Chuong Vu

COMP IV: Project Portfolio

Fall 2016

**Contents:**

**PS0** **Hello World with SFML**

**PS1** Recursive Graphic

**PS2** Linear Feedback Shift Register and Image Encoding

**PS3** N-Body Simulation

**0 PS4** **Edit Distance**

**PS5** Ring Buffer and Guitar Hero

**PS6** Markov Model of Natural Language

**PS7** Kronos **Intouch** Parsing

**PS0 Hello World with SFML**

This is the first assignment and the purpose was making me to get use to the SFML library for C++. Since I have completed the Computing 3 which learned about structure in C++ so this is not really hard to pick up the objects in SFML. I have spent few hours to read the SFML and the assignment requires so I have a good understand what I really need to do.

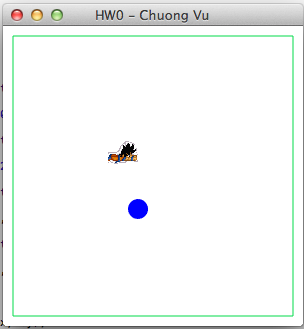
A part of the assignment was setup a Linux machine and install SFML library. Since I’m using MAC and I had an IT degree so it is not really difficult to build the SMFL environment in hour.

I have spent like two hours to read the SFML library and the example code that available on the sfml-dev website. So I tried to run few examples code before I can do the assignment.

This assignment is was a simple movement of SFML that using the Up, Down, Left, and Right key on keyboard to make the object moves in the SFML windows. To make this assignment more interesting, I had make a Blue Dot was random moving inside the wall and I used the Son-goku as a sprite object that chasing the Blue Dot.

Since this is just an assignment to let me get use to the SFML, so there was no advance algorithms or data structures for this assignment.

Output:



1. //-------------------------------
2. //Name: Chuong Vu
3. //Assigment 0, ps0
4. //Class: Computing IV
5. //Professor: Dr. Yelena Rykalova
6. //Due Date: 9/12/2016
7. //Filename: main.cpp
8. //-------------------------------
10. // This program is a simple movement of SFML. Use key Up, Down, Left, Right to move
11. // Son Goku flying sprite. Also, a Blue Dot will random moving inside the red wall.
12. // No object can pass through the wall.
14. #include <iostream>
15. #include <cstdlib>
16. #include <SFML/Graphics.hpp>
17. #include <SFML/Audio.hpp>
19. //Return random number from [-9, 9]
20. **int** getRandom() {
21. **return** std::rand() % 19 + (-9);
22. }
24. **int** main(**int** argc, **char** \*\* argv) {
25. **int** x, y, z, bx, by;
26. x = 0;
27. y = 0;
28. z = 0;
30. sf::ContextSettings settings;
31. settings.antialiasingLevel = 10;
33. sf::RenderWindow window(sf::VideoMode(300, 300),"HW0 - Chuong Vu", sf::Style::Default, settings);
34. window.setFramerateLimit(30);
36. //creta red box
37. sf::VertexArray lines(sf::LinesStrip, 5);
38. lines[0].position = sf::Vector2f(10, 10);
39. lines[0].color = sf::Color::Red;
40. lines[1].position = sf::Vector2f(10, 290);
41. lines[1].color = sf::Color::Red;
42. lines[2].position = sf::Vector2f(290, 290);
43. lines[2].color = sf::Color::Red;
44. lines[3].position = sf::Vector2f(290, 10);
45. lines[3].color = sf::Color::Red;
46. lines[4].position = sf::Vector2f(10, 10);
47. lines[4].color = sf::Color::Red;
49. // Declare and load a texture
50. sf::Texture texture;
51. **if**(!texture.loadFromFile("./sprite.png")) {
52. std::cout << "Error Loading Image!";
53. **return** -1;
54. }
55. texture.setSmooth(**true**);
57. // Create a sprite
58. sf::Sprite sprite;
59. sprite.setTexture(texture);
60. sprite.setTextureRect(sf::IntRect(325, 50, 20, 35));
61. sprite.setPosition(100,100);
63. // Create a blue dot
64. sf::CircleShape circle;
65. circle.setRadius(10);
66. circle.setPosition(30,30);
67. circle.setFillColor(sf::Color::Blue);

70. // It seem like the grader can't complie this music play code.
71. // I tried complie this with Xcode and it works fine
72. // Load a music to play
73. //    sf::Music music;
74. //    if (!music.openFromFile("./500.ogg")) {
75. //        return EXIT\_FAILURE;
76. //    }
77. //    music.setVolume(20);
78. //    music.play();
80. **bool** play = **true**;
81. **bool** ball = **true**;

84. **while** (window.isOpen()) {
85. // check all the window's events that were triggered since the last iteration of the loop
87. sf::Event event;
88. **while** (window.pollEvent(event)) {
89. // Close window: exit
90. **if** (event.type == sf::Event::Closed) {
91. window.close();
92. }
94. // Escape pressed: exit
95. **if** (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape) {
96. window.close();
97. }
99. }
101. **if**((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Up)) {
102. sprite.setTextureRect(sf::IntRect(325, 50, 20, 35));
103. //sprite.move(0, -5);
104. sprite.getPosition().y < 25 ? sprite.move(0, 0) : sprite.move(0, -5);
105. }
106. **if**((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Down)) {
107. sprite.setTextureRect(sf::IntRect(245, 40, 25, 35));
108. //sprite.move(0, 5);
109. sprite.getPosition().y > 225 ? sprite.move(0, 0) : sprite.move(0, 5);
110. }
111. **if**((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Left)) {
112. sprite.setTextureRect(sf::IntRect(265, 50, 45, 25));
113. sprite.setScale(1.0f,1.0f);
114. //sprite.move(-5, 0);
115. sprite.getPosition().x < 25 ? sprite.move(0, 0) : sprite.move(-5, 0);
116. }
117. **if**((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Right)) {
118. sprite.setTextureRect(sf::IntRect(265, 50, 45, 25));
119. sprite.setScale(-1.0f,1.0f);
120. // sprite.move(5, 0);
121. sprite.getPosition().x > 255 ? sprite.move(0, 0) : sprite.move(5, 0);
122. }
124. **if**(play) {
125. **for**( **int** j = 0; j < 50; j++) {
126. x++;
127. y += 2;
128. z += 3;
129. (x > 255) ? x = 0 : x;
130. (y > 255) ? x = 0 : y;
131. (z > 255) ? x = 0 : z;
132. lines[0].color = sf::Color(x , y , z);
133. lines[1].color = sf::Color(x , y , z);
134. lines[2].color = sf::Color(x , y , z);
135. lines[3].color = sf::Color(x , y , z);
136. lines[4].color = sf::Color(x , y , z);
137. }
138. }
140. //moving ball
141. **if**(ball) {
142. bx = getRandom();
143. by = getRandom();
145. **if**(circle.getPosition().x < 20) {
146. circle.move(20, 0);
147. }
148. **else** **if**(circle.getPosition().x > 250) {
149. circle.move(-20, 0);
150. }
151. **else** **if**(circle.getPosition().y < 20) {
152. circle.move(0, 20);
153. }
154. **else** **if**(circle.getPosition().y > 250) {
155. circle.move(0, -20);
156. }
157. **else**{
158. circle.move(bx, by);
159. }
160. }
162. // clear the window with black color
163. window.clear(sf::Color::White);
165. //window.draw(circle);
166. //window.draw(line, 5, sf::LinesStrip);
167. window.draw(sprite);
168. window.draw(lines);
169. window.draw(circle);
171. // end the current frame
172. window.display();
173. }
175. **return** EXIT\_SUCCESS;
176. }

**PS1 Recursive Graphic**

This assignment was to implementing the Sierpinski triangle that based on the described at Princeton website. This assignment requires using sf::Drawable from SFML to draw the Sierpinski triangle.

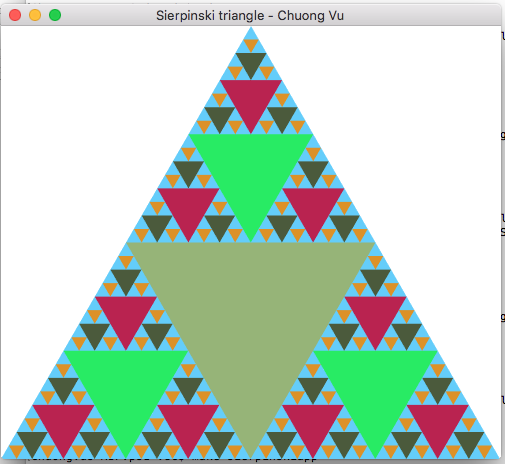
Based on the template that provided by Professor Yelena, I changed some code and wrote new functions that to draw Sierpinski’s Triangle recursive. In the homework I have two difference codes. One is the Sierpinski’s triangle that is required by the instructor and other is original code with is draw squares by my idea. In sierpinski’s triangle, I draw three more triangles on its side and it will keep doing that base on the depth (how many time I want to do recursive). Same with Sierpinski, my original code draw a squares and then other four squares on each its side.

In the code, I used virtual draw function for normal draw windows function; it will draw any shape based on depth and as many as how many times I want.

By using this recursive function, I save more memory since I only need one object for it recursive. Also, this assignment took me 3 days to finish. This assignment helped me understand more about objects and recursive that create another objects. Not really easy to understand the objects but at least it gave me a different view in C++.

1. #
2. #  Make file for ps1
3. #
5. CC = g++
6. OFLAGS = -c -Wall -ansi -pedantic -Werror
7. CFLAGS = -Wall -ansi -pedantic -Werror
8. LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system
10. all: ps1 ps1\_original
12. ps1: main.o sierpinski.o
13. $(CC) main.o sierpinski.o $(CFLAGS) $(LFLAGS) -o sierpinski
15. main.o: main.cpp
16. $(CC) $(OFLAGS) main.cpp
18. sierpinski.o: sierpinski.cpp
19. $(CC) $(OFLAGS) sierpinski.cpp
21. ps1\_original: original.o original\_h.o
22. $(CC) original.o original\_h.o $(CFLAGS) $(LFLAGS) -o original
24. original.o: original.cpp
25. $(CC) $(OFLAGS) original.cpp
27. original\_h.o: original\_h.cpp
28. $(CC) $(OFLAGS) original\_h.cpp
30. clean:
31. \rm -f \*.o  \*~ sierpinski original

Sierpinski Code Output:

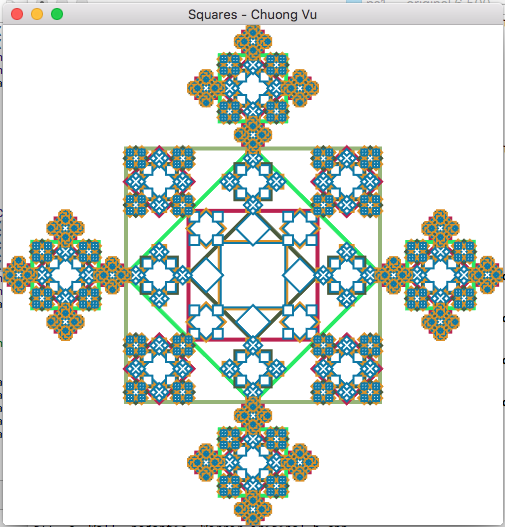


1. //-------------------------------
2. //Name: Chuong Vu
3. //Assigment 1, ps1 - Sierpinski's Triangle
4. //Class: Computing IV
5. //Professor: Dr. Yelena Rykalova
6. //Due Date: 9/18/2016
7. //Filename: main.cpp
8. //-------------------------------
10. // This program use recursive graphics to draw a Sierpinski's Triangle

13. #include <SFML/Graphics.hpp>
14. #include <iostream>
15. #include <cmath>
16. #include "sierpinski.hpp"
18. **using** **namespace** std;
20. **int** main(**int** argc, **char**\*argv[])
21. {
22. **if**(argc<3)
23. {
24. cout << "sierpinski [recursion-depth] [side-length]" << endl;
25. **return** -1;
26. }
28. //store the dide and how many time recurisve triangle call
29. **int** depth = atoi(argv[1]);
30. **int** side = atoi(argv[2]);
32. Sierpinski s(side, depth);
34. sf::ContextSettings settings;
35. settings.antialiasingLevel = 8;
36. sf::RenderWindow window(sf::VideoMode(side,(**int**)(.5\*sqrt(3.)\*(**float**)side)), "Sierpinski triangle - Chuong Vu", sf::Style::Default, settings);
37. window.setFramerateLimit(1);
39. **while**(window.isOpen())
40. {
41. sf::Event event;
42. **while**(window.pollEvent(event))
43. {
44. **if**(event.type == sf::Event::Closed)
45. window.close();
47. **if** (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape)
48. window.close();
50. }
51. window.clear(sf::Color::White);
52. window.draw(s);
53. window.display();
54. }
55. **return** 0;
56. }
57. //-------------------------------
58. //Name: Chuong Vu
59. //Assigment 1, ps1 - Sierpinski's Triangle
60. //Class: Computing IV
61. //Professor: Dr. Yelena Rykalova
62. //Due Date: 9/18/2016
63. //Filename: sierpiniski.hpp
64. //-------------------------------
66. // Header file
68. #ifndef SIERPINSKI\_HPP\_
69. #define SIERPINSKI\_HPP\_
71. #include <SFML/Graphics.hpp>
72. #include <SFML/Window.hpp>
73. #include <iostream>
75. **class** Sierpinski : **public** sf::Drawable {
76. **public**:
78. //top left right points and depth
79. Sierpinski(sf::Vector2f top, sf::Vector2f left, sf::Vector2f right, **int** depth);
81. //side length and depth
82. Sierpinski(**int** side, **int** depth);
84. //destructor
85. ~Sierpinski();
87. **private**:
88. **void** **virtual** draw(sf::RenderTarget& target, sf::RenderStates states) **const**;
90. sf::Vector2f \_top, \_left, \_right;
91. **int** \_depth;
93. Sierpinski\* \_child1;
94. Sierpinski\* \_child2;
95. Sierpinski\* \_child3;
96. };
98. #endif
99. //-------------------------------
100. //Name: Chuong Vu
101. //Assigment 1, ps1 - Sierpinski's Triangle
102. //Class: Computing IV
103. //Professor: Dr. Yelena Rykalova
104. //Due Date: 9/18/2016
105. //Filename: sierpinski.cpp
106. //-------------------------------
108. // This is file to write functions for header

111. #include "sierpinski.hpp"
112. #include <iostream>
113. #include <cmath>
115. //Calucalte mid-point
116. sf::Vector2f calmid(sf::Vector2f start, sf::Vector2f end) {
117. **return** sf::Vector2f((start.x + end.x) / 2, (start.y + end.y) /2);
118. }
120. //Generate sf::Color color
121. sf::Color randomcolor(**int** \_depth) {
122. **return** sf::Color((\_depth \* 110) % 255, (\_depth \* 200) % 255, (\_depth \* 20) % 255);
123. }
125. Sierpinski::Sierpinski(sf::Vector2f top, sf::Vector2f left, sf::Vector2f right, **int** depth) : \_top (top), \_left (left), \_right (right), \_depth (depth) {
126. **int** newdepth = \_depth - 1;
128. **if**(newdepth > 0) {
129. sf::Vector2f p1 = calmid(\_top, \_left);
130. sf::Vector2f p2 = calmid(\_right, \_left);
131. sf::Vector2f p3 = calmid(\_top, \_right);
133. //create 3 triangles that around it's own triangles if depth > 0  (recursive)
134. \_child1 = **new** Sierpinski(\_top, p1, p3, newdepth);
135. \_child2 = **new** Sierpinski(p1, \_left, p2, newdepth);
136. \_child3 = **new** Sierpinski(p3, p2, \_right, newdepth);
137. } **else** {
138. \_child1 = NULL;  //point to null
139. \_child2 = NULL;  //point to null
140. \_child3 = NULL;  //point to null
141. }
142. }
144. //calculate vector of triangle
145. Sierpinski::Sierpinski(**int** side, **int** depth) : \_top (sf::Vector2f(side / 2, 0)), \_depth (depth) {
146. \_top = sf::Vector2f(side / 2, 0);
148. **float** height = 0.5 \* sqrt(3.) \* (**float**)side;
150. \_left = sf::Vector2f(0, height);
151. \_right = sf::Vector2f(side - 1, height);
153. //the 3 childs in the triangles
155. **int** newdepth = \_depth - 1;
157. **if**(newdepth > 0) {
159. sf::Vector2f p1 = calmid(\_top, \_left);
160. sf::Vector2f p2 = calmid(\_right, \_left);
161. sf::Vector2f p3 = calmid(\_top, \_right);
163. //create 3 triangles that around it's own triangles if depth > 0
164. \_child1 = **new** Sierpinski(\_top, p1, p3, newdepth);
165. \_child2 = **new** Sierpinski(p1, \_left, p2, newdepth);
166. \_child3 = **new** Sierpinski(p3, p2, \_right, newdepth);
167. } **else** {
168. \_child1 = NULL;  //point to null
169. \_child2 = NULL;  //point to null
170. \_child3 = NULL;  //point to null
171. }
172. }
174. Sierpinski::~Sierpinski() {
175. //std::cout << "Delete Destructor" << std::endl;
176. **delete**(\_child1);
177. **delete**(\_child2);
178. **delete**(\_child3);
179. }
181. **void** Sierpinski::draw(sf::RenderTarget& target, sf::RenderStates states) **const** {
183. **if**(\_depth == 1) {
184. sf::ConvexShape triangle;
185. triangle.setPointCount(3);
186. triangle.setPoint(0, \_top);
187. triangle.setPoint(1, \_left);
188. triangle.setPoint(2, \_right);
189. triangle.setFillColor(randomcolor(\_depth+190));
190. target.draw(triangle, states);
191. } **else** {
192. sf::Vector2f p1 = calmid(\_top, \_left);
193. sf::Vector2f p2 = calmid(\_right, \_left);
194. sf::Vector2f p3 = calmid(\_top, \_right);
196. sf::ConvexShape triangle;
197. triangle.setPointCount(3);
198. triangle.setPoint(0, p1);
199. triangle.setPoint(1, p2);
200. triangle.setPoint(2, p3);
201. triangle.setFillColor(randomcolor(\_depth));
202. target.draw(triangle, states);
203. }
205. //draw triangles
206. **if**(\_child1 != NULL) {
207. \_child1->draw(target, states);
208. \_child2->draw(target, states);
209. \_child3->draw(target, states);
210. }
211. }

Original Output



1. //-------------------------------
2. //Name: Chuong Vu
3. //Assigment 1, ps1 - Draw Squres
4. //Class: Computing IV
5. //Professor: Dr. Yelena Rykalova
6. //Due Date: 9/18/2016
7. //Filename: original.cpp
8. //-------------------------------
10. // Simmlar to Sierpinski's Triangle, this draw square
12. #include <SFML/Graphics.hpp>
13. #include <iostream>
14. #include <cmath>
15. #include "original.hpp"
17. **using** **namespace** std;
19. **int** main(**int** argc, **char**\*argv[]) {
20. **if**(argc<3){
21. cout << "sierpinski [recursion-depth] [side-length]" << endl;
22. **return** -1;
23. }
25. **int** depth = atoi(argv[1]);
26. **int** side = atoi(argv[2]);
28. Original s(side, depth);
30. sf::ContextSettings settings;
31. settings.antialiasingLevel = 8;
32. sf::RenderWindow window(sf::VideoMode(side, side), "Squares - Chuong Vu", sf::Style::Default, settings);
33. window.setFramerateLimit(1);
35. **while**(window.isOpen()) {
36. sf::Event event;
37. **while**(window.pollEvent(event)) {
38. **if**(event.type == sf::Event::Closed)
39. window.close();
41. **if** (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape)
42. window.close();
44. }
45. window.clear(sf::Color::White);
46. window.draw(s);
47. window.display();
48. }
49. **return** 0;
50. }
51. //-------------------------------
52. //Name: Chuong Vu
53. //Assigment 1, ps1 - Draw Squres
54. //Class: Computing IV
55. //Professor: Dr. Yelena Rykalova
56. //Due Date: 9/18/2016
57. //Filename: original.hpp
58. //-------------------------------
59. // Header files
61. #ifndef ORIGINAL\_HPP\_
62. #define ORIGINAL\_HPP\_
64. #include <SFML/Graphics.hpp>
65. #include <SFML/Window.hpp>
66. #include <iostream>
68. **class** Original : **public** sf::Drawable {
69. **public**:
71. //top left right points and depth
72. Original(sf::Vector2f point1, sf::Vector2f point2, sf::Vector2f point3, sf::Vector2f point4, **int** depth);
74. //side length and depth
75. Original(**int** side, **int** depth);
77. //destructor
78. ~Original();
80. **private**:
81. **void** **virtual** draw(sf::RenderTarget& target, sf::RenderStates states) **const**;
83. sf::Vector2f \_point1, \_point2, \_point3, \_point4;
84. **int** \_depth;
86. Original\* \_child1;
87. Original\* \_child2;
88. Original\* \_child3;
89. Original\* \_child4;
90. Original\* \_child5;
91. };
93. #endif
94. //-------------------------------
95. //Name: Chuong Vu
96. //Assigment 1, ps1 - Draw Squres
97. //Class: Computing IV
98. //Professor: Dr. Yelena Rykalova
99. //Due Date: 9/18/2016
100. //Filename: original\_h.cpp
101. //-------------------------------
102. // Functions for header
104. #include "original.hpp"
105. #include <iostream>
106. #include <cmath>
108. //Calucalte mid-point
109. sf::Vector2f calmid(sf::Vector2f start, sf::Vector2f end) {
110. **return** sf::Vector2f((start.x + end.x) / 2, (start.y + end.y) /2);
111. }
113. //Generate sf::Color color
114. sf::Color randomcolor(**int** \_depth) {
115. **return** sf::Color((\_depth \* 110) % 255, (\_depth \* 200) % 255, (\_depth \* 20) % 255);
116. }
118. Original::Original(sf::Vector2f point1, sf::Vector2f point2, sf::Vector2f point3, sf::Vector2f point4, **int** depth): \_point1 (point1), \_point2 (point2), \_point3 (point3), \_point4 (point4), \_depth (depth) {
119. **int** newdepth = \_depth - 1;
120. **if** (newdepth > 0){
121. sf::Vector2f p1 = calmid(\_point1, \_point2);
122. sf::Vector2f p2 = calmid(\_point2, \_point3);
123. sf::Vector2f p3 = calmid(\_point3, \_point4);
124. sf::Vector2f p4 = calmid(\_point4, \_point1);
126. //draw 4 squares around it's own square if depth > 0 (recursive)
127. \_child1 = **new** Original(p1, p2, p3, p4, newdepth);
128. \_child2 = **new** Original(calmid(\_point1, p1), calmid(p1, p4), calmid(p4, \_point1), \_point1, newdepth);
129. \_child3 = **new** Original(calmid(p1, \_point2), \_point2, calmid(\_point2, p2), calmid(p1, p2), newdepth);
130. \_child4 = **new** Original(calmid(p2, \_point3), \_point3, calmid(\_point3, p3), calmid(p2, p3), newdepth);
131. \_child5 = **new** Original(calmid(p3, \_point4), \_point4, calmid(\_point4, p4), calmid(p3, p4), newdepth);
132. } **else** {
133. \_child1 = NULL;  //point to null
134. \_child2 = NULL;
135. \_child3 = NULL;
136. \_child4 = NULL;
137. \_child5 = NULL;
138. }
140. }
142. Original::Original(**int** side, **int** depth) : \_depth (depth) {
143. \_point1 = sf::Vector2f(side / 2, 1);
144. \_point2 = sf::Vector2f(0, side / 2);
145. \_point3 = sf::Vector2f(side / 2, side);
146. \_point4 = sf::Vector2f(side, side / 2);
148. **int** newdepth = \_depth - 1;
149. **if** (newdepth > 0){
150. sf::Vector2f p1 = calmid(\_point1, \_point2);
151. sf::Vector2f p2 = calmid(\_point2, \_point3);
152. sf::Vector2f p3 = calmid(\_point3, \_point4);
153. sf::Vector2f p4 = calmid(\_point4, \_point1);
155. //draw 4 squares around it's own square
156. \_child1 = **new** Original(p1, p2, p3, p4, newdepth);
157. \_child2 = **new** Original(calmid(\_point1, p1), calmid(p1, p4), calmid(p4, \_point1), \_point1, newdepth);
158. \_child3 = **new** Original(calmid(p1, \_point2), \_point2, calmid(\_point2, p2), calmid(p1, p2), newdepth);
159. \_child4 = **new** Original(calmid(p2, \_point3), \_point3, calmid(\_point3, p3), calmid(p2, p3), newdepth);
160. \_child5 = **new** Original(calmid(p3, \_point4), \_point4, calmid(\_point4, p4), calmid(p3, p4), newdepth);
161. } **else** {
162. \_child1 = NULL;  //point to null
163. \_child2 = NULL;
164. \_child3 = NULL;
165. \_child4 = NULL;
166. \_child5 = NULL;
167. }
169. }
171. Original::~Original() {
172. //std::cout <<"Delete destructor" << std::endl;
173. **delete**(\_child1);
174. **delete**(\_child2);
175. **delete**(\_child3);
176. **delete**(\_child4);
177. **delete**(\_child5);
178. }
180. **void** Original::draw(sf::RenderTarget& target, sf::RenderStates states) **const** {
181. **if**(\_depth == 1) {
182. sf::ConvexShape square;
183. square.setPointCount(4);
184. square.setPoint(0, \_point1);
185. square.setPoint(1, \_point2);
186. square.setPoint(2, \_point3);
187. square.setPoint(3, \_point4);
188. square.setOutlineColor(randomcolor(\_depth + 20));
189. square.setOutlineThickness(2.0);
190. target.draw(square, states);
191. } **else** {
192. sf::Vector2f p1 = calmid(\_point1, \_point2);
193. sf::Vector2f p2 = calmid(\_point2, \_point3);
194. sf::Vector2f p3 = calmid(\_point3, \_point4);
195. sf::Vector2f p4 = calmid(\_point4, \_point1);
197. sf::ConvexShape square;
198. square.setPointCount(4);
199. square.setPoint(0, p1);
200. square.setPoint(1, p2);
201. square.setPoint(2, p3);
202. square.setPoint(3, p4);
203. square.setOutlineColor(randomcolor(\_depth));
204. square.setOutlineThickness(4.0);
205. target.draw(square, states);
206. }
208. //draw from function
209. **if**(\_child1 != NULL) {
210. \_child1->draw(target, states);
211. \_child2->draw(target, states);
212. \_child3->draw(target, states);
213. \_child4->draw(target, states);
214. \_child5->draw(target, states);
215. }
216. }

**PS2 Linear Feedback Shift Register and Image Encoding**

This assignment was about completing the linear feedback shift register. This assignment divided into two parts.

