Resaltador de sintaxis v1.0

Delivery Date	@September 12, 2021
Equipo	Individual
Materia	Implementación de métodos computacionales

Carla Oñate Gardella A01653555

Conclusiones

Esta actividad la sentí un poco complicada porque tuve algunos problemas encontrando las reglas correctas. Pero al final logré resaltar todos los problemas necesarios. Mi código lo hice en el mismo archivo main ya que no ví la necesidad de generar una clase. En general lo que hice fue primero crear las expresiones, luego checar con cada expresión cada línea y hacer los cambios. Para cambiar comentarios no logré hacerlo con una expresión regular ya que la expresión regular para reconocer identificadores reconocía al igual los comentarios y no logré corregir esto. Creo que esta actividad que hice cumple con todo lo que se pidió en la actividad. Aunque el diseño de mi resaltar esta un poco feo porque soy mala escogiendo colores y luego contrastan mucho entonces ya no se ve bien. Esta actividad creo tiene cosas que se pueden mejorar pero como se entregó funciona bien también.

Cosas a mejorar

Para reconocer los **identificadores y palabras reservadas** tuve que crear una **función** en donde se pudiera **diferenciar** entre una y otra. Esta parte es la que primero se reemplaza porque la regla que forme tomaba las palabras span o class que estaban en la línea y habían sido modificadas antes y las volvía a intercambiar con tags de HTML entonces se hacían varios tags uno dentro de otro. **Este error** solo lo pude resolver al **reemplazar primero** en la línea los **identificadores** y luego los demás lexemas que no confunden las palabras dentro de los tags de HTML.

Creo que hay otras funcionalidad que tiene c++ para manejar las regex que no conozco. Por lo que **siento** que mi **solución podría estar mejor y más sencilla**. Pero tuve conflictos con algunos símbolos que se pueden usar en regex pero mi programa no los aceptaba entonces eso al igual hizo difícil la resolución de esta situación problema.

Intente poner un **archivo de CSS** por **separado** pero este **no funcionaba**, ya que no son muchos estilos decidí meterlos cono inline style en el archivo. Sin embargo esto no me gusta mucho y preferiría que estuviera en un archivo separado.

Para reconocer los **comentarios** tuve que hacerlo con un `if` pero el problema de esto es que **asume** que **toda la línea es un comentario**. Por lo que si el comentario empieza después se marcaría toda la línea como comentario.

En la parte de números mi regla **reconoce bien si es número entero o decimal**. Y esto funciona bien en las primeras dos líneas **pero en la línea 7** donde se tiene `3.1416` este **no es reconocido** y se deja en blanco por lo que no se le agregan estilos. No logré entender porque ese en especifico no se coloreaba. Todas mis regex las probé en un **sitio web** y ahí **mostraba que la regex si reconocía ese número** pero en mi programa no se le agregaba el tag. Intenté resolver este problema pero no tuve éxito.

En general me hubiera gustado que se pudiera hacer esta actividad teniendo solo la parte de `replaceMatchforHTML()` en donde se tiene un switch y la función de `regex_replace` hiciera todo el trabajo. Así es como lo intente hacer desde un inicio pero mientras surgían problemas tuve que cambiar de estrategia y resultó en un código más largo.

Github link

Código

```
#include <iostream>
#include <regex>
#include <fstream>
#include <vector>

//Carla Oñate Gardella A01653555

std::string identifierToHtml(std::string line, std::regex reg){
    std::vector<std::string> reservedWords = {"define", "lambda", "if", "cond",
 "else", "true", "false", "nil", "car", "cdr", "cons", "list", "apply",
 "map", "let", "begin", "null?", "eq?", "set!"};

//Store matches inside vector - we do this to be able to later
//remove duplicates of same match
   std::vector<std::string> nonDuplicatesMatches;
   std::regex_iterator<std::string::iterator> it (line.begin(), line.end(), reg);
   std::regex_iterator<std::string::iterator> end;
   while(it != end){
        nonDuplicatesMatches.push_back(it->str());
        ++it;
   }


  //Removing duplicate matches from vector
 std::sort(nonDuplicatesMatches.begin(), nonDuplicatesMatches.end());
 auto last = std::unique(nonDuplicatesMatches.begin(), nonDuplicatesMatches.end());
 nonDuplicatesMatches.erase(last, nonDuplicatesMatches.end());

 //Replace matches with HTML tag
 std::string replace;                                  //string to store html tag to be replaced later
 std::string replacedString = line;                    //modifications of line are being stored here
 for(const auto& el : nonDuplicatesMatches){
//If match is reserved word then class is different
    if(std::find(reservedWords.begin(), reservedWords.end(), el) != reservedWords.end()){   //If match is found on reserved words vector
        replace = "<span class=\"reserved\">";
    } else {
        replace = "<span class=\"identifier\">";
    }
    replace.append(R"($&)");
    replace.append("</span>");

    //Create string to turn to regex -> \b(el)\b - this regex is the one
//used in the regex_replace function, this to prevent errors in matching
  std::string regexString = "\\b("; regexString.append(el); regexString.append(")\\b");
  std::regex reservedReg(regexString);  //Regular expression equal to the match word
  replacedString = std::regex_replace(replacedString, reservedReg, replace);
  }
  return replacedString;  //Return modified string with HTML tags for the identifiers or reserved words only
}



std::string replaceMatchforHTML(std::string& line, std::regex reg, int type){
    if(type == 0){
        //If string has ; then manually set tag to comment
        if(line.find(';') != std::string::npos){
            return "<span class=\"comment\">" + line + "</span>";
        }
        return identifierToHtml(line, reg);   //Function to filter
//identifiers and reserved words to then replace them in the line.
    } else {
        //Switch to change span tag class depending the type of regex
        std::string replaceString = "<span class=";
            switch (type) {
                case 1:
                    replaceString.append("\"specials\"");
                break;
                case 2:
                    replaceString.append("\"decimal\"");
                break;
                case 3:
                    replaceString.append("\"exp\"");
                break;
                case 4:
                    replaceString.append("\"parenthesis\"");
                break;
                default:
                    replaceString.append("\"default\"");
                break;
```

