (temp->data[temp->data.length()-1] == '}') ){
                int s1 = count(temp->data.begin(),temp->data.end(),']');
                int s2 = count(temp->data.begin(),temp->data.end(),'}');
                if(s1<0){s1=0;}
                if(s2<0){s2=0;}
                int sum = s1+s2;
                temp->data.insert(temp->data.length()-sum,"}");
            }else{
                temp->data = temp->data + "}";
            }
        }else if( case3 ){
            Node* temp = getLastChild(root->children[x]);
            if(temp->data[temp->data.length()-1] == ']' || (temp->data[temp->data.length()-1] == '}') ){
                int s1 = count(temp->data.begin(),temp->data.end(),']');
                int s2 = count(temp->data.begin(),temp->data.end(),'}');
                if(s1<0){s1=0;}
                if(s2<0){s2=0;}
                int sum = s1+s2;
                temp->data.insert(temp->data.length()-sum,"]");
            }else{
                temp->data = temp->data + "]";
            }
        }
       makeBrackets(root->children[x]);
    }}
```

7) **makeOneNodeForRepeatedChild**(Node* root) is used to make one node for repeated children.



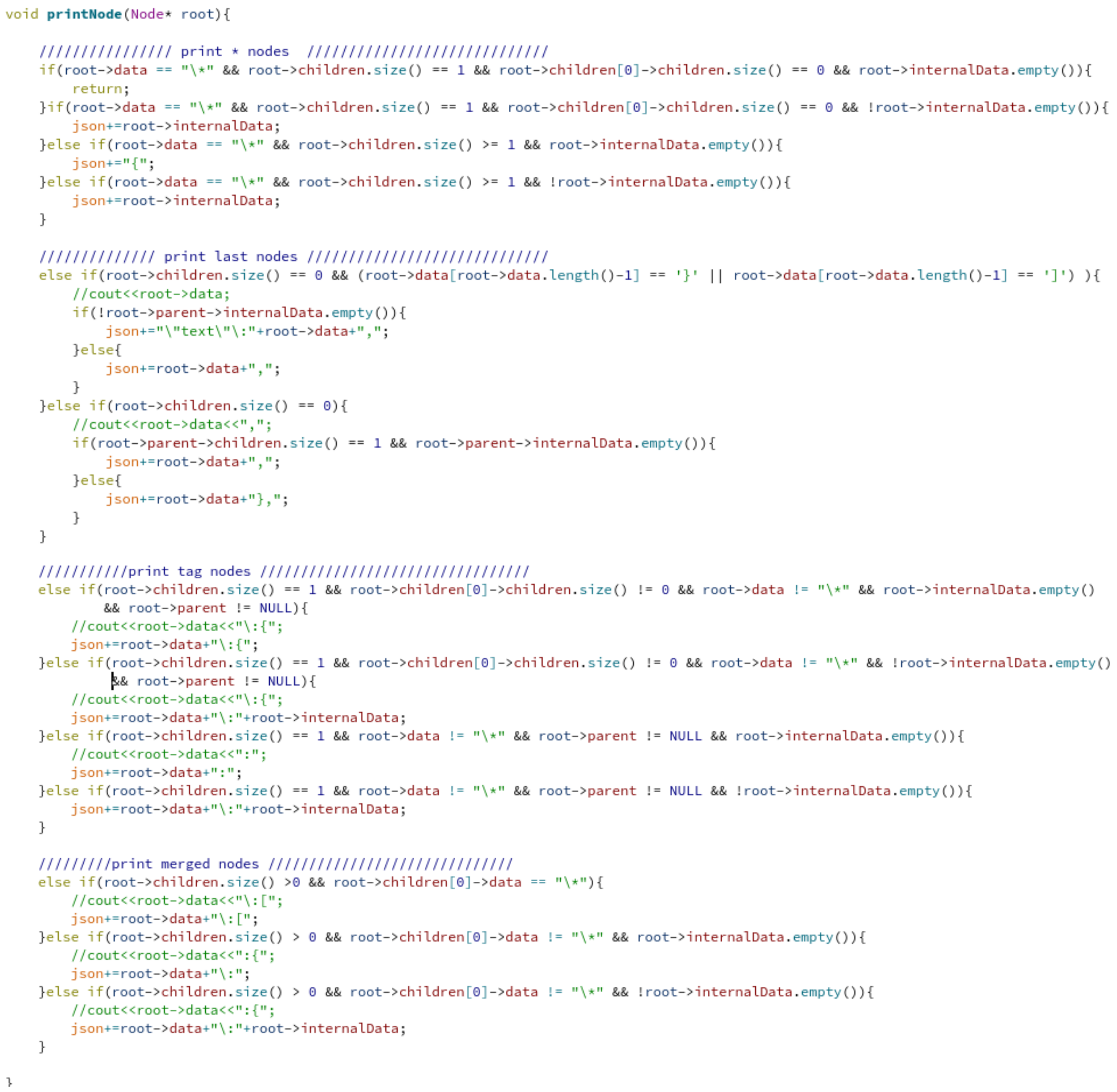
```
void makeOneNodeForRepeatedChild(Node* root){

      if(root->children.size() < 1){
            return;
      }
      std::vector<std::string>temp;
      std::vector<std::string>names;

     for(unsigned int x=0;x<root->children.size();x++){
        if(root->children[x]->data != "*"){
            temp.push_back(root->children[x]->data);
        }
     }

     for(unsigned int x=0;x<temp.size();x++){
        if(count(temp.begin(),temp.end(),temp[x]) > 1  && count(names.begin(),names.end(),temp[x]) == 0){
            names.push_back(temp[x]);
        }
     }

     for(unsigned int x=0;x<names.size();x++){
        Node* simp = makeNewNode(names[x]);
        for(unsigned int y=0;y<root->children.size();y++){
            if(root->children[y]->data == names[x]){
                root->children[y]->data = '*';
                root->children[y]->parent = simp;
                addChildren(simp,root->children[y]);
                root->children.erase(root->children.begin()+y);
                y--;
            }

        }
        addChildren(root,simp);
        simp->parent = root;
     }
return;
```

8) Recursive functions: **organizeTree**(Node* root) which is used to organize tree after merging nodes for repeated children, and **makeQutation**(Node* root) which is used to make quotations for JSON conversion were implemented recursively to reduce time.



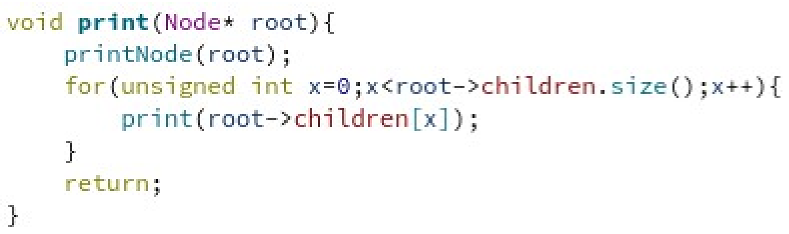
```
void organizeTree(Node* root){
if(root==NULL){return ;}
    makeOneNodeForRepeatedChild(root);
    for(unsigned int x=0;x<root->children.size();x++){
            organizeTree(root->children[x]);
    }
return;
}

void makeQutation(Node* root){
    for(unsigned int x=0;x<root->children.size();x++){
        if(root->children[x]->data != "*"){
            root->children[x]->data = "\"" + root->children[x]->data + "\"";
        }
        makeQutation(root->children[x]);
    }
}
```

9) **printNode**(Node* root) function is used to print normal, tag, merged and last nodes.



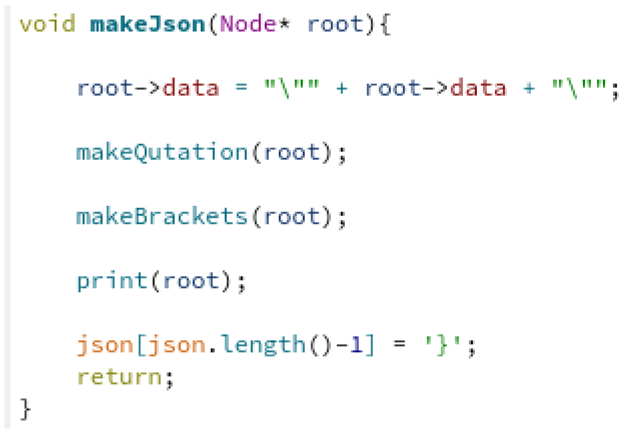
```
void printNode(Node* root){

    //////////////// print * nodes  /////////////////////////////
    if(root->data == "\*" && root->children.size() == 1 && root->children[0]->children.size() == 0 && root->internalData.empty()){
        return;
    }if(root->data == "\*" && root->children.size() == 1 && root->children[0]->children.size() == 0 && !root->internalData.empty()){
        json+=root->internalData;
    }else if(root->data == "\*" && root->children.size() >= 1 && root->internalData.empty()){
        json+="{";
    }else if(root->data == "\*" && root->children.size() >= 1 && !root->internalData.empty()){
        json+=root->internalData;
    }

    ////////////// print last nodes /////////////////////////////
    else if(root->children.size() == 0 && (root->data[root->data.length()-1] == '}' || root->data[root->data.length()-1] == ']') ){
        //cout<<root->data;
        if(!root->parent->internalData.empty()){
            json+="\"text\"\:"+root->data+",";
        }else{
            json+=root->data+",";
        }
    }else if(root->children.size() == 0){
        //cout<<root->data<<",";
        if(root->parent->children.size() == 1 && root->parent->internalData.empty()){
            json+=root->data+",";
        }else{
            json+=root->data+"},";
        }
    }