*Part PS2a.*

Based on the header LFSR.hpp and test.cpp that provided by Professor Yelena, my LFSR simply using an string input that represent as a binary string to \_seed. Also, by following the instruction that from the link provide by Professor Yelena (http://www.cs.princeton.edu/courses/archive/fall13/cos126/assignments/lfsr.html)

I know that I have to get the value of tap position and the last position from string using XOR to get the new bit. And then I shift the current string bit to left by 1 and put the new bit to the end. So I will get a new bit string as a linear feedback shift register supposed to be.

Overall, the hardest think for this assignment is how to figure out how the generate function work. To calculate the return number from generate function, first I call the function step() as how many as I want to shift. And then I check if the step() function return 1 or 0, if it return 1, I will calculate the sum of step() that return 1 so I will get the value that generate() function return.

Test Case

1) I use the test case provide by the professor is 00111 with tap = 2 and shift 8 time. So the generate(8) return 198 which is no error.

2^7 + 2^6 + 2^2 + 2^1 = 198

2) I use another two test cases to make sure that my method is correct;

a) 010111, tap = 3 and shift 4 times. The result is 2^1 = 2. True

b) 1101010010, tap = 6 and shift 6 times. The result is 2^4 + 2^3 + 2^2 = 28. True

c) 000, tap = 1. This is short seed strings and I except step return is 0 and generate = 0. True

Output for part a:

1. #
2. #  Make file for ps2a
3. #
5. CC = g++
6. CFLAGS = -lboost\_unit\_test\_framework -Wall -ansi -pedantic -Werror
7. OFLAGS =  -ansi -pedantic -Wall -Werror
9. all: ps2a
11. ps2a: test.o LFSR.o
12. $(CC) -o ps2a test.o LFSR.o $(CFLAGS)
14. test.o: test.cpp
15. $(CC) -c test.cpp $(OFLAGS)
17. LFSR.o: LFSR.hpp LFSR.cpp
18. $(CC) -c LFSR.cpp $(OFLAGS)
20. clean:
21. rm -f \*.o \*~ ps2a
22. //test.cpp
23. #include <iostream>
24. #include <string>
26. #include "LFSR.hpp"
28. #define BOOST\_TEST\_DYN\_LINK
29. #define BOOST\_TEST\_MODULE Main
30. #include <boost/test/unit\_test.hpp>
32. BOOST\_AUTO\_TEST\_CASE(fiveBitsTapAtTwo) {
34. LFSR l("00111", 2);
35. BOOST\_REQUIRE(l.step() == 1);
36. BOOST\_REQUIRE(l.step() == 1);
37. BOOST\_REQUIRE(l.step() == 0);
38. BOOST\_REQUIRE(l.step() == 0);
39. BOOST\_REQUIRE(l.step() == 0);
40. BOOST\_REQUIRE(l.step() == 1);
41. BOOST\_REQUIRE(l.step() == 1);
42. BOOST\_REQUIRE(l.step() == 0);
44. LFSR l2("00111", 2);
45. BOOST\_REQUIRE(l2.generate(8) == 198);
46. }
48. /\*
49. six bit tap a three
50. 010111  0
51. 101110  1
52. 011100  2
53. 111001  3
54. 110010  4
55. \*/
56. BOOST\_AUTO\_TEST\_CASE(sixBitsTapAtThree) {
58. LFSR l3("010111", 3);
59. BOOST\_REQUIRE(l3.step() == 0);
60. BOOST\_REQUIRE(l3.step() == 0);
61. BOOST\_REQUIRE(l3.step() == 1);
62. BOOST\_REQUIRE(l3.step() == 0);
64. LFSR l4("010111", 3);
65. BOOST\_REQUIRE(l4.generate(4) == 2);
66. }

69. /\*
70. Ten bits tap at six
71. 1101010010  0
72. 1010100100  1
73. 0101001001  2
74. 1010010011  3
75. 0100100111  4
76. 1001001110  5
77. 0010011100  6
78. \*/
79. BOOST\_AUTO\_TEST\_CASE(tenBitsTapAtSix){
81. LFSR l5("1101010010", 6);
82. BOOST\_REQUIRE(l5.step() == 0);
83. BOOST\_REQUIRE(l5.step() == 1);
84. BOOST\_REQUIRE(l5.step() == 1);
85. BOOST\_REQUIRE(l5.step() == 1);
86. BOOST\_REQUIRE(l5.step() == 0);
87. BOOST\_REQUIRE(l5.step() == 0);
89. LFSR l6("1101010010", 6);
90. BOOST\_REQUIRE(l5.generate(4) == 1);
91. }

94. BOOST\_AUTO\_TEST\_CASE(threeBitsTapAtOne){
96. LFSR l5("000", 1);
97. BOOST\_REQUIRE(l5.step() == 0);
98. BOOST\_REQUIRE(l5.step() == 0);
99. BOOST\_REQUIRE(l5.step() == 0);
101. LFSR l6("000", 1);
102. BOOST\_REQUIRE(l5.generate(4) == 0);
103. }
104. //-------------------------------
105. //Name: Chuong Vu
106. //Assigment 1, ps2 - Encoding images with LFSR
107. //Class: Computing IV
108. //Professor: Dr. Yelena Rykalova
109. //Filename: main.cpp
110. //-------------------------------
112. #include <iostream>
113. #include <string>
115. #include "LFSR.hpp"
117. #define BOOST\_TEST\_DYN\_LINK
118. #define BOOST\_TEST\_MODULE Main
119. #include <boost/test/unit\_test.hpp>
121. // Starter-case
122. BOOST\_AUTO\_TEST\_CASE(fiveBitsTapAtTwo) {
124. LFSR l("00111", 2);
125. BOOST\_REQUIRE(l.step() == 1);
126. BOOST\_REQUIRE(l.step() == 1);
127. BOOST\_REQUIRE(l.step() == 0);
128. BOOST\_REQUIRE(l.step() == 0);
129. BOOST\_REQUIRE(l.step() == 0);
130. BOOST\_REQUIRE(l.step() == 1);
131. BOOST\_REQUIRE(l.step() == 1);
132. BOOST\_REQUIRE(l.step() == 0);
134. LFSR l2("00111", 2);
135. BOOST\_REQUIRE(l2.generate(8) == 198);
136. }
138. // Testing arbitrary size and tap, all zeros.
139. BOOST\_AUTO\_TEST\_CASE(twentyBitsAllZeros) {
141. LFSR l("00000000000000000000", 2);
142. BOOST\_REQUIRE(l.step() == 0);
143. BOOST\_REQUIRE(l.step() == 0);
144. BOOST\_REQUIRE(l.step() == 0);
145. BOOST\_REQUIRE(l.step() == 0);
146. BOOST\_REQUIRE(l.step() == 0);
147. BOOST\_REQUIRE(l.step() == 0);
148. BOOST\_REQUIRE(l.step() == 0);
149. BOOST\_REQUIRE(l.step() == 0);
150. BOOST\_REQUIRE(l.step() == 0);
151. BOOST\_REQUIRE(l.step() == 0);
152. BOOST\_REQUIRE(l.step() == 0);
154. LFSR l2("00000000000000000000", 2);
155. BOOST\_REQUIRE(l2.generate(9) == 0);
156. }
158. // Testing to see if tap at zero works properly and in the right order
159. BOOST\_AUTO\_TEST\_CASE(fiveBitsTapAtZero) {
160. LFSR l("11001", 0);
161. BOOST\_REQUIRE(l.step() == 0);
162. BOOST\_REQUIRE(l.step() == 1);
163. BOOST\_REQUIRE(l.step() == 1);
164. BOOST\_REQUIRE(l.step() == 1);
165. BOOST\_REQUIRE(l.step() == 0);
166. BOOST\_REQUIRE(l.step() == 0);
168. LFSR l2("11001", 0);
169. BOOST\_REQUIRE(l2.generate(6) == 28);
170. }
172. // Arbitrary size, testing to see if tap at the first element in the string
173. // (the most significant bit) works properly.
174. BOOST\_AUTO\_TEST\_CASE(thirtyBitsTapAt29) {
175. LFSR l("110011001100110011001100110011", 29);
176. BOOST\_REQUIRE(l.step() == 0);
177. BOOST\_REQUIRE(l.step() == 0);
178. BOOST\_REQUIRE(l.step() == 0);
179. BOOST\_REQUIRE(l.step() == 0);
180. BOOST\_REQUIRE(l.step() == 0);
182. LFSR l2("110011001100110011001100110011", 29);
183. BOOST\_REQUIRE(l2.generate(29) == 0);
184. }
185. /\*
186. \* LFSR.hpp
187. \*
188. \*  Created on: Sun Mar  2 09:27:35 2014
189. \*      Author: Fred Martin, fredm@cs.uml.edu
190. \*  ed: Y. Rykalova fall 2016
191. \*/
192. #ifndef \_LFSR\_H
193. #define \_LFSR\_H
194. #include <iostream>
195. #include <string>

198. **class** LFSR {
200. **public**:
201. LFSR(std::string seed, **int** tap);
202. ~LFSR();
203. **int** step();
204. **int** generate(**int** k);
205. **friend** std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
207. **private**:
208. std::string \_seed;        //string data
209. **int** \_tap;                 //index location
210. };
212. #endif
213. //-------------------------------
214. //Name: Chuong Vu
215. //Assigment 1, ps2a - Linear Feedback Shift Register and Unit Testing
216. //Class: Computing IV
217. //Professor: Dr. Yelena Rykalova
218. //Filename: LFSR.cpp
219. //-------------------------------

222. #include <cstdlib>
223. #include <sstream>    // std::ostringstream
224. #include <cmath>
225. #include "LFSR.hpp"

228. //default contructor of the class
229. LFSR::LFSR(std::string seed, **int** tap) : \_seed(seed), \_tap(tap){ }
231. LFSR::~LFSR(){}
233. //get return value, either 0 or 1
234. **int** LFSR::step() {
235. std::ostringstream \_str;
237. //Shiff the bit to left by 1
238. unsigned i = 1;
239. **while**(i < \_seed.length())
240. {
241. \_str << \_seed[i];
242. i++;
243. }
245. //XOR for the last bit after shift
246. **int** bit = (\_seed[0] - 48) ^ (\_seed[\_seed.length() - (\_tap + 1)] - 48);
248. //Add last bit to shifted bits
249. \_str << bit;
251. //Store back shifted bits to \_seed
252. \_seed = \_str.str();
254. //Return XOR bit, either 0 or 1
255. **return** bit;
256. }
258. //get return value, either 0 or 1
259. //convert binary to decimal
260. **int** LFSR::generate(**int** k)
261. {
262. **int** num = 0;
263. **int** i = k - 1;
264. **while**(i >= 0)
265. {
266. **if**(step() == 1)
267. num += pow(2, i);
268. i--;
269. }
270. //std::cout << num << std::endl;;
271. **return** num;
272. }
274. //print the output
275. std::ostream& operator<< (std::ostream &out, LFSR &lfsr)
276. {
277. out << lfsr.\_seed;
278. **return** out;
279. }

*Part PS2b.*

Based on the ps2a, I wrote a PhotoMagic.cpp for encode and decode image. Using PNG image for encode because it is 8-bits so it can be encode. The program using the same LFSR files from ps2a but I add more function to get picture encode. I note that the generate function from LFSR will determine the running time for encoding. The bigger bit shift, the slower the encode time run. So I set it to 9 is just enough to encode in a short.

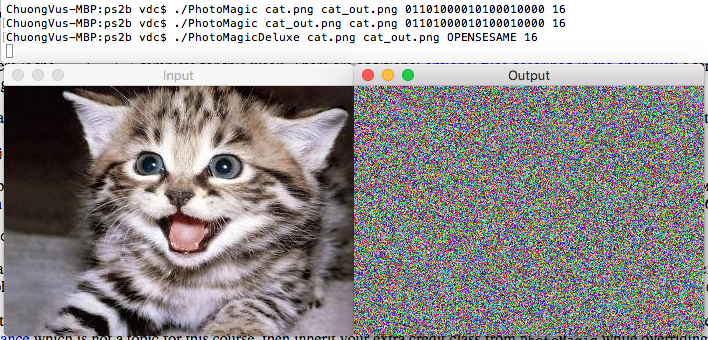
For extra credit:

I note that every single character from a string is 6 bits string. So I take every single character from alphanumeric password and compare to the base64 if it is in there or not. If yes then I take that character and using bit set library to convert it to 6 bits binary then I join them together to get the final binary bits string for encode.

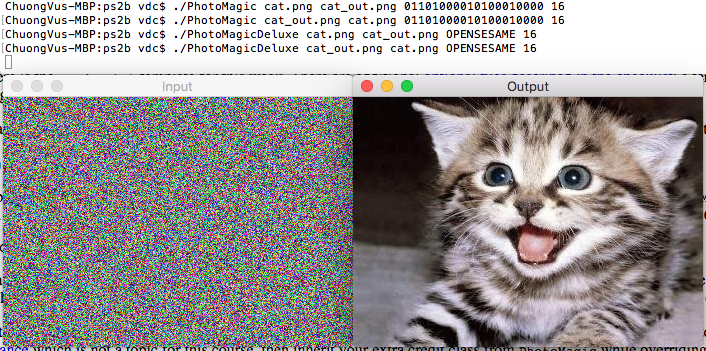
After this assignment, I had known more about bit shift in digital which I have learned from the Logic Design class and how to apply the bit-shift to daily life. This can be use full if I want to share encoded pictures to my friend on the internet and only people who know the key can decode the pictures.

1. #
2. #  Make file for ps2b
3. #
4. CC = g++
5. CFLAGS = -c -Wall -pedantic -Werror
6. OFLAGS = -Wall -pedantic -Werror
7. LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system
9. all: PhotoMagic PhotoMagicDeluxe
11. PhotoMagic: PhotoMagic.o LFSR.o
12. $(CC) -o PhotoMagic PhotoMagic.o LFSR.o $(LFLAGS) $(OFLAGS)
14. PhotoMagic.o: PhotoMagic.cpp
15. $(CC) $(CFLAGS) PhotoMagic.cpp
17. PhotoMagicDeluxe: PhotoMagicDeluxe.o LFSR.o
18. $(CC) -o PhotoMagicDeluxe PhotoMagicDeluxe.o LFSR.o $(LFLAGS) $(OFLAGS)
20. PhotoMagicDeluxe.o: PhotoMagicDeluxe.cpp
21. $(CC) $(CFLAGS) PhotoMagicDeluxe.cpp
23. LFSR.o: LFSR.hpp LFSR.cpp
24. $(CC) $(CFLAGS) LFSR.cpp
26. clean:
27. rm -f \*.o \*~ PhotoMagic PhotoMagicDeluxe

Encode output:



Decode Output:



1. //
2. //  PhotoMagic.cpp
3. //  ps2b
4. //
5. //  Created by ChuongVu on 9/28/16.
6. //  Copyright © 2016 ChuongVu. All rights reserved.
7. //  Using Xcode
8. //
9. #include <iostream>
10. #include <cstring>
11. #include <cstdlib>
12. #include <bitset>
13. #include "LFSR.hpp"
14. #include <SFML/System.hpp>
15. #include <SFML/Window.hpp>
16. #include <SFML/Graphics.hpp>

19. **int** main(**int** argc, **char**\* argv[])
20. {
21. **int** pos = 200;
23. **if**(argc < 5) {
24. std::cout << "Not enough command-line arguments." << std::endl;
25. **return** -1;
26. }
27. **if**(argc > 5) {
28. std::cout << "Too many command-line arguments." << std::endl;
29. **return** -1;
30. }

33. // Initialize LFSR with command-line arguments
34. LFSR lfsr(std::string(argv[3]), atoi(argv[4]));
36. // Initialize images
37. sf::Image image1;
38. sf::Image image2;

41. // Load inputted image
42. **if**(!image1.loadFromFile(argv[1]))
43. **return** -1;
45. //output image
46. **if**(!image2.loadFromFile(argv[1]))
47. **return** -1;
49. // p is a pixel
50. sf::Color p;
52. sf::Vector2u size = image2.getSize();
54. // create photographic negative image of upper-left 200 px square
55. **for** (unsigned **int** x = 0; x < size.x; x++) {
56. **for** (unsigned **int** y = 0; y < size.y; y++) {
57. p = image2.getPixel(x, y);
58. p.r = (p.r ^ lfsr.generate(8));
59. p.g = (p.g ^ lfsr.generate(8));
60. p.b = (p.b ^ lfsr.generate(8));
61. image2.setPixel(x, y, p);
62. }
63. }
65. sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
66. sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output");
68. sf::Texture texture1;   //input
69. sf::Texture texture2;   //ouput
70. texture1.loadFromImage(image1);
71. texture2.loadFromImage(image2);
73. sf::Sprite sprite1;     //input
74. sf::Sprite sprite2;     //output
75. sprite1.setTexture(texture1);
76. sprite2.setTexture(texture2);
78. **while** (window1.isOpen() && window2.isOpen())
79. {
80. sf::Event event;
81. **while** (window1.pollEvent(event))
82. {
83. **if** (event.type == sf::Event::Closed)
84. window1.close();
85. }
86. **while** (window2.pollEvent(event))
87. {
88. **if** (event.type == sf::Event::Closed)
89. window2.close();
90. }
92. window1.clear(sf::Color::White);
93. window1.setPosition(sf::Vector2i(pos,size.y));
94. window1.draw(sprite1);
95. window1.display();
97. window2.clear(sf::Color::White);
98. window2.setPosition(sf::Vector2i(pos + size.x,size.y));
99. window2.draw(sprite2);
100. window2.display();
101. }
103. // fredm: saving a PNG segfaults for me, though it does properly
104. //   write the file
105. **if**(!image2.saveToFile(argv[2]))
106. **return** -1;
108. **return** 0;
109. }
110. //
111. //  PhotoMagicDeluxe.cpp
112. //  ps2b
113. //
114. //  Created by ChuongVu on 9/28/16.
115. //  Copyright © 2016 ChuongVu. All rights reserved.
116. //  Using Xcode
117. //
118. #include <iostream>
119. #include <cstring>
120. #include <cstdlib>
121. #include <bitset>
122. #include "LFSR.hpp"
123. #include <SFML/System.hpp>
124. #include <SFML/Window.hpp>
125. #include <SFML/Graphics.hpp>
127. //Use to convert string to Binary 6N-bit string
128. std::string string\_convert(std::string str)
129. {
130. std::string temp;
131. std::string base64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
132. **for**(**int** i = 0; i < str.size(); i++)
133. {
134. **char** c =  str.at(i);
135. std::**size\_t** check = base64.find(c);
136. **if**(check==std::string::npos)
137. {
138. std::cout << "Cant find char" << c << "in base64";
139. **return** 0;
140. }
141. std::bitset<6> b(check);
142. //std::cout << b
143. temp += b.to\_string();
144. }
145. **return** temp;
146. }

149. **int** main(**int** argc, **char**\* argv[])
150. {
151. **int** pos = 200;
153. **if**(argc < 5) {
154. std::cout << "Not enough command-line arguments." << std::endl;
155. **return** -1;
156. }
157. **if**(argc > 5) {
158. std::cout << "Too many command-line arguments." << std::endl;
159. **return** -1;
160. }

163. // Initialize LFSR with command-line arguments
164. LFSR lfsr(std::string(string\_convert(argv[3])), atoi(argv[4]));
166. // Initialize images
167. sf::Image image1;
168. sf::Image image2;
170. // Load inputted image
171. **if**(!image1.loadFromFile(argv[1]))
172. **return** -1;
174. //output image
175. **if**(!image2.loadFromFile(argv[1]))
176. **return** -1;
178. // p is a pixel
179. sf::Color p;
181. sf::Vector2u size = image2.getSize();
183. // create photographic negative image of upper-left 200 px square
184. //the biger of generate number, the slower of encoding picture is so I set it to 8
185. **for** (unsigned **int** x = 0; x < size.x; x++) {
186. **for** (unsigned **int** y = 0; y < size.y; y++) {
187. p = image2.getPixel(x, y);
188. p.r = (p.r ^ lfsr.generate(8));
189. p.g = (p.g ^ lfsr.generate(8));
190. p.b = (p.b ^ lfsr.generate(8));
191. image2.setPixel(x, y, p);
192. }
193. }
195. sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
196. sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output");
198. sf::Texture texture1;   //input
199. sf::Texture texture2;   //ouput
200. texture1.loadFromImage(image1);
201. texture2.loadFromImage(image2);
203. sf::Sprite sprite1;     //input
204. sf::Sprite sprite2;     //output
205. sprite1.setTexture(texture1);
206. sprite2.setTexture(texture2);
208. **while** (window1.isOpen() && window2.isOpen())
209. {
210. sf::Event event;
211. **while** (window1.pollEvent(event))
212. {
213. **if** (event.type == sf::Event::Closed)
214. window1.close();
215. }
216. **while** (window2.pollEvent(event))
217. {
218. **if** (event.type == sf::Event::Closed)
219. window2.close();
220. }
222. window1.clear(sf::Color::White);
223. window1.setPosition(sf::Vector2i(pos,size.y));
224. window1.draw(sprite1);
225. window1.display();
227. window2.clear(sf::Color::White);
228. window2.setPosition(sf::Vector2i(pos + size.x,size.y));
229. window2.draw(sprite2);
230. window2.display();
231. }
233. // fredm: saving a PNG segfaults for me, though it does properly
234. //   write the file
235. **if**(!image2.saveToFile(argv[2]))
236. **return** -1;
238. **return** 0;
239. }
240. /\*
241. \* LFSR.hpp
242. \*
243. \*  Created on: Sun Mar  2 09:27:35 2014
244. \*  Author: Fred Martin, fredm@cs.uml.edu
245. \*  ed: Y. Rykalova fall 2016
246. \*/
247. #ifndef \_LFSR\_H
248. #define \_LFSR\_H
249. #include <iostream>
250. #include <string>
252. **class** LFSR {
253. **public**:
254. LFSR(std::string seed, **int** tap);
255. ~LFSR();
256. **int** step();
257. **int** generate(**int** k);
258. **friend** std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
260. **private**:
261. std::string \_seed;      //string data
262. **int** \_tap;               //index location
263. };
265. #endif
266. //-------------------------------
267. //Name: Chuong Vu
268. //Assigment 1, ps2b - Encoding images with LFSR
269. //Class: Computing IV
270. //Professor: Dr. Yelena Rykalova
271. //Filename: LFSR.cpp
272. //-------------------------------
274. #include <cstdlib>
275. #include <sstream>    // std::ostringstream
276. #include <cmath>
277. #include "LFSR.hpp"