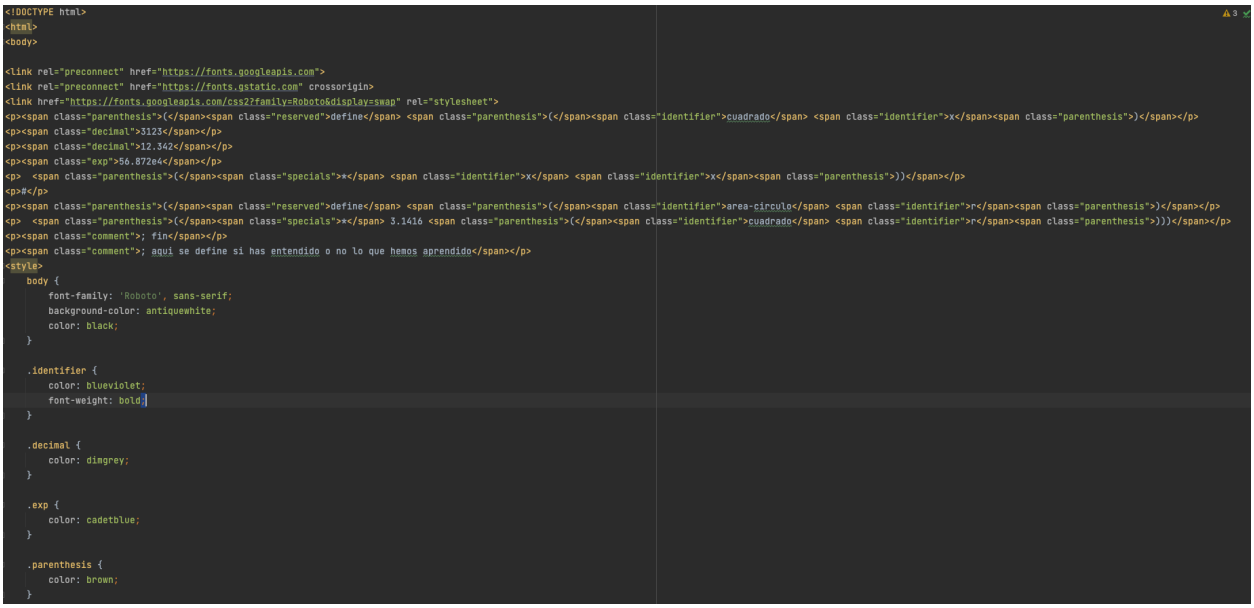
```
            }
        replaceString.append(">");
        replaceString.append(R"($&)");
        replaceString.append("</span>");
//Returns the line with the new HTML tags
        return std::regex_replace(line, reg, replaceString);
    }
}


void createHTMLfile(const std::vector<std::string>& result){
    std::fstream outputFile;
    outputFile.open("output.html", std::ios::out | std::ios::app);
    //We write the basic tags for any HTML file
    outputFile << "<!DOCTYPE html>\n" << "<html>\n" << "<body>\n" << std::endl;
    outputFile << "<link rel=\"preconnect\" href=\"https://fonts.googleapis.com\">\n"
                  "<link rel=\"preconnect\" href=\"https://fonts.gstatic.com\" crossorigin>\n"
                  "<link href=\"https://fonts.googleapis.com/css2?family=Roboto&display=swap\" rel=\"stylesheet\">" << std::endl;
//We add the changes done to the file using the result vector
    for(const std::string& el : result){
//Write the changes done to the file
        outputFile << "<p>" << el << "</p>" << std::endl;
    }
    //Add the styles for each lexema - tried to add a separate
//CSS file but it was not working. Also the design is not very nice
//but I'm not very good at choosing colors
    outputFile << "<style>\n"
                  "    body {\n"
                  "        font-family: 'Roboto', sans-serif;\n"
                  "        background-color: antiquewhite;\n"
                  "        color: black;\n"
                  "    }\n"
                  "\n"
                  "    .identifier {\n"
                  "        color: blueviolet;\n"
                  "        font-weight: bold;\n"
                  "    }\n"
                  "\n"
                  "    .decimal {\n"
                  "        color: dimgrey;\n"
                  "    }\n"
                  "\n"
                  "    .exp {\n"
                  "        color: cadetblue;\n"
                  "    }\n"
                  "\n"
                  "    .parenthesis {\n"
                  "        color: brown;\n"
                  "    }\n"
                  "\n"
                  "    .reserved {\n"
                  "        color: chocolate;\n"
                  "    }\n"
                  "\n"
                  "    .comment {\n"
                  "        color: blue;\n"
                  "    }\n"
                  "    .specials {\n"
                  "        color: green;\n"
                  "    }\n"
                  "</style>" << std::endl;
    outputFile << "</body>\n" << "</html>\n" << std::endl;
    outputFile.close();
}


int main() {
    //Regex for lexemas
    std::regex decimalNumbersReg(R"(\b(?!.*?e|E)\d+(\.)*\d+)");
    std::regex expNumbersReg(R"(\d+\.\d+[e|E]\d)");
    std::regex identifierReg(R"((?!e|E)[a-zA-Z]+(\-|\w)*(?!\.\d))");
    std::regex parenthesisReg(R"([\(|\)\{|\}|\[|\]]+)");
    std::regex specialsReg(R"((?!\>)[\+\-\/\<\>\=\*]+(?!\S|\<+))");

    //We open the file
    std::fstream file("file.txt");
    std::string line;
    //Create vector to store changes
    std::vector<std::string> resultFile;  //Here all the modified lines
//will be stored and then will be written in a new file -
//this to prevent unwanted changes to the original file
```

```
//Here we read the file line by line and then call function to do the replacement
    if(file.is_open()){
    std::string changedLine;
        while(std::getline(file, line)){
          changedLine = line;
          std::vector<std::regex> regularExps = {identifierReg, specialsReg,
           decimalNumbersReg, expNumbersReg, parenthesisReg};
          for(int i=0; i < regularExps.size(); i++){
      // 0 - Identifier, 1- Specials,  2 - Decimal, 3 - expNum, 4 - Parenthesis  - index from regex vector
              changedLine = replaceMatchforHTML(changedLine, regularExps[i], i);
          }
          resultFile.push_back(changedLine);  //push the changes done to the line
      }
   file.close();
  }
  createHTMLfile(resultFile);
}
```

