# What goals were set?

First of all we create a sufficiently compact HTML parser capable of creating DOM and which can select some elements from DOM by tag name or some attributes. For example with jQuery selector we can say: Hey, jQuery, find my HTML element with id equals to “bla-bla” and class equals to “coo-coo” `$.("#bla-bla .coo-coo)` and then we can work with this element. It's really very easy to use. But it will take too much time to implements all such features. That's why I decided to implement only some part of features that can do any production-ready library. So I implemented:

- Downloading HTML page via HTTP
- Making DOM
- Finding some HTML errors (unclosed tags, for example)
- Finding elements by tag name
- Parsing and saving attributes for HTML tags

# Introduction

What is a HTML parser and parsing at all?

***Parsing*** or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. So, that's why HTML parser should analyze the all HTML document(string) with some HTML rules and 'find interesting information' in this document. And may be it can show some errors or just a fact that there are some errors;

Parsing HTML is a automated task, performed by (so called) HTML parsers. They have two main purposes:

- HTML traversal: offer a interface for programmers to easily access and modify of the "HTML string code". Canonical example: DOM parsers.
- HTML clean: to fix invalid HTML and to improve the layout and indent style of the resulting markup. Canonical example: HTML Tidy.

My work is more similar to HTML traversal type of parsers.

### Which HTML parsers we have on C/C++

- Gumbo
- HTML Tidy
- Hubhub
- libxml
- HTMLReader
- etc.

## What is DOM

The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents.Objects in the DOM tree may be addressed and manipulated by using methods on the objects. For example we have root element `html` tag and then we have his children. For `html` it usually `body,head` or some other. And every this children can has his own elements(children). And this children will be sublings for each other and they will have one parent.

# CParser.class

So, I wrote a very simple class to parse html `CParser`. It hasn't default constructor but it has constructor with one parameter(string) and we can pass to it HTML inline: `<html><body><div class='btn'>....</div></body></html>` for example or we can pass string with URI likes this `http://localhost` (it doesn't use any external libraries, just Windows sockets). Then DOM will be created and you can use some methods.

Sometimes people use regular expressions to parse HTML. Because it's hardcorder and don't wont to use external libraries for the small task. In C++11, language has lib , which partly came from boost.

### For example we can write all HTML tags simple:

```
for (auto a : parser.allTags){
    for (int i = 0; i < a->deep; i++) cout << " ";
        cout << a->tagName;
        cout << "\n";
}
```

### And result will be looks like this:

```
html
  title
  /title
  a
   font
   /font
  /a
  div
   div
    div
     br/
    /div
   /div
  /div
 /html
```

**Not so interesting maybe, but we can do not just it.**

So, if some HTML tags have some attributes, for example HTML tag `a` usually have `href` or something similar to `click` method for JavaScript. Or it can have `id` or `class` for extending with JS or CSS. So, for every HTML tag we have list of all attributes and we can see all atributes very easy. Likes this:

```
tagToFind = "div";
for (auto a : parser.getByTag(tagToFind)){
        cout << "Tag name: " << a->tagName;
        cout << "\n  " << "Attributes: ";
            for (auto m : a->getAttributes()){
                for (auto ii = m.begin(); ii != m.end(); ii++)cout <<"\n    " << ii->first << "=" << ii->second;
        }
```

### And result will be looks likes this:

```
Tag name: div
  Attributes:
    id=firstDiv
----------
Tag name: div
  Attributes:
    id=secondDiv
----------
Tag name: div
  Attributes:
    class=123
    id=thirdDiv
```

**In this example** I used some other method `getByTag()`. In these method we pass string with tag name and get `vector<CNode*>` as a result. `CNode` is a class for every tag in DOM.

## Text inside tag

So, sometimes we need to select some text in container (between open and closed tag). And we can do it easilly. Every node has field `innerText` and we can see text inside this tag. For example, with following C++ code

```
for (auto a : parser.allTags){
    for (int i = 0; i < a->deep; i++) cout << " ";
        cout << a->tagName << "<" << a->deep << ">" << "<<< " << a->innerText << " >>>";
        cout << "\n";
}
```

We will se output likes this

```
html<<<
hello page begin
hello end
 >>>
  title<<< myTitle >>>
  /title<<<  >>>
  a<<< LinkName >>>
   font<<< LocalHost >>>
   /font<<<  >>>
  /a<<<  >>>
  div<<< Some interesting text inside first div >>>
   div<<< Some text inside second div >>>
    div<<< text inside third div >>>
     br/<<< text new line>>>
    /div<<<  >>>
   /div<<<  >>>
  /div<<<  >>>
 /html<<< >>>
```

