

Computing 2040 Portfolio

Troy Wang

Table of Contents

[PS0: Hello World with SFML 4](#_Toc58170278)

[Discussion: 4](#_Toc58170279)

[Screenshot: 4](#_Toc58170280)

[Code: 5](#_Toc58170281)

[Main.cpp: 5](#_Toc58170282)

[PS1: Linear Feedback Shift Register 7](#_Toc58170283)

[PS1a: Linear Feedback Shift Register (Part A) 7](#_Toc58170284)

[Discussion: 7](#_Toc58170285)

[Console Output (Running Boost Test): 7](#_Toc58170286)

[Code: 7](#_Toc58170287)

[Makefile: 7](#_Toc58170288)

[test.cpp 7](#_Toc58170289)

[fibLFSR.h 9](#_Toc58170290)

[fibLFSR.cpp: 10](#_Toc58170291)

[PS1b Linear Feedback Shift Register (Part B) 12](#_Toc58170292)

[Discussion: 12](#_Toc58170293)

[Screenshots: 12](#_Toc58170294)

[Encoding: 12](#_Toc58170295)

[Decoding: 13](#_Toc58170296)

[Code: 13](#_Toc58170297)

[Makefile: 13](#_Toc58170298)

[Photomagic.cpp: 13](#_Toc58170299)

[PS2: Nbody Simulation 16](#_Toc58170300)

[PS2a: Nbody Simulation (Part A) 16](#_Toc58170301)

[Discussion: 16](#_Toc58170302)

[Screenshot: 17](#_Toc58170303)

[Code: 18](#_Toc58170304)

[Makefile: 18](#_Toc58170305)

[main.cpp 18](#_Toc58170306)

[Universe.hpp 19](#_Toc58170307)

[Universe.cpp 20](#_Toc58170308)

[Celestialbody.hpp 21](#_Toc58170309)

[Celestialbody.cpp 22](#_Toc58170310)

[PS2b: Nbody Simulation (Part B) 24](#_Toc58170311)

[Discussion: 24](#_Toc58170312)

[Screenshot: 25](#_Toc58170313)

[Code: 26](#_Toc58170314)

[main.cpp 26](#_Toc58170315)

[Universe.hpp 28](#_Toc58170316)

[Universe.cpp 29](#_Toc58170317)

[Celestialbody.hpp 31](#_Toc58170318)

[Celestialbody.cpp 32](#_Toc58170319)

[PS3: Synthesizing a Plucked String 36](#_Toc58170320)

[PS3a: CircularBuffer implementation 36](#_Toc58170321)

[Discussion: 36](#_Toc58170322)

[Console Output (Running Boost test): 36](#_Toc58170323)

[Code: 36](#_Toc58170324)

[Makefile: 36](#_Toc58170325)

[test.cpp: 36](#_Toc58170326)

[CircularBuffer.h 38](#_Toc58170327)

[CircularBuffer.cpp 39](#_Toc58170328)

[PS3b: StringSound Implementation and SFML audio output 40](#_Toc58170329)

[Discussion: 40](#_Toc58170330)

[Screenshot: 41](#_Toc58170331)

[Makefile 41](#_Toc58170332)

[KSGuitarSim.cpp 41](#_Toc58170333)

[StringSound.h 43](#_Toc58170334)

[StringSound.cpp 44](#_Toc58170335)

[PS4: DNA Sequence Alignment 46](#_Toc58170336)

[Discussion: 46](#_Toc58170337)

[Console Output when run on example10.txt: 46](#_Toc58170338)

[Code: 47](#_Toc58170339)

[main.cpp 47](#_Toc58170340)

[ED.hpp 47](#_Toc58170341)

[ED.cpp 48](#_Toc58170342)

[PS5: Markov Model of Natural Language 51](#_Toc58170343)

[Discussion: 51](#_Toc58170344)

[Console Output: 51](#_Toc58170345)

[When running TextGenerator: 51](#_Toc58170346)

[When running Test: 51](#_Toc58170347)

[Code: 51](#_Toc58170348)

[Makefile 51](#_Toc58170349)

[TextGenerator.cpp 52](#_Toc58170350)

[test.cpp 52](#_Toc58170351)

[MModel.h 56](#_Toc58170352)

[MModel.cpp 56](#_Toc58170353)

[PS6: Kronos Time Clock 59](#_Toc58170354)

[Discussion: 59](#_Toc58170355)

[File Output (when run using device5\_intouch.log as the log file): 59](#_Toc58170356)

[Code: 60](#_Toc58170357)

[Makefile 60](#_Toc58170358)

