Upload: cpp2html.cpp
Type: application/octet-stream
Size: 4.669921875 Kb
Upload complete. Save this receipt for your records:

Receipt: 026a77535d8ef5438677aed847f5201bf5c21b45

Below is the content of your submission

========== START OF FILE ==========

```
/*
 * CSc103 Project 5: Syntax highlighting, part two.
 * See readme.html for details.
 * Please list all references you made use of in order to complete the
 * assignment: your classmates, websites, etc.  Aside from the lecture notes
 * and the book, please list everything.  And remember- citing a source does
 * NOT mean it is okay to COPY THAT SOURCE.  What you submit here **MUST BE
 * YOUR OWN WORK**.
 * References:
 * Sultan Alreyashi, classmate
 *
 * Finally, please indicate approximately how many hours you spent on this:
 * #hours: ~ 3-5 Hours
 */

#include "fsm.h"
using namespace cppfsm;
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <string>
using std::string;
#include <set>
using std::set;
#include <map>
using std::map;
#include <initializer_list> // for setting up maps without constructors.

// enumeration for our highlighting tags:
enum {
        hlstatement,  // used for "if,else,for,while" etc...
        hlcomment,    // for comments
        hlstrlit,     // for string literals
        hlpreproc,    // for preprocessor directives (e.g., #include)
        hltype,       // for datatypes and similar (e.g. int, char, double)
        hlnumeric,    // for numeric literals (e.g. 1234)
        hlescseq,     // for escape sequences
        hlerror,      // for parse errors, like a bad numeric or invalid escape
        hlident       // for other identifiers.  Probably won't use this.
};

// usually global variables are a bad thing, but for simplicity,
// we'll make an exception here.
// initialize our map with the keywords from our list:
map<string, short> hlmap = {
#include "keywords.txt"
};
// note: the above is not a very standard use of #include...

// map of highlighting spans:
map<int, string> hlspans = {
        {hlstatement, "<span class='statement'>"},
        {hlcomment, "<span class='comment'>"},
        {hlstrlit, "<span class='strlit'>"},
        {hlpreproc, "<span class='preproc'>"},
        {hltype, "<span class='type'>"},
        {hlnumeric, "<span class='numeric'>"},
        {hlescseq, "<span class='escseq'>"},
        {hlerror, "<span class='error'>"}
};
// note: initializing maps as above requires the -std=c++0x compiler flag,
```

```
// as well as #include<initializer_list>.  Very convenient though.
// to save some typing, store a variable for the end of these tags:
string spanend = "</span>";

string translateHTMLReserved(char c) {
        switch (c) {
                case '"':
                        return "&quot;";
                case '\'':
                        return "&apos;";
                case '&':
                        return "&amp;";
                case '<':
                        return "&lt;";
                case '>':
                        return "&gt;";
                case '\t': // make tabs 4 spaces instead.
                        return "    ";
                default:
                        char s[2] = {c,0};
                        return s;
        }
}

string syntaxHighlight(string line){
string temp = "";
string output = "";
int state, oldstate;
map<string, short>::iterator it;
state = 0;

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

        oldstate = updateState(state, line[i]);

        if (oldstate == state){
                temp += translateHTMLReserved(line[i]);
        }

        else if (oldstate != state || line.length() == i){ //if the state has changed
                switch (oldstate){

                        case start:
                        output += temp;
                        break;

                        case scanid:
                        it = hlmap.find(temp);
                        if (it != hlmap.end())
                                output += hlspans[hlmap[temp]] + temp + spanend;
                        else
                                output += temp;
                        break;

                        case strlit:
                        if (state == start){
                                output += hlspans[hlstrlit]+ temp + '\"' + spanend;
                                i++;
                                oldstate = updateState(state,line[i]); //move it up one for the quotes
                                }
                        else
                                output += hlspans[hlstrlit] + temp + spanend;
                        break;

                        case readfs:
                        if (state == comment){ //it's comments all the way
                                output += hlspans[hlcomment];
                                for (int j = i; j < line.length(); j++){
                                        temp += line[j];
                                }
                                output += temp + spanend;
                                return output;
                        }
                        break;
```

```
                        case readesc:
                        if (state == error){
                                output += hlspans[hlerror];
                                for (int j = i; j < line.length(); j++){
                                        temp += line[j];
                                }
                                output += temp + spanend;
                                return output;
                        }
                        else{
                                output += hlspans[hlescseq] + temp + line[i] + spanend;
                                i++; //advance it past the escape sequence
                                oldstate = updateState(state, line[i]);
                        }
                        break;

                        case scannum:
                        if (state == error){
                                output += hlspans[hlnumeric] + temp + spanend + hlspans[hlerror];
                                temp = ""; //numbers in 99XYY remain highlighted as numbers
                                for (int j = i; j < line.length(); j++){//go until the end of the string!
                                        temp += line[j];
                                }
                                output += temp + spanend;
                                return output;
                                        }
                                output += hlspans[hlnumeric] + temp + spanend;
                                break;
                        }
                        temp = translateHTMLReserved(line[i]);
                }
        }
        output += temp;
        return output;
}


int main() {
        string input, output;

        while (getline(cin, input)){
                output = syntaxHighlight(input);
                cout << output << "\n";
        }
        return 0;
}
```

========== END OF FILE ==========