280. //default contructor of the class
281. LFSR::LFSR(std::string seed, **int** tap) : \_seed(seed), \_tap(tap)
282. {
283. }
285. LFSR::~LFSR()
286. {
287. }
289. //get return value, either 0 or 1
290. **int** LFSR::step()
291. {
292. std::ostringstream \_str;
294. //Shiff the bit to left by 1
295. unsigned i = 1;
296. **while**(i < \_seed.length())
297. {
298. \_str << \_seed[i];
299. i++;
300. }
302. //XOR for the last bit after shift
303. **int** bit = (\_seed[0] - 48) ^ (\_seed[\_seed.length() - (\_tap + 1)] - 48);
305. //Add last bit to shifted bits
306. \_str << bit;
308. //Store back shifted bits to \_seed
309. \_seed = \_str.str();
311. //Return XOR bit, either 0 or 1
312. **return** bit;
313. }
315. //get return value, either 0 or 1
316. //convert binary to decimal
317. **int** LFSR::generate(**int** k)
318. {
319. **int** num = 0;
320. **int** i = k - 1;
321. **while**(i >= 0)
322. {
323. **if**(step() == 1)
324. num += pow(2, i);
325. i--;
326. }
327. //std::cout << num << std::endl;;
328. **return** num;
329. }
331. //print the output
332. std::ostream& operator<< (std::ostream &out, LFSR &lfsr)
333. {
334. out << lfsr.\_seed;
335. **return** out;
336. }

**PS3 N-Body Simulation**

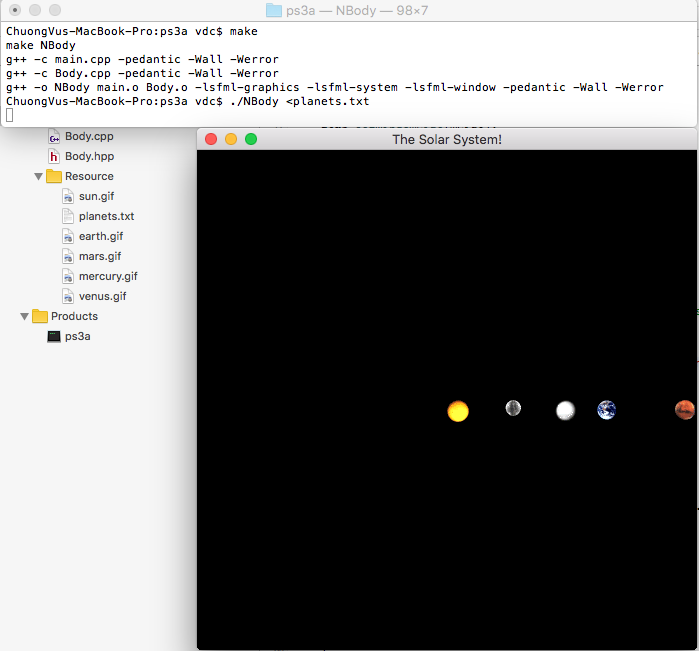
This assignment also had two parts A and B.

The main point for this assignment was read the information from the planets.txt file and put the object to the right position on the SFML windows. I pretty much watched the online lecture from Fred Martin that he went through the lecture and explained about the homework. It made me have a better idea how to write the header and its functions to make the program run as homework.

What I did for this assignment was create a Body class with hold all the information form the file such as x, y position and mass. I used the <vector> library to create an array of objects and push all the information need to calculate in the Body, after I got all the information I wanted, I will call it out and display it on the SFML windows.

The hard part for this assignment was part B which added the physics simulation and the animation that I had created in part A. The header file was almost the same but I do need to work on the main file which was for the planets moving around the Sun. I got to calculate the Acceleration and the Velocity for each planet to make sure that it is responding correctly to the assignment wanted. This assignment took me like three days to think about the calculation and remind me of most physics laws.

Output for ps3a:



1. #
2. # Makefile for ps3a
3. #
5. all:
6. make NBody
8. NBody: main.o Body.o
9. g++ -o NBody main.o Body.o -lsfml-graphics -lsfml-system -lsfml-window -pedantic -Wall -Werror
11. main.o: Body.hpp main.cpp
12. g++ -c main.cpp -pedantic -Wall -Werror
14. Body.o: Body.hpp Body.cpp
15. g++ -c Body.cpp -pedantic -Wall -Werror
17. clean:
18. rm -f \*.o \*~ NBody
19. //-------------------------------
20. //Name: Chuong Vu
21. //Assigment ps3a - N-Body Simulation
22. //Class: Computing IV
23. //Professor: Dr. Yelena Rykalova
24. //Filename: main.cpp
25. //-------------------------------
27. //
28. // Disclaimer:
29. // ----------
30. //
31. // This code will work only if you selected window, graphics and audio.
32. //
33. // In order to load the resources like cute\_image.png, you have to set up
34. // your target scheme:
35. //
36. // - Select "Edit Scheme…" in the "Product" menu;
37. // - Check the box "use custom working directory";
38. // - Fill the text field with the folder path containing your resources;
39. //        (e.g. your project folder)
40. // - Click OK.
41. //  Prof. Yelena Rykalova
42. //
43. #include <iostream>
44. #include <sstream>
45. #include <cstdlib>
46. #include <vector>
47. #include "Body.hpp"
49. #include <SFML/Graphics.hpp>
50. #include <SFML/Window.hpp>
51. #include <SFML/System.hpp>
53. **int** main(**int** argc, **char** **const**\*\* argv) {
54. **int** wSize = 500;
55. std::string input;
56. **int** n, i;
57. unsigned y;
58. **double** uSize;
60. //read integer to n
61. getline(std::cin, input);
62. n = (atoi(input.c\_str()));
64. //read double to universeSize
65. getline(std::cin, input);
66. uSize = (atof(input.c\_str()));

69. Body temp;
70. temp.setWindowSize(wSize);
71. temp.setUniverseSize(uSize);

74. std::vector<Body> universe;
76. //fill universe with n size
77. **for**(i = 0; i < n; i++)
78. universe.push\_back(temp);
80. //read value from text file to universe
81. **for**(i = 0; i < n; i++)
82. {
83. getline(std::cin, input);
84. std::istringstream iss(input); //get value between space from file
85. iss >> universe[i];
87. }
88. sf::RenderWindow window(sf::VideoMode(wSize, wSize), "The Solar System!");

91. **while**(window.isOpen())
92. {
93. sf::Event event;
94. **while**(window.pollEvent(event))
95. {
96. **if**(event.type == sf::Event::Closed)
97. window.close();
99. **if**(event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape)
100. window.close();
101. }
102. window.clear();
104. **for**(y = 0; y < universe.size(); y++)
105. {
106. window.draw(universe[y]);
107. }
108. window.display();
110. }
112. **return** EXIT\_SUCCESS;
113. }
114. //
115. //  Body.hpp
116. //  ps3a
117. //
118. //  Created by ChuongVu on 10/7/16.
119. //  Copyright © 2016 ChuongVu. All rights reserved.
120. //  Xcode
121. //  Prof. Yelena Rykalova
123. #ifndef Body\_hpp
124. #define Body\_hpp
126. #include <iostream>
127. #include <cstring>
128. #include <SFML/Graphics.hpp>
130. **class** Body : **public** sf::Drawable {
132. **public**:
133. Body();
134. Body(**double** \_xpos, **double** \_ypos, **double** \_xvel, **double** \_yvel, **double** \_mass, std::string \_fname);
135. ~Body();
136. **void** setWindowSize(**int** \_size);
137. **void** setUniverseSize(**double** \_size);
138. **friend** std::istream &operator>>(std::istream &input, Body &arg);
140. **private**:
141. **double** xpos, ypos, xvel, yvel, mass, universeSize;
142. **int** windowSize;
143. sf::Sprite sprite;
144. sf::Texture texture;
145. **void** draw(sf::RenderTarget& target, sf::RenderStates states) **const**;
147. };
149. #endif /\* Body\_hpp \*/
150. //
151. //  Body.cpp
152. //  ps3a
153. //
154. //  Created by ChuongVu on 10/7/16.
155. //  Copyright © 2016 ChuongVu. All rights reserved.
156. //
157. //  Prof. Yelena Rykalova
158. //
159. #include <iostream>
160. #include <sstream>
161. #include "Body.hpp"

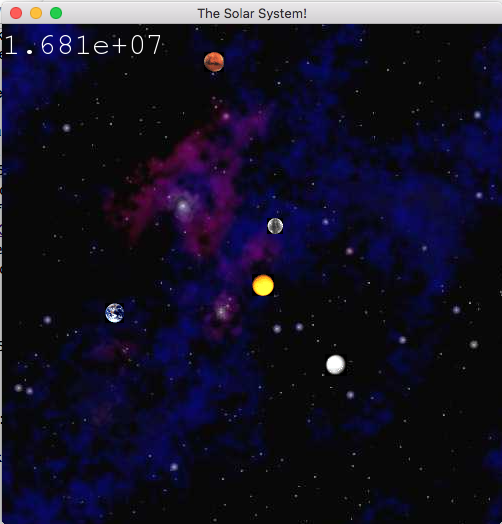
164. //default constructor
165. Body::Body() : xpos(0), ypos(0), xvel(0), yvel(0), mass(0){
166. }

169. //non-default constructor
170. Body::Body(**double** \_xpos, **double** \_ypos, **double** \_xvel, **double** \_yvel, **double** \_mass, std::string \_fname) : xpos(\_xpos), ypos(\_ypos), xvel(\_xvel), yvel(\_yvel), mass(\_mass) {
171. texture.loadFromFile(\_fname);
172. sprite.setTexture(texture);
173. }

176. //de-constructor
177. Body::~Body() {}

180. //set windows size
181. **void** Body::setWindowSize(**int** \_size) {
182. windowSize = \_size;
183. }
185. //set universe size
186. **void** Body::setUniverseSize(**double** \_size) {
187. universeSize = \_size;
188. }
190. //draw planets based on position
191. **void** Body::draw(sf::RenderTarget& target, sf::RenderStates states) **const** {
192. sf::Sprite sprite\_temp = sprite;
193. **double** ratio = (windowSize / 2) / universeSize;
194. **double** rxpos = xpos \* ratio + (windowSize / 2);
195. **double** rypos = ypos \* ratio + (windowSize / 2);
197. sprite\_temp.setPosition(rxpos, rypos);
199. target.draw(sprite\_temp);
200. }
202. //Get position and file name from file
203. std::istream &operator>>(std::istream &input, Body &arg) {
204. std::string fname;
205. input >> arg.xpos;
206. input >> arg.ypos;
207. input >> arg.xvel;
208. input >> arg.yvel;
209. input >> arg.mass;
210. input >> fname;
211. arg.texture.loadFromFile(fname);
212. arg.sprite.setTexture(arg.texture);
214. **return** input;
215. }

Output for ps3b:



1. #
2. # Makefile for ps3b
3. #
5. all:
6. make NBody
8. NBody: main.o Body.o
9. g++ -o NBody main.o Body.o -lsfml-graphics -lsfml-system -lsfml-window -pedantic -Wall -Werror
11. main.o: Body.hpp main.cpp
12. g++ -c main.cpp -pedantic -Wall -Werror
14. Body.o: Body.hpp Body.cpp
15. g++ -c Body.cpp -pedantic -Wall -Werror
17. clean:
18. rm -f \*.o \*~ NBody
19. //-------------------------------
20. //Name: Chuong Vu
21. //Assigment ps3b - N-Body Simulation
22. //Class: Computing IV
23. //Professor: Dr. Yelena Rykalova
24. //Filename: main.cpp
25. //-------------------------------
26. //
27. // Disclaimer:
28. // ----------
29. //
30. // This code will work only if you selected window, graphics and audio.
31. //
32. // In order to load the resources like cute\_image.png, you have to set up
33. // your target scheme:
34. //
35. // - Select "Edit Scheme…" in the "Product" menu;
36. // - Check the box "use custom working directory";
37. // - Fill the text field with the folder path containing your resources;
38. //        (e.g. your project folder)
39. // - Click OK.
40. //  Prof. Yelena Rykalova
41. //
42. #include <iostream>
43. #include <sstream>
44. #include <cstdlib>
45. #include <vector>
46. #include <cmath>
47. #include <cstring>
48. #include "Body.hpp"
50. #include <SFML/Graphics.hpp>
51. #include <SFML/Window.hpp>
52. #include <SFML/System.hpp>
54. **int** main(**int** argc, **char** **const**\*\* argv)
55. {
56. **int** wSize = 500;
57. **double** G = 6.67e-11;    //Constant Gravitation
58. std::string input;
59. **int** n, i;
60. **double** uSize, totalTime, stepTime, elapsedTime;
62. **if**(argc < 3) {
63. std::cout << "Missing command line args" << std::endl;
64. **return** 1;
65. } **else** **if**(argc > 3) {
66. std::cout << "Too many command line args" << std::endl;
67. **return** 1;
68. }
70. // Prepare time variables
71. totalTime = atof(argv[1]);
72. stepTime = atof(argv[2]);
73. elapsedTime = 0.0;
75. // Prepare the time text
76. sf::Font courier\_new;
77. courier\_new.loadFromFile("cour.ttf");
79. sf::Text timeText("0.0", courier\_new);
81. // Prepare the image background
82. sf::Image backgroundI;
83. backgroundI.loadFromFile("starfield.jpg");
85. sf::Texture backgroundT;
86. backgroundT.loadFromImage(backgroundI);
88. sf::Sprite backgroundP(backgroundT);
90. //read integer to n
91. getline(std::cin, input);
92. n = (atoi(input.c\_str()));
94. //read double to universeSize
95. getline(std::cin, input);
96. uSize = (atof(input.c\_str()));
98. Body temp;
99. temp.setWindowSize(wSize);
100. temp.setUniverseSize(uSize);
102. std::vector<Body> universe;
104. //fill universe with n size
105. **for**(i = 0; i < n; i++)
106. universe.push\_back(temp);
108. //read value from text file to universe
109. **for**(i = 0; i < n; i++) {
110. getline(std::cin, input);
111. std::istringstream iss(input); //get value between space from file
112. iss >> universe[i];
113. }
115. sf::RenderWindow window(sf::VideoMode(wSize, wSize), "The Solar System!");
117. **while**(window.isOpen()) {
118. sf::Event event;
119. **while**(window.pollEvent(event)) {
120. **if**(event.type == sf::Event::Closed)
121. window.close();
123. **if**(event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape)
124. window.close();
126. }
127. // Vector of Force, initialized to 0
128. std::vector<**double**> forceX;
129. std::vector<**double**> forceY;
131. **for**(**int** i = 0; i < n; i++) {
132. forceX.push\_back(0.0);
133. forceY.push\_back(0.0);
134. }
136. // Calculate Acceleration
137. **for**(unsigned i = 0; i < universe.size(); i++) {
138. **for**(unsigned j = 0; j < universe.size(); j++) {
139. **if**(i == j) {
140. forceX[i] += 0;
141. forceY[i] += 0;
142. } **else** {
143. **double** deltaX = universe[j].getxPos() - universe[i].getxPos();
144. **double** deltaY = universe[j].getyPos() - universe[i].getyPos();
145. **double** r = sqrt(deltaX \* deltaX + deltaY \* deltaY);
146. **double** F = (G \* universe[i].getMass() \* universe[j].getMass()) / (r \* r);
147. **double** Fx = F \* (deltaX / r);
148. **double** Fy = F \* (deltaY / r);
149. forceX[i] += Fx;
150. forceY[i] += Fy;
151. }
152. }
153. }
155. // Calculate New Velocity
156. **for**(unsigned i = 0; i < universe.size(); i++) {
157. **double** Ax = forceX[i] / universe[i].getMass();
158. **double** Ay = forceY[i] / universe[i].getMass();
160. universe[i].updateXVel(stepTime \* Ax);
161. universe[i].updateYVel(stepTime \* Ay);
163. // Update position
164. universe[i].step(stepTime);
165. }
167. window.clear();
169. window.draw(backgroundP);
171. // Draw each body in the vector
172. **for**(unsigned i = 0; i < universe.size(); i++)
173. window.draw(universe[i]);
175. // Update Time, close if max is reached
176. elapsedTime += stepTime;
177. **if**(elapsedTime > totalTime)
178. window.close();
180. // Update timeText
181. std::stringstream ss;
182. ss << elapsedTime;
183. timeText.setString(ss.str());
185. // Draw Current Time
186. window.draw(timeText);
188. window.display();
190. }
192. **for**(unsigned i = 0; i < universe.size(); i++)
193. std::cout << universe[i] << std::endl;
195. **return** EXIT\_SUCCESS;
196. }
197. //
198. //  Body.hpp
199. //  ps3b
200. //
201. //  Created by ChuongVu on 10/7/16.
202. //  Copyright © 2016 ChuongVu. All rights reserved.
203. //  Xcode
204. //  Prof. Yelena Rykalova
206. #ifndef Body\_hpp
207. #define Body\_hpp
209. #include <iostream>
210. #include <cstring>
211. #include <SFML/Graphics.hpp>
213. **class** Body : **public** sf::Drawable {
215. **public**:
216. Body();
217. Body(**int** \_windowSize, **double** \_universeSize);
218. ~Body();
219. **void** setWindowSize(**int** \_size);
220. **void** setUniverseSize(**double** \_size);
221. **friend** std::istream &operator>>(std::istream &input, Body &arg);
222. **friend** std::ostream &operator<<(std::ostream &output, Body &arg);
223. **double** getxPos();
224. **double** getyPos();
225. **double** getMass();
226. **void** updateXVel(**double** AxT);
227. **void** updateYVel(**double** AxT);
228. **void** step(**double** time);
230. **private**:
231. **double** xpos, ypos, xvel, yvel, mass, universeSize;
232. **int** windowSize;
233. sf::Sprite sprite;
234. sf::Texture texture;
235. std::string fname;
236. **void** draw(sf::RenderTarget& target, sf::RenderStates states) **const**;
237. };
239. #endif /\* Body\_hpp \*/
240. //
241. //  Body.cpp
242. //  ps3b
243. //
244. //  Created by ChuongVu on 10/7/16.
245. //  Copyright © 2016 ChuongVu. All rights reserved.
246. //
247. //  Prof. Yelena Rykalova
248. //
249. #include <iostream>
250. #include <sstream>
251. #include <iomanip>
252. #include "Body.hpp"

255. //default constructor
256. Body::Body() : xpos(0), ypos(0), xvel(0), yvel(0), mass(0)
257. {
258. }

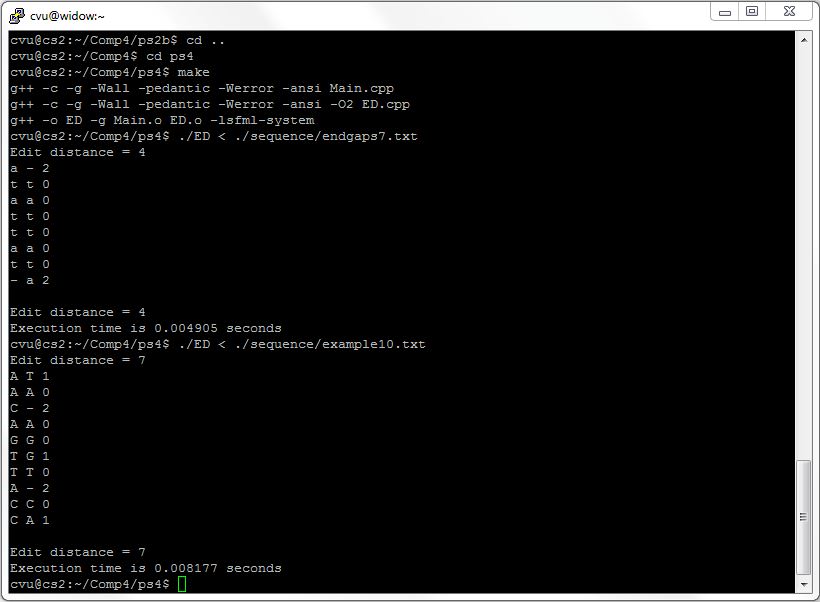
261. //non-default constructor
262. Body::Body(**int** \_windowSize, **double** \_universeSize) : xpos(0), ypos(0), xvel(0), yvel(0), mass(0)
263. {
264. windowSize = \_windowSize;
265. universeSize = \_universeSize;
266. }
268. **double** Body::getxPos()
269. {
270. **return** xpos;
271. }
273. **double** Body::getyPos()
274. {
275. **return** ypos;
276. }
278. **double** Body::getMass()
279. {
280. **return** mass;
281. }
283. **void** Body::updateXVel(**double** AxT)
284. {
285. xvel -= AxT;
286. }
288. **void** Body::updateYVel(**double** AxT)
289. {
290. yvel -= AxT;
291. }
293. **void** Body::step(**double** time)
294. {
295. xpos = xpos - time \* xvel;
296. ypos = ypos - time \* yvel;
297. }
299. //de-constructor
300. Body::~Body() {}
302. //set windows size
303. **void** Body::setWindowSize(**int** \_size)
304. {
305. windowSize = \_size;
306. }
308. //set universe size
309. **void** Body::setUniverseSize(**double** \_size)
310. {
311. universeSize = \_size;
312. }
314. //draw planets based on position
315. **void** Body::draw(sf::RenderTarget& target, sf::RenderStates states) **const**
316. {
317. sf::Sprite sprite\_temp = sprite;
318. **double** ratio = (windowSize / 2) / universeSize;
319. **double** rxpos = xpos \* ratio + (windowSize / 2);
320. **double** rypos = ypos \* ratio + (windowSize / 2);
322. sprite\_temp.setPosition(rxpos, rypos);
324. target.draw(sprite\_temp);
325. }
327. //Get position and file name from file
328. std::istream &operator>>(std::istream &input, Body &arg)
329. {
330. std::string fname;
331. input >> arg.xpos;
332. input >> arg.ypos;
333. input >> arg.xvel;
334. input >> arg.yvel;
335. input >> arg.mass;
336. input >> fname;
337. arg.texture.loadFromFile(fname);
338. arg.sprite.setTexture(arg.texture);
340. **return** input;
341. }
343. std::ostream &operator<<(std::ostream &output, Body &arg) {
344. output << std::setw(14) << arg.xpos;
345. output << std::setw(14) << arg.ypos;
346. output << std::setw(14) << arg.xvel;
347. output << std::setw(14) << arg.yvel;
348. output << std::setw(14) << arg.mass;
349. output << std::setw(14) << arg.fname;
351. **return** output;
352. }

**PS4 Edit Distance**

This assignment was implementing the DNA sequence alignment. This one really helpful for who want to know about analysis DNA (gene), also next semester I will take bioinformatics class which is writing a code to analysis difference DNA sequence so this one was a start point for me to understand how the DNA works.