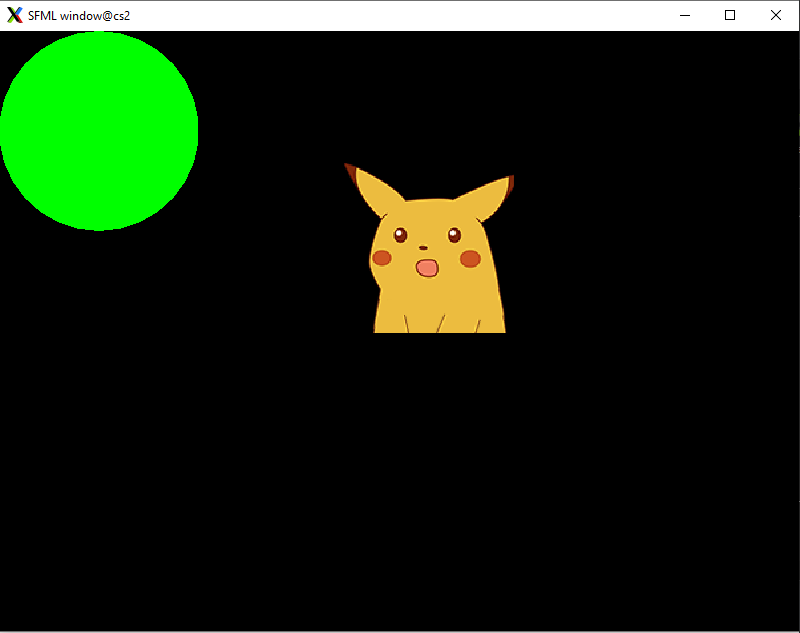
[PS6.cpp 60](#_Toc58170359)

# PS0: Hello World with SFML

### Discussion:

This project was an introduction to SFML. I was able to modify the SFML tutorial demo and include an additional sprite and have it respond to key presses on the keyboard. To do this, I utilized the SFML framework and a while loop that constantly checked for new events so that the program could modify the sprite. Through this project, I was able to become familiar with the SFML libraries and increase familiarity with event listeners.

### Screenshot:



### Code:

#### Main.cpp:

01**|** #include **<**SFML**/**Audio**.**hpp**>**

02**|** #include **<**SFML**/**Graphics**.**hpp**>**

03**|** int main**()**

04**|** **{**

05**|** // Create the main window

06**|** sf**::**RenderWindow window**(**sf**::**VideoMode**(**800**,** 600**),** "SFML window"**);**

07**|** //make green circle

08**|** sf**::**CircleShape shape**(**100.f**);**

09**|** shape**.**setFillColor**(**sf**::**Color**::**Green**);**

10**|** // Load a sprite to display

11**|** sf**::**Texture texture**;**

12**|** **if** **(!**texture**.**loadFromFile**(**"sprite.png"**))**

13**|** **return** EXIT\_FAILURE**;**

14**|** sf**::**Sprite sprite**(**texture**);**

15**|**

16**|** // showSprite variable to maintain toggle outside of game loop.

17**|** bool showSprite **=** **true;**

18**|**

19**|** // Start the game loop

20**|** **while** **(**window**.**isOpen**())**

21**|** **{**

22**|** // Process events

23**|** sf**::**Event event**;**

24**|** **while** **(**window**.**pollEvent**(**event**))**

25**|** **{**

26**|** //move sprite when arrow keys are pressed

27**|** **if(**event**.**type **==** sf**::**Event**::**KeyPressed**){**

28**|** **switch(**event**.**key**.**code**){**

29**|** **case** sf**::**Keyboard**::**Up**:**

30**|** **if(**showSprite**)**

31**|** sprite**.**move**(**0**,-**3**);**

32**|** **break;**

33**|** **case** sf**::**Keyboard**::**Down**:**

34**|** **if(**showSprite**)**

35**|** sprite**.**move**(**0**,**3**);**

36**|** **break;**

37**|** **case** sf**::**Keyboard**::**Right**:**

38**|** **if(**showSprite**)**

39**|** sprite**.**move**(**5**,**0**);**

40**|** **break;**

41**|** **case** sf**::**Keyboard**::**Left**:**

42**|** **if(**showSprite**)**

43**|** sprite**.**move**(-**5**,**0**);**

44**|** **break;**

45**|** **case** sf**::**Keyboard**::**Space**:**

46**|** showSprite **=** showSprite **==** **true** **?** **false** **:** **true;**