    ///////////print tag nodes /////////////////////////////////
    else if(root->children.size() == 1 && root->children[0]->children.size() != 0 && root->data != "\*" && root->internalData.empty()
            && root->parent != NULL){
        //cout<<root->data<<"\:{";
        json+=root->data+"\:{";
    }else if(root->children.size() == 1 && root->children[0]->children.size() != 0 && root->data != "\*" && !root->internalData.empty()
            && root->parent != NULL){
        //cout<<root->data<<"\:{";
        json+=root->data+"\:"+root->internalData;
    }else if(root->children.size() == 1 && root->data != "\*" && root->parent != NULL && root->internalData.empty()){
        //cout<<root->data<<":";
        json+=root->data+":";
    }else if(root->children.size() == 1 && root->data != "\*" && root->parent != NULL && !root->internalData.empty()){
        json+=root->data+"\:"+root->internalData;
    }

    /////////print merged nodes /////////////////////////////
    else if(root->children.size() >0 && root->children[0]->data == "\*"){
        //cout<<root->data<<"\:[";
        json+=root->data+"\:[";
    }else if(root->children.size() > 0 && root->children[0]->data != "\*" && root->internalData.empty()){
        //cout<<root->data<<":{";
        json+=root->data+"\:";
    }else if(root->children.size() > 0 && root->children[0]->data != "\*" && !root->internalData.empty()){
        //cout<<root->data<<":{";
        json+=root->data+"\:"+root->internalData;
    }

}
```

Which is called by the recursive function **print**(Node* root) to print the nodes



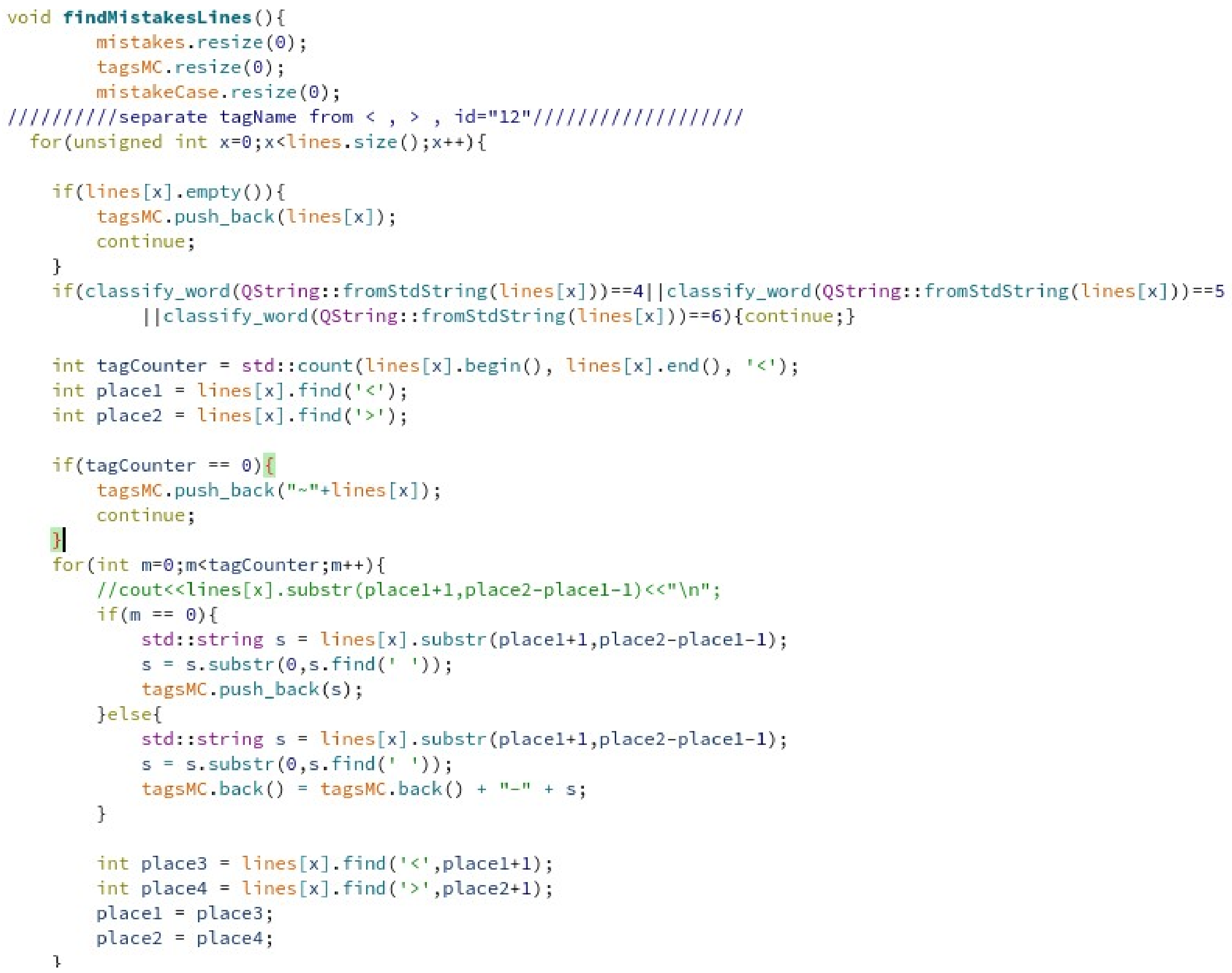
```
void print(Node* root){
    printNode(root);
    for(unsigned int x=0;x<root->children.size();x++){
        print(root->children[x]);
    }
    return;
}
```

10) **makeJson**(Node* root) function calls makeQutation(root), makeBrackets(root), print(root) functions, and is used to convert the xml file to JSON.



```
void makeJson(Node* root){

    root->data = "\"" + root->data + "\"";

    makeQutation(root);

    makeBrackets(root);

    print(root);

    json[json.length()-1] = '}';
    return;
}
```

11) **findMistakesLines**() function is used to find and declare mistakes in lines.



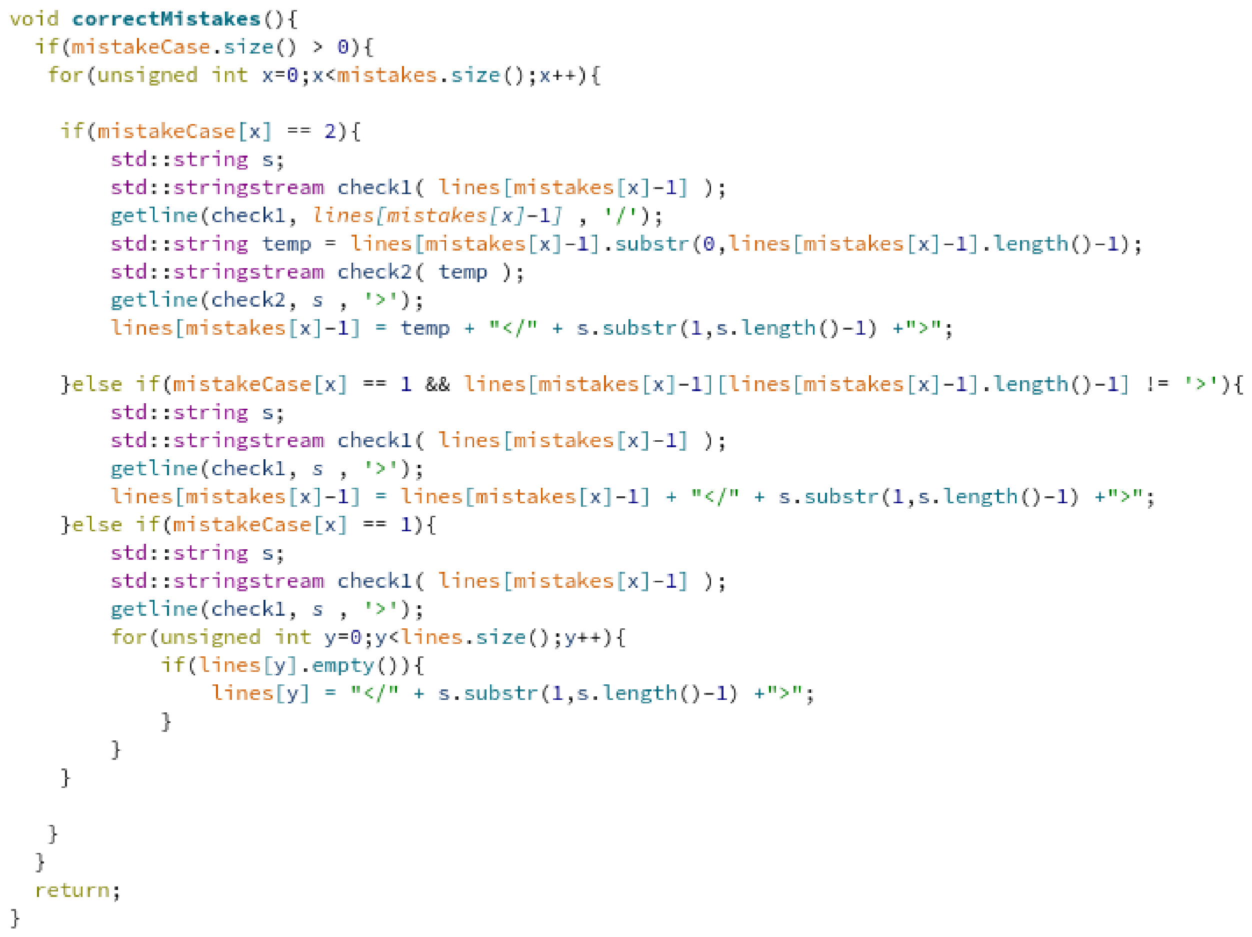
```
void findMistakesLines(){
        mistakes.resize(0);
        tagsMC.resize(0);
        mistakeCase.resize(0);
//////////separate tagName from < , > , id="12"//////////////////
  for(unsigned int x=0;x<lines.size();x++){