I based on what professor went over in class how to get the alignment. And also I use the idea from Princeton to calculate the alignment. To find the alignment itself, I check from the top-left to bottom-right. At the top-left, if both letters is same, I will go diagonal, else, it will check to find the smallest number from bottom-left to bottom-right and top-right. Finally, it based on where it so it can either get penalty or gap and final will calculate the edit distance. The method mostly comes from Princeton website. The hardest part for this assignment was known how to fill out the DNA matrix table and understand how to get the correct edit distance and the optimal alignment.

Output:



1. #
2. #  Make file for ps4
3. #
5. CC = g++
6. CFLAGS = -c -g -Wall -pedantic -Werror -ansi
7. LFLAGS = -lsfml-system
9. all: ED
11. ED: Main.o ED.o
12. $(CC) -o ED -g Main.o ED.o $(LFLAGS)
14. Main.o: ED.cpp Main.cpp
15. $(CC) $(CFLAGS) Main.cpp
17. ED.o: ED.hpp ED.cpp
18. $(CC) $(CFLAGS) -O2 ED.cpp
20. clean:
21. rm -f \*.o \*~ ED
22. //
23. //  main.cpp
24. //  ps4
25. //
26. //  Created by ChuongVu on 10/16/16.
27. //  Copyright © 2016 ChuongVu. All rights reserved.
28. //
30. #include <iostream>
31. #include <cstring>
32. #include "ED.hpp"
33. #include <SFML/System.hpp>      //for time clock
35. **int** main(**int** argc, **const** **char** \* argv[]) {
37. sf::Clock clock;
38. sf::Time t;
40. std::string stringX;
41. std::string stringY;
43. std::cin >> stringX;
44. std::cin >> stringY;

47. ED DNA(stringX, stringY);
49. // Get what we want from the class
50. std::cout << "Edit distance = " << DNA.getEditDistance() << std::endl;
51. std::cout << DNA.getEditString() << std::endl;
52. std::cout << "Edit distance = " << DNA.getEditDistance() << std::endl;

55. t = clock.getElapsedTime();
57. std::cout << "Execution time is " << t.asSeconds() << " seconds \n";
59. **return** 0;
60. }
61. //
62. //  ED.hpp
63. //  ps4
64. //
65. //  Created by ChuongVu on 10/16/16.
66. //  Copyright © 2016 ChuongVu. All rights reserved.
67. //
69. #ifndef ED\_hpp
70. #define ED\_hpp
72. #include <iostream>
73. #include <cstring>
74. #include <vector>
76. **class** ED {
77. **public**:
78. ED(std::string \_stringX, std::string \_stringY);
79. **int** getEditDistance() **const**;
80. std::string getEditString() **const**;
81. **void** print() **const**;
83. **private**:
84. // Input Variables
85. std::string strX;
86. std::string strY;
88. // Constructed Variables
89. std::vector< std::vector<**int**> > opt;    //2-d vector opt[i][j] for X and Y
91. // Private Functions
92. **int** OptDistance();
93. **int** editDistance;   // value return distance opt[0][0]
94. **int** penalty(**char** a, **char** b);
95. **int** min(**int** a, **int** b, **int** c);
97. std::string Alignment() **const**;
98. std::string editStr;

101. };
102. #endif /\* ED\_hpp \*/
103. //
104. //  ED.cpp
105. //  ps4
106. //
107. //  Created by ChuongVu on 10/16/16.
108. //  Copyright © 2016 ChuongVu. All rights reserved.
109. //
110. #include <iostream>
111. #include <cstring>
112. #include <vector>
113. #include <sstream>
114. #include "ED.hpp"

117. ED::ED(std::string \_stringX, std::string \_stringY) : strX(\_stringX), strY(\_stringY)
118. {
119. //create a temp vector
120. std::vector<**int**> temp;
122. //fill matrix table with value 0
123. **for**(**int** i = 0; i < **static\_cast**<**int**>(strY.length()) + 1; i++)
124. temp.push\_back(0);
125. **for**(**int** i = 0; i < **static\_cast**<**int**>(strX.length()) + 1; i++)
126. opt.push\_back(temp);
128. //Call optDistance to calculate the matrix values
129. editDistance = OptDistance();
130. editStr = Alignment();
131. //print();
133. }

136. //Fill matrix table
137. **int** ED::OptDistance()
138. {
139. //fill from bottom right to left and upper
141. **for**(**int** i = opt.size() - 1; i >= 0; i--)
142. {
143. **for**(**int** j = opt[i].size() - 1; j >= 0; j--)
144. {
145. **if**((i == **static\_cast**<**int**>(opt.size() - 1)) && (j == **static\_cast**<**int**>(opt[i].size() - 1)))
146. opt[i][j] = 0;                  //base case
147. **else** **if**(i == **static\_cast**<**int**>(opt.size() - 1))
148. opt[i][j] = opt[i][j + 1] + 2;  //Y col + grap
149. **else** **if** (j == **static\_cast**<**int**>(opt[i].size() - 1))
150. opt[i][j] = opt[i + 1][j] + 2;  //Y col + grap
151. **else**
152. opt[i][j] = min(**static\_cast**<**int**>(opt[i+1][j+1] + penalty(strX[i], strY[j])), **static\_cast**<**int**>(opt[i + 1][j] + 2), **static\_cast**<**int**>(opt[i][j + 1] + 2));
153. }
154. }
156. //Get Edit Distance value
157. **return** opt[0][0];
159. }
161. //get penalty
162. **int** ED::penalty(**char** a, **char** b)
163. {
164. **return** a == b ? 0 : 1;
165. }
167. //Get min value
168. **int** ED::min(**int** a, **int** b, **int** c)
169. {
170. **return** a < b ? (a < c ? a : c) : (b < c ? b : c);
171. }

174. std::string ED::Alignment() **const**
175. {
176. **int** i = 0, j = 0;
177. std::stringstream ss;
179. **while**(i < **static\_cast**<**int**>(opt.size() - 1) || j < **static\_cast**<**int**>(opt[0].size() - 1))
180. {
181. **if**((i < **static\_cast**<**int**>(opt.size() - 1)) && (j < **static\_cast**<**int**>(opt[0].size() - 1)) && (opt[i+1][j+1] <= opt[i+1][j] + 1) && (opt[i+1][j+1] <= opt[i][j+1] + 1))
182. {
183. ss << strX[i] << " " << strY[j] << " " << opt[i][j] - opt[i+1][j+1] << '\n';
184. i++;
185. j++;
186. }
187. **else** **if**(((i < **static\_cast**<**int**>(opt.size() - 1)) && (opt[i+1][j] <= opt[i][j+1])) || (j == **static\_cast**<**int**>(opt[0].size() - 1)))
188. {
189. ss << strX[i] << " " << "-" << " " << opt[i][j] - opt[i+1][j] << '\n';
190. i++;
191. }
192. **else**
193. {
194. ss << "-" << " " << strY[j] << " " << opt[i][j] - opt[i][j+1] << '\n';
195. j++;
196. }
197. }
199. **return** ss.str();
200. }

203. **int** ED::getEditDistance() **const** {
204. **return** editDistance;
205. }
207. std::string ED::getEditString() **const** {
208. **return** editStr;
209. }

212. **void** ED::print() **const**
213. {
214. //print table test
215. **for**(unsigned i = 0; i < opt.size(); i ++)
216. {
217. **for**(unsigned j = 0; j < opt[i].size(); j++)
218. {
219. std::cout << "\t" << opt[i][j];
220. }
221. std::cout << std::endl;
222. }
223. }

**PS5** **Ring Buffer and Guitar Hero**

This assignment was create a simulate plucking a guitar strung using the *Karplus-Strong* algorithms. This algorithm played a seminal role in the emergence of physically modeled sound synthesis. Basically when I press a key that represent the guitar-string, the program will calculate the frequency to make a sound.

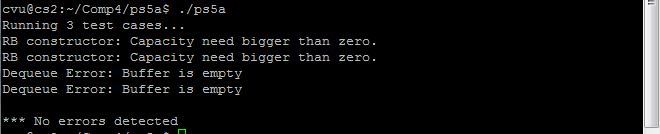
I pretty much using header template that provided by Prof. Rykalov after I read the assignment on the princeton.edu website and also I have watch some video from Prof. Fred in previous semester so it helped me have more idea how to write a cyclic wrap around buffer.

The problem first I got is I use the push\_back in enqueue so after a certain enqueue, I will get error because the size of vector keep growth and dequeue can't delete in peek() so I got stuck at error where the size keep growth and I can't use pop\_back for dequeuer.

Later, I realize that I need to create cyclic-wrap around buffer it means that the size and the capacity are fixed. The only thing is change is the first and last index in the buffer so in the constructor method, I reserve the capacity and also initial all element in the vector 0 so it will have fixed size with all 0 element. And then keep follow the princeton.edu cation website with the test file that provide by the Prof. Rykalov, I have fixed all the error.

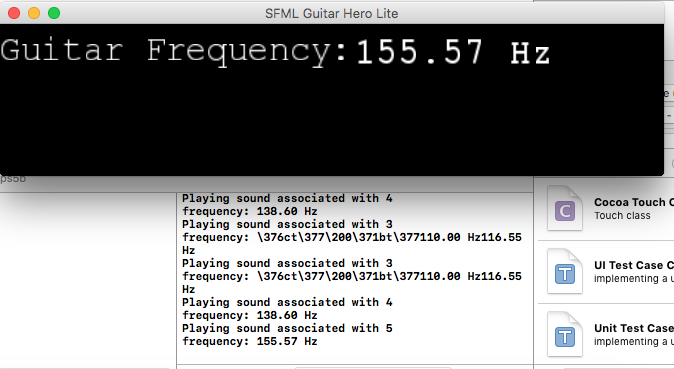
After complete the part A then the part B was much easier since I only need to accept the input key and display. By finish this assignment, I know more about <vector> library and it really useful in C++ rather than using array because with vector, I can push, pop without need to malloc variable in memory because <vector> library take care all of it.

Ps5a output:



1. #
2. # Make file for ps5a
3. #
4. CC = g++
5. CFLAGS = -c -Wall -pedantic -Werror -ansi
6. LFLAGS = -lboost\_unit\_test\_framework
8. all: ps5a
10. ps5a: test.o RingBuffer.o
11. $(CC) -o ps5a  test.o RingBuffer.o $(LFLAGS)
13. test.o: RingBuffer.cpp test.cpp
14. $(CC) $(CFLAGS) test.cpp
16. RingBuffer.o: RingBuffer.hpp RingBuffer.cpp
17. $(CC) $(CFLAGS) RingBuffer.cpp
19. clean:
20. rm -f \*.o \*~ ps5a
21. // Copyright 2015 fredm@cs.uml.edu for 91.204 Computing IV
22. // Wed Mar 25 06:32:17 2015
24. #define BOOST\_TEST\_DYN\_LINK
25. #define BOOST\_TEST\_MODULE Main
26. #include <boost/test/unit\_test.hpp>
28. #include <stdint.h>
29. #include <iostream>
30. #include <string>
31. #include <exception>
32. #include <stdexcept>
34. #include "RingBuffer.hpp"
36. BOOST\_AUTO\_TEST\_CASE(RBcontructor) {
37. // normal constructor
38. BOOST\_REQUIRE\_NO\_THROW(RingBuffer(100));
40. // this should fail
41. BOOST\_REQUIRE\_THROW(RingBuffer(0), std::exception);
42. BOOST\_REQUIRE\_THROW(RingBuffer(0), std::invalid\_argument);
43. }
45. BOOST\_AUTO\_TEST\_CASE(RBenque\_dequeue) {
46. RingBuffer rb(100);
48. rb.enqueue(2);
49. rb.enqueue(1);
50. rb.enqueue(0);
52. BOOST\_REQUIRE(rb.dequeue() == 2);
53. BOOST\_REQUIRE(rb.dequeue() == 1);
54. BOOST\_REQUIRE(rb.dequeue() == 0);
56. BOOST\_REQUIRE\_THROW(rb.dequeue(), std::runtime\_error);
57. }
59. BOOST\_AUTO\_TEST\_CASE(testcase3) {
60. RingBuffer testcase3(100);
62. testcase3.enqueue(50);
63. testcase3.enqueue(20);
65. BOOST\_REQUIRE(testcase3.dequeue() == 50);
66. BOOST\_REQUIRE(testcase3.dequeue() == 20);
68. BOOST\_REQUIRE\_THROW(testcase3.dequeue(), std::runtime\_error);
69. }
70. // Copyright Chuong Vu
71. // RingBuffer.hpp
73. #ifndef \_RingBuffer\_HPP
74. #define \_RingBuffer\_HPP
75. #include <stdint.h>
76. #include <vector>
78. **class** RingBuffer{
79. **public**:
80. **explicit** RingBuffer(**int** capacity);
81. ~RingBuffer();
82. // add item x to the end
83. **void** enqueue(int16\_t x);
84. // delete and return item from the front
85. int16\_t dequeue();
86. // return (but do not delete) item from the front
87. int16\_t peek() **const**;
88. // return number of items currently in the buffer
89. **int** size() **const**;
90. // check empty
91. **bool** isEmpty() **const**;
92. // is the buffer full  (size equals capacity)?
93. **bool** isFull() **const**;
95. **private**:
96. std::vector<int16\_t> buffer;
97. // keep track number of items currently in buffer
98. **int** \_size;
99. // capacity
100. **int** \_capacity;
101. // first index
102. **int** \_first;
103. // last index
104. **int** \_last;
105. };
107. #endif
108. // Copyright Chuong Vu
109. // Filename: RingBuffer.cpp
111. #include <stdint.h>
112. #include <iostream>
113. #include <stdexcept>
114. #include "RingBuffer.hpp"
116. // static\_cast<int>
117. RingBuffer::RingBuffer(**int** capacity) {
118. **try** {
119. **if** (capacity < 1)
120. **throw** std::invalid\_argument("Capacity need bigger than zero");
121. } **catch**(std::invalid\_argument& invalid\_arg) {
122. // catch invalid\_argument type.
123. std::cerr << "RB constructor: Capacity need bigger than zero.\n";
124. **throw** invalid\_arg;
125. }
127. // researve capacity for buffer
128. buffer.reserve(capacity);
129. // initial value 0 for every element in buffer
130. // so later on first and last just move around
131. // the size of vector will stay the same
132. **for** (**int** i = 0; i < capacity; i++)
133. buffer.push\_back(0);
135. \_first = 0;
136. \_last = 0;
137. \_size = 0;
138. \_capacity = capacity;
139. }
141. // return size of buffer
142. **int** RingBuffer::size() **const** {
143. **return** \_size;
144. }
146. // if \_size >0 then RingBuffer is not empty
147. **bool** RingBuffer::isEmpty() **const** {
148. **return** \_size > 0 ? **false** : **true**;
149. }
151. // is the buffer full  (size equals capacity)?
152. **bool** RingBuffer::isFull() **const** {
153. **return** \_size == \_capacity ? **true** : **false**;
154. }
156. // add item x to the end
157. **void** RingBuffer::enqueue(int16\_t x) {
158. **try** {
159. **if** (isFull())
160. **throw** std::runtime\_error("Buffer is full");
161. } **catch**(std::runtime\_error& run\_err) {
162. // catch run time error type
163. std::cerr << "Enqueue Error: Buffer is full\n";
164. **throw** run\_err;
165. }
167. // assgin value x to location \_last
168. buffer[\_last] = x;
170. // When either index equals capacity
171. // make it wrap-around by changing the index to 0.
172. **if** (\_last == \_capacity - 1)
173. \_last = 0;
174. **else**
175. \_last++;
177. // update \_size
178. \_size++;
179. }
181. RingBuffer::~RingBuffer() {}
183. // delete and return item from the front
184. int16\_t RingBuffer::dequeue() {
185. **try** {
186. **if** (isEmpty())
187. **throw** std::runtime\_error("Buffer is empty");
188. } **catch**(std::runtime\_error& run\_err) {
189. // catch run time error type
190. std::cerr << "Dequeue Error: Buffer is empty\n";
191. **throw** run\_err;
192. }
194. // create temp variable = peek
195. int16\_t temp = peek();
197. // move \_first index
198. **if** (\_first == \_capacity - 1)
199. \_first = 0;
200. **else**
201. \_first++;
203. // update \_size
204. \_size--;
206. **return** temp;
207. }
209. // return (but do not delete) item from the front
210. int16\_t RingBuffer::peek() **const** {
211. **try** {
212. **if** (isEmpty())
213. **throw** std::runtime\_error("Buffer is empty");
214. } **catch**(std::runtime\_error& run\_err) {
215. // catch run time error type
216. std::cerr << "Peek Error: Buffer is empty\n";
217. **throw** run\_err;
218. }
219. **return** buffer[\_first];
220. }

Ps5b output:



1. #
2. # Make file for ps5b
3. #
5. CC = g++
6. OFLAGS = -c -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -ansi -pedantic -Wall -Werror -lboost\_unit\_test\_framework
7. CFLAGS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -ansi -pedantic -Wall -Werror -lboost\_unit\_test\_framework
9. all: GuitarHero
11. GuitarHero: RingBuffer.o GuitarString.o GuitarHero.o
12. $(CC) -o GuitarHero GuitarHero.o GuitarString.o RingBuffer.o $(CFLAGS)
14. GuitarHero.o: GuitarHero.cpp GuitarString.hpp RingBuffer.hpp
15. $(CC) $(OFLAGS) GuitarHero.cpp
17. GuitarString.o: GuitarString.cpp GuitarString.hpp RingBuffer.hpp
18. $(CC) $(OFLAGS) GuitarString.cpp
20. RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
21. $(CC) $(OFLAGS) RingBuffer.cpp
23. permissions: all
24. chmod o+x GuitarHero
26. clean:
27. rm -f \*.o \*~GuitarHero
28. // Copyright Chuong Vu
29. // Filename: GuitarHero.cpp
30. #include <SFML/Graphics.hpp>
31. #include <SFML/System.hpp>
32. #include <SFML/Audio.hpp>
33. #include <SFML/Window.hpp>
35. #include <math.h>
36. #include <limits.h>
37. #include <stdint.h>
38. #include <sstream>
39. #include <iostream>
40. #include <string>
41. #include <exception>
42. #include <stdexcept>
43. #include <vector>
45. #include "RingBuffer.hpp"
46. #include "GuitarString.hpp"
48. #define CONCERT\_A 220.0
49. #define SAMPLES\_PER\_SEC 44100
51. std::vector<int16\_t> makeSamplesFromString(GuitarString gs) {
52. std::vector<int16\_t> samples;
54. gs.pluck();
55. **int** duration = 8;  // seconds
56. **int** i;
57. **for** (i= 0; i < SAMPLES\_PER\_SEC \* duration; i++) {
58. gs.tic();
59. samples.push\_back(gs.sample());
60. }
62. **return** samples;
63. }
65. **int** main() {
66. sf::RenderWindow window(sf::VideoMode(450, 100), "SFML Guitar Hero Lite");
67. sf::Event event;
69. **double** freq;
70. // keyboard layout
71. std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
72. std::string str;
73. std::stringstream ss;
74. std::vector<std::string> frequency;
76. std::vector<std::vector<int16\_t> > samples(37);
77. std::vector<sf::Sound> sounds(37);
78. std::vector<sf::SoundBuffer> soundBuffers(37);
80. // calculate and store frequency to the sound vector
81. **for** (**int** i = 0; i < 37; i++) {
82. freq = (CONCERT\_A \* 2) \* pow(2, (i - 24) / 12.0);
84. str = "";
85. ss << ceil(freq \* 100) / 100;
86. str = ss.str();
87. str.erase(str.begin()+6, str.end());
88. str = str + " Hz";
89. frequency.push\_back(str);
90. ss.clear();
92. GuitarString gs(freq);
93. samples[i] = makeSamplesFromString(gs);
95. **if** (!soundBuffers[i].loadFromSamples(&samples[i][0], samples[i].size(),
96. 2, SAMPLES\_PER\_SEC))
97. **throw** std::runtime\_error("sf::SoundBuffer: failed to load from samples.");
99. sounds[i].setBuffer(soundBuffers[i]);
100. }
102. sf::Font font;
103. font.loadFromFile("cour.ttf");
104. sf::Text text1("Guitar Frequency:", font, 24);
105. sf::Text text("", font, 24);
106. text.setStyle(sf::Text::Bold);
107. text.setPosition(240, 2);
109. **while** (window.isOpen()) {
110. **while** (window.pollEvent(event)) {
111. **switch** (event.type) {
112. **case** sf::Event::Closed:
113. window.close();
114. **break**;
116. **case** sf::Event::KeyPressed:
117. **if** (event.key.code == sf::Keyboard::Escape)
118. window.close();
119. **break**;
121. **case** sf::Event::TextEntered:
122. **if** (event.text.unicode < 128) {
123. std::string keypress;
124. keypress += **static\_cast**<**char**>(event.text.unicode);
125. std::cout << "Playing sound associated with " << keypress;
126. std::cout << std::endl;
127. **int** index = keyboard.find(keypress);
128. sounds[index].play();
129. std::cout << "frequency: " << frequency[index] << std::endl;
130. text.setString(frequency[index]);
131. window.draw(text);
132. }
133. **break**;
134. **default**:
135. **break**;
136. }
138. window.clear();
139. window.draw(text1);
140. window.draw(text);
141. window.display();
142. }
143. }
145. **return** 0;
146. }
147. // Copyright Chuong Vu
148. // Filename: GuitarHeroExtra.cpp
149. #include <SFML/Graphics.hpp>
150. #include <SFML/System.hpp>
151. #include <SFML/Audio.hpp>
152. #include <SFML/Window.hpp>
154. #include <math.h>
155. #include <limits.h>
156. #include <stdint.h>
157. #include <sstream>
158. #include <iostream>
159. #include <string>
160. #include <exception>
161. #include <stdexcept>
162. #include <vector>
164. #include "RingBuffer.hpp"
165. #include "GuitarString.hpp"
167. #define CONCERT\_A 220.0
168. #define SAMPLES\_PER\_SEC 44100
170. std::vector<int16\_t> makeSamplesFromString(GuitarString gs) {
171. std::vector<int16\_t> samples;
173. gs.pluck();
174. **int** duration = 8;  // seconds
175. **int** i;
176. **for** (i= 0; i < SAMPLES\_PER\_SEC \* duration; i++) {
177. gs.tic();
178. samples.push\_back(gs.sample());
179. }
181. **return** samples;
182. }
184. **int** main() {
185. sf::RenderWindow window(sf::VideoMode(450, 100), "SFML Guitar Hero Lite");
186. sf::Event event;
188. **double** freq;
189. // keyboard layout
190. std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
191. std::string str;
192. std::stringstream ss;
193. std::vector<std::string> frequency;
195. std::vector<std::vector<int16\_t> > samples(37);
196. std::vector<sf::Sound> sounds(37);
197. std::vector<sf::SoundBuffer> soundBuffers(37);
199. // calculate and store frequency to the sound vector
200. **for** (**int** i = 0; i < 37; i++) {
201. freq = (CONCERT\_A \* 2) \* pow(2, (i - 24) / 12.0);
203. str = "";
204. ss << ceil(freq \* 100) / 100;
205. str = ss.str();
206. str.erase(str.begin()+6, str.end());
207. str = str + " Hz";
208. frequency.push\_back(str);
209. ss.clear();
211. GuitarString gs(freq);
212. samples[i] = makeSamplesFromString(gs);
214. **if** (!soundBuffers[i].loadFromSamples(&samples[i][0], samples[i].size(),
215. 2, SAMPLES\_PER\_SEC))
216. **throw** std::runtime\_error("sf::SoundBuffer: failed to load from samples.");
218. sounds[i].setBuffer(soundBuffers[i]);
219. }
221. sf::Font font;
222. font.loadFromFile("cour.ttf");
223. sf::Text text1("Guitar Frequency:", font, 24);
224. sf::Text text("", font, 24);
225. text.setStyle(sf::Text::Bold);
226. text.setPosition(240, 2);
228. **while** (window.isOpen()) {
229. **while** (window.pollEvent(event)) {
230. **switch** (event.type) {
231. **case** sf::Event::Closed:
232. window.close();
233. **break**;
235. **case** sf::Event::KeyPressed:
236. **if** (event.key.code == sf::Keyboard::Escape)
237. window.close();
238. **break**;
240. **case** sf::Event::TextEntered:
241. **if** (event.text.unicode < 128) {
242. std::string keypress;
243. keypress += **static\_cast**<**char**>(event.text.unicode);
244. std::cout << "Playing sound associated with " << keypress;
245. std::cout << std::endl;
246. **int** index = keyboard.find(keypress);
247. sounds[index].play();
248. std::cout << "frequency: " << frequency[index] << std::endl;
249. text.setString(frequency[index]);
250. window.draw(text);
251. }
252. **break**;
253. **default**:
254. **break**;
255. }
257. window.clear();
258. window.draw(text1);
259. window.draw(text);
260. window.display();
261. }
262. }
264. **return** 0;
265. }
266. // Copyright Chuong Vu
267. // Filename: GuitarString.hpp
269. #ifndef \_GUITARSSTRING\_HPP
270. #define \_GUITARSSTRING\_HPP
272. #include <stdint.h>
273. #include <vector>
274. #include "RingBuffer.hpp"
276. **class** GuitarString {
277. **private**:
278. // create ringbuffer
279. RingBuffer buffer;
281. // for counting
282. **int** count;
284. **public**:
285. // create a guitar string of the given frequency
286. // using a sampling rate of 44,100
287. **explicit** GuitarString(**double** frequency);
289. // create a guitar string with
290. // size and initial values are given by the vector
291. **explicit** GuitarString(std::vector<int16\_t> init);
293. // pluck the guitar string by replacing the buffer
294. // with random values, representing white noise
295. **void** pluck();
297. // advance the simulation one time step
298. **void** tic();
300. // return the current sample
301. int16\_t sample() **const**;
303. // return number of times tic was called so far
304. **int** time() **const**;
305. };