47**|** **break;**

48**|** **default:**

49**|** **break;**

50**|** **}**

51**|** **}**

52**|** //

53**|** **if(**event**.**type **==** sf**::**Event**::**MouseWheelScrolled**){**

54**|** **if(**showSprite**){**

55**|** **if(**event**.**mouseWheelScroll**.**delta **>** 0**)**

56**|** sprite**.**scale**(**1.1**,** 1.1**);**

57**|** **else{**

58**|** sprite**.**scale**(**0.9**,**0.9**);**

59**|** **}**

60**|** **}**

61**|** **}**

62**|** // Close window: exit

63**|** **if** **(**event**.**type **==** sf**::**Event**::**Closed**)**

64**|** window**.**close**();**

65**|** **}**

66**|** // Clear screen

67**|** window**.**clear**();**

68**|** //draw circle

69**|** window**.**draw**(**shape**);**

70**|** // Draw the sprite

71**|** **if(**showSprite**)**

72**|** window**.**draw**(**sprite**);**

73**|** // Update the window

74**|** window**.**display**();**

75**|** **}**

76**|** **return** EXIT\_SUCCESS**;**

77**|** **}**

78**|**

# PS1: Linear Feedback Shift Register

## PS1a: Linear Feedback Shift Register (Part A)

### Discussion:

In this portion of the project, I familiarized myself with a linear feedback shift register(LFSR). This allowed me to make pseudo-random numbers based off a seed and several hardcoded tap positions. To implement this LFSR, I simply used a string to keep track of the register and the binary XOR operation to find the next number based on the tap positions. I also used cppboost to test the program, so I became more familiar with unit testing.

### Console Output (Running Boost Test):

Running 3 test cases...

\*\*\* No errors detected

### Code:

#### Makefile:

01| CC **=** g++

02| CFLAGS **=** -std=c++11 -c -g -Og -Wall -Werror -pedantic

03| OBJ **=** test.o

04| DEPS **=** FibLFSR.h

05| LIBS **=** FibLFSR.cpp

06| EXE **=** boosttest

07|

08| all**:** $(OBJ)

09| $(CC) $(OBJ) -o $(EXE) $(LIBS)

10|

11| %.o**:** %.cpp $(DEPS)

12| $(CC) $(CFLAGS) -o $@ $<

13|

14| clean**:**

15| rm $(OBJ) $(EXE)

16|

#### test.cpp

01**|** #include **<**iostream**>**

02**|** #include **<**string**>**

03**|** #include "FibLFSR.h"

04**|** #define BOOST\_TEST\_DYN\_LINK

05**|** #define BOOST\_TEST\_MODULE Main

06**|** #include **<**boost**/**test**/**included**/**unit\_test**.**hpp**>**

07**|**

08**|** //initial test in file.

09**|** BOOST\_AUTO\_TEST\_CASE**(**sixteenBitsThreeTaps**)** **{**

10**|**

11**|** FibLFSR l**(**"1011011000110110"**);**

12**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

13**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

14**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

15**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 1**);**

16**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 1**);**

17**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

18**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

19**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 1**);**

20**|**

21**|** FibLFSR l2**(**"1011011000110110"**);**

22**|** BOOST\_REQUIRE**(**l2**.**generate**(**9**)** **==** 51**);**

23**|** **}**

24**|**

25**|** BOOST\_AUTO\_TEST\_CASE**(**constructors**){**

26**|** FibLFSR normalConstructorTest**(**"1011011000110110"**);** //makes a constructor with 16 bits, which should work

27**|**

28**|** boost**::**test\_tools**::**output\_test\_stream output**;** //redirects the constructors overloaded << to the boost test stream.

29**|** output **<<** normalConstructorTest**;**

30**|** BOOST\_REQUIRE**(**output**.**is\_equal**(**"1011011000110110"**));** //checks to see if output matches what it is supposed to be, which is the parameter for the constructor

31**|**

32**|** BOOST\_CHECK\_THROW**(**FibLFSR normalConstructorTest**(**"10110110001"**),** std**::**length\_error**);** //makes a constructor with 11 bits, which should throw an exception due to length

33**|**

34**|** BOOST\_CHECK\_THROW**(**FibLFSR normalConstructorTest**(**"10110110001000000"**),** std**::**length\_error**);** //makes a constructor with 17 bits, which should throw an exception due to length

35**|**

36**|** BOOST\_CHECK\_THROW**(**FibLFSR normalConstructorTest**(**"101101100011011A"**),** std**::**invalid\_argument**);** //makes a constructor with 11 bits, which should throw an exception due to invalid argument (bits aren't 0's and 1's)

37**|** **}**

38**|**

39**|** //more extensive generate tests

40**|** BOOST\_AUTO\_TEST\_CASE**(**moreGenerate**)** **{**

41**|**

42**|** FibLFSR generatorTest**(**"1010101111000111"**);**

43**|**

44**|** //just checks generate function at different k values.

