///////////////////////////////////////////////////////////////////

// XmlDocument.cpp - a container of XmlElement nodes //

// Ver 1.3 //

// Application: CSE687 Pr#2, Spring 2015 //

// Platform: Dell XPS L520, Core i7, Windows 7 SP3 //

// Author: Vijayendra Ghadge, Syracuse University //

// (315) 728 8953, vvgahdge@syr.edu //

///////////////////////////////////////////////////////////////////

#include <iostream>

#include "XmlDocument.h"

#include <fstream>

#include <vector>

using namespace XmlProcessing;

using sPtr = std::shared\_ptr <AbstractXmlElement>;

XmlProcessing::XmlDocument::XmlDocument(const std::string& src, sourceType srcType)

{

/\* todo \*/

}

sPtr XmlDocument::constructtree(std::string s, bool mode)

{

sPtr Doc\_root=nullptr;

Doc\_root = parseElem(s, mode,Doc\_root);

return Doc\_root;

}

bool XmlDocument::addnewchild(std::string s, bool mode,sPtr parent)

{

parseElem(s, mode, parent);

return true;

}

bool XmlDocument::removechildren(sPtr node)

{

for (auto a : node->getchild())

node->removeChild(a);

return true;

}

std::shared\_ptr < AbstractXmlElement > XmlDocument::parseElem(std::string text,bool mode,sPtr pt)

{

sPtr ptr;

if (pt == nullptr)

{

DocElement doc\_element(pDocElement\_);

ptr = makeDocElement(pDocElement\_);

node.push(ptr);

}

else

{ node.push(pt); }

int size = -1;

try

{

Toker toker; toker.setMode(Toker::xml); toker.attach(text, mode); XmlParts parts(&toker); //parts.verbose(); // uncomment to show token details

while (parts.get())

{

size = parts.length() - 1; //std::cout << parts.show().c\_str() << std::endl;

if (parts[0] == "<" && parts[1] == "?" && parts[2] == "xml" && parts[size - 1] == "?" && parts[size] == ">")

{ ptr = createDecElem(parts); (node.top())->addChild(ptr); }

else if (parts[0] == "<" && parts[1] == "!" && parts[2] == "--" && parts[size - 1] == "--" && parts[size] == ">")

{ ptr = createCommElem(parts); (node.top())->addChild(ptr); }

else if (parts[0] == "<" && parts[1] == "?" && parts[size - 1] == "?" && parts[size] == ">")

{ ptr = createProcElem(parts); (node.top())->addChild(ptr); }

else if (parts[0] == "<" && parts[size] == ">")

{

if (parts[1] != "/")

{

ptr = createTagElem(parts);

if (node.empty()) { node.push(ptr); }

else { (node.top())->addChild(ptr); node.push(ptr); }

}

else if (parts[2] == (node.top())->value() && !node.empty()) { node.pop(); }

else{ std::cout << "\n..............................Malformed XML.........................................\n"; } //exit(0); //std::cout << temp->toString();

}

else if (parts[0] != "<" && parts[size] != ">") { ptr = createTextElem(parts); (node.top())->addChild(ptr); }

}

}

catch (std::exception ex)

{

std::cout << "\n " << ex.what() << "\n\n";

}

if (!node.empty()){

ptr = node.top(); node.pop();

}

return ptr;

}

std::shared\_ptr < AbstractXmlElement > XmlDocument::createDecElem(XmlParts p)

{

sPtr ptr = makeXmlDeclarElement();

for (int i = 1; i < p.length()-1; ++i)

{

if (p[i] == "=")

{

ptr->addAttrib(p[i - 1], getval(p[i + 1]));

}

} //std::cout << ptr->toString();

return ptr;

}

std::shared\_ptr < AbstractXmlElement > XmlDocument::createCommElem(XmlParts p)

{

sPtr ptr;

std::string temp;

for (int i = 3; i < p.length() - 2; ++i)

{

temp = temp + p[i]+" ";

}

temp.erase(temp.end() - 1);

ptr = makeCommentElement(temp);

//std::cout << ptr->toString();

return ptr;

}

std::shared\_ptr < AbstractXmlElement > XmlDocument::createProcElem(XmlParts p)

{

sPtr ptr;

std::string temp;

int flag = 0;

ptr = makeProcInstrElement(p[2]);

for (int i = 2; i < p.length()-1; ++i)

{

if (p[i] == "=")

{

ptr->addAttrib(p[i - 1], getval(p[i + 1]));

flag = 1;

}

}

if (flag == 0)

{

for (int i = 2; i < p.length() - 3; ++i)

{

ptr->addAttrib(p[i], p[i + 1]);

}

}

//std::cout << ptr->toString();

return ptr;

}

std::shared\_ptr < AbstractXmlElement > XmlDocument::createTagElem(XmlParts p)

{

sPtr ptr;

std::string temp;

ptr = makeTaggedElement(p[1]);

for (int i = 2; i < p.length() - 1; ++i)

{

if (p[i] == "=")