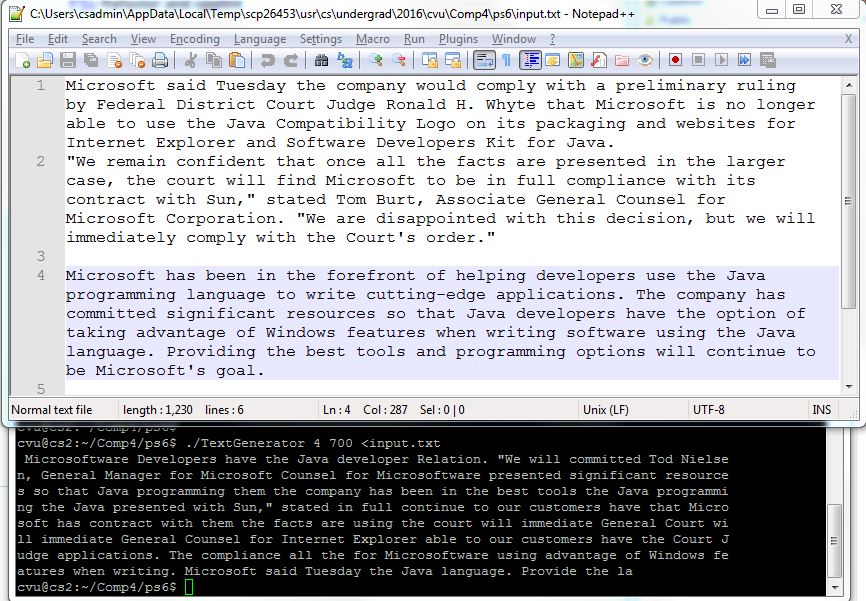
308. #endif
309. // Copyright Chuong Vu
310. // Filename: GuitarString.cpp
311. #include <iostream>
312. #include <stdint.h>  //NOLINT
313. #include <vector>
314. #include <cmath>    // for ceil function which use for rounding integer number
315. #include <cstdlib>
316. #include "GuitarString.hpp"
317. #include "RingBuffer.hpp"
319. // create a guitar string of the given frequency
320. // using a sampling rate of 44,100
321. GuitarString::GuitarString(**double** frequency) : buffer(ceil(44100/frequency)) {
322. // std::cout << ceil(44100/frequency) << std::endl;
323. count = 0;
324. }
326. // create a guitar string with
327. // size and initial values are given by the vector
328. GuitarString::GuitarString(std::vector<int16\_t> init) : buffer(init.size()) {
329. **for** (**size\_t** i = 0; i < init.size(); i++) {
330. buffer.enqueue(init[i]);
331. }
332. count = 0;
333. }
335. // pluck the guitar string by replacing the buffer
336. // with random values, representing white noise
337. **void** GuitarString::pluck() {
338. // epmpy the buffer for replacing value
339. buffer.emptyBuffer();
341. // check if the buffer is not full for make sure before enqueue
342. **while** (!buffer.isFull()) {
343. buffer.enqueue((int16\_t)(rand()) & 0xFFFF);  //NOLINT
344. }
345. }
347. // advance the simulation one time step
348. **void** GuitarString::tic() {
349. **int** n1 = buffer.dequeue();
350. **int** n2 = buffer.peek();
351. // std::cout << n2 << std::endl;
352. buffer.enqueue(0.5 \* 0.996 \* (n1 + n2));
353. count++;
354. }
356. // return the current sample
357. int16\_t GuitarString::sample() **const** {
358. **return** buffer.peek();
359. }
361. // return number of times tic was called so far
362. **int** GuitarString::time() **const** {
363. **return** count;
364. }

**PS6** Markov Model of Natural Language

This assignment was analyzing an input text for transitions between k-grams. I had to read everything on Princeton to get some idea about it and also with in class exercise so I can understand clearly how Markov chain works.

Again, I use vector to keep track my string and k-grams. I mostly based on the source code that Prof. Rykalov showed in class and also watched the video from last semester to get example header code and how to implement the code. The key was based on the k-gram and letter T characters for simulating the trajectory through the corresponding Markov chain. The code was pretty much simple and easy, the only hard part is push the string to alphabet vector and then using the k-gram key to analysis and push it to the Kgram vector which used later on for calculate the next character in the sequence and the percent that next character will be.

From this assignment, I learned that I can use the Markov Model to generate new Paragraphs with randomly from the current one. This was a fun assignment to play around with English.

Output for PS6: 

1. #
2. # Makefile for ps6
3. #
5. all:
6. make TextGenerator
8. TextGenerator: MarkovModel.o TextGenerator.o
9. g++ -o TextGenerator TextGenerator.o MarkovModel.o -ansi -pedantic -Wall -Werror
11. TextGenerator.o: MarkovModel.hpp TextGenerator.cpp
12. g++ -c -g TextGenerator.cpp -ansi -pedantic -Wall -Werror
14. MarkovModel.o: MarkovModel.hpp MarkovModel.cpp
15. g++ -c -g MarkovModel.cpp -ansi -pedantic -Wall -Werror
17. clean:
18. rm -f \*.o \*~ TextGenerator
20. run:
21. ./TextGenerator 2 700 <input.txt
22. //
23. //  main.cpp
24. //  ps6
25. //
26. //  Created by ChuongVu on 11/12/16.
27. //  Copyright © 2016 ChuongVu. All rights reserved.
28. //
30. #include <iostream>
31. #include <string>
32. #include <sstream>
33. #include <cstdlib>
34. #include "MarkovModel.hpp"
36. **int** main(**int** argc, **const** **char** \* argv[]) {
37. **int** i, k, T;
38. **if** (argc < 3) {
39. std::cout << "Missing command line args" << std::endl;
40. **return** 1;
41. } **else** **if** (argc > 3) {
42. std::cout << "Too many command line args" << std::endl;
43. **return** 1;
44. }
46. // Gather User Input
47. k = atoi(argv[1]);
48. T = atoi(argv[2]);
49. std::string input;
50. std::string current;
51. **while** (std::cin >> current) {
52. input += " " + current;
53. current = "";
54. }
56. // Create the MarkovModel
57. MarkovModel model(input, k);
59. // Generate a new string
60. std::string temp;
61. **for** (i = 0; i < k; i++) {
62. temp.push\_back(input[i]);
63. }
65. std::cout << model.gen(temp, T) << std::endl;
67. **return** 0;
68. }
69. //
70. //  MarkovModel.hpp
71. //  ps6
72. //
73. //  Created by ChuongVu on 11/12/16.
74. //  Copyright © 2016 ChuongVu. All rights reserved.
75. //  Xcode
76. /\*
77. \* MarkovModel.hpp
78. \* Copyright Fred Martin, fredm@cs.uml.edu
79. \* Tue Apr  7 21:54:53 2015
80. \*/
82. #ifndef MarkovModel\_hpp
83. #define MarkovModel\_hpp
85. #include <string>
86. #include <map>
88. **using** std::string;
90. **class** MarkovModel {
91. **public**:
92. MarkovModel(string text, **int** k);
93. ~MarkovModel();
94. **int** order();
95. **int** freq(string kgram);
96. **int** freq(string kgram, **char** c);
97. **char** randk(string kgram);
98. string gen(string kgram, **int** T);
100. **friend** std::ostream& operator<< (std::ostream &out, MarkovModel &mm);
102. **private**:
103. **int** \_order;
104. std::map <string, **int**> \_kgrams;  // must #include <map>
105. string \_alphabet;
106. string \_init;
107. };

110. #endif /\* MarkovModel\_hpp \*/
111. //
112. //  MarkovModel.cpp
113. //  ps6
114. //
115. //  Created by ChuongVu on 11/12/16.
116. //  Copyright © 2016 ChuongVu. All rights reserved.
117. //
119. #include <iostream>
120. #include <string>
121. #include <vector>
122. #include <map>
123. #include <stdexcept>
124. #include <cstdlib>
125. #include <ctime>
126. #include "MarkovModel.hpp"
128. **using** std::string;
130. // create a Markov model of order k from given text
131. // Assume that text has length at least k.
132. MarkovModel::MarkovModel(string text, **int** k) : \_order(k), \_init(text) {
133. **int** i, j;
134. srand(**static\_cast**<unsigned **int**>(time(NULL)));
136. // Collect characters for a list of the alphabet
137. **for** (i = 0; i < **static\_cast**<**int**>(text.size()); i++) {
138. **if** (string::npos == \_alphabet.find(text[i])) {
139. \_alphabet.push\_back(text[i]);
140. }
141. }
143. // Create a map of all the kgrams
144. **for** (i = 0; i < **static\_cast**<**int**>(text.size()); i++) {
145. string temp1;
146. string temp2;
148. // Build up a string of k consecutive characters
149. **for** (j = i; j < i + k; j++) {
150. **if** (j >= **static\_cast**<**int**>(text.size())) {
151. temp1.push\_back(text[j - text.size()]);
152. } **else** {
153. temp1.push\_back(text[j]);
154. }
155. }
156. // Put the occurance of the string into the kgram map
157. **if** (\_kgrams.end() == \_kgrams.find(temp1)) {
158. \_kgrams[temp1] = 1;
159. } **else** {
160. \_kgrams[temp1] += 1;
161. }
163. // Put in all possible k+1 strings (regardless of existence)
164. **for** (j = 0; j < **static\_cast**<**int**>(\_alphabet.size()); j++) {
165. **if** (\_kgrams.end() == \_kgrams.find(temp1 + \_alphabet[j])) {
166. \_kgrams[temp1 + \_alphabet[j]] = 0;
167. }
168. }
170. // Build up a string of k+1 consecutive characters
171. **for** (j = i; j < i + k + 1; j++) {
172. **if** (j >= **static\_cast**<**int**>(text.size())) {
173. temp2.push\_back(text[j - text.size()]);
174. } **else** {
175. temp2.push\_back(text[j]);
176. }
177. }
178. // Put the occurance of the string into the kgram map
179. \_kgrams[temp2] += 1;
180. }
181. }
183. MarkovModel::~MarkovModel() { }
185. // order k of Markov model
186. **int** MarkovModel::order() {
187. **return** \_order;
188. }
190. // number of occurrences of kgram in text
191. // (throw an exception if kgram is not of length k)
192. **int** MarkovModel::freq(string kgram) {
193. **if** (**static\_cast**<**int**>(kgram.size()) != \_order) {
194. **throw** std::runtime\_error("kgram is not size k");
195. }
197. **if** (\_order == 0) {
198. **return** **static\_cast**<**int**>(\_init.size());
199. } **else** {
200. **return** \_kgrams[kgram];
201. }
202. }
204. // number of times that character c follows kgram
205. // if order=0, return num of times char c appears
206. // (throw an exception if kgram is not of length k)
207. **int** MarkovModel::freq(string kgram, **char** c) {
208. **int** i;
210. **if** (**static\_cast**<**int**>(kgram.size()) != \_order) {
211. **throw** std::runtime\_error("kgram is not size k");
212. }
214. **if** (\_order ==0) {
215. **int** count = 0;
216. **for** (i = 0; i < **static\_cast**<**int**>(\_init.size()); i++) {
217. **if** (\_init[i] == c) {
218. count++;
219. }
220. }
221. **return** count;
222. } **else** {
223. **return**  \_kgrams[kgram + c];
224. }
226. **return** 0;
227. }
229. // random character following given kgram
230. // (Throw an exception if kgram is not of length k.
231. //  Throw an exception if no such kgram.)
232. **char** MarkovModel::randk(string kgram) {
233. **if** (**static\_cast**<**int**>(kgram.size()) != \_order ||
234. \_kgrams.end() == \_kgrams.find(kgram)) {
235. **throw** std::runtime\_error("Error: randk kgram not valid");
236. }
238. **int** i, j;
239. string temp;
241. **for** (i = 0; i < **static\_cast**<**int**>(\_alphabet.size()); i++) {
242. **for** (j = 0; j < \_kgrams[kgram + \_alphabet[i]]; j++) {
243. temp.push\_back(\_alphabet[i]);
244. }
245. }
247. **return** temp[rand() % temp.size()];
248. }
250. // generate a string of length T characters
251. // by simulating a trajectory through the corresponding
252. // Markov chain.  The first k characters of the newly
253. // generated string should be the argument kgram.
254. // Throw an exception if kgram is not of length k.
255. // Assume that T is at least k.
256. string MarkovModel::gen(string kgram, **int** T) {
257. string temp = kgram;
258. string rValue = kgram;
259. **char** c;
260. **int** i;
262. **for** (i = 0; i < T - \_order; i++) {
263. c = randk(temp);
264. rValue.push\_back(c);
265. temp.erase(temp.begin());
266. temp.push\_back(c);
267. }
269. **return** rValue;
270. }
272. // overload the stream insertion operator and display
273. // the internal state of the Markov Model. Print out
274. // the order, the alphabet, and the frequencies of
275. // the k-grams and k+1-grams.
276. std::ostream& operator<<(std::ostream& os, MarkovModel& mm) {
277. **int** i;
278. os << "\n" << "Original: \"" << mm.\_init << "\"" << std::endl;
279. os << "Order: " << mm.\_order << std::endl;
280. os << "Alphabet: \"" << mm.\_alphabet << "\"" << "\n" << std::endl;
282. os << "MarkovModel Map" << std::endl;
283. std::map <string, **int**> temp = mm.\_kgrams;
284. **for** (std::map<string, **int**>::iterator ma = temp.begin();
285. ma != temp.end(); ++ma) {
286. os << ma->first << " " << ma->second << " => ";
287. **for** (i = 0; i < **static\_cast**<**int**>(mm.\_alphabet.size()); i++) {
288. ma++;
289. os << ma->first << " " << ma->second << " ";
290. }
291. os << std::endl;
292. }
293. **return** os;
294. }

**PS7 Kronos Intouch Parsing**

This assignment for me was using regex to analysis data. This was the last assignment also was the most import assignment for me since my academic plan is go for Database. I had worked at Teradyne and Autoliv and my job was analysis data but without regex. So this time was the first time I use regex to search for a string that I want to get from the log file. This assignment has two parts, A and B.

For the part A, to be able to do this one, I need to understand how regex work. I need to know how to write a correct regular expression before I can actually write a code. I have used regex101.com as a reference to practice regex. Once I understand how to make a correct regex then the code was very easy to me. I just need to check line by line based on regex and then print it out based on the matched groups. I think the hardest one for this assignment was need a good logic for both part A and B so when the code run, each line will be only read on time and not need read again.

After this assignment, I was able to understand how to use regex and applied the regex to my last assignment GUI I which using JavaScript and jQuery. And I believe this regex will help me later on in the future for data analysis.

1. # Makefile for ps7a
2. all:
3. make ps7a
5. ps7a: ps7a.cpp
6. g++ -o ps7a ps7a.cpp -ansi -pedantic -Wall -Werror -lboost\_regex -lboost\_date\_time
8. clean:
9. rm -f ps7a \*~
10. // Copyright Chuong Vu
11. // Filename: ps7a.cpp
13. #include <boost/regex.hpp>  // for regex\_match library
14. #include <boost/date\_time/gregorian/gregorian.hpp>
15. #include <boost/date\_time/posix\_time/posix\_time.hpp>
16. #include <iostream>
17. #include <fstream>  // for read input file
18. #include <string>
19. #include <vector>
21. **using** std::cout;
22. **using** std::cin;
23. **using** std::endl;
24. **using** std::string;
25. **using** std::ifstream;
26. **using** std::ofstream;
28. **using** boost::regex;
29. **using** boost::gregorian::date;
30. **using** boost::gregorian::from\_simple\_string;  // convert string to date library
31. **using** boost::gregorian::date\_period;
32. **using** boost::gregorian::date\_duration;
33. **using** boost::posix\_time::ptime;
34. **using** boost::posix\_time::time\_duration;
36. // The function based on Prof. Victor Grinberg lecture capture
37. // instantiate the template for any type has the str() member
38. // function that returns an std::string
39. **template** <**typename** T>
40. **int** to\_int(**const** T& sm) {
41. **return** atoi(sm.str().c\_str());
42. }
44. **int** main(**int** argc, **char**\* argv[]) {
45. string line;
46. **int** line\_order = 1;
47. **bool** boot = **false**;
48. // Posix Time
49. ptime t1, t2;
50. std::vector<string> checklist;
51. string outputName;
52. ofstream outputFile;
53. // Construct regular expression
54. // YYYY -MM -DD format
55. string sdate("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2}) ");
56. // Time format
57. string stime("([0-9]{2}):([0-9]{2}):([0-9]{2})");
58. // boot start
59. string sboot("(.\*log.c.166.\*)");
60. // boot start success
61. string sdone("(.\*oejs.AbstractConnector:Started SelectChannelConnector.\*)");
63. // Check # of Command Line Args
64. **if** (argc != 2) {
65. **throw** std::runtime\_error("Usage: enter one log file");
66. }
68. // Open Device Log, end if it can't be opened.
69. ifstream inputFile(argv[1], ifstream::in);
70. **if** (!inputFile.is\_open()) {
71. **throw** std::runtime\_error("Cannot open file");
72. }
74. outputName = string(argv[1]) + ".rpt";
75. outputFile.open(outputName.c\_str());
77. // Regular expression for regex use to compare
78. regex e\_begin\_boot(sdate + stime + sboot);
79. regex e\_end\_boot(sdate + stime + sdone);
81. // hold matches values
82. boost::smatch m;
84. **while** (getline(inputFile, line)) {
85. **if** (regex\_match(line, m, e\_begin\_boot)) {
86. date d(from\_simple\_string(m[0]));
87. ptime temp(d, time\_duration(to\_int(m[4]),
88. to\_int(m[5]), to\_int(m[6])));
89. t1 = temp;
90. **if** (boot) {
91. outputFile << "\*\*\*\* Incomplete boot \*\*\*\*\n" << std::endl;
92. boot = **false**;
93. }
94. outputFile << "=== Device boot ===" << endl
95. << line\_order << "(" << argv[1] << "): "
96. << m[1] << "-" << m[2] << "-" << m[3] << " "
97. << m[4] << ":" << m[5] << ":" << m[6] << " "
98. << "Boot Start" << endl;
99. boot = **true**;
100. } **else** **if** (regex\_match(line, m, e\_end\_boot)) {
101. **if** (boot) {
102. date d(from\_simple\_string(m[0]));
103. ptime temp(d, time\_duration(to\_int(m[4]),
104. to\_int(m[5]), to\_int(m[6])));
105. t2 = temp;
106. time\_duration td = t2 - t1;
107. outputFile << line\_order << "(" << argv[1] << "): "
108. << m[1] << "-" << m[2] << "-" << m[3] << " "
109. << m[4] << ":" << m[5] << ":" << m[6] << " "
110. << "Boot Completed\n"
111. << "    " << "Boot Time: "
112. << td.total\_milliseconds() << "ms \n" << endl;
113. boot = **false**;
114. } **else** {
115. outputFile << "\*\*\*\* Incomplete boot \*\*\*\* \n" << endl;
116. }
117. }
118. line\_order++;
119. }
120. **return** 0;
121. }

Output for ps7a:

1. // device1\_intouch.log.rpt
3. === Device boot ===
4. 435369(device1\_intouch.log): 2014-03-25 19:11:59 Boot Start
5. 435759(device1\_intouch.log): 2014-03-25 19:15:02 Boot Completed
6. Boot Time: 183000ms
8. === Device boot ===
9. 436500(device1\_intouch.log): 2014-03-25 19:29:59 Boot Start
10. 436859(device1\_intouch.log): 2014-03-25 19:32:44 Boot Completed
11. Boot Time: 165000ms
13. === Device boot ===
14. 440719(device1\_intouch.log): 2014-03-25 22:01:46 Boot Start
15. 440791(device1\_intouch.log): 2014-03-25 22:04:27 Boot Completed
16. Boot Time: 161000ms
18. === Device boot ===
19. 440866(device1\_intouch.log): 2014-03-26 12:47:42 Boot Start
20. 441216(device1\_intouch.log): 2014-03-26 12:50:29 Boot Completed
21. Boot Time: 167000ms
23. === Device boot ===
24. 442094(device1\_intouch.log): 2014-03-26 20:41:34 Boot Start
25. 442432(device1\_intouch.log): 2014-03-26 20:44:13 Boot Completed
26. Boot Time: 159000ms
28. === Device boot ===
29. 443073(device1\_intouch.log): 2014-03-27 14:09:01 Boot Start
30. 443411(device1\_intouch.log): 2014-03-27 14:11:42 Boot Completed
31. Boot Time: 161000ms
32. // device2\_intouch.log.rpt
34. === Device boot ===
35. 498921(device2\_intouch.log): 2014-03-11 15:42:26 Boot Start
36. 499030(device2\_intouch.log): 2014-03-11 15:45:08 Boot Completed
37. Boot Time: 162000ms
38. // device3\_intouch.log.rpt
40. === Device boot ===
41. 31063(device3\_intouch.log): 2014-01-26 09:55:07 Boot Start
42. 31176(device3\_intouch.log): 2014-01-26 09:58:04 Boot Completed
43. Boot Time: 177000ms
45. === Device boot ===
46. 31274(device3\_intouch.log): 2014-01-26 12:15:18 Boot Start
47. \*\*\*\* Incomplete boot \*\*\*\*
49. === Device boot ===
50. 31293(device3\_intouch.log): 2014-01-26 14:02:39 Boot Start
51. 31401(device3\_intouch.log): 2014-01-26 14:05:24 Boot Completed
52. Boot Time: 165000ms
54. === Device boot ===
55. 32623(device3\_intouch.log): 2014-01-27 12:27:55 Boot Start
56. \*\*\*\* Incomplete boot \*\*\*\*
58. === Device boot ===
59. 32641(device3\_intouch.log): 2014-01-27 12:30:23 Boot Start
60. \*\*\*\* Incomplete boot \*\*\*\*
62. === Device boot ===
63. 32656(device3\_intouch.log): 2014-01-27 12:32:51 Boot Start
64. \*\*\*\* Incomplete boot \*\*\*\*
66. === Device boot ===
67. 32674(device3\_intouch.log): 2014-01-27 12:35:19 Boot Start
68. \*\*\*\* Incomplete boot \*\*\*\*
70. === Device boot ===
71. 32693(device3\_intouch.log): 2014-01-27 14:02:38 Boot Start
72. 32801(device3\_intouch.log): 2014-01-27 14:05:21 Boot Completed
73. Boot Time: 163000ms
75. === Device boot ===
76. 33709(device3\_intouch.log): 2014-01-28 12:44:17 Boot Start
77. \*\*\*\* Incomplete boot \*\*\*\*
79. === Device boot ===
80. 33725(device3\_intouch.log): 2014-01-28 14:02:33 Boot Start
81. 33833(device3\_intouch.log): 2014-01-28 14:05:15 Boot Completed
82. Boot Time: 162000ms
84. === Device boot ===
85. 34594(device3\_intouch.log): 2014-01-29 12:43:07 Boot Start
86. \*\*\*\* Incomplete boot \*\*\*\*
88. === Device boot ===
89. 34613(device3\_intouch.log): 2014-01-29 14:02:35 Boot Start
90. 34721(device3\_intouch.log): 2014-01-29 14:05:19 Boot Completed
91. Boot Time: 164000ms
93. === Device boot ===
94. 37428(device3\_intouch.log): 2014-01-30 12:43:05 Boot Start
95. \*\*\*\* Incomplete boot \*\*\*\*
97. === Device boot ===
98. 37447(device3\_intouch.log): 2014-01-30 14:02:40 Boot Start
99. 37555(device3\_intouch.log): 2014-01-30 14:05:22 Boot Completed
100. Boot Time: 162000ms
102. === Device boot ===
103. 38258(device3\_intouch.log): 2014-01-31 14:02:33 Boot Start
104. 38366(device3\_intouch.log): 2014-01-31 14:05:16 Boot Completed
105. Boot Time: 163000ms
107. === Device boot ===
108. 39150(device3\_intouch.log): 2014-02-01 12:39:38 Boot Start
109. \*\*\*\* Incomplete boot \*\*\*\*
111. === Device boot ===
112. 39166(device3\_intouch.log): 2014-02-01 12:42:07 Boot Start
113. \*\*\*\* Incomplete boot \*\*\*\*
115. === Device boot ===
116. 39182(device3\_intouch.log): 2014-02-01 14:02:32 Boot Start
117. 39290(device3\_intouch.log): 2014-02-01 14:05:16 Boot Completed
118. Boot Time: 164000ms
120. === Device boot ===
121. 40288(device3\_intouch.log): 2014-02-02 14:02:39 Boot Start
122. 40397(device3\_intouch.log): 2014-02-02 14:05:31 Boot Completed
123. Boot Time: 172000ms
125. === Device boot ===
126. 41615(device3\_intouch.log): 2014-02-03 12:35:55 Boot Start
127. \*\*\*\* Incomplete boot \*\*\*\*
129. === Device boot ===
130. 41633(device3\_intouch.log): 2014-02-03 12:38:22 Boot Start
131. \*\*\*\* Incomplete boot \*\*\*\*
133. === Device boot ===
134. 41648(device3\_intouch.log): 2014-02-03 12:40:48 Boot Start
135. \*\*\*\* Incomplete boot \*\*\*\*
137. === Device boot ===
138. 41666(device3\_intouch.log): 2014-02-03 12:43:17 Boot Start
139. \*\*\*\* Incomplete boot \*\*\*\*
141. === Device boot ===
142. 41684(device3\_intouch.log): 2014-02-03 12:45:46 Boot Start
143. \*\*\*\* Incomplete boot \*\*\*\*
145. === Device boot ===
146. 41694(device3\_intouch.log): 2014-02-03 14:02:34 Boot Start
147. 41802(device3\_intouch.log): 2014-02-03 14:05:18 Boot Completed
148. Boot Time: 164000ms
149. // device4\_intouch.log.rpt
151. === Device boot ===
152. 4(device4\_intouch.log): 2013-10-02 18:42:38 Boot Start
153. 112(device4\_intouch.log): 2013-10-02 18:45:23 Boot Completed
154. Boot Time: 165000ms
156. === Device boot ===
157. 747(device4\_intouch.log): 2013-10-03 12:23:21 Boot Start
158. 855(device4\_intouch.log): 2013-10-03 12:26:15 Boot Completed
159. Boot Time: 174000ms
161. === Device boot ===
162. 1459(device4\_intouch.log): 2013-10-04 16:20:03 Boot Start
163. 1568(device4\_intouch.log): 2013-10-04 16:23:06 Boot Completed
164. Boot Time: 183000ms
166. === Device boot ===
167. 31848(device4\_intouch.log): 2013-12-03 16:21:13 Boot Start
168. 31956(device4\_intouch.log): 2013-12-03 16:24:08 Boot Completed
169. Boot Time: 175000ms
171. === Device boot ===
172. 32789(device4\_intouch.log): 2013-12-04 21:50:27 Boot Start
173. 33032(device4\_intouch.log): 2013-12-04 21:52:57 Boot Completed
174. Boot Time: 150000ms
176. === Device boot ===
177. 33145(device4\_intouch.log): 2013-12-04 21:58:45 Boot Start
178. 33390(device4\_intouch.log): 2013-12-04 22:01:14 Boot Completed
179. Boot Time: 149000ms
181. === Device boot ===
182. 33677(device4\_intouch.log): 2013-12-04 22:21:03 Boot Start
183. 33920(device4\_intouch.log): 2013-12-04 22:23:31 Boot Completed
184. Boot Time: 148000ms
186. === Device boot ===
187. 45295(device4\_intouch.log): 2013-12-05 13:34:25 Boot Start
188. 45538(device4\_intouch.log): 2013-12-05 13:36:55 Boot Completed
189. Boot Time: 150000ms
191. === Device boot ===
192. 45615(device4\_intouch.log): 2013-12-05 14:12:25 Boot Start
193. 45858(device4\_intouch.log): 2013-12-05 14:14:53 Boot Completed
194. Boot Time: 148000ms
196. === Device boot ===
197. 46117(device4\_intouch.log): 2013-12-05 15:39:02 Boot Start
198. 46353(device4\_intouch.log): 2013-12-05 15:41:29 Boot Completed
199. Boot Time: 147000ms
201. === Device boot ===
202. 46357(device4\_intouch.log): 2013-12-05 20:20:24 Boot Start
203. 46600(device4\_intouch.log): 2013-12-05 20:22:54 Boot Completed
204. Boot Time: 150000ms
206. === Device boot ===
207. 46792(device4\_intouch.log): 2013-12-10 13:20:43 Boot Start
208. 47035(device4\_intouch.log): 2013-12-10 13:23:12 Boot Completed
209. Boot Time: 149000ms
211. === Device boot ===
212. 47700(device4\_intouch.log): 2013-12-10 19:40:58 Boot Start
213. 47991(device4\_intouch.log): 2013-12-10 19:43:55 Boot Completed
214. Boot Time: 177000ms
216. === Device boot ===
217. 48100(device4\_intouch.log): 2013-12-11 14:09:11 Boot Start
218. 48341(device4\_intouch.log): 2013-12-11 14:11:41 Boot Completed
219. Boot Time: 150000ms
221. === Device boot ===
222. 48345(device4\_intouch.log): 2013-12-11 14:17:49 Boot Start
223. 48636(device4\_intouch.log): 2013-12-11 14:20:46 Boot Completed
224. Boot Time: 177000ms
225. // device5\_intouch.log.rpt
227. === Device boot ===
228. 31063(device5\_intouch.log): 2014-01-26 09:55:07 Boot Start
229. 31176(device5\_intouch.log): 2014-01-26 09:58:04 Boot Completed
230. Boot Time: 177000ms
232. === Device boot ===
233. 31274(device5\_intouch.log): 2014-01-26 12:15:18 Boot Start
234. \*\*\*\* Incomplete boot \*\*\*\*
236. === Device boot ===
237. 31293(device5\_intouch.log): 2014-01-26 14:02:39 Boot Start
238. 31401(device5\_intouch.log): 2014-01-26 14:05:24 Boot Completed
239. Boot Time: 165000ms
241. === Device boot ===
242. 32623(device5\_intouch.log): 2014-01-27 12:27:55 Boot Start
243. \*\*\*\* Incomplete boot \*\*\*\*
245. === Device boot ===
246. 32641(device5\_intouch.log): 2014-01-27 12:30:23 Boot Start
247. \*\*\*\* Incomplete boot \*\*\*\*
249. === Device boot ===
250. 32656(device5\_intouch.log): 2014-01-27 12:32:51 Boot Start
251. \*\*\*\* Incomplete boot \*\*\*\*
253. === Device boot ===
254. 32674(device5\_intouch.log): 2014-01-27 12:35:19 Boot Start
255. \*\*\*\* Incomplete boot \*\*\*\*
257. === Device boot ===
258. 32693(device5\_intouch.log): 2014-01-27 14:02:38 Boot Start
259. 32801(device5\_intouch.log): 2014-01-27 14:05:21 Boot Completed
260. Boot Time: 163000ms
262. === Device boot ===
263. 33709(device5\_intouch.log): 2014-01-28 12:44:17 Boot Start
264. \*\*\*\* Incomplete boot \*\*\*\*
266. === Device boot ===
267. 33725(device5\_intouch.log): 2014-01-28 14:02:33 Boot Start
268. 33833(device5\_intouch.log): 2014-01-28 14:05:15 Boot Completed
269. Boot Time: 162000ms
271. === Device boot ===
272. 34594(device5\_intouch.log): 2014-01-29 12:43:07 Boot Start
273. \*\*\*\* Incomplete boot \*\*\*\*
275. === Device boot ===
276. 34613(device5\_intouch.log): 2014-01-29 14:02:35 Boot Start
277. 34721(device5\_intouch.log): 2014-01-29 14:05:19 Boot Completed
278. Boot Time: 164000ms
280. === Device boot ===
281. 37428(device5\_intouch.log): 2014-01-30 12:43:05 Boot Start
282. \*\*\*\* Incomplete boot \*\*\*\*
284. === Device boot ===
285. 37447(device5\_intouch.log): 2014-01-30 14:02:40 Boot Start
286. 37555(device5\_intouch.log): 2014-01-30 14:05:22 Boot Completed
287. Boot Time: 162000ms
289. === Device boot ===
290. 38258(device5\_intouch.log): 2014-01-31 14:02:33 Boot Start
291. 38366(device5\_intouch.log): 2014-01-31 14:05:16 Boot Completed
292. Boot Time: 163000ms
294. === Device boot ===
295. 39150(device5\_intouch.log): 2014-02-01 12:39:38 Boot Start
296. \*\*\*\* Incomplete boot \*\*\*\*
298. === Device boot ===
299. 39166(device5\_intouch.log): 2014-02-01 12:42:07 Boot Start
300. \*\*\*\* Incomplete boot \*\*\*\*
302. === Device boot ===
303. 39182(device5\_intouch.log): 2014-02-01 14:02:32 Boot Start
304. 39290(device5\_intouch.log): 2014-02-01 14:05:16 Boot Completed
305. Boot Time: 164000ms
307. === Device boot ===
308. 40288(device5\_intouch.log): 2014-02-02 14:02:39 Boot Start
309. 40397(device5\_intouch.log): 2014-02-02 14:05:31 Boot Completed
310. Boot Time: 172000ms
312. === Device boot ===
313. 41615(device5\_intouch.log): 2014-02-03 12:35:55 Boot Start
314. \*\*\*\* Incomplete boot \*\*\*\*
316. === Device boot ===
317. 41633(device5\_intouch.log): 2014-02-03 12:38:22 Boot Start
318. \*\*\*\* Incomplete boot \*\*\*\*
320. === Device boot ===
321. 41648(device5\_intouch.log): 2014-02-03 12:40:48 Boot Start
322. \*\*\*\* Incomplete boot \*\*\*\*
324. === Device boot ===
325. 41666(device5\_intouch.log): 2014-02-03 12:43:17 Boot Start
326. \*\*\*\* Incomplete boot \*\*\*\*
328. === Device boot ===
329. 41684(device5\_intouch.log): 2014-02-03 12:45:46 Boot Start
330. \*\*\*\* Incomplete boot \*\*\*\*
332. === Device boot ===
333. 41694(device5\_intouch.log): 2014-02-03 14:02:34 Boot Start
334. 41802(device5\_intouch.log): 2014-02-03 14:05:18 Boot Completed
335. Boot Time: 164000ms
336. // device6\_intouch.log.rpt
338. === Device boot ===
339. 2(device6\_intouch.log): 2014-04-03 20:27:48 Boot Start
340. 161(device6\_intouch.log): 2014-04-03 20:31:01 Boot Completed
341. Boot Time: 193000ms
343. === Device boot ===
344. 82079(device6\_intouch.log): 2014-04-09 14:51:15 Boot Start
345. 82303(device6\_intouch.log): 2014-04-09 14:54:39 Boot Completed
346. Boot Time: 204000ms
348. === Device boot ===
349. 85398(device6\_intouch.log): 2014-04-10 18:13:13 Boot Start
350. 85564(device6\_intouch.log): 2014-04-10 18:16:37 Boot Completed
351. Boot Time: 204000ms
353. === Device boot ===
354. 85957(device6\_intouch.log): 2014-04-10 19:11:05 Boot Start
355. 86123(device6\_intouch.log): 2014-04-10 19:14:24 Boot Completed
356. Boot Time: 199000ms
358. === Device boot ===
359. 86127(device6\_intouch.log): 2014-04-10 19:18:36 Boot Start
360. 86293(device6\_intouch.log): 2014-04-10 19:21:56 Boot Completed
361. Boot Time: 200000ms
363. === Device boot ===
364. 86568(device6\_intouch.log): 2014-04-10 19:32:16 Boot Start
365. 86732(device6\_intouch.log): 2014-04-10 19:35:36 Boot Completed
366. Boot Time: 200000ms
368. === Device boot ===
369. 86750(device6\_intouch.log): 2014-04-10 20:06:27 Boot Start
370. 86821(device6\_intouch.log): 2014-04-10 20:09:07 Boot Completed
371. Boot Time: 160000ms
373. === Device boot ===
374. 86939(device6\_intouch.log): 2014-04-11 00:15:56 Boot Start
375. 87111(device6\_intouch.log): 2014-04-11 00:18:49 Boot Completed
376. Boot Time: 173000ms
378. === Device boot ===
379. 87116(device6\_intouch.log): 2014-04-11 13:28:25 Boot Start
380. 87286(device6\_intouch.log): 2014-04-11 13:31:12 Boot Completed
381. Boot Time: 167000ms
383. === Device boot ===
384. 87836(device6\_intouch.log): 2014-04-11 13:58:02 Boot Start
385. 88009(device6\_intouch.log): 2014-04-11 14:00:49 Boot Completed
386. Boot Time: 167000ms
388. === Device boot ===
389. 88983(device6\_intouch.log): 2014-04-11 14:23:42 Boot Start
390. 89155(device6\_intouch.log): 2014-04-11 14:26:31 Boot Completed
391. Boot Time: 169000ms
393. === Device boot ===
394. 90112(device6\_intouch.log): 2014-04-14 12:13:59 Boot Start
395. \*\*\*\* Incomplete boot \*\*\*\*
397. === Device boot ===
398. 90135(device6\_intouch.log): 2014-04-14 12:16:13 Boot Start
399. \*\*\*\* Incomplete boot \*\*\*\*
401. === Device boot ===
402. 90176(device6\_intouch.log): 2014-04-14 12:18:44 Boot Start
403. 90307(device6\_intouch.log): 2014-04-14 12:21:25 Boot Completed
404. Boot Time: 161000ms

Ps7b:

1. # Makefile for ps7b
2. all:
3. make ps7b
5. ps7b: ps7b.o kronos\_parse.o
6. g++ -o ps7b ps7b.o kronos\_parse.o -lboost\_regex -lboost\_date\_time
8. ps7b.o:
9. g++ -c ps7b.cpp -ansi -pedantic -Wall -Werror
11. kronos\_parse.o: kronos\_parse.hpp kronos\_parse.cpp
12. g++ -c kronos\_parse.cpp -ansi -pedantic -Wall -Werror
14. clean:
15. rm -f \*.o \*~ ps7b
17. test:
19. g++ -o test stdin\_boost.cpp -lboost\_regex
21. r:
22. ./test
23. run:
24. ./ps7b device4\_intouch.log
25. // Copyright Chuong Vu
26. // Filename: ps7b.cpp
27. #include <iostream>
28. #include <fstream>  // for read input file
29. #include <sstream>  // Required for stringstreams
30. #include <string>
31. #include "boost/regex.hpp"  // for regex\_match library
32. #include "kronos\_parse.hpp"
33. #include "boost/date\_time/gregorian/gregorian.hpp"
34. #include "boost/date\_time/posix\_time/posix\_time.hpp"
36. **using** std::cout;
37. **using** std::cin;
38. **using** std::endl;
39. **using** std::string;
40. **using** std::ifstream;
41. **using** std::ofstream;
42. **using** std::ostringstream;
44. **using** boost::regex;
45. **using** boost::gregorian::date;
46. **using** boost::gregorian::from\_simple\_string;  // convert string to date library
47. **using** boost::gregorian::date\_period;
48. **using** boost::gregorian::date\_duration;
49. **using** boost::posix\_time::ptime;
50. **using** boost::posix\_time::time\_duration;