45**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**5**)** **==** 4**);**

46**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**7**)** **==** 32**);**

47**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**11**)** **==** 893**);**

48**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**13**)** **==** 3929**);**

49**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**4**)** **==** 8**);**

50**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**15**)** **==** 7711**);**

51**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**16**)** **==** 21874**);**

52**|** BOOST\_CHECK\_THROW**(**generatorTest**.**generate**(**20**),** std**::**invalid\_argument**);** // should throw an invalid argument exception because k is larger than the 16 bit register.

53**|** BOOST\_CHECK\_THROW**(**generatorTest**.**generate**(-**5**),** std**::**invalid\_argument**);** // should throw an invalid argument exception because k is negative

54**|** **}**

55**|**

#### fibLFSR.h

01**|** #include **<**iostream**>**

02**|** #include **<**string**>**

03**|** #include "FibLFSR.h"

04**|** #define BOOST\_TEST\_DYN\_LINK

05**|** #define BOOST\_TEST\_MODULE Main

06**|** #include **<**boost**/**test**/**included**/**unit\_test**.**hpp**>**

07**|**

08**|** //initial test in file.

09**|** BOOST\_AUTO\_TEST\_CASE**(**sixteenBitsThreeTaps**)** **{**

10**|**

11**|** FibLFSR l**(**"1011011000110110"**);**

12**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

13**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

14**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

15**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 1**);**

16**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 1**);**

17**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

18**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 0**);**

19**|** BOOST\_REQUIRE**(**l**.**step**()** **==** 1**);**

20**|**

21**|** FibLFSR l2**(**"1011011000110110"**);**

22**|** BOOST\_REQUIRE**(**l2**.**generate**(**9**)** **==** 51**);**

23**|** **}**

24**|**

25**|** BOOST\_AUTO\_TEST\_CASE**(**constructors**){**

26**|** FibLFSR normalConstructorTest**(**"1011011000110110"**);** //makes a constructor with 16 bits, which should work

27**|**

28**|** boost**::**test\_tools**::**output\_test\_stream output**;** //redirects the constructors overloaded << to the boost test stream.

29**|** output **<<** normalConstructorTest**;**

30**|** BOOST\_REQUIRE**(**output**.**is\_equal**(**"1011011000110110"**));** //checks to see if output matches what it is supposed to be, which is the parameter for the constructor

31**|**

32**|** BOOST\_CHECK\_THROW**(**FibLFSR normalConstructorTest**(**"10110110001"**),** std**::**length\_error**);** //makes a constructor with 11 bits, which should throw an exception due to length

33**|**

34**|** BOOST\_CHECK\_THROW**(**FibLFSR normalConstructorTest**(**"10110110001000000"**),** std**::**length\_error**);** //makes a constructor with 17 bits, which should throw an exception due to length

35**|**

36**|** BOOST\_CHECK\_THROW**(**FibLFSR normalConstructorTest**(**"101101100011011A"**),** std**::**invalid\_argument**);** //makes a constructor with 11 bits, which should throw an exception due to invalid argument (bits aren't 0's and 1's)

37**|** **}**

38**|**

39**|** //more extensive generate tests

40**|** BOOST\_AUTO\_TEST\_CASE**(**moreGenerate**)** **{**

41**|**

42**|** FibLFSR generatorTest**(**"1010101111000111"**);**

43**|**

44**|** //just checks generate function at different k values.

45**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**5**)** **==** 4**);**

46**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**7**)** **==** 32**);**

47**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**11**)** **==** 893**);**

48**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**13**)** **==** 3929**);**

49**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**4**)** **==** 8**);**

50**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**15**)** **==** 7711**);**

51**|** BOOST\_REQUIRE**(**generatorTest**.**generate**(**16**)** **==** 21874**);**

52**|** BOOST\_CHECK\_THROW**(**generatorTest**.**generate**(**20**),** std**::**invalid\_argument**);** // should throw an invalid argument exception because k is larger than the 16 bit register.

53**|** BOOST\_CHECK\_THROW**(**generatorTest**.**generate**(-**5**),** std**::**invalid\_argument**);** // should throw an invalid argument exception because k is negative

54**|** **}**

55**|**

#### fibLFSR.cpp:

01**|** #include **<**iostream**>**

02**|** #include **<**string**>**

03**|** #include **<**stdexcept**>**

04**|** #include "FibLFSR.h"

05**|**

06**|** FibLFSR**::**FibLFSR**(**std**::**string seed**){**

07**|** **if(**seed**.**length**()** **==** 16**){** //makes sure that the length of the seed is 16/

08**|** **for(**char**&** c **:** seed**){**

09**|** **if(**c **!=** '0' **&&** c **!=** '1'**){** //checks the characters to make sure they are 0's and 1's

10**|** **throw** std**::**invalid\_argument**(**"this is not a string of 0's and 1's"**);** //exception thrown if characters are not 0's and 1's

11**|** **}**

12**|** **}**

13**|** //sets the seed to bitString variable.