For the following HTML file

```
<html>
<title>myTitle</title>
hello page begin
<a href="localhost"><font color="red">LocalHost</font>LinkName</a>
<div id="firstDiv">
Some interesting text inside first div
<div id="secondDiv">
Some text inside second div
<div class="123" id="thirdDiv">
text inside third div<br/>text new line
</div>
</div>
</div>
hello end
</html>
```

It's so dirsty example. May be it will be better to print only some text inside one tag. For example, if we can to choose title of our HTML pag ewhich will we shown in title of windows or tab in our browser we can with the following C++ code

```
string selectTitleInsideText = "title";
for (auto a : parser.getByTag(selectTitleInsideText)){
cout << "Title of HTML page is: " << a->innerText;
}
```

And output will be likes this: Title of HTML page is: myTitle

## Finding HTML errors

Sometimes frontend developers forget to close tag(hm,of course all modern IDEs create auto and closing tag... but... nothing can help some people) thats's parser can show some errors likes unclosed tag or closing tag without opening tag. CParser.class hasn't a method or flags to detect this error but we can write some additional code by ourselves to implements this feature. So, we have DOM that's we we have all elements in our HTML file. And if we have unclosed/overclosed tags we can find it. In the following code we find opening and closing tags and put them into 2 `vectors`

```
vector<CNode*> openingTags, closingTags;
for (auto a : this->allTags){
    if ((a->tagName != "br" || a->tagName != "br/") && !a->isSelfClosing){
        if (a->tagName[0] != '/'){
            openingTags.push_back(a);
        }
        else{
            a->tagName = a->tagName.erase(0,1);
            closingTags.push_back(a);
        }
    }
}
```

I these code we write `br` because it's self-closing and additionally we won't to see in opening and closing tags. For example, we can do the same thing with some other tags

Then we should find differences between two vectors with custom type `CNode*`. It's not the best idea to use something like this

```
vector<string> diff;
set<string> openingTagsString, closingTagsString;

for (auto a : openingTags){
    openingTagsString.insert(a->tagName);
}

for (auto a : closingTags){
    closingTagsString.insert(a->tagName);
}
std::set_symmetric_difference(openingTagsString.begin(),
 openingTagsString.end(), closingTagsString.begin(), closingTagsString.end(),
    inserter(diff, diff.begin()));
```

Because it will cause problem. It will find only unique elements. For example if we have two tags `div` it will be equal to one. Similar problem will be and with `lists,vector`, for example.

But to continue we can write some short lambda expression or write code which will try do find for the every opening element closing and delete these two elements from list.

But it's not interesting and we will write as much code as we can.

```
    int aPosition = 0, bPosition = 0;

    for (auto a : openingTags){
        bPosition = 0;
        aPosition++;
        for (int i = 0; i < closingTags.size(); i++){
            this->aPositions.push_back(4);
        }

        for (auto b : closingTags){
            bPosition++;
            if (!(a->tagName.compare(b->tagName))){
                openingTags.erase(openingTags.begin()+aPosition );
                aPosition--;
                closingTags.erase(closingTags.begin()+bPosition);
                break;
            }
        }
```

So, and if we have some problems, in the opening or closing vectors will be some elements. The type of this element will be CNode*. So, we can find text inside this tag/container and find tag name.

Source code of all files

## How it works? just in a few words.

We download all HTML file from the network or just from constructor if it's inline. Then we try to find symbol `<`. If we find it, that's mean that then will be name of the tag `<tagName...` and after space can be attributes of the tag: `<tagName attr1="value1" attrr2="value2"...` but it's not necessarily for tag. Then we try to find symbol `/` before closing tag. If we find `/` it means that it's self-closing tag and he hasn't any children. Moreover we know that there are some self-closing tag by default or without string error:

- br

- tr
- th
- td
- li
- dt
- dd
- dl
- p
- nobt
- b

## Main.cpp - examples of using

```
#include "stdafx.h"
#include <string>
#include <iostream>
#include "Parser.h"

using namespace std;

auto main() -> int
{
    //auto parser = *new CParser("<html> start html <head> inside head </head> <body>
    <div insideDIV1 /> <div SecondInner2 /><div> inside DIV3 </div>BodyText</body>
    before close html</html> <div ll> third div4</div> ");
    auto parser = *new CParser("http://localhost/index.html");
    cout << "\n\n";

    cout << "+++++++++ Printing DOM - All tags +++++++++\n";

    for (auto a : parser.allTags){
        for (int i = 0; i < a->deep; i++) cout << " ";
            cout << a->tagName ;
            cout << "\n";
    }
    cout << "+++++++++ END +++++++++\n";
    cout << "\n\n";
    cout << "\n";


    cout << "+++++++++ Printing DOM - All tags with inner text +++++++++\n";

    for (auto a : parser.allTags){
        for (int i = 0; i < a->deep; i++) cout << " ";
            cout << a->tagName << "<" << a->deep << ">" << "<<< " << a->innerText << " >>>";
            cout << "\n";
    }
    cout << "+++++++++ END +++++++++\n";
    cout << "\n\n";

    string tagToFind = "div";

    cout << "+++++++++ Finding in DOM by tag: \"" << tagToFind <<"\" +++++++++\n";

    for (auto a : parser.getByTag(tagToFind)){
        cout << "Tag name: " << a->tagName;
        cout << "\n  " << "Attributes: ";

        for (auto m : a->getAttributes()){
            for (auto ii = m.begin(); ii != m.end(); ii++)cout <<"\n" << ii->first << "=" << ii->second;
        }
        cout << "\n" << "----------------------" << "\n";
    }
    string selectTitleInsideText = "title";

    cout << "\n\n";
    cout << "+++++++++ Finding and printing title of HTML page +++++++++\n";
    for (auto a : parser.getByTag(selectTitleInsideText)){
    cout << "Title of HTML page is: " << a->innerText;
    }

    cout << "\n" << "----------------------" << "\n";



    cout << "\n";

    return 0;

}
```

## Parser.h

```
#pragma once

#include <string>
#include "Node.h"
using namespace std;

class CParser
{
public:
    ~CParser();
    // Url (http://localhost) or html inline //
    CParser(string url);

    auto makeDOM()->void;
    auto getByTag(string tag)->vector<CNode*>;
    bool hasError = false;
    vector<CNode*> allTags;

private:
    CNode* rootNode = nullptr;
    string html;

    CParser();

};
```

## Node.h

```
#pragma once

#include <vector>
#include <map>


using namespace std;



class CNode
{
public:
    CNode();
    ~CNode();

    auto setTagName(string tag)->void;
    auto setTagAttributes(vector<string> attributes) -> void;
    auto getAttributes()->vector<map<string, string>>;
    auto setDeep(int d) -> void;

    bool isSelfClosing = false;

    int deep=0;
    string tagName;
    vector<map<string, string>> attributes;
    string innerText;

private:


    string classHTML;
    string idHTML;

};
```

## Node.cpp

```
#include "stdafx.h"
#include "Node.h"



CNode::CNode()
{
}



CNode::~CNode()
{
}

auto CNode::setDeep(int deep)->void{
    this->deep = deep;
}

auto CNode::setTagName(string tag)->void{
    this->tagName = tag;
}

auto CNode::getAttributes()->vector<map<string, string>>{
    return this->attributes;
}
```