    if(lines[x].empty()){
        tagsMC.push_back(lines[x]);
        continue;
    }
    if(classify_word(QString::fromStdString(lines[x]))==4||classify_word(QString::fromStdString(lines[x]))==5
            ||classify_word(QString::fromStdString(lines[x]))==6){continue;}

    int tagCounter = std::count(lines[x].begin(), lines[x].end(), '<');
    int place1 = lines[x].find('<');
    int place2 = lines[x].find('>');

    if(tagCounter == 0){
        tagsMC.push_back("~"+lines[x]);
        continue;
    }
    for(int m=0;m<tagCounter;m++){
        //cout<<lines[x].substr(place1+1,place2-place1-1)<<"\n";
        if(m == 0){
            std::string s = lines[x].substr(place1+1,place2-place1-1);
            s = s.substr(0,s.find(' '));
            tagsMC.push_back(s);
        }else{
            std::string s = lines[x].substr(place1+1,place2-place1-1);
            s = s.substr(0,s.find(' '));
            tagsMC.back() = tagsMC.back() + "-" + s;
        }

        int place3 = lines[x].find('<',place1+1);
        int place4 = lines[x].find('>',place2+1);
        place1 = place3;
        place2 = place4;
    }
```

```
//////////////////////declare mistakes lines //////////////////////////////
  std::vector<std::string> xx;
  std::vector<int> index;
  for(unsigned int x=1;x<tagsMC.size()+1;x++){
    if(tagsMC[x-1].empty()){
        mistakes.push_back(x);
        continue;
    }else if(tagsMC[x-1][0] == '~'){
        continue;
    }
    if(tagsMC[x-1].find('/') == std::string::npos){
        xx.push_back(tagsMC[x-1]);
        index.push_back(x);
    }else{
    std::stringstream check1(tagsMC[x-1]);
    std::string intermediate;
    while(getline(check1, intermediate, '-'))
    {
        if(intermediate.find('/') == std::string::npos){
            xx.push_back(intermediate);
            index.push_back(x);
        }else{
            std::string s = intermediate.substr(1,intermediate.length()-1);
            if( xx.back() == s ){
                xx.pop_back();
                index.pop_back();
            }else if( xx[xx.size()-2] == s ){
                mistakes.push_back(index.back());
                mistakeCase.push_back(1);
                xx.pop_back();
                index.pop_back();
                x--;
            }else{
                mistakes.push_back(x);
                mistakeCase.push_back(2);
                xx.pop_back();
                index.pop_back();
            }}}}}
```

12) **countSynset**(Node* root) function is used to count the number of synsets.

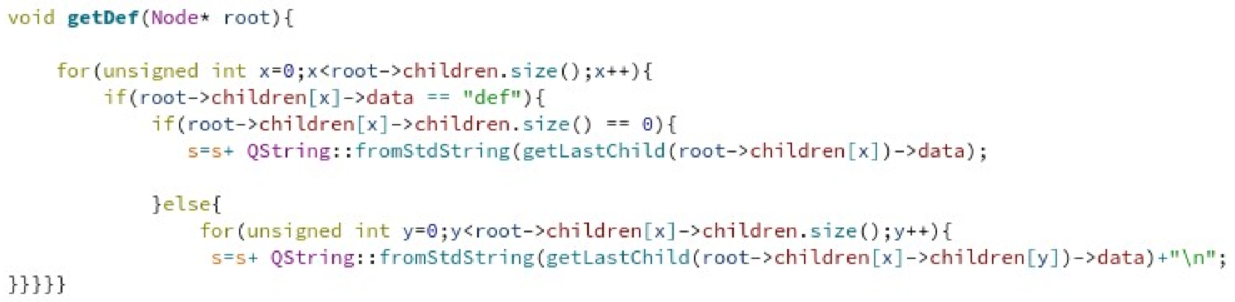
```
void countSynset(Node* root){

    for(unsigned int x=0;x<root->children.size();x++){
        if(root->children[0]->data == "synset"){
            if(root->children[0]->children[0]->data == "*"){
                synsetCounter=synsetCounter+root->children[0]->children.size();
            }else{
                synsetCounter=synsetCounter+1;
            }
        }
        countSynset(root->children[x]);
    }

}
```

13) **correctMistakes**() function is used to correct mistakes.



```
void correctMistakes(){
  if(mistakeCase.size() > 0){
   for(unsigned int x=0;x<mistakes.size();x++){

    if(mistakeCase[x] == 2){
        std::string s;
        std::stringstream check1( lines[mistakes[x]-1] );
        getline(check1, lines[mistakes[x]-1] , '/');
        std::string temp = lines[mistakes[x]-1].substr(0,lines[mistakes[x]-1].length()-1);
        std::stringstream check2( temp );
        getline(check2, s , '>');
        lines[mistakes[x]-1] = temp + "</" + s.substr(1,s.length()-1) +">";

    }else if(mistakeCase[x] == 1 && lines[mistakes[x]-1][lines[mistakes[x]-1].length()-1] != '>'){
        std::string s;
        std::stringstream check1( lines[mistakes[x]-1] );
        getline(check1, s , '>');
        lines[mistakes[x]-1] = lines[mistakes[x]-1] + "</" + s.substr(1,s.length()-1) +">";
    }else if(mistakeCase[x] == 1){
        std::string s;
        std::stringstream check1( lines[mistakes[x]-1] );
        getline(check1, s , '>');
        for(unsigned int y=0;y<lines.size();y++){
            if(lines[y].empty()){
                lines[y] = "</" + s.substr(1,s.length()-1) +">";
            }
        }
    }

   }
  }
  return;
}
```

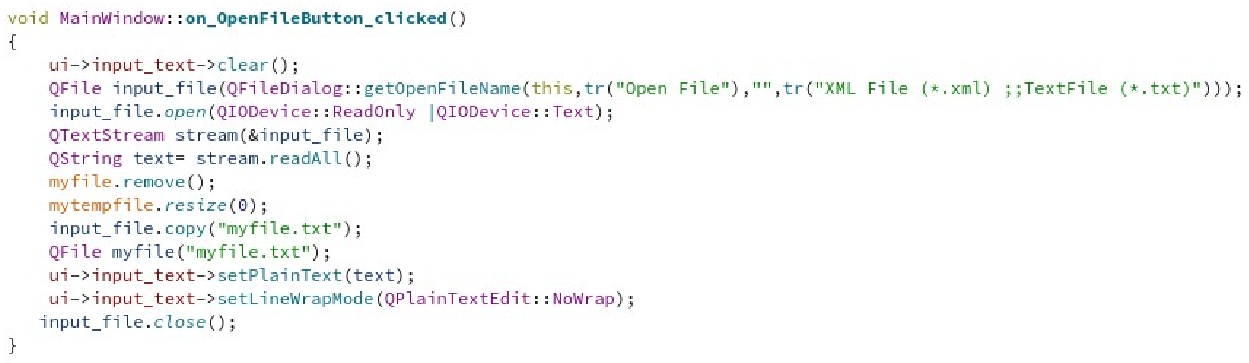
14) **getDef**(Node* root) function is used to get the definition of a given word.



```
void getDef(Node* root){

    for(unsigned int x=0;x<root->children.size();x++){
        if(root->children[x]->data == "def"){
            if(root->children[x]->children.size() == 0){
               s=s+ QString::fromStdString(getLastChild(root->children[x])->data);