14**|** bitString **=** seed**;**

15**|** **}**

16**|** **else{**

17**|** **throw** std**::**length\_error**(**"length is not 16 bits."**);**

18**|** **}**

19**|** **}**

20**|**

21**|** int FibLFSR**::**step**(){**

22**|** //xor equation of ((left most XOR 13th bit) XOR 12th bit)XOR 10th bit). since the first two are both characters, i'm not "-'0'" them to get their value as an int because it wouldn't change the XOR value. However, XORing 0000000001 with '0' would give a wildly different value, so I "-'0'" to change their value to their numerical value at the bit level.

23**|** int add **=** **((**bitString**.**at**(**0**)^(**bitString**.**at**(**2**))^(**bitString**.**at**(**3**)-**'0'**))^(**bitString**.**at**(**5**)-**'0'**));**

24**|** bitString **=** bitString**.**substr**(**1**)** **+** std**::**to\_string**(**add**);** //add the result while ditching the most significant bit.

25**|** // std::cout << bitString << " " << add << std::endl;

26**|** **return** add**;**

27**|** **}**

28**|**

29**|** int FibLFSR**::**generate**(**int k**){**

30**|** **if(**k**>**16 **||** k**<**0**){** // checks to make sure we aren't checking more bits than the register is. also negative numbers make no sense, so those are checked for as well.

31**|** **throw** std**::**invalid\_argument**(** "k is larger than register or negative"**);** //exception thrown if k is larger than the register or negative.

32**|** **}**

33**|** int count **=** 0**;**

34**|** **for(**int i **=** 0**;** i**<**k**;** i**++){**

35**|** count **=** count**\***2 **+** step**();**

36**|** **}**

37**|** // std::cout << bitString << " " << count << std::endl;

38**|** **return** count**;**

39**|** **}**

40**|**

41**|** //prints out the current bitString saved in LFSR

42**|** std**::**ostream**&** **operator<<** **(**std**::**ostream **&**out**,** FibLFSR **&**a**)**

43**|** **{**

44**|** out **<<** a**.**bitString**;**

45**|** **return** out**;**

46**|** **}**

47**|**

## PS1b Linear Feedback Shift Register (Part B)

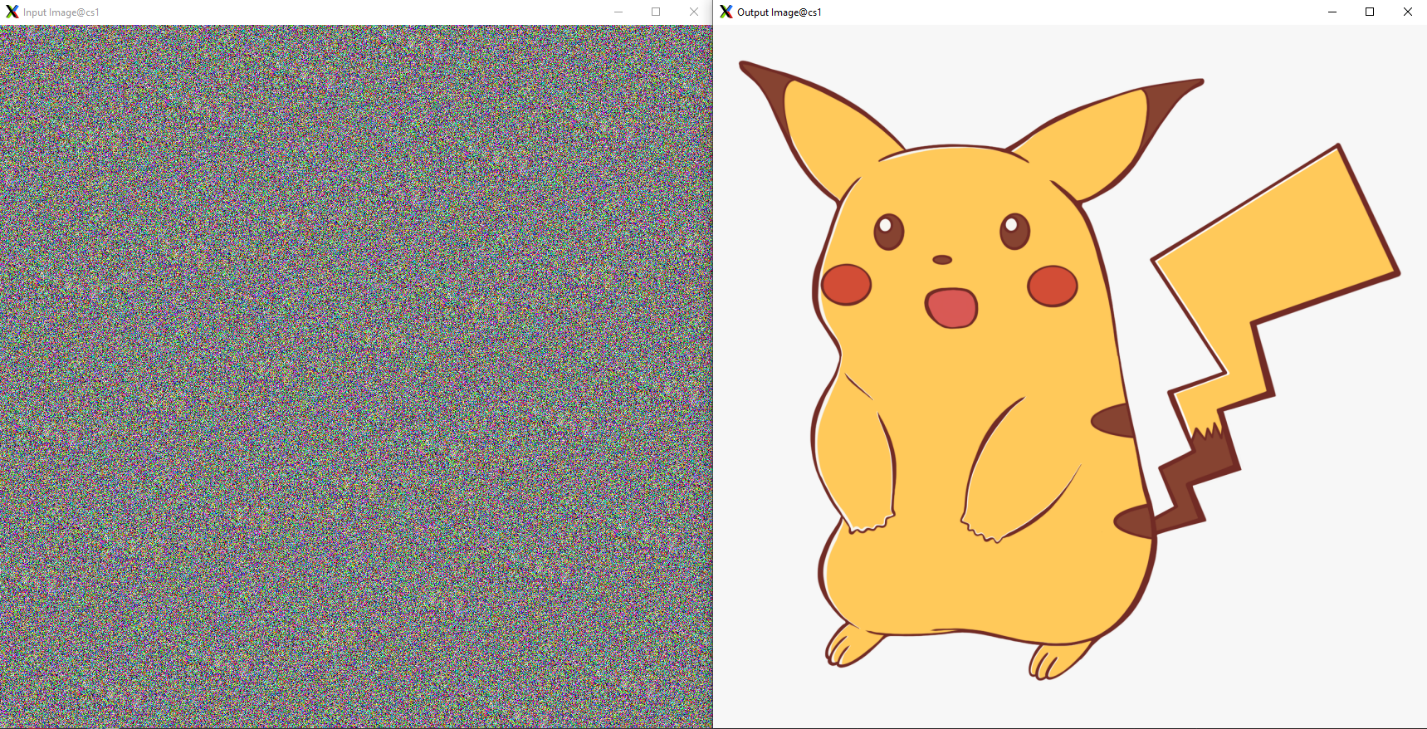
### Discussion:

I extended the LFSR to be able to encode image files so that they become static if they are not decoded with the right algorithm. I integrated SFML, which I had utilized in PS0, to open the resulting encoded/decoded image in its own window. The resulting image is also saved as a png file. This project helped me become more familiar with the encoding process and how we scramble our data from others.

### Screenshots:

#### Encoding:

#### Decoding:



### Code:

Note: Since PS1b builds off PS1a, reused files will not be included in the code section if they were not modified.

#### Makefile:

01| CC **=** g++

02| CFLAGS **=** -std=c++11 -c -g -Og -Wall -Werror -pedantic

03| OBJ **=** PhotoMagic.o

04| DEPS **=** FibLFSR.h

05| LIBS **=** FibLFSR.cpp -lsfml-graphics -lsfml-window -lsfml-system

06| EXE **=** PhotoMagic

07|

08| all**:** $(OBJ)

09| $(CC) $(OBJ) -o $(EXE) $(LIBS)

10|

11| %.o**:** %.cpp $(DEPS)

12| $(CC) $(CFLAGS) -o $@ $<

13|

14| clean**:**

15| rm $(OBJ) $(EXE)

16|

#### Photomagic.cpp:

001**|** #include **<**SFML**/**System**.**hpp**>**

002**|** #include **<**SFML**/**Window**.**hpp**>**

003**|** #include **<**SFML**/**Graphics**.**hpp**>**

004**|** #include "FibLFSR.h"

005**|**

006**|** sf**::**Image transform**(**sf**::**Image**&** image**,** FibLFSR**\*** randomizer**);**

007**|**

008**|** sf**::**Image transform**(**sf**::**Image**&** image**,** FibLFSR**\*** randomizer**){**

009**|** sf**::**Color p**;**

010**|** sf**::**Vector2u size **=** image**.**getSize**();**

011**|** // Randomize the bits in the image

012**|** **for(**int x**=** 0**;** x **<** **(**signed**)**size**.**x**;** x**++){**

013**|** **for(**int y **=** 0**;** y **<** **(**signed**)**size**.**y**;** y**++){**

014**|** // Get the current pixel from the input image

015**|** p **=** image**.**getPixel**(**x**,** y**);**

016**|**

017**|** // XOR the pixels

018**|** p**.**r **=** p**.**r **^** randomizer**->**generate**(**8**);**

019**|** p**.**g **=** p**.**g **^** randomizer**->**generate**(**8**);**

020**|** p**.**b **=** p**.**b **^** randomizer**->**generate**(**8**);**

021**|**

022**|** // Modify just the output image

023**|** image**.**setPixel**(**x**,** y**,** p**);**

024**|** **}**

025**|** **}**

026**|** **return** image**;**

027**|** **}**

028**|**

029**|**

030**|** int main**(**int argc**,** char**\*** argv**[]){**

031**|** //check that 4 command line inputs were entered

032**|** **if(**argc **!=** 4**){**

033**|** **throw** std**::**length\_error **(**"There are not 4 command line arguments. Please use ./executable inputImage outputImage seed"**);**

034**|** **}**

035**|**

036**|** // Create the input image

037**|** sf**::**Image inputImage**;**

038**|** **if** **(!**inputImage**.**loadFromFile**(**argv**[**1**])){**

039**|** **throw** std**::**invalid\_argument**(**"This is not a valid image file"**);**