53. // The function based on Prof. Victor Grinberg lecture capture
54. // instantiate the template for any type has the str() member
55. // function that returns an std::string
56. **template** <**typename** T>
57. **int** to\_int(**const** T& sm) {
58. **return** atoi(sm.str().c\_str());
59. }
61. string IntToString(**int** a) {
62. ostringstream temp;
63. temp << a;
64. **return** temp.str();
65. }
67. **int** main(**int** argc, **char**\* argv[]) {
68. string line;
69. **int** line\_order = 1;
70. **bool** boot = **false**;
71. **bool** check = **false**;
72. **int** time1, time2;
73. // Posix Time
74. ptime t1, t2;
76. string time;
77. string outputName;
78. ofstream outputFile;
80. ostringstream ss;
81. ss.str("");
83. // Check # of Command Line Args
84. **if** (argc != 2) {
85. **throw** std::runtime\_error("Usage: enter one log file");
86. }
88. // Open Device Log, end if it can't be opened.
89. ifstream inputFile(argv[1], ifstream::in);
90. **if** (!inputFile.is\_open()) {
91. **throw** std::runtime\_error("Cannot open file");
92. }
93. string inputName = string(argv[1]);
94. outputName = string(argv[1]) + ".rpt";
95. outputFile.open(outputName.c\_str());
97. // Construct regular expression
98. // YYYY -MM -DD format
99. string sdate("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2}) ");
100. // Time format
101. string stime("([0-9]{2}):([0-9]{2}):([0-9]{2})");
102. // boot start
103. string sboot("(.\*log.c.166.\*)");
104. // boot start success
105. string sdone("(.\*oejs.AbstractConnector:Started SelectChannelConnector.\*)");
107. // Regular expression for regex use to compare
108. // Boot time
109. regex e\_begin\_boot(sdate + stime + sboot);
110. regex e\_end\_boot(sdate + stime + sdone);
111. //  Start and succe started
112. string sstart("(.\*Starting Service.)");
113. string ssuccess("(.\*Service started successfully.)");
114. string sruntime("\\((.\*)\\)");
116. //  1.Logging
117. string slogging("(..Logging.\*)");
118. regex e\_slog\_start(sstart + slogging);
119. regex e\_slog\_done(ssuccess + slogging + sruntime);
121. //  2.DatabaseInitialize
122. string sdatainit("(..DatabaseInitialize.\*)");
123. regex e\_datainit\_start(sstart + sdatainit);
124. regex e\_datainit\_done(ssuccess + sdatainit + sruntime);
126. //  3.MessagingService
127. string smesser("(..MessagingService.\*)");
128. regex e\_messer\_start(sstart + smesser);
129. regex e\_messer\_done(ssuccess + smesser + sruntime);
131. //  4.HealthMonitorService
132. string shealth("(..HealthMonitorService.\*)");
133. regex e\_health\_start(sstart + shealth);
134. regex e\_health\_done(ssuccess + shealth + sruntime);
136. //  5.Persistence
137. string sperser("(..Persistence.\*)");
138. regex e\_sperser\_start(sstart + sperser);
139. regex e\_sperser\_done(ssuccess + sperser + sruntime);
141. //  6.ConfigurationService
142. string sconser("(..ConfigurationService.\*)");
143. regex e\_sconser\_start(sstart + sconser);
144. regex e\_sconser\_done(ssuccess + sconser + sruntime);
146. //  7.LandingPadService
147. string slser("(..LandingPadService.\*)");
148. regex e\_slser\_start(sstart + slser);
149. regex e\_slser\_done(ssuccess + slser + sruntime);
151. //  8.PortConfigurationService
152. string sportcon("(..PortConfigurationService.\*)");
153. regex e\_sportcon\_start(sstart + sportcon);
154. regex e\_sportcon\_done(ssuccess + sportcon + sruntime);
156. //  9.CacheService
157. string scser("(..CacheService.\*)");
158. regex e\_scser\_start(sstart + scser);
159. regex e\_scser\_done(ssuccess + scser + sruntime);
161. //  10.ThemingService
162. string sthemser("(..ThemingService.\*)");
163. regex e\_sthemser\_start(sstart + sthemser);
164. regex e\_sthemser\_done(ssuccess + sthemser + sruntime);
166. //  11.StagingService
167. string sstaser("(..StagingService.\*)");
168. regex e\_sstaser\_start(sstart + sstaser);
169. regex e\_sstaser\_done(ssuccess + sstaser + sruntime);
171. //  12.DeviceIOService
172. string sdevser("(..DeviceIOService.\*)");
173. regex e\_sdevser\_start(sstart + sdevser);
174. regex e\_sdevser\_done(ssuccess + sdevser + sruntime);
176. //  13.BellService
177. string sbellser("(..BellService.\*)");
178. regex e\_sbellser\_start(sstart + sbellser);
179. regex e\_sbellser\_done(ssuccess + sbellser + sruntime);
181. //  14.GateService
182. string sgateser("(..GateService.\*)");
183. regex e\_sgateser\_start(sstart + sgateser);
184. regex e\_sgateser\_done(ssuccess + sgateser + sruntime);
186. //  15.ReaderDataService
187. string sreadser("(..ReaderDataService.\*)");
188. regex e\_sreadser\_start(sstart + sreadser);
189. regex e\_sreadser\_done(ssuccess + sreadser + sruntime);
191. //  16.BiometricService
192. string sbioser("(..BiometricService.\*)");
193. regex e\_sbioser\_start(sstart + sbioser);
194. regex e\_sbioser\_done(ssuccess + sbioser + sruntime);
196. //  17.StateManager
197. string stateser("(..StateManager.\*)");
198. regex e\_stateser\_start(sstart + stateser);
199. regex e\_stateser\_done(ssuccess + stateser + sruntime);
201. //  18.OfflineSmartviewService
202. string soffser("(..OfflineSmartviewService.\*)");
203. regex e\_soffser\_start(sstart + soffser);
204. regex e\_soffser\_done(ssuccess + soffser + sruntime);
206. //  19.AVFeedbackService
207. string savfser("(..AVFeedbackService.\*)");
208. regex e\_savfser\_start(sstart + savfser);
209. regex e\_savfser\_done(ssuccess + savfser + sruntime);
211. //  20.DatabaseThreads
212. string sdataser("(..DatabaseThreads.\*)");
213. regex e\_sdataser\_start(sstart + sdataser);
214. regex e\_sdataser\_done(ssuccess + sdataser + sruntime);
216. //  21.SoftLoadService
217. string ssoftser("(..SoftLoadService.\*)");
218. regex e\_ssoftser\_start(sstart + ssoftser);
219. regex e\_ssoftser\_done(ssuccess + ssoftser + sruntime);
221. //  22.WATCHDOG
222. string swatser("(..WATCHDOG.\*)");
223. regex e\_swatser\_start(sstart + swatser);
224. regex e\_swatser\_done(ssuccess + swatser + sruntime);
226. //  23.ProtocolService
227. string sprotoser("(..ProtocolService.\*)");
228. regex e\_sprotoser\_start(sstart + sprotoser);
229. regex e\_sprotoser\_done(ssuccess + sprotoser + sruntime);
231. //  24.DiagnosticsService
232. string sdiagser("(..DiagnosticsService.\*)");
233. regex e\_sdiagser\_start(sstart + sdiagser);
234. regex e\_sdiagser\_done(ssuccess + sdiagser + sruntime);
236. //  Softload start
237. string softdate("(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|Dec)");
238. string softdate1("([\\s]+)([0-9]{1,2}).");
239. string softload(".\*(SOFTLOADSERVICE.\*)");
240. string softstart("(.\*Install started.\*)");
241. regex e\_softload\_start(softdate + softdate1 + stime + softload + softstart);
243. //  Softload original
244. string softori1("(.\*removing intouch-application-base-)");
245. string softori2("(.\*)(.armv6jel\_vfp.rpm.\*)");
246. regex e\_soffser\_orgi(softori1 + softori2);
248. //  Softload newverion
249. string softnew1("(.\*Processing.[0-9]{1,2}.of.[0-9]{1,2}.)");
250. string softnew2("(intouch-application-base-)(.\*)(.armv6jel\_vfp.rpm.\*)");
251. regex e\_softload\_new(softnew1 + softnew2);
253. //  Softload done
254. string softdone1("(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)");
255. string softdone2("([\\s]+)([0-9]{1,2}).([0-9]{2}):([0-9]{2}):([0-9]{2})");
256. string softdone3("(.\*ExitValue from install command.\*)");
257. regex e\_softload\_done(softdone1 + softdone2 + softdone3);
259. //  create default construct class Kronos
260. Kronos logfile(inputName);
262. // hold matches values
263. boost::smatch m;
265. **while** (getline(inputFile, line)) {
266. **if** (regex\_match(line, m, e\_begin\_boot)) {
267. date d(from\_simple\_string(m[0]));
268. ptime temp(d, time\_duration(to\_int(m[4]),
269. to\_int(m[5]), to\_int(m[6])));
270. t1 = temp;
272. **if** (boot) {
273. outputFile << "\*\*\*\* Incomplete boot \*\*\*\*\n" << std::endl;
274. outputFile << logfile.getStr();
275. logfile.reset();
276. boot = **false**;
277. check = **false**;
278. }
279. outputFile << "=== Device boot ===" << endl
280. << line\_order << "(" << inputName << "): "
281. << m[1] << "-" << m[2] << "-" << m[3] << " "  //  date
282. << m[4] << ":" << m[5] << ":" << m[6] << " "  //  time
283. << "Boot Start" << endl;
284. boot = **true**;
285. check = **true**;
286. } **else** **if** (regex\_match(line, m, e\_end\_boot)) {
287. **if** (boot) {
288. date d(from\_simple\_string(m[0]));
289. ptime temp(d, time\_duration(to\_int(m[4]),
290. to\_int(m[5]), to\_int(m[6])));
291. t2 = temp;
292. time\_duration td = t2 - t1;
293. outputFile << line\_order << "(" << inputName << "): "
294. << m[1] << "-" << m[2] << "-" << m[3] << " "  //  date
295. << m[4] << ":" << m[5] << ":" << m[6] << " "  //  time
296. << "Boot Completed" << endl
297. << "\t" << "Boot Time: "
298. << td.total\_milliseconds() << "ms\n" << endl;
299. outputFile << logfile.getStr();
300. logfile.reset();
301. boot = **false**;
302. check = **false**;
303. } **else** {
304. outputFile << "\*\*\*\* Incomplete boot12345 \*\*\*\* \n" << endl;
305. }
306. check = **false**;
307. }
309. **if** (check) {
310. //  1.Logging
311. **if** (regex\_match(line, m, e\_slog\_start)) {
312. logfile.update(1, 1, IntToString(line\_order), "");
313. } **else** **if** (regex\_match(line, m, e\_slog\_done)) {
314. time = m[3];
315. logfile.update(1, 2, IntToString(line\_order), time);
316. //  2.DatabaseInitialize
317. } **else** **if** (regex\_match(line, m, e\_datainit\_start)) {
318. logfile.update(2, 1, IntToString(line\_order), "");
319. } **else** **if** (regex\_match(line, m, e\_datainit\_done)) {
320. time = m[3];
321. logfile.update(2, 2, IntToString(line\_order), time);
322. //  3.MessagingService
323. } **else** **if** (regex\_match(line, m, e\_messer\_start)) {
324. logfile.update(3, 1, IntToString(line\_order), "");
325. } **else** **if** (regex\_match(line, m, e\_messer\_done)) {
326. time = m[3];
327. logfile.update(3, 2, IntToString(line\_order), time);
328. //  4.HealthMonitorService
329. } **else** **if** (regex\_match(line, m, e\_health\_start)) {
330. logfile.update(4, 1, IntToString(line\_order), "");
331. } **else** **if** (regex\_match(line, m, e\_health\_done)) {
332. time = m[3];
333. logfile.update(4, 2, IntToString(line\_order), time);
334. //  5.Persistence
335. } **else** **if** (regex\_match(line, m, e\_sperser\_start)) {
336. logfile.update(5, 1, IntToString(line\_order), "");
337. } **else** **if** (regex\_match(line, m, e\_sperser\_done)) {
338. time = m[3];
339. logfile.update(5, 2, IntToString(line\_order), time);
340. //  6.ConfigurationService
341. } **else** **if** (regex\_match(line, m, e\_sconser\_start)) {
342. logfile.update(6, 1, IntToString(line\_order), "");
343. } **else** **if** (regex\_match(line, m, e\_sconser\_done)) {
344. time = m[3];
345. logfile.update(6, 2, IntToString(line\_order), time);
346. //  7.LandingPadService
347. } **else** **if** (regex\_match(line, m, e\_slser\_start)) {
348. logfile.update(7, 1, IntToString(line\_order), "");
349. } **else** **if** (regex\_match(line, m, e\_slser\_done)) {
350. time = m[3];
351. logfile.update(7, 2, IntToString(line\_order), time);
352. //  8.PortConfigurationService
353. } **else** **if** (regex\_match(line, m, e\_sportcon\_start)) {
354. logfile.update(8, 1, IntToString(line\_order), "");
355. } **else** **if** (regex\_match(line, m, e\_sportcon\_done)) {
356. time = m[3];
357. logfile.update(8, 2, IntToString(line\_order), time);
358. //  9.CacheService
359. } **else** **if** (regex\_match(line, m, e\_scser\_start)) {
360. logfile.update(9, 1, IntToString(line\_order), "");
361. } **else** **if** (regex\_match(line, m, e\_scser\_done)) {
362. time = m[3];
363. logfile.update(9, 2, IntToString(line\_order), time);
364. //  10.ThemingService
365. } **else** **if** (regex\_match(line, m, e\_sthemser\_start)) {
366. logfile.update(10, 1, IntToString(line\_order), "");
367. } **else** **if** (regex\_match(line, m, e\_sthemser\_done)) {
368. time = m[3];
369. logfile.update(10, 2, IntToString(line\_order), time);
370. //  11.StagingService
371. } **else** **if** (regex\_match(line, m, e\_sstaser\_start)) {
372. logfile.update(11, 1, IntToString(line\_order), "");
373. } **else** **if** (regex\_match(line, m, e\_sstaser\_done)) {
374. time = m[3];
375. logfile.update(11, 2, IntToString(line\_order), time);
376. //  12.DeviceIOService
377. } **else** **if** (regex\_match(line, m, e\_sdevser\_start)) {
378. logfile.update(12, 1, IntToString(line\_order), "");
379. } **else** **if** (regex\_match(line, m, e\_sdevser\_done)) {
380. time = m[3];
381. logfile.update(12, 2, IntToString(line\_order), time);
382. //  13.BellService
383. } **else** **if** (regex\_match(line, m, e\_sbellser\_start)) {
384. logfile.update(13, 1, IntToString(line\_order), "");
385. } **else** **if** (regex\_match(line, m, e\_sbellser\_done)) {
386. time = m[3];
387. logfile.update(13, 2, IntToString(line\_order), time);
388. //  14.GateService
389. } **else** **if** (regex\_match(line, m, e\_sgateser\_start)) {
390. logfile.update(14, 1, IntToString(line\_order), "");
391. } **else** **if** (regex\_match(line, m, e\_sgateser\_done)) {
392. time = m[3];
393. logfile.update(14, 2, IntToString(line\_order), time);
394. //  15.ReaderDataService
395. } **else** **if** (regex\_match(line, m, e\_sreadser\_start)) {
396. logfile.update(15, 1, IntToString(line\_order), "");
397. } **else** **if** (regex\_match(line, m, e\_sreadser\_done)) {
398. time = m[3];
399. logfile.update(15, 2, IntToString(line\_order), time);
400. //  16.BiometricService
401. } **else** **if** (regex\_match(line, m, e\_sbioser\_start)) {
402. logfile.update(16, 1, IntToString(line\_order), "");
403. } **else** **if** (regex\_match(line, m, e\_sbioser\_done)) {
404. time = m[3];
405. logfile.update(16, 2, IntToString(line\_order), time);
406. //  17.StateManager
407. } **else** **if** (regex\_match(line, m, e\_stateser\_start)) {
408. logfile.update(17, 1, IntToString(line\_order), "");
409. } **else** **if** (regex\_match(line, m, e\_stateser\_done)) {
410. time = m[3];
411. logfile.update(17, 2, IntToString(line\_order), time);
412. //  18.OfflineSmartviewService
413. } **else** **if** (regex\_match(line, m, e\_soffser\_start)) {
414. logfile.update(18, 1, IntToString(line\_order), "");
415. } **else** **if** (regex\_match(line, m, e\_soffser\_done)) {
416. time = m[3];
417. logfile.update(18, 2, IntToString(line\_order), time);
418. //  19.AVFeedbackService
419. } **else** **if** (regex\_match(line, m, e\_savfser\_start)) {
420. logfile.update(19, 1, IntToString(line\_order), "");
421. } **else** **if** (regex\_match(line, m, e\_savfser\_done)) {
422. time = m[3];
423. logfile.update(19, 2, IntToString(line\_order), time);
424. //  20.DatabaseThreads
425. } **else** **if** (regex\_match(line, m, e\_sdataser\_start)) {
426. logfile.update(20, 1, IntToString(line\_order), "");
427. } **else** **if** (regex\_match(line, m, e\_sdataser\_done)) {
428. time = m[3];
429. logfile.update(20, 2, IntToString(line\_order), time);
430. //  21.SoftLoadService
431. } **else** **if** (regex\_match(line, m, e\_ssoftser\_start)) {
432. logfile.update(21, 1, IntToString(line\_order), "");
433. } **else** **if** (regex\_match(line, m, e\_ssoftser\_done)) {
434. time = m[3];
435. logfile.update(21, 2, IntToString(line\_order), time);
436. //  22.WATCHDOG
437. } **else** **if** (regex\_match(line, m, e\_swatser\_start)) {
438. logfile.update(22, 1, IntToString(line\_order), "");
439. } **else** **if** (regex\_match(line, m, e\_swatser\_done)) {
440. time = m[3];
441. logfile.update(22, 2, IntToString(line\_order), time);
442. //  23.ProtocolService
443. } **else** **if** (regex\_match(line, m, e\_sprotoser\_start)) {
444. logfile.update(23, 1, IntToString(line\_order), "");
445. } **else** **if** (regex\_match(line, m, e\_sprotoser\_done)) {
446. time = m[3];
447. logfile.update(23, 2, IntToString(line\_order), time);
448. //  24.DiagnosticsService
449. } **else** **if** (regex\_match(line, m, e\_sdiagser\_start)) {
450. logfile.update(24, 1, IntToString(line\_order), "");
451. } **else** **if** (regex\_match(line, m, e\_sdiagser\_done)) {
452. time = m[3];
453. logfile.update(24, 2, IntToString(line\_order), time);
454. }
455. }
457. //  SoftLoad start
458. **if** (regex\_match(line, m, e\_softload\_start)) {
459. time1 = (to\_int(m[4]) \* 3600) + (to\_int(m[5]) \* 60) + to\_int(m[6]);
460. time = m[1] + " " + m[3] + " " + m[4] + ":" + m[5] + ":" + m[6];
461. // cout << time << endl;
462. logfile.update(25, 1, IntToString(line\_order), time);
463. } **else** **if** (regex\_match(line, m, e\_softload\_new)) {  //  SoftLoad new
464. time = m[3];
465. // cout << time << endl;
466. logfile.update(25, 2, IntToString(line\_order), time);
467. } **else** **if** (regex\_match(line, m, e\_soffser\_orgi)) {  //  SoftLoad orig
468. time = m[2];
469. // cout << time << endl;
470. logfile.update(25, 3, IntToString(line\_order), time);
471. } **else** **if** (regex\_match(line, m, e\_softload\_done)) {  //  SoftLoad end
472. time2 = (to\_int(m[4]) \* 3600) + (to\_int(m[5]) \* 60) + to\_int(m[6]);
473. time = m[1] + " " + m[3] + " " + m[4] + ":" + m[5] + ":" + m[6];
474. // cout << time << endl;
475. logfile.update(25, 4, IntToString(line\_order), time);
476. logfile.update(25, 5, IntToString(time2 - time1), time);
477. outputFile << logfile.getsl();
478. }
479. line\_order++;
480. }
481. **return** 0;
482. }
483. // Copyright Chuong Vu
484. // Filename: kronos\_parse.hpp
485. #include <string>
486. #include <sstream>  // Required for stringstreams
488. **class** Kronos {
489. **public**:
490. **explicit** Kronos(std::string input);
491. std::string getStr(**void**);
492. std::string getsl(**void**);
493. **void** update(**int** name, **int** check, **const** std::string line,
494. **const** std::string time);
495. **void** reset(**void**);
497. **private**:
498. std::stringstream ss;  //  hold the output string
499. std::string \_Name;  // device name
501. // 24 steps
502. std::string \_logstart, \_logcom, \_datainitstart, \_datainitcom, \_messerstart,
503. \_messercom, \_heathst, \_heathco, \_perserst, \_perserco, \_sconserst,
504. \_sconserco, \_slserst, \_slserco, \_sportconst, \_sportconco, \_scserst,
505. \_scserco, \_sthemserst, \_sthemserco, \_sstaserst, \_sstaserco, \_sdevserst,
506. \_sdevserco, \_sbellserst, \_sbellserco, \_sgateserst, \_sgateserco, \_sreadserst,
507. \_sreadserco, \_sbioserst, \_sbioserco, \_stateserst, \_stateserco, \_soffserst,
508. \_soffserco, \_savfserst, \_savfserco, \_sdataserst, \_sdataserco, \_ssoftserst,
509. \_ssoftserco, \_swatserst, \_swatserco, \_sprotoserst, \_sprotoserco,
510. \_sdiagserst, \_sdiagserco;
512. //  for softload
513. std::string \_softloadstart, \_softloador, \_softloadnew, \_softloadtime,
514. \_softloadelap;
516. **int** \_logging, \_datainit, \_messer, \_health, \_sperser, \_sconser,
517. \_slser, \_sportcon, \_scser, \_sthemser, \_sstaser, \_sdevser, \_sbellser,
518. \_sgateser, \_sreadser, \_sbioser, \_stateser, \_soffser, \_savfser, \_sdataser,
519. \_ssoftser, \_swatser, \_sprotoser, \_sdiagser;
520. **bool** first;  // first value use to check first failed
521. **bool** \_softload;  //  check softload start or not.
522. };
523. // Copyright Chuong Vu
524. // kronos\_parse.cpp
525. #include <iostream>
526. #include <string>
527. #include "kronos\_parse.hpp"
529. Kronos::Kronos(std::string input) : \_Name(input) {
530. \_logging = \_datainit = \_messer = \_health = \_sperser = \_sconser = \_slser
531. = \_sportcon = \_scser = \_sthemser = \_sstaser = \_sdevser = \_sbellser
532. = \_sgateser = \_sreadser = \_sbioser = \_stateser = \_soffser = \_savfser
533. = \_sdataser = \_ssoftser = \_swatser = \_sprotoser = \_sdiagser = 0;
534. first = **true**;
535. \_softload = **false**;
536. ss.str("");
537. //  1
538. \_logstart = "\tLogging\n\t\tStart: Not started(" + \_Name + ")\n";
539. \_logcom = "\t\tCompleted: Not completed(" + \_Name + ")\n"
540. + "\t\tElapsed Time: \n";
541. //  2
542. \_datainitstart = "\tDatabaseInitialize\n\t\tStart: Not started("
543. + \_Name + ")\n";
544. \_datainitcom = "\t\tCompleted: Not completed(" + \_Name + ")\n"
545. + "\t\tElapsed Time: \n";
546. //  3
547. \_messerstart = "\tMessagingService\n\t\tStart: Not started(" + \_Name
548. + ")\n";
549. \_messercom = "\t\tCompleted: Not completed(" + \_Name + ")\n"
550. + "\t\tElapsed Time: \n";
551. //  4
552. \_heathst = "\tHealthMonitorService\n\t\tStart: Not started(" + \_Name
553. + ")\n";
554. \_heathco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
555. + "\t\tElapsed Time: \n";
556. //  5
557. \_perserst = "\tPersistence\n\t\tStart: Not started(" + \_Name + ")\n";
558. \_perserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
559. + "\t\tElapsed Time: \n";
560. //  6
561. \_sconserst = "\tConfigurationService\n\t\tStart: Not started(" + \_Name
562. + ")\n";
563. \_sconserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
564. + "\t\tElapsed Time: \n";
565. //  7
566. \_slserst = "\tLandingPadService\n\t\tStart: Not started(" + \_Name
567. + ")\n";
568. \_slserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
569. + "\t\tElapsed Time: \n";
570. //  8
571. \_sportconst = "\tPortConfigurationService\n\t\tStart: Not started("
572. + \_Name + ")\n";
573. \_sportconco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
574. + "\t\tElapsed Time: \n";
575. //  9
576. \_scserst = "\tCacheService\n\t\tStart: Not started(" + \_Name + ")\n";
577. \_scserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
578. + "\t\tElapsed Time: \n";
579. //  10
580. \_sthemserst = "\tThemingService\n\t\tStart: Not started(" + \_Name + ")\n";
581. \_sthemserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
582. + "\t\tElapsed Time: \n";
583. //  11
584. \_sstaserst = "\tStagingService\n\t\tStart: Not started(" + \_Name + ")\n";
585. \_sstaserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
586. + "\t\tElapsed Time: \n";
587. //  12
588. \_sdevserst = "\tDeviceIOService\n\t\tStart: Not started(" + \_Name + ")\n";
589. \_sdevserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
590. + "\t\tElapsed Time: \n";
591. //  13
592. \_sbellserst = "\tBellService\n\t\tStart: Not started(" + \_Name + ")\n";
593. \_sbellserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
594. + "\t\tElapsed Time: \n";
595. //  14
596. \_sgateserst = "\tGateService\n\t\tStart: Not started(" + \_Name + ")\n";
597. \_sgateserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
598. + "\t\tElapsed Time: \n";
599. //  15
600. \_sreadserst = "\tReaderDataService\n\t\tStart: Not started(" + \_Name
601. + ")\n";
602. \_sreadserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
603. + "\t\tElapsed Time: \n";
604. //  16
605. \_sbioserst = "\tBiometricService\n\t\tStart: Not started(" + \_Name + ")\n";
606. \_sbioserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
607. + "\t\tElapsed Time: \n";
608. //  17
609. \_stateserst = "\tStateManager\n\t\tStart: Not started(" + \_Name + ")\n";
610. \_stateserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
611. + "\t\tElapsed Time: \n";
612. //  18
613. \_soffserst = "\tOfflineSmartviewService\n\t\tStart: Not started(" + \_Name
614. + ")\n";
615. \_soffserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
616. + "\t\tElapsed Time: \n";
617. //  19
618. \_savfserst = "\tAVFeedbackService\n\t\tStart: Not started(" + \_Name
619. + ")\n";
620. \_savfserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
621. + "\t\tElapsed Time: \n";
622. //  20
623. \_sdataserst = "\tDatabaseThreads\n\t\tStart: Not started(" + \_Name + ")\n";
624. \_sdataserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
625. + "\t\tElapsed Time: \n";
626. //  21
627. \_ssoftserst = "\tSoftLoadService\n\t\tStart: Not started(" + \_Name + ")\n";
628. \_ssoftserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
629. + "\t\tElapsed Time: \n";
630. //  22
631. \_swatserst = "\tWATCHDOG\n\t\tStart: Not started(" + \_Name + ")\n";
632. \_swatserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
633. + "\t\tElapsed Time: \n";
634. //  23
635. \_sprotoserst = "\tProtocolService\n\t\tStart: Not started(" + \_Name + ")\n";
636. \_sprotoserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
637. + "\t\tElapsed Time: \n";
638. //  24
639. \_sdiagserst = "\tDiagnosticsService\n\t\tStart: Not started(" + \_Name
640. + ")\n";
641. \_sdiagserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
642. + "\t\tElapsed Time: \n";
644. //  Softload
645. \_softloadstart = "";
646. \_softloador = "";
647. \_softloadnew = "";
648. \_softloadtime = "";
649. \_softloadelap = "";
650. }