            }else{
                for(unsigned int y=0;y<root->children[x]->children.size();y++){
                 s=s+ QString::fromStdString(getLastChild(root->children[x]->children[y])->data)+"\n";
}}}}}
```

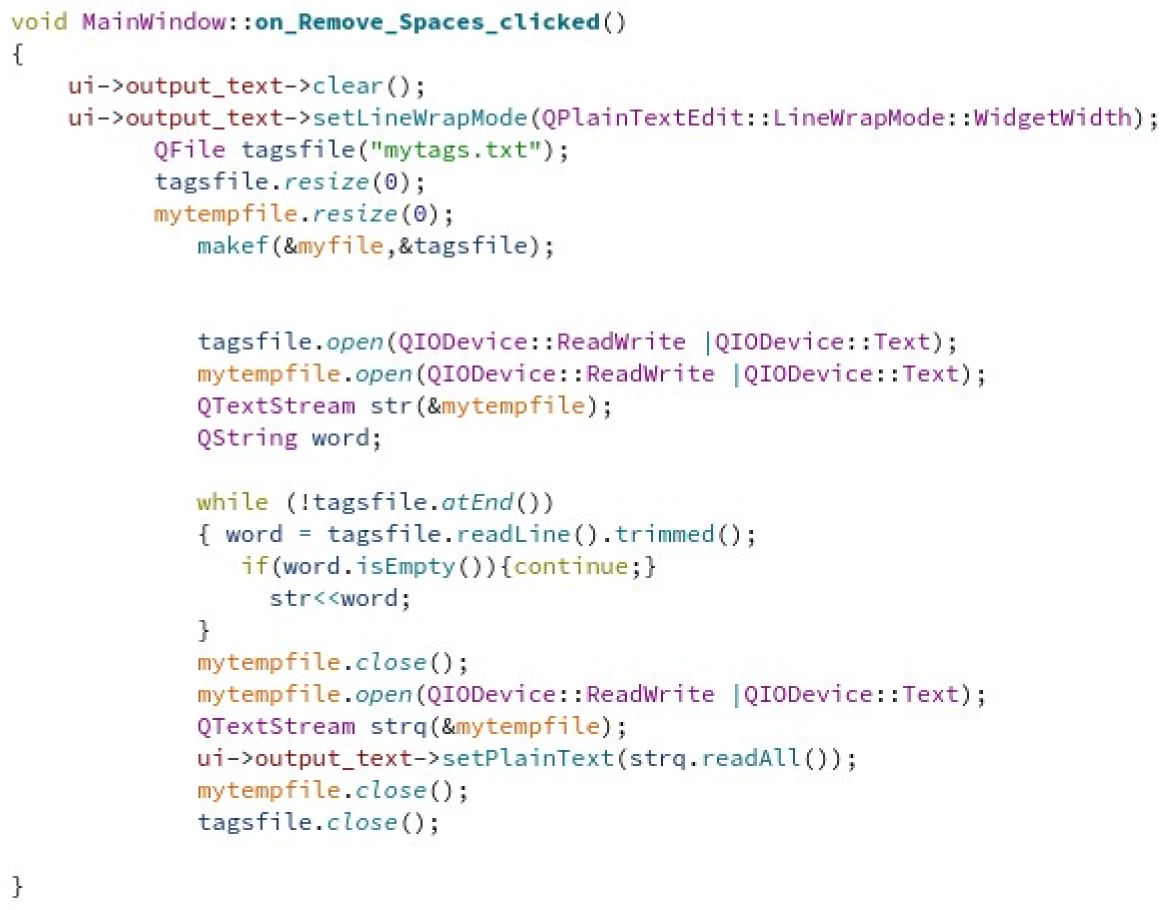
3) Slots:

1) **on_OpenFileButton_clicked**() slot is called when the open file button is clicked to open the file.



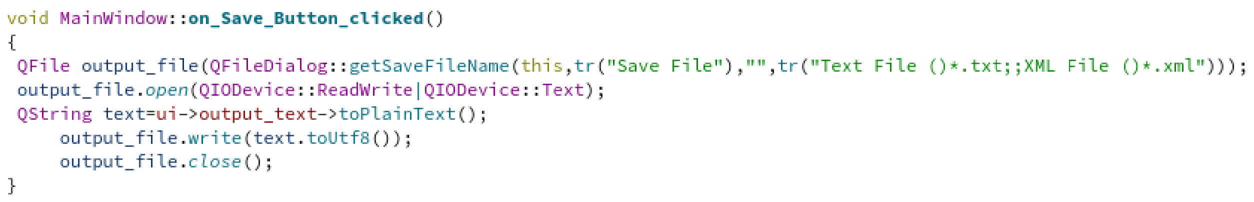
```
void MainWindow::on_OpenFileButton_clicked()
{
    ui->input_text->clear();
    QFile input_file(QFileDialog::getOpenFileName(this,tr("Open File"),"",tr("XML File (*.xml) ;;TextFile (*.txt)")));
    input_file.open(QIODevice::ReadOnly |QIODevice::Text);
    QTextStream stream(&input_file);
    QString text= stream.readAll();
    myfile.remove();
    mytempfile.resize(0);
    input_file.copy("myfile.txt");
    QFile myfile("myfile.txt");
    ui->input_text->setPlainText(text);
    ui->input_text->setLineWrapMode(QPlainTextEdit::NoWrap);
   input_file.close();
}
```

2) **on_Remove_Spaces_clicked**() slot is called when the remove spaces button is clicked to remove the spaces to reduce size.



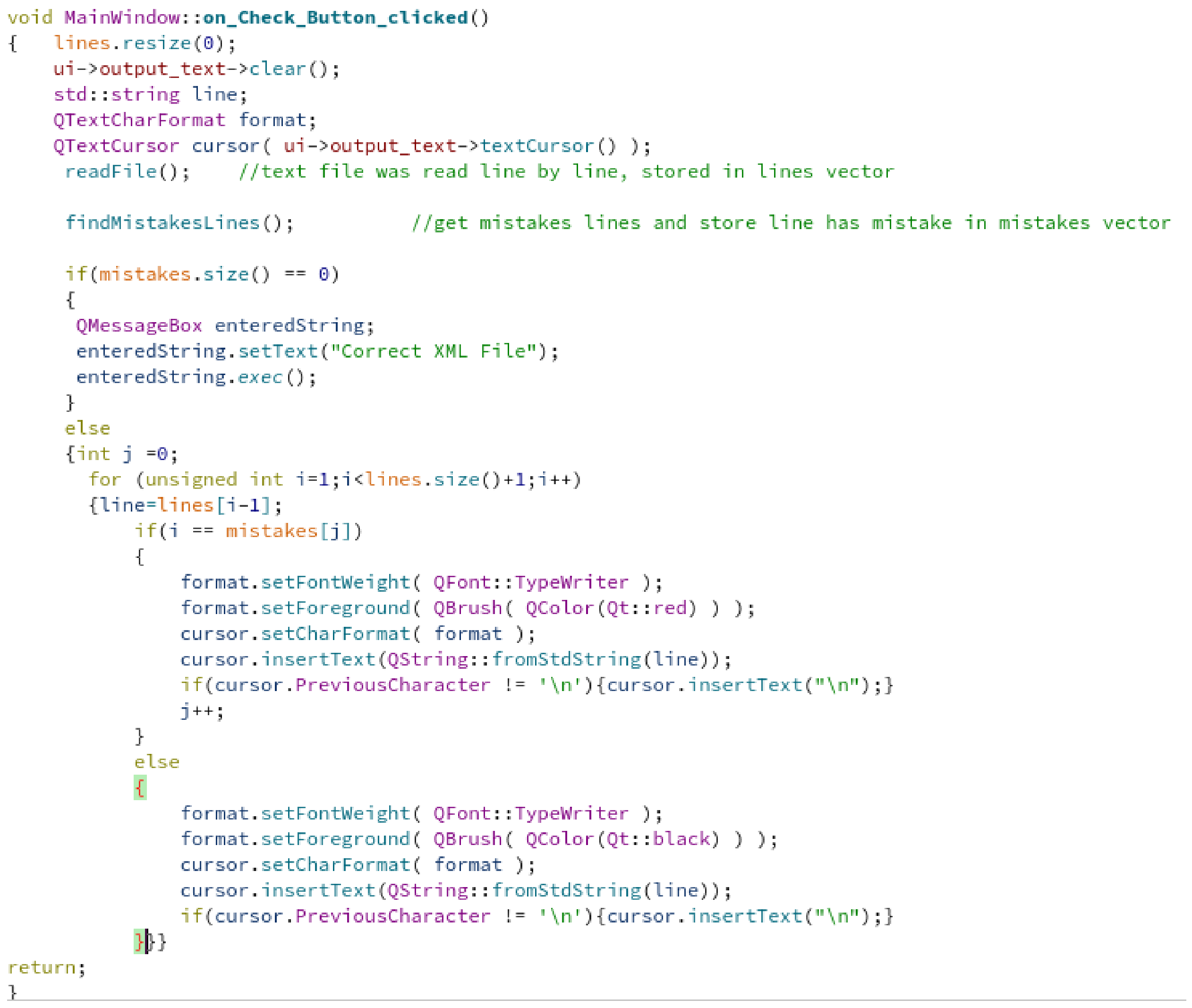
```
void MainWindow::on_Remove_Spaces_clicked()
{
    ui->output_text->clear();
    ui->output_text->setLineWrapMode(QPlainTextEdit::LineWrapMode::WidgetWidth);
          QFile tagsfile("mytags.txt");
          tagsfile.resize(0);
          mytempfile.resize(0);
             makef(&myfile,&tagsfile);


             tagsfile.open(QIODevice::ReadWrite |QIODevice::Text);
             mytempfile.open(QIODevice::ReadWrite |QIODevice::Text);
             QTextStream str(&mytempfile);
             QString word;

             while (!tagsfile.atEnd())
             { word = tagsfile.readLine().trimmed();
                if(word.isEmpty()){continue;}
                  str<<word;
             }
             mytempfile.close();
             mytempfile.open(QIODevice::ReadWrite |QIODevice::Text);
             QTextStream strq(&mytempfile);
             ui->output_text->setPlainText(strq.readAll());
             mytempfile.close();
             tagsfile.close();