040**|** **}**

041**|**

042**|** // Create the output image

043**|** sf**::**Image outputImage**;**

044**|** **if** **(!**outputImage**.**loadFromFile**(**argv**[**1**])){**

045**|** **throw** std**::**invalid\_argument**(**"This is not a valid image file"**);**

046**|** **}**

047**|**

048**|** //Create the FibLFSR randomizer

049**|** FibLFSR randomizer**(**argv**[**3**]);**

050**|**

051**|** // Create Vector24 variable to get dimensions for windows

052**|** sf**::**Vector2u size **=** inputImage**.**getSize**();**

053**|**

054**|** // Modify the output file and save it

055**|** **if** **(!(**transform**(**outputImage**,** **&**randomizer**).**saveToFile**(**argv**[**2**]))){**

056**|** **return** **-**1**;**

057**|** **}**

058**|**

059**|** //make window for original image

060**|** sf**::**RenderWindow original**(**sf**::**VideoMode**(**size**.**x**,** size**.**y**),** "Input Image"**);**

061**|**

062**|** //make window for transformed image

063**|** sf**::**RenderWindow transformed**(**sf**::**VideoMode**(**size**.**x**,** size**.**y**),** "Output Image"**);**

064**|**

065**|**

066**|** //Create texture for input for original

067**|** sf**::**Texture inputTexture**;**

068**|** inputTexture**.**loadFromImage**(**inputImage**);**

069**|**

070**|** //Create sprite from texture for original

071**|** sf**::**Sprite inputSprite**;**

072**|** inputSprite**.**setTexture**(**inputTexture**);**

073**|**

074**|** //Create texture for input for transformed

075**|** sf**::**Texture outputTexture**;**

076**|** outputTexture**.**loadFromImage**(**outputImage**);**

077**|**

078**|** //Create sprite from texture for transformed

079**|** sf**::**Sprite outputSprite**;**

080**|** outputSprite**.**setTexture**(**outputTexture**);**

081**|**

082**|** // Start the game loop

083**|** **while** **(**original**.**isOpen**()** **&&** transformed**.**isOpen**())**

084**|** **{**

085**|** sf**::**Event event**;**

086**|**

087**|** **while** **(**original**.**pollEvent**(**event**))**

088**|** **{**

089**|** **if** **(**event**.**type **==** sf**::**Event**::**Closed**)**

090**|** **{**

091**|** original**.**close**();**

092**|** **}**

093**|** **}**

094**|**

095**|** **while** **(**transformed**.**pollEvent**(**event**))**

096**|** **{**

097**|** **if** **(**event**.**type **==** sf**::**Event**::**Closed**)**

098**|** **{**

099**|** transformed**.**close**();**

100**|** **}**

101**|** **}**

102**|**

103**|** original**.**clear**();**

104**|** original**.**draw**(**inputSprite**);** // Input image

105**|** original**.**display**();**

106**|**

107**|** transformed**.**clear**();**

108**|** transformed**.**draw**(**outputSprite**);** // Output image

109**|** transformed**.**display**();**

110**|** **}**

111**|**

112**|**

113**|** **return** EXIT\_SUCCESS**;**

114**|** **}**

115**|**

# PS2: Nbody Simulation

## PS2a: Nbody Simulation (Part A)

### Discussion:

In this project, I used SFML to create a window to show the planets given to me for the assignment. It reads the information from a planets.txt file. During this project, I learned how to redirect a file as input using terminal. Previously, I had not known of this and had been hand typing any inputs that I needed. A key algorithm I used was to calculate at what distance I should be showing the planets. I first used the dimensions of the window that I made in SFML as a benchmark (halved it because the planets distance is from the center of the universe). Then, based on the radius of the universe and the distance of a given body form the center, I made a ratio to use so that I knew where to place the planet. That ratio was then converted to pixels so that I could place it on the window via SFML. I also used a vector to hold all the bodies in the universe class so that all the celestial bodies were all in one place and very intuitive to access as well as iterate over.

### Screenshot:

### Code:

#### Makefile:

01| CC **=** g++

02| CFLAGS **=** -std=c++14 -c -g -Og -Wall -Werror -pedantic

03| OBJ **=** Universe.o CelestialBody.o main.o

04| LIBS **=** -lsfml-graphics -lsfml-system -lsfml-window

05| EXE **=** NBody

06|

07| all**:** NBody

08|

09| NBody**:** $(OBJ)

10| $(CC) $(OBJ) -o $(EXE) $(LIBS)

11|

12| CelestialBody.o**:** CelestialBody.cpp CelestialBody.hpp

13| $(CC) $(CFLAGS) -o $@ $<

14|

15| Universe.o**:** Universe.cpp Universe.hpp

16| $(CC) $(CFLAGS) -o $@ $<

17|

18| main.o**:** main.cpp Universe.hpp

19| $(CC) $(CFLAGS) -o $@ $<

20|

21| clean**:**

22| rm \*.o

23| rm $(EXE)