## Parser.cpp

```
#include "stdafx.h"
#include "Parser.h"
#include <vector>
#include <iostream>
#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <map>
#include <string>
#include <algorithm>
#include <set>

#pragma comment(lib,"ws2_32.lib")



using namespace std;
```

```
auto CParser::getByTag(string tag) -> vector<CNode*>{

    vector<CNode*> result;
    vector<string> tagToFind;

    vector<string> t;
    string sr;

    for (int i = 0; i < tag.size(); i++){
        if (i == 0){ sr += tag[i]; }
            if (tag[i] == ' '){
                tagToFind.push_back(sr);
                sr.clear();
        }
    }

    for (auto r : this->allTags){
        if (r->tagName == tag) result.push_back(r);
    }

    return result;
}


auto CParser::makeDOM() -> void{

    string html = this->html;

    bool onTagName = false;
    bool onTagLocation = false;
    string currentTag;
    int currentDeep = 0;

    vector<map<string, string>> NodeAttributes;
    map<string, string> s;

    string tempStringForAttributeName, tempStringForAttributeValue;
    bool onAttributeName = false, onAttributeValue = false;

    CNode* prevElement = nullptr;
    CNode* parentElement = nullptr;
    CNode* rootNode = nullptr;

    auto reversedAllTags = this->allTags;

    char currentChar = 0;
    char prevChar = 0;

    for (unsigned int i = 0; i < html.length(); i++){

        prevChar = currentChar;
        currentChar = html[i];

        /// If it is closed tag we shoud use parent Node
        if (!onAttributeName && !onAttributeValue && !onTagLocation
        && !onTagName && currentChar != '<' && currentChar != '>'){
            if (prevElement != nullptr){
                if (prevElement->tagName[0] == '/'){
                    reversedAllTags = this->allTags;
                    std::reverse(reversedAllTags.begin(), reversedAllTags.end());
                    for (auto a : reversedAllTags){
                        if (a->deep == currentDeep && a->tagName != "br/" &&  a->tagName != "br"){
                            prevElement = a;
                            break;
                        }
                    }
                }
                prevElement->innerText += currentChar;
            }
        }

        // If we are in tag and this thar is last
        if (onTagLocation && (currentChar == '>')){
            onTagName = false;
            onTagLocation = false;

            currentDeep++;

            if (prevChar == '/') currentDeep -= 1; //  <tagName "/">

            if (currentTag[0] == '/'){ // </...>
                currentDeep -= 1;
            }

            if (prevChar == '/' && currentTag == "br/") currentDeep++;

            CNode* node = new CNode();
            node->setTagName(currentTag);
            node->setDeep(currentDeep);
            node->attributes = NodeAttributes;

            if (currentTag == "br") {
                currentDeep--;
            }

            if (prevChar == '/'){
                currentDeep -= 1; //  <tagName "/">
                node->isSelfClosing = true;
            }

            if (currentTag[0] == '/'){ // </...>
                currentDeep -= 1;
            }

            if (prevElement == nullptr){
                // If it's root element
                if (rootNode == nullptr) rootNode = node;

                prevElement = node;
            }
            else{
                prevElement = node;
                // Not root element
            }
            this->allTags.push_back(node);
            currentTag.clear();
        }

//////////////////////////////////////////////////////////////////////////
        if (onTagName){
            if (currentChar == ' '){
                onTagName = false;
            }
            else{
                currentTag += currentChar;
            }
        }

        if (currentChar == '<'){
            onTagName = true;
            onTagLocation = true;
            NodeAttributes.clear();
        }

        ///HTML attribute
        if (onTagLocation && !onTagName){
            if (onAttributeValue && (currentChar == '"' || currentChar == '\'')){
                s[tempStringForAttributeName] = tempStringForAttributeValue;
                NodeAttributes.push_back(s);

                tempStringForAttributeName.clear();
                tempStringForAttributeValue.clear();

                s.clear();

                onAttributeName = false;
                onAttributeValue = false;
            }


            if (currentChar != ' ' && (currentChar != '"' && currentChar != '\'')){
                if (onAttributeValue && !onAttributeName) tempStringForAttributeValue += currentChar;
                if (onAttributeName && currentChar == '=') {
                    i++;

                    onAttributeValue = true;
                    onAttributeName = false;

                }
                else{
                    if (onAttributeValue == false){
                        onAttributeName = true;
                        tempStringForAttributeName += currentChar;
                    }
                }
            }
        }
    }

    // Unclosed tag or one more closed tag
    if (currentDeep != 0) this->hasError = true;

}

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


CParser::CParser(string url){
    // TODO: refactoring, not funny
    if (url[0] == 'h' && url[1] == 't' && url[2] == 't' && url[3] == 'p'){

        /// Get DATA from url
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            cout << "WSAStartup failed.\n";
        }

        string hostname;
        hostname = url.substr(7);
        string t, suburl;

        for (int i = 0; i < hostname.length(); i++){
            if (hostname[i] != '/') { t += hostname[i]; }
            else{
                suburl = hostname.substr(t.length() + 1);
                break;
            }
        }
        /// t - real hostname ( "localhost")
        /// suburl - all after / "localhost/blabla"

        SOCKET Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        struct hostent *host;
        host = gethostbyname(t.c_str());
        SOCKADDR_IN SockAddr;
        SockAddr.sin_port = htons(80);
        SockAddr.sin_family = AF_INET;
        SockAddr.sin_addr.s_addr = *((unsigned long*)host->h_addr);
        if (connect(Socket, (SOCKADDR*)(&SockAddr), sizeof(SockAddr)) != 0){
            cout << "Could not connect";
        }
        string result;
        string s1 = "GET /" + suburl + " HTTP/1.1\r\nHOST: "+t+"\r\nConnection: close\r\n\r\n";

        send(Socket, s1.c_str(), strlen(s1.c_str()), 0);
        char buffer[10];
        int nDataLength;
        while ((nDataLength = recv(Socket, buffer, 10, 0)) > 0){
            int i = 0;
            while (buffer[i] >= 32 || buffer[i] == '\n' || buffer[i] == '\r') {
                result += buffer[i];
                i += 1;
            }
        }
        closesocket(Socket);
        WSACleanup();
        url = result;
    }

    url.erase(url.find_last_not_of(" \n\r\t") + 1); // trim
    url.erase(std::remove(url.begin(), url.end(), '\t'), url.end());

    this->html = url;
    this->makeDOM();

}


CParser::CParser()
{
}


CParser::~CParser()
{
}
```

## What I can do more for this project

So, this work was very interesting for me and **I believe that you will appreciate it well**.

It will be interesting to implements some features but a lot of some other such as working with parent/subling/children element in the DOM, and modification DOM, complex finding such as `findByTag(".body .div a")`

# Thank's for reading!