}
```

3) **on_Save_Button_clicked**() slot is called when the save file button is clicked to save the file.



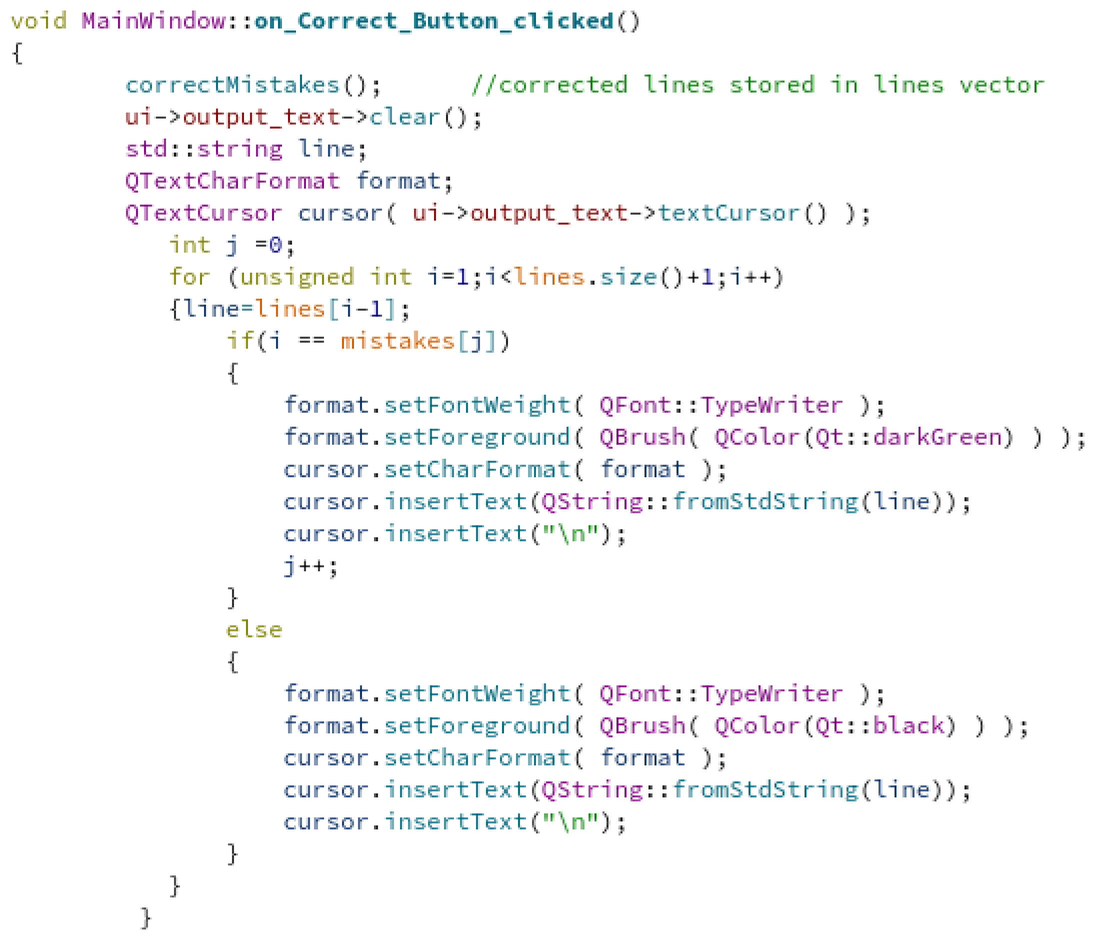
```
void MainWindow::on_Save_Button_clicked()
{
 QFile output_file(QFileDialog::getSaveFileName(this,tr("Save File"),"",tr("Text File ()*.txt;;XML File ()*.xml")));
 output_file.open(QIODevice::ReadWrite|QIODevice::Text);
 QString text=ui->output_text->toPlainText();
     output_file.write(text.toUtf8());
     output_file.close();
}
```

4) **on_Check_Button_clicked**() slot is called when the check button is clicked to check the consistency of the XML file, where it highlights the mistakes in **red**, and if there aren't any mistakes it shows a message to the user that the file is correct.



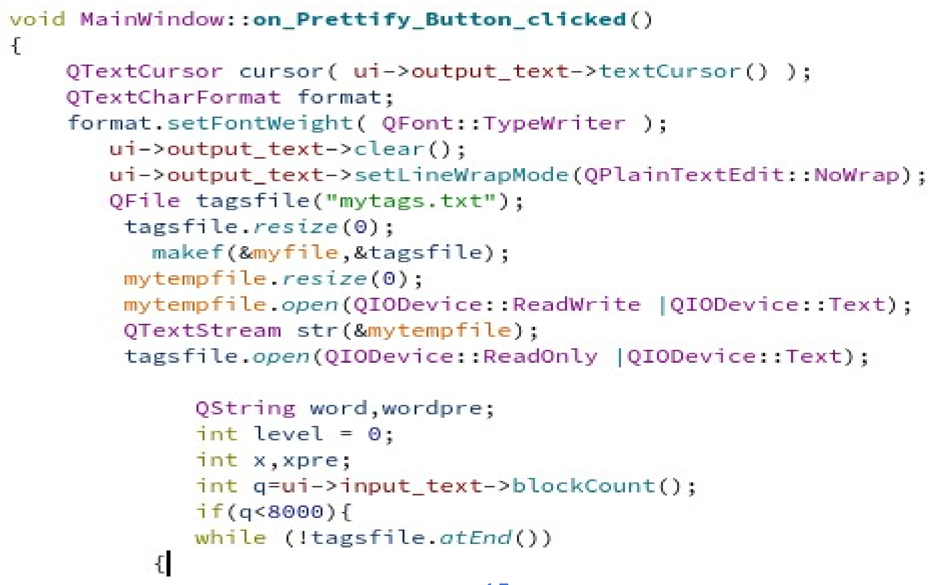
```
void MainWindow::on_Check_Button_clicked()
{   lines.resize(0);
    ui->output_text->clear();
    std::string line;
    QTextCharFormat format;
    QTextCursor cursor( ui->output_text->textCursor() );
     readFile();    //text file was read line by line, stored in lines vector

     findMistakesLines();          //get mistakes lines and store line has mistake in mistakes vector

     if(mistakes.size() == 0)
     {
      QMessageBox enteredString;
      enteredString.setText("Correct XML File");
      enteredString.exec();
     }
     else
     {int j =0;
       for (unsigned int i=1;i<lines.size()+1;i++)
       {line=lines[i-1];
           if(i == mistakes[j])
           {
               format.setFontWeight( QFont::TypeWriter );
               format.setForeground( QBrush( QColor(Qt::red) ) );
               cursor.setCharFormat( format );
               cursor.insertText(QString::fromStdString(line));
               if(cursor.PreviousCharacter != '\n'){cursor.insertText("\n");}
               j++;
           }
           else
           {
               format.setFontWeight( QFont::TypeWriter );
               format.setForeground( QBrush( QColor(Qt::black) ) );
               cursor.setCharFormat( format );
               cursor.insertText(QString::fromStdString(line));
               if(cursor.PreviousCharacter != '\n'){cursor.insertText("\n");}
           }}}
return;
}
```

5) **on_Correct_Button_clicked**() slot is called when the correct button is clicked to correct the mistakes that were found in the XML file, where it highlights the corrected mistakes in **green**.



```
void MainWindow::on_Correct_Button_clicked()
{
        correctMistakes();      //corrected lines stored in lines vector
        ui->output_text->clear();
        std::string line;
        QTextCharFormat format;
        QTextCursor cursor( ui->output_text->textCursor() );
           int j =0;
           for (unsigned int i=1;i<lines.size()+1;i++)
           {line=lines[i-1];
               if(i == mistakes[j])
               {
                   format.setFontWeight( QFont::TypeWriter );
                   format.setForeground( QBrush( QColor(Qt::darkGreen) ) );
                   cursor.setCharFormat( format );
                   cursor.insertText(QString::fromStdString(line));
                   cursor.insertText("\n");
                   j++;
               }
               else
               {
                   format.setFontWeight( QFont::TypeWriter );
                   format.setForeground( QBrush( QColor(Qt::black) ) );
                   cursor.setCharFormat( format );
                   cursor.insertText(QString::fromStdString(line));
                   cursor.insertText("\n");
               }
           }
         }
```

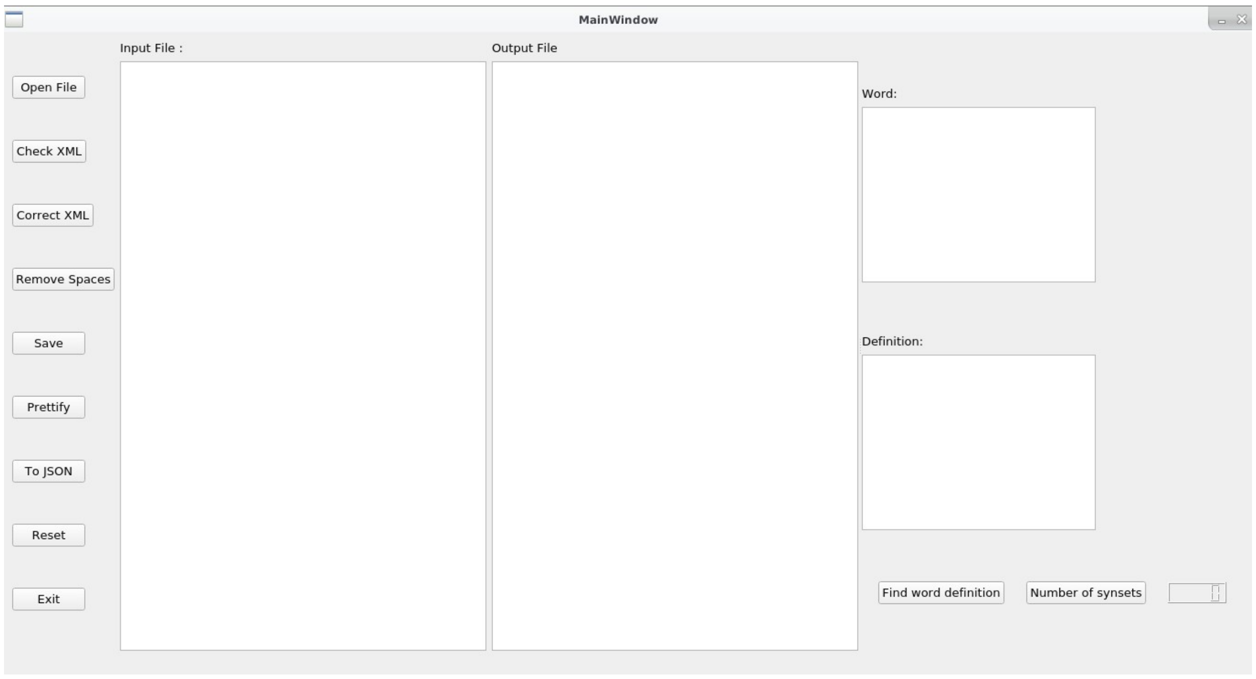
6) **on_Prettify_Button_clicked**() slot is called when the prettify button is clicked to prettify the XML file, and also changes the tags color to blue (a small part is shown).



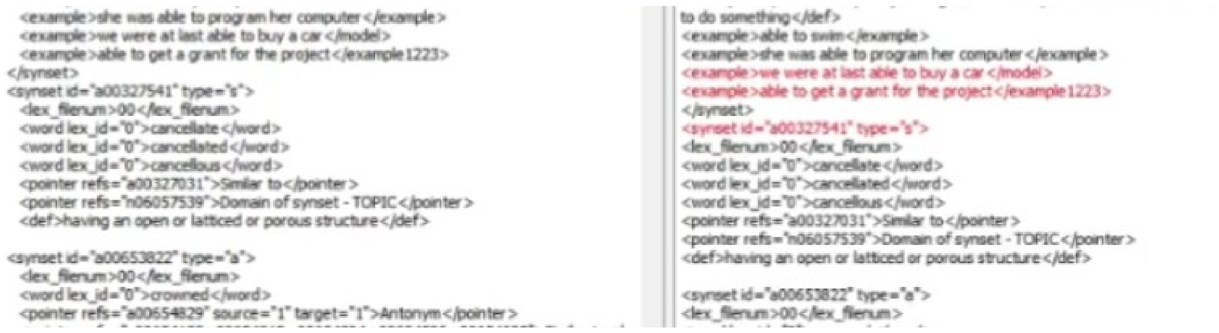
```
void MainWindow::on_Prettify_Button_clicked()
{
    QTextCursor cursor( ui->output_text->textCursor() );
    QTextCharFormat format;
    format.setFontWeight( QFont::TypeWriter );
       ui->output_text->clear();
       ui->output_text->setLineWrapMode(QPlainTextEdit::NoWrap);
       QFile tagsfile("mytags.txt");
        tagsfile.resize(0);
          makef(&myfile,&tagsfile);
        mytempfile.resize(0);
        mytempfile.open(QIODevice::ReadWrite |QIODevice::Text);
        QTextStream str(&mytempfile);
        tagsfile.open(QIODevice::ReadOnly |QIODevice::Text);