HTML resultante

```
<!DOCTYPE html>
<html>
<body>

<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap" rel="stylesheet">
<p><span class="parenthesis">(</span><span class="reserved">define</span> <span class="parenthesis">(</span><span class="identifier">cuadra
<p><span class="decimal">3123</span></p>
<p><span class="decimal">12.342</span></p>
<p><span class="exp">56.872e4</span></p>
<p>  <span class="parenthesis">(</span><span class="specials">*</span> <span class="identifier">x</span> <span class="identifier">x</span><
<p>#</p>
<p><span class="parenthesis">(</span><span class="reserved">define</span> <span class="parenthesis">(</span><span class="identifier">area-c
<p>  <span class="parenthesis">(</span><span class="specials">*</span> 3.1416 <span class="parenthesis">(</span><span class="identifier">cu
<p><span class="comment">; fin</span></p>
<p><span class="comment">; aqui se define si has entendido o no lo que hemos aprendido</span></p>
<style>
    body {
        font-family: 'Roboto', sans-serif;
        background-color: antiquewhite;
        color: black;
    }

    .identifier {
        color: blueviolet;
        font-weight: bold;
    }

    .decimal {
        color: dimgrey;
    }

    .exp {
        color: cadetblue;
    }

    .parenthesis {
        color: brown;
    }

    .reserved {
        color: chocolate;
    }

    .comment {
        color: blue;
    }
    .specials {
        color: green;
    }
</style>
</body>
</html>
```

Capturas pantalla



```
(define (cuadrado x)
3123
12.342
56.872e4
(* x x))
#
(define (area-circulo r)
(* 3.1416 (cuadrado r)))
; fin
; aqui se define si has entendido o no lo que hemos aprendido
```