653. **void** Kronos::reset(**void**) {
654. \_logging = \_datainit = \_messer = \_health = \_sperser = \_sconser = \_slser
655. = \_sportcon = \_scser = \_sthemser = \_sstaser = \_sdevser = \_sbellser
656. = \_sgateser = \_sreadser = \_sbioser = \_stateser = \_soffser = \_savfser
657. = \_sdataser = \_ssoftser = \_swatser = \_sprotoser = \_sdiagser = 0;
658. first = **true**;
659. \_softload = **false**;
660. ss.str("");
661. //  1
662. \_logstart = "\tLogging\n\t\tStart: Not started(" + \_Name + ")\n";
663. \_logcom = "\t\tCompleted: Not completed(" + \_Name + ")\n"
664. + "\t\tElapsed Time: \n";
665. //  2
666. \_datainitstart = "\tDatabaseInitialize\n\t\tStart: Not started("
667. + \_Name + ")\n";
668. \_datainitcom = "\t\tCompleted: Not completed(" + \_Name + ")\n"
669. + "\t\tElapsed Time: \n";
670. //  3
671. \_messerstart = "\tMessagingService\n\t\tStart: Not started(" + \_Name
672. + ")\n";
673. \_messercom = "\t\tCompleted: Not completed(" + \_Name + ")\n"
674. + "\t\tElapsed Time: \n";
675. //  4
676. \_heathst = "\tHealthMonitorService\n\t\tStart: Not started(" + \_Name
677. + ")\n";
678. \_heathco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
679. + "\t\tElapsed Time: \n";
680. //  5
681. \_perserst = "\tPersistence\n\t\tStart: Not started(" + \_Name + ")\n";
682. \_perserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
683. + "\t\tElapsed Time: \n";
684. //  6
685. \_sconserst = "\tConfigurationService\n\t\tStart: Not started(" + \_Name
686. + ")\n";
687. \_sconserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
688. + "\t\tElapsed Time: \n";
689. //  7
690. \_slserst = "\tLandingPadService\n\t\tStart: Not started(" + \_Name
691. + ")\n";
692. \_slserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
693. + "\t\tElapsed Time: \n";
694. //  8
695. \_sportconst = "\tPortConfigurationService\n\t\tStart: Not started("
696. + \_Name + ")\n";
697. \_sportconco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
698. + "\t\tElapsed Time: \n";
699. //  9
700. \_scserst = "\tCacheService\n\t\tStart: Not started(" + \_Name + ")\n";
701. \_scserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
702. + "\t\tElapsed Time: \n";
703. //  10
704. \_sthemserst = "\tThemingService\n\t\tStart: Not started(" + \_Name + ")\n";
705. \_sthemserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
706. + "\t\tElapsed Time: \n";
707. //  11
708. \_sstaserst = "\tStagingService\n\t\tStart: Not started(" + \_Name + ")\n";
709. \_sstaserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
710. + "\t\tElapsed Time: \n";
711. //  12
712. \_sdevserst = "\tDeviceIOService\n\t\tStart: Not started(" + \_Name + ")\n";
713. \_sdevserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
714. + "\t\tElapsed Time: \n";
715. //  13
716. \_sbellserst = "\tBellService\n\t\tStart: Not started(" + \_Name + ")\n";
717. \_sbellserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
718. + "\t\tElapsed Time: \n";
719. //  14
720. \_sgateserst = "\tGateService\n\t\tStart: Not started(" + \_Name + ")\n";
721. \_sgateserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
722. + "\t\tElapsed Time: \n";
723. //  15
724. \_sreadserst = "\tReaderDataService\n\t\tStart: Not started(" + \_Name
725. + ")\n";
726. \_sreadserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
727. + "\t\tElapsed Time: \n";
728. //  16
729. \_sbioserst = "\tBiometricService\n\t\tStart: Not started(" + \_Name + ")\n";
730. \_sbioserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
731. + "\t\tElapsed Time: \n";
732. //  17
733. \_stateserst = "\tStateManager\n\t\tStart: Not started(" + \_Name + ")\n";
734. \_stateserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
735. + "\t\tElapsed Time: \n";
736. //  18
737. \_soffserst = "\tOfflineSmartviewService\n\t\tStart: Not started(" + \_Name
738. + ")\n";
739. \_soffserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
740. + "\t\tElapsed Time: \n";
741. //  19
742. \_savfserst = "\tAVFeedbackService\n\t\tStart: Not started(" + \_Name
743. + ")\n";
744. \_savfserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
745. + "\t\tElapsed Time: \n";
746. //  20
747. \_sdataserst = "\tDatabaseThreads\n\t\tStart: Not started(" + \_Name + ")\n";
748. \_sdataserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
749. + "\t\tElapsed Time: \n";
750. //  21
751. \_ssoftserst = "\tSoftLoadService\n\t\tStart: Not started(" + \_Name + ")\n";
752. \_ssoftserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
753. + "\t\tElapsed Time: \n";
754. //  22
755. \_swatserst = "\tWATCHDOG\n\t\tStart: Not started(" + \_Name + ")\n";
756. \_swatserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
757. + "\t\tElapsed Time: \n";
758. //  23
759. \_sprotoserst = "\tProtocolService\n\t\tStart: Not started(" + \_Name + ")\n";
760. \_sprotoserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
761. + "\t\tElapsed Time: \n";
762. //  24
763. \_sdiagserst = "\tDiagnosticsService\n\t\tStart: Not started(" + \_Name
764. + ")\n";
765. \_sdiagserco = "\t\tCompleted: Not completed(" + \_Name + ")\n"
766. + "\t\tElapsed Time: \n";
768. //  Softload
769. \_softloadstart = "";
770. \_softloador = "";
771. \_softloadnew = "";
772. \_softloadtime = "";
773. \_softloadelap = "";
774. }
776. **void** Kronos::update(**int** name, **int** check, **const** std::string line,
777. **const** std::string \_time) {
778. **switch** (name) {
779. **case** 1:
780. **if** (check == 1) {
781. // \_logstart = "";
782. \_logstart = "\tLogging\n\t\tStart: " + line + "(" + \_Name
783. + ")\n";
784. // Check failed flag
785. \_logging == 1 ? \_logging = 1 : \_logging++;
786. } **else** **if** (check == 2) {
787. // \_logcom = "";
788. \_logcom = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
789. + "\t\tElapsed Time: " + \_time + "\n";
790. \_logging++;
791. }
792. **break**;
794. **case** 2:
795. **if** (check == 1) {
796. // \_datainitstart = "";
797. \_datainitstart = "\tDatabaseInitialize\n\t\tStart: " + line
798. + "(" + \_Name + ")\n";
799. // Check failed flag
800. \_datainit == 1 ? \_datainit = 1 : \_datainit++;
801. } **else** **if** (check == 2) {
802. // \_datainitcom = "";
803. \_datainitcom = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
804. + "\t\tElapsed Time: " + \_time + "\n";
805. \_datainit++;
806. }
807. **break**;
809. **case** 3:
810. **if** (check == 1) {
811. // \_messerstart = "";
812. \_messerstart = "\tMessagingService\n\t\tStart: " + line + "("
813. + \_Name + ")\n";
814. // Check failed flag
815. \_messer == 1 ? \_messer = 1 : \_messer++;
816. } **else** **if** (check == 2) {
817. // \_messercom = "";
818. \_messercom = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
819. + "\t\tElapsed Time: " + \_time + "\n";
820. \_messer++;
821. }
822. **break**;
824. **case** 4:
825. **if** (check == 1) {
826. // \_heathst = "";
827. \_heathst = "\tHealthMonitorService\n\t\tStart: " + line + "("
828. + \_Name + ")\n";
829. // Check failed flag
830. \_health == 1 ? \_health = 1 : \_health++;
831. } **else** **if** (check == 2) {
832. // \_heathco = "";
833. \_heathco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
834. + "\t\tElapsed Time: " + \_time + "\n";
835. \_health++;
836. }
837. **break**;
839. **case** 5:
840. **if** (check == 1) {
841. // \_perserst = "";
842. \_perserst = "\tPersistence\n\t\tStart: " + line + "(" + \_Name
843. + ")\n";
844. // Check failed flag
845. \_sperser == 1 ? \_sperser = 1 : \_sperser++;
846. } **else** **if** (check == 2) {
847. // \_perserco = "";
848. \_perserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
849. + "\t\tElapsed Time: " + \_time + "\n";
850. \_sperser++;
851. }
852. **break**;
854. **case** 6:
855. **if** (check == 1) {
856. // \_sconserst = "";
857. \_sconserst = "\tConfigurationService\n\t\tStart: " + line
858. + "(" + \_Name + ")\n";
859. // Check failed flag
860. \_sconser == 1 ? \_sconser = 1 : \_sconser++;
861. } **else** **if** (check == 2) {
862. // \_sconserco = "";
863. \_sconserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
864. + "\t\tElapsed Time: " + \_time + "\n";
865. \_sconser++;
866. }
867. **break**;
869. **case** 7:
870. **if** (check == 1) {
871. // \_slserst = "";
872. \_slserst = "\tLandingPadService\n\t\tStart: " + line + "("
873. + \_Name + ")\n";
874. // Check failed flag
875. \_slser == 1 ? \_slser = 1 : \_slser++;
876. } **else** **if** (check == 2) {
877. // \_slserco = "";
878. \_slserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
879. + "\t\tElapsed Time: " + \_time + "\n";
880. \_slser++;
881. }
882. **break**;
884. **case** 8:
885. **if** (check == 1) {
886. // \_sportconst = "";
887. \_sportconst = "\tPortConfigurationService\n\t\tStart: " + line
888. + "(" + \_Name + ")\n";
889. // Check failed flag
890. \_sportcon == 1 ? \_sportcon = 1 : \_sportcon++;
891. } **else** **if** (check == 2) {
892. // \_sportconco = "";
893. \_sportconco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
894. + "\t\tElapsed Time: " + \_time + "\n";
895. \_sportcon++;
896. }
897. **break**;
899. **case** 9:
900. **if** (check == 1) {
901. //  \_scserst = "";
902. \_scserst = "\tCacheService\n\t\tStart: " + line + "(" + \_Name
903. + ")\n";
904. // Check failed flag
905. \_scser == 1 ? \_scser = 1 : \_scser++;
906. } **else** **if** (check == 2) {
907. // \_scserco = "";
908. \_scserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
909. + "\t\tElapsed Time: " + \_time + "\n";
910. \_scser++;
911. }
912. **break**;
914. **case** 10:
915. **if** (check == 1) {
916. // \_sthemserst = "";
917. \_sthemserst = "\tThemingService\n\t\tStart: " + line + "("
918. + \_Name + ")\n";
919. // Check failed flag
920. \_sthemser == 1 ? \_sthemser = 1 : \_sthemser++;
921. } **else** **if** (check == 2) {
922. // \_sthemserco = "";
923. \_sthemserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
924. + "\t\tElapsed Time: " + \_time + "\n";
925. \_sthemser++;
926. }
927. **break**;
929. **case** 11:
930. **if** (check == 1) {
931. //  \_sstaserst = "";
932. \_sstaserst = "\tStagingService\n\t\tStart: " + line + "("
933. + \_Name + ")\n";
934. // Check failed flag
935. \_sstaser == 1 ? \_sstaser = 1 : \_sstaser++;
936. } **else** **if** (check == 2) {
937. // \_sstaserco = "";
938. \_sstaserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
939. + "\t\tElapsed Time: " + \_time + "\n";
940. \_sstaser++;
941. }
942. **break**;
944. **case** 12:
945. **if** (check == 1) {
946. //  \_sdevserst = "";
947. \_sdevserst = "\tDeviceIOService\n\t\tStart: " + line + "("
948. + \_Name + ")\n";
949. // Check failed flag
950. \_sdevser == 1 ? \_sdevser = 1 : \_sdevser++;
951. } **else** **if** (check == 2) {
952. //  \_sdevserco = "";
953. \_sdevserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
954. + "\t\tElapsed Time: " + \_time + "\n";
955. \_sdevser++;
956. }
957. **break**;
959. **case** 13:
960. **if** (check == 1) {
961. //   \_sbellserst = "";
962. \_sbellserst = "\tBellService\n\t\tStart: " + line + "("
963. + \_Name + ")\n";
964. // Check failed flag
965. \_sbellser == 1 ? \_sbellser = 1 : \_sbellser++;
966. } **else** **if** (check == 2) {
967. //  \_sbellserco = "";
968. \_sbellserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
969. + "\t\tElapsed Time: " + \_time + "\n";
970. \_sbellser++;
971. }
972. **break**;
974. **case** 14:
975. **if** (check == 1) {
976. //  \_sgateserst = "";
977. \_sgateserst = "\tGateService\n\t\tStart: " + line + "("
978. + \_Name + ")\n";
979. // Check failed flag
980. \_sgateser == 1 ? \_sgateser = 1 : \_sgateser++;
981. } **else** **if** (check == 2) {
982. //  \_sgateserco = "";
983. \_sgateserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
984. + "\t\tElapsed Time: " + \_time + "\n";
985. \_sgateser++;
986. }
987. **break**;
989. **case** 15:
990. **if** (check == 1) {
991. //  \_sreadserst = "";
992. \_sreadserst = "\tReaderDataService\n\t\tStart: " + line + "("
993. + \_Name + ")\n";
994. // Check failed flag
995. \_sreadser == 1 ? \_sreadser = 1 : \_sreadser++;
996. } **else** **if** (check == 2) {
997. //   \_sreadserco = "";
998. \_sreadserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
999. + "\t\tElapsed Time: " + \_time + "\n";
1000. \_sreadser++;
1001. }
1002. **break**;
1004. **case** 16:
1005. **if** (check == 1) {
1006. //  \_sbioserst = "";
1007. \_sbioserst = "\tBiometricService\n\t\tStart: " + line + "("
1008. + \_Name + ")\n";
1009. // Check failed flag
1010. \_sbioser == 1 ? \_sbioser = 1 : \_sbioser++;
1011. } **else** **if** (check == 2) {
1012. //   \_sbioserco = "";
1013. \_sbioserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1014. + "\t\tElapsed Time: " + \_time + "\n";
1015. \_sbioser++;
1016. }
1017. **break**;
1019. **case** 17:
1020. **if** (check == 1) {
1021. //   \_stateserst = "";
1022. \_stateserst = "\tStateManager\n\t\tStart: " + line + "("
1023. + \_Name + ")\n";
1024. // Check failed flag
1025. \_stateser == 1 ? \_stateser = 1 : \_stateser++;
1026. } **else** **if** (check == 2) {
1027. //   \_stateserco = "";
1028. \_stateserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1029. + "\t\tElapsed Time: " + \_time + "\n";
1030. \_stateser++;
1031. }
1032. **break**;
1034. **case** 18:
1035. **if** (check == 1) {
1036. //   \_soffserst = "";
1037. \_soffserst = "\tOfflineSmartviewService\n\t\tStart: " + line
1038. + "(" + \_Name + ")\n";
1039. // Check failed flag
1040. \_soffser == 1 ? \_soffser = 1 : \_soffser++;
1041. } **else** **if** (check == 2) {
1042. //   \_soffserco = "";
1043. \_soffserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1044. + "\t\tElapsed Time: " + \_time + "\n";
1045. \_soffser++;
1046. }
1047. **break**;
1049. **case** 19:
1050. **if** (check == 1) {
1051. //   \_savfserst = "";
1052. \_savfserst = "\tAVFeedbackService\n\t\tStart: " + line + "("
1053. + \_Name + ")\n";
1054. // Check failed flag
1055. \_savfser == 1 ? \_savfser = 1 : \_savfser++;
1056. } **else** **if** (check == 2) {
1057. //  \_savfserco = "";
1058. \_savfserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1059. + "\t\tElapsed Time: " + \_time + "\n";
1060. \_savfser++;
1061. }
1062. **break**;
1064. **case** 20:
1065. **if** (check == 1) {
1066. //   \_sdataserst = "";
1067. \_sdataserst = "\tDatabaseThreads\n\t\tStart: " + line + "("
1068. + \_Name + ")\n";
1069. // Check failed flag
1070. \_sdataser == 1 ? \_sdataser = 1 : \_sdataser++;
1071. } **else** **if** (check == 2) {
1072. //  \_sdataserco = "";
1073. \_sdataserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1074. + "\t\tElapsed Time: " + \_time + "\n";
1075. \_sdataser++;
1076. }
1077. **break**;
1079. **case** 21:
1080. **if** (check == 1) {
1081. //    \_ssoftserst = "";
1082. \_ssoftserst = "\tSoftLoadService\n\t\tStart: " + line + "("
1083. + \_Name + ")\n";
1084. // Check failed flag
1085. \_ssoftser == 1 ? \_ssoftser = 1 : \_ssoftser++;
1086. } **else** **if** (check == 2) {
1087. //  \_ssoftserco = "";
1088. \_ssoftserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1089. + "\t\tElapsed Time: " + \_time + "\n";
1090. \_ssoftser++;
1091. }
1092. **break**;
1094. **case** 22:
1095. **if** (check == 1) {
1096. //     \_swatserst = "";
1097. \_swatserst = "\tWATCHDOG\n\t\tStart: " + line + "(" + \_Name
1098. + ")\n";
1099. // Check failed flag
1100. \_swatser == 1 ? \_swatser = 1 : \_swatser++;
1101. } **else** **if** (check == 2) {
1102. //     \_swatserco = "";
1103. \_swatserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1104. + "\t\tElapsed Time: " + \_time + "\n";
1105. \_swatser++;
1106. }
1107. **break**;
1109. **case** 23:
1110. **if** (check == 1) {
1111. //     \_sprotoserst = "";
1112. \_sprotoserst = "\tProtocolService\n\t\tStart: " + line + "("
1113. + \_Name + ")\n";
1114. // Check failed flag
1115. \_sprotoser == 1 ? \_sprotoser = 1 : \_sprotoser++;
1116. } **else** **if** (check == 2) {
1117. //      \_sprotoserco = "";
1118. \_sprotoserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1119. + "\t\tElapsed Time: " + \_time + "\n";
1120. \_sprotoser++;
1121. }
1122. **break**;
1124. **case** 24:
1125. **if** (check == 1) {
1126. //   \_sdiagserst = "";
1127. \_sdiagserst = "\tDiagnosticsService\n\t\tStart: " + line + "("
1128. + \_Name + ")\n";
1129. // Check failed flag
1130. \_sdiagser == 1 ? \_sdiagser = 1 : \_sdiagser++;
1131. } **else** **if** (check == 2) {
1132. //   \_sdiagserco = "";
1133. \_sdiagserco = "\t\tCompleted: " + line + "(" + \_Name + ")\n"
1134. + "\t\tElapsed Time: " + \_time + "\n";
1135. \_sdiagser++;
1136. }
1137. **break**;
1139. **case** 25:
1140. **if** (check == 1) {
1141. \_softloadstart = line + "(" + \_Name + ") : " + \_time
1142. + " Softload Start\n";
1143. \_softload = **true**;
1144. } **else** **if** (check == 2) {
1145. \_softloadnew = "\tNew version ==> " + \_time + "\n";
1146. } **else** **if** (check == 3) {
1147. \_softloador = "\tOriginal version ==> " + \_time + "\n";
1148. } **else** **if** (check == 4) {
1149. \_softloadtime = line + "(" + \_Name + ") : " + \_time
1150. + " Softload Completed\n";
1151. } **else** **if** (check == 5) {
1152. \_softloadelap = "\tElapsed time (sec) ==> " + line + "\n";
1153. }
1154. **break**;
1156. **default**:
1157. // Code
1158. **break**;
1159. }
1160. }

1163. std::string Kronos::getStr(**void**) {
1164. ss << "Services\n";
1165. ss << \_logstart << \_logcom << \_datainitstart << \_datainitcom
1166. << \_messerstart << \_messercom << \_heathst << \_heathco
1167. << \_perserst << \_perserco << \_sconserst << \_sconserco
1168. << \_slserst << \_slserco << \_sportconst << \_sportconco
1169. << \_scserst << \_scserco << \_sthemserst << \_sthemserco
1170. << \_sstaserst << \_sstaserco << \_sdevserst << \_sdevserco
1171. << \_sbellserst <<\_sbellserco << \_sgateserst << \_sgateserco
1172. << \_sreadserst << \_sreadserco << \_sbioserst << \_sbioserco
1173. << \_stateserst << \_stateserco << \_soffserst << \_soffserco
1174. << \_savfserst << \_savfserco << \_sdataserst << \_sdataserco
1175. << \_ssoftserst << \_ssoftserco << \_swatserst << \_swatserco
1176. << \_sprotoserst << \_sprotoserco << \_sdiagserst << \_sdiagserco;
1178. **if** (\_logging < 2) {
1179. **if** (first) {
1180. ss << "\t\*\*\* Services not successfully started: Logging";
1181. first = **false**;
1182. } **else** {
1183. ss << ", Logging";
1184. }
1185. }
1187. **if** (\_datainit < 2) {
1188. **if** (first) {
1189. ss << "\t\*\*\* Services not successfully started: DatabaseInitialize";
1190. first = **false**;
1191. } **else** {
1192. ss << ", DatabaseInitialize";
1193. }
1194. }
1196. **if** (\_messer < 2) {
1197. **if** (first) {
1198. ss << "\t\*\*\* Services not successfully started: MessagingService";
1199. first = **false**;
1200. } **else** {
1201. ss << ", MessagingService";
1202. }
1203. }
1205. **if** (\_health < 2) {
1206. **if** (first) {
1207. ss << "\t\*\*\* Services not successfully started:"
1208. << "HealthMonitorService";
1209. first = **false**;
1210. } **else** {
1211. ss << ", HealthMonitorService";
1212. }
1213. }
1215. **if** (\_sperser < 2) {
1216. **if** (first) {
1217. ss << "\t\*\*\* Services not successfully started: Persistence";
1218. first = **false**;
1219. } **else** {
1220. ss << ", Persistence";
1221. }
1222. }
1224. **if** (\_sconser < 2) {
1225. **if** (first) {
1226. ss << "\t\*\*\* Services not successfully started:"
1227. << "ConfigurationService";
1228. first = **false**;
1229. } **else** {
1230. ss << ", ConfigurationService";
1231. }
1232. }
1234. **if** (\_slser < 2) {
1235. **if** (first) {
1236. ss << "\t\*\*\* Services not successfully started: LandingPadService";
1237. first = **false**;
1238. } **else** {
1239. ss << ", LandingPadService";
1240. }
1241. }
1243. **if** (\_sportcon < 2) {
1244. **if** (first) {
1245. ss << "\t\*\*\* Services not successfully started:"
1246. << "PortConfigurationService";
1247. first = **false**;
1248. } **else** {
1249. ss << ", PortConfigurationService";
1250. }
1251. }
1253. **if** (\_scser < 2) {
1254. **if** (first) {
1255. ss << "\t\*\*\* Services not successfully started: CacheService";
1256. first = **false**;
1257. } **else** {
1258. ss << ", CacheService";
1259. }
1260. }
1262. **if** (\_sthemser < 2) {
1263. **if** (first) {
1264. ss << "\t\*\*\* Services not successfully started: ThemingService";
1265. first = **false**;
1266. } **else** {
1267. ss << ", ThemingService";
1268. }
1269. }
1271. **if** (\_sstaser < 2) {
1272. **if** (first) {
1273. ss << "\t\*\*\* Services not successfully started: StagingService";
1274. first = **false**;
1275. } **else** {
1276. ss << ", StagingService";
1277. }
1278. }
1280. **if** (\_sdevser < 2) {
1281. **if** (first) {
1282. ss << "\t\*\*\* Services not successfully started: DeviceIOService";
1283. first = **false**;
1284. } **else** {
1285. ss << ", DeviceIOService";
1286. }
1287. }
1289. **if** (\_sbellser < 2) {
1290. **if** (first) {
1291. ss << "\t\*\*\* Services not successfully started: BellService";
1292. first = **false**;
1293. } **else** {
1294. ss << ", BellService";
1295. }
1296. }
1298. **if** (\_sgateser < 2) {
1299. **if** (first) {
1300. ss << "\t\*\*\* Services not successfully started: GateService";
1301. first = **false**;
1302. } **else** {
1303. ss << ", GateService";
1304. }
1305. }
1307. **if** (\_sreadser < 2) {
1308. **if** (first) {
1309. ss << "\t\*\*\* Services not successfully started: ReaderDataService";
1310. first = **false**;
1311. } **else** {
1312. ss << ", ReaderDataService";
1313. }
1314. }
1316. **if** (\_sbioser < 2) {
1317. **if** (first) {
1318. ss << "\t\*\*\* Services not successfully started: BiometricService";
1319. first = **false**;
1320. } **else** {
1321. ss << ", BiometricService";
1322. }
1323. }
1325. **if** (\_stateser < 2) {
1326. **if** (first) {
1327. ss << "\t\*\*\* Services not successfully started: StateManager";
1328. first = **false**;
1329. } **else** {
1330. ss << ", StateManager";
1331. }
1332. }
1334. **if** (\_soffser < 2) {
1335. **if** (first) {
1336. ss << "\t\*\*\* Services not successfully started:"
1337. << " OfflineSmartviewService";
1338. first = **false**;
1339. } **else** {
1340. ss << ", OfflineSmartviewService";
1341. }
1342. }
1344. **if** (\_savfser < 2) {
1345. **if** (first) {
1346. ss << "\t\*\*\* Services not successfully started: AVFeedbackService";
1347. first = **false**;
1348. } **else** {
1349. ss << ", AVFeedbackService";
1350. }
1351. }
1353. **if** (\_sdataser < 2) {
1354. **if** (first) {
1355. ss << "\t\*\*\* Services not successfully started: DatabaseThreads";
1356. first = **false**;
1357. } **else** {
1358. ss << ", DatabaseThreads";
1359. }
1360. }
1362. **if** (\_ssoftser < 2) {
1363. **if** (first) {
1364. ss << "\t\*\*\* Services not successfully started: SoftLoadService";
1365. first = **false**;
1366. } **else** {
1367. ss << ", SoftLoadService";
1368. }
1369. }
1371. **if** (\_swatser < 2) {
1372. **if** (first) {
1373. ss << "\t\*\*\* Services not successfully started: WATCHDOG";
1374. first = **false**;
1375. } **else** {
1376. ss << ", WATCHDOG";
1377. }
1378. }
1380. **if** (\_sprotoser < 2) {
1381. **if** (first) {
1382. ss << "\t\*\*\* Services not successfully started: ProtocolService";
1383. first = **false**;
1384. } **else** {
1385. ss << ", ProtocolService";
1386. }
1387. }
1389. **if** (\_sdiagser < 2) {
1390. **if** (first) {
1391. ss << "\t\*\*\* Services not successfully started: DiagnosticsService";
1392. first = **false**;
1393. } **else** {
1394. ss << ", DiagnosticsService";
1395. }
1396. }
1398. ss << "\n";
1400. **return** ss.str();
1401. }
1403. std::string Kronos::getsl(**void**) {
1404. std::stringstream a;
1405. **if** (\_softload) {
1406. a << "=== Softload ===\n"
1407. << \_softloadstart << \_softloador << \_softloadnew
1408. << \_softloadelap << \_softloadtime;
1409. }
1410. // std::cout << a.str();
1411. **return** a.str();
1412. }