             QString word,wordpre;
             int level = 0;
             int x,xpre;
             int q=ui->input_text->blockCount();
             if(q<8000){
             while (!tagsfile.atEnd())
           {
```

4) Graphical user interface:

1) General view:



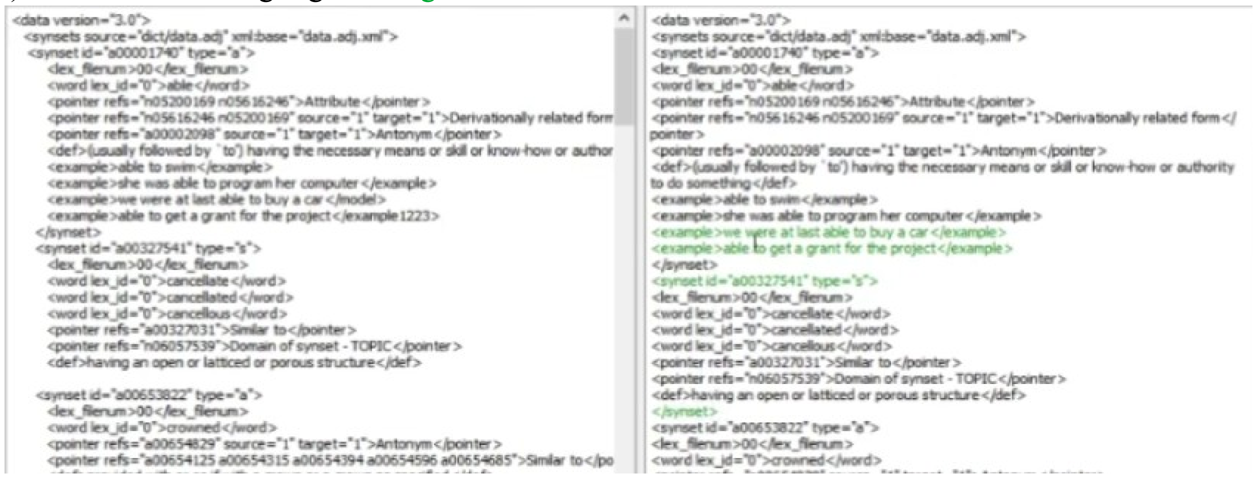
2) Mistakes are highlighted in red.



```
    <example>she was able to program her computer</example>
    <example>we were at last able to buy a car</model>
    <example>able to get a grant for the project</example1223>
  </synset>
  <synset id="a00327541" type="s">
    <lex_filenum>00</lex_filenum>
    <word lex_id="0">cancellate</word>
    <word lex_id="0">cancellated</word>
    <word lex_id="0">cancellous</word>
    <pointer refs="a00327031">Similar to</pointer>
    <pointer refs="n06057539">Domain of synset - TOPIC</pointer>
    <def>having an open or latticed or porous structure</def>

  <synset id="a00653822" type="a">
    <lex_filenum>00</lex_filenum>
    <word lex_id="0">crowned</word>
    <pointer refs="a00654829" source="1" target="1">Antonym</pointer>
```

```
to do something</def>
<example>able to swim</example>
<example>she was able to program her computer</example>
<example>we were at last able to buy a car</model>
<example>able to get a grant for the project</example1223>
</synset>
<synset id="a00327541" type="s">
<lex_filenum>00</lex_filenum>
<word lex_id="0">cancellate</word>
<word lex_id="0">cancellated</word>
<word lex_id="0">cancellous</word>
<pointer refs="a00327031">Similar to</pointer>
<pointer refs="n06057539">Domain of synset - TOPIC</pointer>
<def>having an open or latticed or porous structure</def>

<synset id="a00653822" type="a">
<lex_filenum>00</lex_filenum>
```

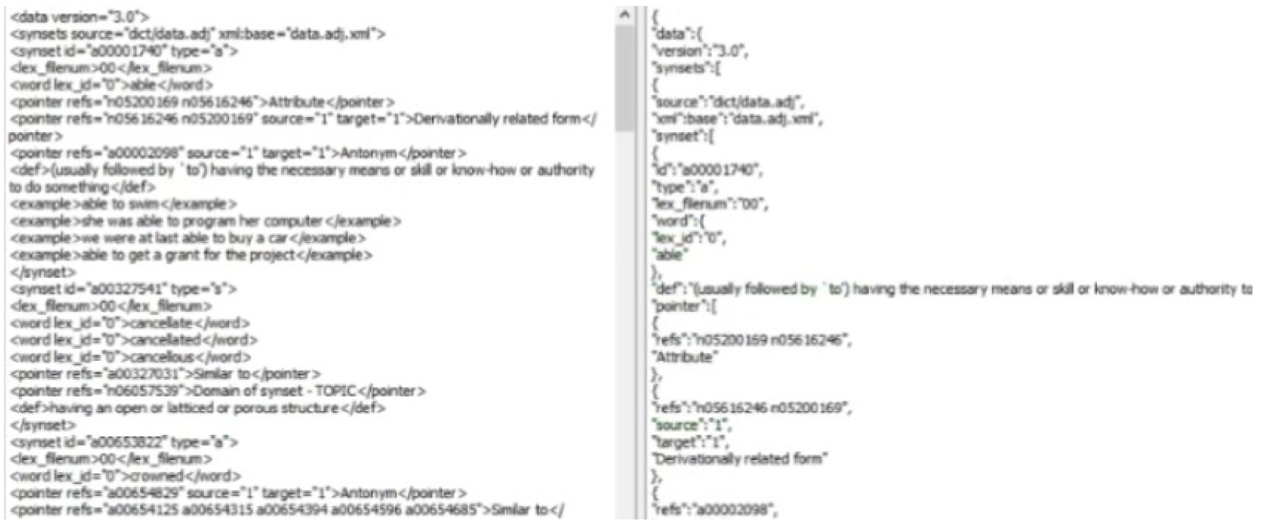
3) Corrections are highlighted in green.