{

ptr->addAttrib(p[i - 1], getval(p[i + 1]));

}

}

//std::cout << ptr->toString();

return ptr;

}

std::shared\_ptr < AbstractXmlElement > XmlDocument::createTextElem(XmlParts p)

{

sPtr ptr;

std::string temp;

for (int i = 0; i < p.length(); ++i)

{

temp = temp + " " + p[i];

}

ptr = makeTextElement(temp);

//std::cout << ptr->toString();

return ptr;

}

std::string XmlDocument::getval(std::string x)

{

x.erase(x.begin());

x.erase(x.end() - 1);

//std::cout << x;

return x;

}

//----< recursive left-to-right Depth First Search >-------------------------

std::vector<sPtr> XmlDocument::findElembytag(sPtr pNode,const std::string tag)

{

for (auto pChild : pNode->getchild())

{

if (pChild->value() == tag)

{

found\_.push\_back(pChild);

}

findElembytag(pChild,tag);

}

return found\_;

}

std::vector<sPtr> XmlDocument::findElembyVal(sPtr pNode, const std::string val)

{

std::vector<std::pair<std::string, std::string>> att;

for (auto pChild : pNode->getchild())

{

att = pChild->getattrib();

for (auto a : att)

{

if (a.second==val)

found\_.push\_back(pChild);

}

findElembyVal(pChild, val);

}

return found\_;

}

std::vector<sPtr> XmlDocument::findElembyId(sPtr pNode, const std::string id)

{

std::vector<std::pair<std::string, std::string>> att;

for (auto pChild : pNode->getchild())

{

att = pChild->getattrib();

for (auto a : att)

{

if (a.first == id)

found\_.push\_back(pChild);

}

findElembyId(pChild, id);

}

return found\_;

}

void XmlDocument::clearvector()

{

found\_.clear();

}

#ifdef TEST\_XMLDOCUMENT

int main()

{

title("Testing XmlDocument class");

sPtr root,x,hold,y;

std::vector<sPtr> temp,temp2;

std::vector<std::pair<std::string, std::string>> att;

/\*const std::string s = "<author>Jeff Prosise <hello> <men>Men will be men!</men> </hello><note Company = 'Wintellect'>< / note>< / author>";

XmlDocument doc(s, XmlDocument::string);

root = doc.constructtree(s,false);\*/

std::string s = "LectureNote.xml";

XmlDocument doc(s, XmlDocument::filename);

root = doc.constructtree(s, true);

std::cout << root->toString();

//root=doc.constructtree();

/\* std::string addchilren = "<author>Jeff Prosise <hello> <men>Men will be men!</men> </hello><note Company = 'Wintellect'>< / note>< / author>"; root = doc.addnewchild(addchilren, false,root); std::cout << root->toString();\*/

//std::cout << root->toString(); //temp=doc.findElembytag(root,"note"); //for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) //{ //x=\*i; // std::cout <<x->toString(); //}

/\* temp = doc.findElembyId(root, "Company"); for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) { x = \*i; std::cout << x->toString(); }\*/

/\* temp = doc.findElembyVal(root, "1.0"); for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) { x = \*i; std::cout << x->toString(); } \*/

// temp=doc.findElembytag(root,"date"); // for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) // { // x=\*i; //// std::cout <<x->toString(); // std::string addchilren = "<author>Jeff Prosise <hello> <men>Men will be men!</men> </hello><note Company = 'Wintellect'>< / note>< / author>"; // if (doc.addnewchild(addchilren, false, x)) // std::cout << root->toString(); // }

//temp = doc.findElembytag(root, "reference"); // for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) // { // x=\*i; // std::cout <<x->toString(); // std::cout << "\n\n"; // doc.removechildren(x); // std::cout << root->toString(); // }

//temp = doc.findElembyId(root, "version"); // for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) // { // x=\*i; // std::cout <<x->toString(); // std::cout << "\n\n"; // att=x->getattrib(); // for (auto a : att) // { // std::cout << a.first; // std::cout << a.second; // } // }

//temp = doc.findElembytag(root, "author"); // for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) // { // x=\*i; // std::cout <<x->toString(); // std::cout << "\n\n"; // temp2=x->getchild(); // for (std::vector<sPtr>::iterator i = temp2.begin(); i != temp2.end(); ++i) // { // y=\*i; // std::cout <<y->toString(); // } // }

//temp = doc.findElembytag(root, "LectureNote"); // for (std::vector<sPtr>::iterator i = temp.begin(); i != temp.end(); ++i) // { // x=\*i; // std::cout <<x->toString(); // std::cout << "\n\n"; // std::string str1, str2; // str1 = "attribute\_name"; // str2 = "attribute\_value";; // x->addAttrib(str1, str2); // std::cout << x->toString(); // x->removeAttrib(str1); // x->removeAttrib("course"); // std::cout << x->toString(); // }

/\* std::ofstream myfile; myfile.open("output.xml"); myfile << root->toString(); myfile.close();\*/

std::cout << "\n\n";

return 0;

}

#endif