```
<data version="3.0">
  <synsets source="dict/data.adj" xml:base="data.adj.xml">
   <synset id="a00001740" type="a">
      <lex_filenum>00</lex_filenum>
      <word lex_id="0">able</word>
      <pointer refs="n05200169 n05616246">Attribute</pointer>
      <pointer refs="n05616246 n05200169" source="1" target="1">Derivationally related form
      <pointer refs="a00002098" source="1" target="1">Antonym</pointer>
      <def>(usually followed by `to') having the necessary means or skill or know-how or author
      <example>able to swim</example>
      <example>she was able to program her computer</example>
      <example>we were at last able to buy a car</model>
      <example>able to get a grant for the project</example1223>
    </synset>
    <synset id="a00327541" type="s">
      <lex_filenum>00</lex_filenum>
      <word lex_id="0">cancellate</word>
      <word lex_id="0">cancellated</word>
      <word lex_id="0">cancellous</word>
      <pointer refs="a00327031">Similar to</pointer>
      <pointer refs="n06057539">Domain of synset - TOPIC</pointer>
      <def>having an open or latticed or porous structure</def>

    <synset id="a00653822" type="a">
      <lex_filenum>00</lex_filenum>
      <word lex_id="0">crowned</word>
      <pointer refs="a00654829" source="1" target="1">Antonym</pointer>
      <pointer refs="a00654125 a00654315 a00654394 a00654596 a00654685">Similar to</po
```

```
<data version="3.0">
<synsets source="dict/data.adj" xml:base="data.adj.xml">
<synset id="a00001740" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">able</word>
<pointer refs="n05200169 n05616246">Attribute</pointer>
<pointer refs="n05616246 n05200169" source="1" target="1">Derivationally related form</
pointer>
<pointer refs="a00002098" source="1" target="1">Antonym</pointer>
<def>(usually followed by `to') having the necessary means or skill or know-how or authority
to do something</def>
<example>able to swim</example>
<example>she was able to program her computer</example>
<example>we were at last able to buy a car</example>
<example>able to get a grant for the project</example>
</synset>
<synset id="a00327541" type="s">
<lex_filenum>00</lex_filenum>
<word lex_id="0">cancellate</word>
<word lex_id="0">cancellated</word>
<word lex_id="0">cancellous</word>
<pointer refs="a00327031">Similar to</pointer>
<pointer refs="n06057539">Domain of synset - TOPIC</pointer>
<def>having an open or latticed or porous structure</def>
</synset>
<synset id="a00653822" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">crowned</word>
```

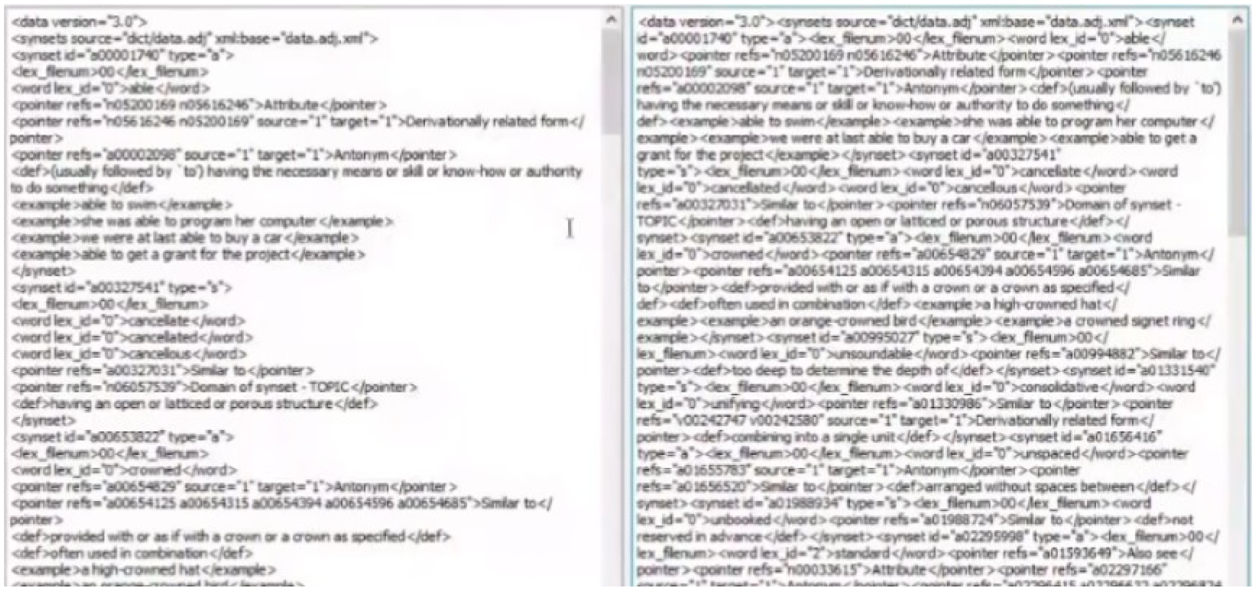
4) Conversion of XML file to JSON.



```
<data version="3.0">
<synsets source="dict/data.adj" xml:base="data.adj.xml">
<synset id="a00001740" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">able</word>
<pointer refs="n05200169 n05616246">Attribute</pointer>
<pointer refs="n05616246 n05200169" source="1" target="1">Derivationally related form</
pointer>
<pointer refs="a00002098" source="1" target="1">Antonym</pointer>
<def>(usually followed by `to') having the necessary means or skill or know-how or authority
to do something</def>
<example>able to swim</example>
<example>she was able to program her computer</example>
<example>we were at last able to buy a car</example>
<example>able to get a grant for the project</example>
</synset>
<synset id="a00327541" type="s">
<lex_filenum>00</lex_filenum>
<word lex_id="0">cancellate</word>
<word lex_id="0">cancellated</word>
<word lex_id="0">cancellous</word>
<pointer refs="a00327031">Similar to</pointer>
<pointer refs="n06057539">Domain of synset - TOPIC</pointer>
<def>having an open or latticed or porous structure</def>
</synset>
<synset id="a00653822" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">crowned</word>
<pointer refs="a00654829" source="1" target="1">Antonym</pointer>
<pointer refs="a00654125 a00654315 a00654394 a00654596 a00654685">Similar to</
```

```
{
"data":{
"version":"3.0",
"synsets":[
{
"source":"dict/data.adj",
"xml":base":"data.adj.xml",
"synset":[
{
"id":"a00001740",
"type":"a",
"lex_filenum":"00",
"word":{
"lex_id":"0",
"able"
},
"def":"(usually followed by `to') having the necessary means or skill or know-how or authority to
"pointer":[
{
"refs":"n05200169 n05616246",
"Attribute"
},
{
"refs":"n05616246 n05200169",
"source":"1",
"target":"1",
"Derivationally related form"
},
{
"refs":"a00002098",
```

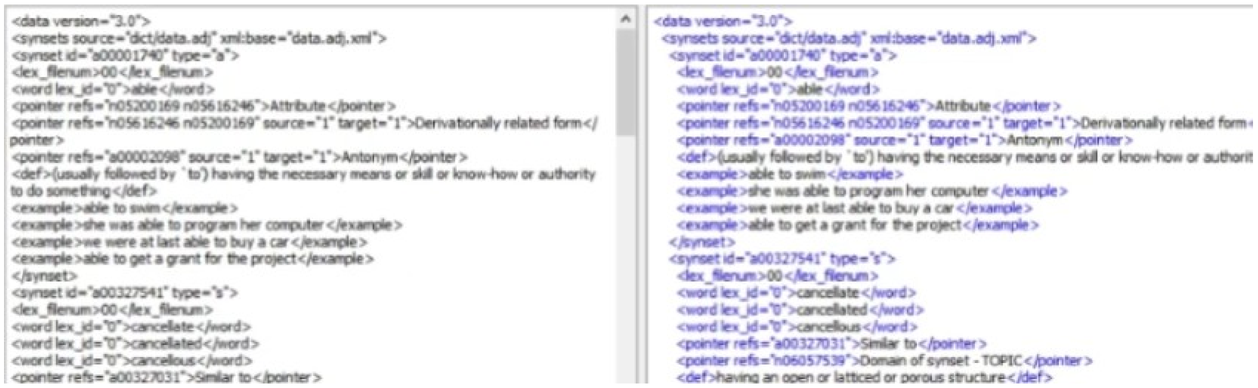
5) Space Removal.



```
<data version="3.0">
<synsets source="dict/data.adj" xml:base="data.adj.xml">
<synset id="a00001740" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">able</word>
<pointer refs="n05200169 n05616246">Attribute</pointer>
<pointer refs="n05616246 n05200169" source="1" target="1">Derivationally related form</
pointer>
<pointer refs="a00002098" source="1" target="1">Antonym</pointer>
<def>(usually followed by `to') having the necessary means or skill or know-how or authority
to do something</def>
<example>able to swim</example>
<example>she was able to program her computer</example>
<example>we were at last able to buy a car</example>
<example>able to get a grant for the project</example>
</synset>
<synset id="a00327541" type="s">
<lex_filenum>00</lex_filenum>
<word lex_id="0">cancellate</word>
<word lex_id="0">cancellated</word>
<word lex_id="0">cancellous</word>
<pointer refs="a00327031">Similar to</pointer>
<pointer refs="n06057539">Domain of synset - TOPIC</pointer>
<def>having an open or latticed or porous structure</def>
</synset>
<synset id="a00653822" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">crowned</word>
<pointer refs="a00654829" source="1" target="1">Antonym</pointer>
<pointer refs="a00654125 a00654315 a00654394 a00654596 a00654685">Similar to</
pointer>
<def>provided with or as if with a crown or a crown as specified</def>
<def>often used in combination</def>
<example>a high-crowned hat</example>
```

```
<data version="3.0"><synsets source="dict/data.adj" xml:base="data.adj.xml"><synset
id="a00001740" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">able</
word><pointer refs="n05200169 n05616246">Attribute</pointer><pointer refs="n05616246
n05200169" source="1" target="1">Derivationally related form</pointer><pointer
refs="a00002098" source="1" target="1">Antonym</pointer><def>(usually followed by `to')
having the necessary means or skill or know-how or authority to do something</
def><example>able to swim</example><example>she was able to program her computer</
example><example>we were at last able to buy a car</example><example>able to get a
grant for the project</example></synset><synset id="a00327541"
type="s"><lex_filenum>00</lex_filenum><word lex_id="0">cancellate</word><word
lex_id="0">cancellated</word><word lex_id="0">cancellous</word><pointer
refs="a00327031">Similar to</pointer><pointer refs="n06057539">Domain of synset -
TOPIC</pointer><def>having an open or latticed or porous structure</def></
synset><synset id="a00653822" type="a"><lex_filenum>00</lex_filenum><word
lex_id="0">crowned</word><pointer refs="a00654829" source="1" target="1">Antonym</
pointer><pointer refs="a00654125 a00654315 a00654394 a00654596 a00654685">Similar
to</pointer><def>provided with or as if with a crown or a crown as specified</
def><def>often used in combination</def><example>a high-crowned hat</
example><example>an orange-crowned bird</example><example>a crowned signet ring</
example></synset><synset id="a00995027" type="s"><lex_filenum>00</
lex_filenum><word lex_id="0">unsoundable</word><pointer refs="a00994882">Similar to</
pointer><def>too deep to determine the depth of</def></synset><synset id="a01331540"
type="s"><lex_filenum>00</lex_filenum><word lex_id="0">consolidative</word><word
lex_id="0">unifying</word><pointer refs="a01330986">Similar to</pointer><pointer
refs="v00242747 v00242580" source="1" target="1">Derivationally related form</
pointer><def>combining into a single unit</def></synset><synset id="a01656416"
type="a"><lex_filenum>00</lex_filenum><word lex_id="0">unspaced</word><pointer
refs="a01655783" source="1" target="1">Antonym</pointer><pointer
refs="a01656520">Similar to</pointer><def>arranged without spaces between</def></
synset><synset id="a01988934" type="s"><lex_filenum>00</lex_filenum><word
lex_id="0">unbooked</word><pointer refs="a01988724">Similar to</pointer><def>not
reserved in advance</def></synset><synset id="a02295998" type="a"><lex_filenum>00</
lex_filenum><word lex_id="2">standard</word><pointer refs="a01593649">Also see</
pointer><pointer refs="n00033615">Attribute</pointer><pointer refs="a02297166"
```

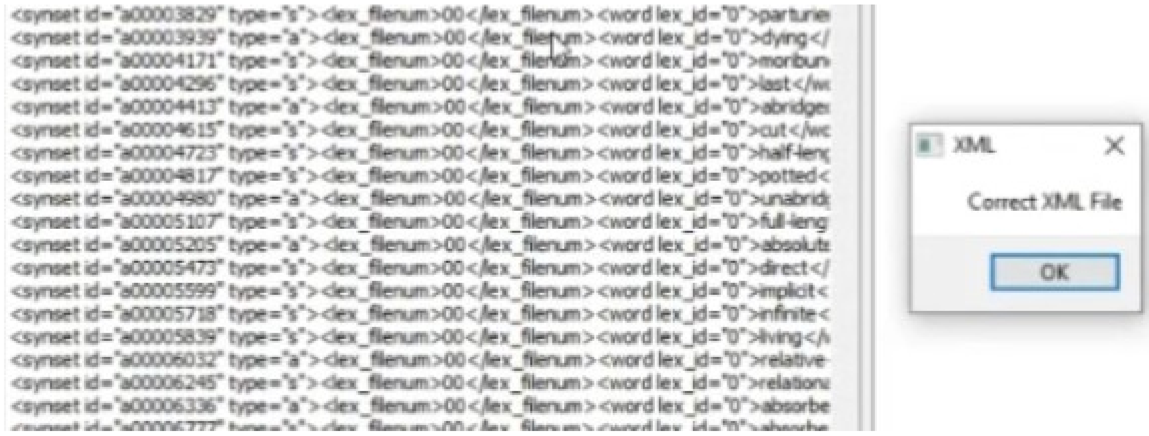
6) Prettify (tags are shown in blue).



```
<data version="3.0">
<synsets source="dict/data.adj" xml:base="data.adj.xml">
<synset id="a00001740" type="a">
<lex_filenum>00</lex_filenum>
<word lex_id="0">able</word>
<pointer refs="n05200169 n05616246">Attribute</pointer>
<pointer refs="n05616246 n05200169" source="1" target="1">Derivationally related form</
pointer>
<pointer refs="a00002098" source="1" target="1">Antonym</pointer>
<def>(usually followed by `to') having the necessary means or skill or know-how or authority
to do something</def>
<example>able to swim</example>
<example>she was able to program her computer</example>
<example>we were at last able to buy a car</example>
<example>able to get a grant for the project</example>
</synset>
<synset id="a00327541" type="s">
<lex_filenum>00</lex_filenum>
<word lex_id="0">cancellate</word>
<word lex_id="0">cancellated</word>
<word lex_id="0">cancellous</word>
<pointer refs="a00327031">Similar to</pointer>
```

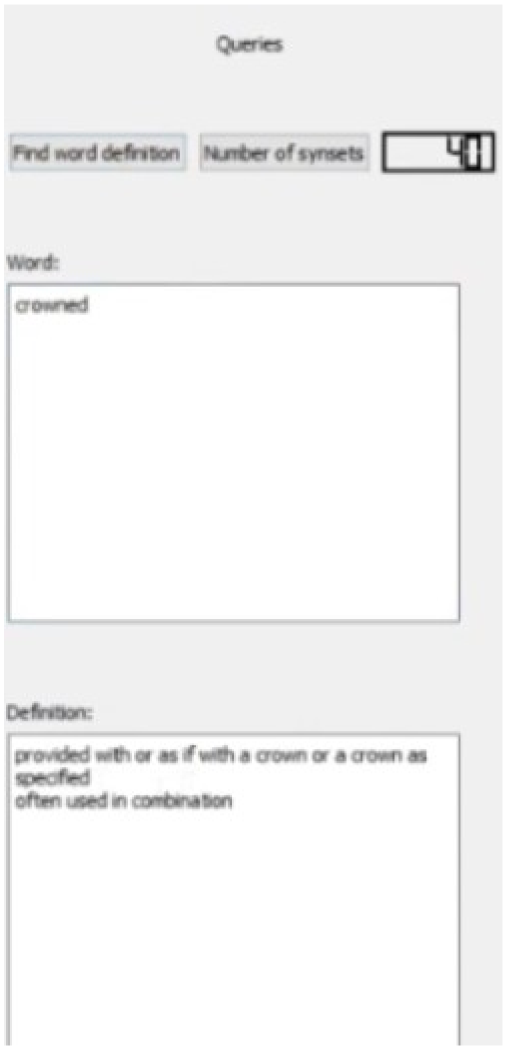
```
<data version="3.0">
 <synsets source="dict/data.adj" xml:base="data.adj.xml">
  <synset id="a00001740" type="a">
   <lex_filenum>00</lex_filenum>
   <word lex_id="0">able</word>
   <pointer refs="n05200169 n05616246">Attribute</pointer>
   <pointer refs="n05616246 n05200169" source="1" target="1">Derivationally related form
   <pointer refs="a00002098" source="1" target="1">Antonym</pointer>
   <def>(usually followed by `to') having the necessary means or skill or know-how or authorit
   <example>able to swim</example>
   <example>she was able to program her computer</example>
   <example>we were at last able to buy a car</example>
   <example>able to get a grant for the project</example>
  </synset>
  <synset id="a00327541" type="s">
   <lex_filenum>00</lex_filenum>
   <word lex_id="0">cancellate</word>
   <word lex_id="0">cancellated</word>
   <word lex_id="0">cancellous</word>
   <pointer refs="a00327031">Similar to</pointer>
   <pointer refs="n06057539">Domain of synset - TOPIC</pointer>
   <def>having an open or latticed or porous structure</def>
```

7) Correct XML file message



```
<synset id="a00003829" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">parturie
<synset id="a00003939" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">dying</
<synset id="a00004171" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">moribun
<synset id="a00004296" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">last</w
<synset id="a00004413" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">abridge
<synset id="a00004615" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">cut</wc
<synset id="a00004723" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">half-leng
<synset id="a00004817" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">potted<
<synset id="a00004980" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">unabrid
<synset id="a00005107" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">full-leng
<synset id="a00005205" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">absolut
<synset id="a00005473" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">direct</
<synset id="a00005599" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">implicit<
<synset id="a00005718" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">infinite<
<synset id="a00005839" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">living</
<synset id="a00006032" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">relative
<synset id="a00006245" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">relation
<synset id="a00006336" type="a"><lex_filenum>00</lex_filenum><word lex_id="0">absorbe
<synset id="a00006777" type="s"><lex_filenum>00</lex_filenum><word lex_id="0">absorbe
```

8) Showing the number of synsets, and the definition of a certain query word.



Word:

crowned

Definition:

provided with or as if with a crown or a crown as specified
often used in combination

03

Third Topic

Complexity of Operations

n: number of synsets.

void on_OpenFileButton_clicked() → o(1)

void on_Prettify_Button_clicked() → o(n)

void on_Save_Button_clicked() → o(1)

void on_JSON_Button_clicked() → o(n)

void on_Remove_Spaces_clicked() → o(n)

void on_Check_Button_clicked() → o(n)

void on_Correct_Button_clicked() → o(n)

void getTagsAndLines() → o(n)

makePureTags() → o(n)

makePureTagsLinesWithoutSlash() → o(n)

Node* makeTree(std::vector<std::string> pureTagsLinesWithoutSlash, Node* current_root) o(n)

Node* getMainParent(Node* root) → o(1)

Node* getLastChild(Node* root) → o(1)

void makeBrackets(Node* root)→ o(n)

void makeOneNodeForRepeatedChild(Node* root)→ o(n)

void organizeTree(Node* root)→ o(n)

void printNode(Node* root)→ o(1)

void print(Node* root)→ o(n)

void makeQutation(Node* root)→ o(n)

void makeJson(Node* root)→ o(1)

void findMistakesLines()→ o(n)

void countSynset(Node* root) → o(n)

void getDef(Node* root)→ o(n)

04

Fourth Topic

References

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). *Introduction to Algorithms, Third Edition*

[2] Tetsuji Kuboyama (2007). "Matching and learning in trees" Doctoral Thesis, University of Tokyo.

[3] "Processing XML with E4X". Mozilla Developer Center. Mozilla Foundation.