24|

#### main.cpp

01**|** #include "Universe.hpp"

02**|** #include **<**SFML**/**Graphics**.**hpp**>**

03**|**

04**|** int main**(){**

05**|** int numPlanets**;**

06**|** float radius**;**

07**|** std**::**cin **>>** numPlanets **>>** radius**;**

08**|**

09**|**

10**|** universe galaxy **=** universe**(**numPlanets**,** radius**);**

11**|**

12**|** sf**::**RenderWindow window**(**sf**::**VideoMode**(**defaultWindowSize**.**x**,** defaultWindowSize**.**y**),** "The Galaxy"**);**

13**|**

14**|** //create image for background

15**|** sf**::**Image background**;**

16**|**

17**|** //if load fails, throw error

18**|** **if(!**background**.**loadFromFile**(**"space.png"**)){**

19**|** **throw** std**::**invalid\_argument**(**"no file for celestialImage"**);**

20**|** **}**

21**|**

22**|** //creating texture for background

23**|** sf**::**Texture backgroundTexture**;**

24**|** backgroundTexture**.**loadFromImage**(**background**);**

25**|**

26**|** //creating sprite for background

27**|** sf**::**Sprite backgroundSprite**;**

28**|** backgroundSprite**.**setTexture**(**backgroundTexture**);**

29**|**

30**|** **while** **(**window**.**isOpen**()){**

31**|** // Process events

32**|** sf**::**Event event**;**

33**|** **while** **(**window**.**pollEvent**(**event**)){**

34**|** // Close window: exit

35**|** **if** **(**event**.**type **==** sf**::**Event**::**Closed**){**

36**|** window**.**close**();**

37**|** **}**

38**|** **}**

39**|** window**.**clear**();**

40**|**

41**|** //draw background

42**|** window**.**draw**(**backgroundSprite**);**

43**|**

44**|**

45**|** //function in universe class to draw all celestialBodies in universe object uses target.draw() in the function.

46**|** galaxy**.**draw**(**window**);**

47**|**

48**|** window**.**display**();**

49**|** **}**

50**|**

51**|**

52**|** **return** 0**;**

53**|** **}**

54**|**

#### Universe.hpp

01**|** #ifndef UNIVERSE\_H

02**|** #define UNIVERSE\_H

03**|** #include "CelestialBody.hpp"

04**|** #include **<**vector**>**

05**|** #include **<**memory**>**

06**|**

07**|** class universe**{**

08**|** public**:**

09**|** //default constructor

10**|** universe**();**

11**|**

12**|** //parameter constructor

13**|** universe**(**float radius**);**

14**|**

15**|** //constructor for taking in input from txt doc.

16**|** universe**(**int numOfCelestialBodies**,** float radius**);**

17**|**

18**|** //set radius if needed

19**|** void set\_radius**(**float radius**);**

20**|**

21**|** //calculate positions based on window size.

22**|** void set\_position**(**sf**::**Vector2u windowSize **=** defaultWindowSize**);**

23**|**

24**|** // Draw method for universe

25**|** void virtual draw**(**sf**::**RenderTarget**&** target**)** const**;**

26**|**

27**|** //for testing

28**|** void print **();**

29**|**

30**|** private**:**

31**|** double universeRadius**;**

32**|**

33**|** //using a vector of shared\_ptr to hold celestialBodies

34**|** std**::**vector**<**std**::**shared\_ptr**<**celestialBody**>>** celestialBodies**;**

35**|** **};**

36**|**

37**|**

38**|** #endif //UNIVERSE\_H

#### Universe.cpp

01**|** #include "Universe.hpp"

02**|**

03**|** universe**::**universe**(){**

04**|** **return;**

05**|** **}**

06**|**

07**|** universe**::**universe**(**float radius**){**

08**|** universeRadius **=** radius**;**

09**|** **}**

10**|**

11**|** //constructor for taking in input from txt doc.

12**|** universe**::**universe**(**int numOfCelestialBodies**,** float radius**){**

13**|**

14**|** //for loop to collect data for the number of celestial bodies.

15**|** **for(**int i**=**0**;** i**<**numOfCelestialBodies**;** i**++){**

16**|** std**::**shared\_ptr**<**celestialBody**>** temp **=** std**::**make\_shared**<**celestialBody**>** **();**

17**|** std**::**cin **>>** **\***temp**;**

18**|**

19**|** //calculate initial position for all bodies as we loop through

20**|** temp**->**set\_position**(**radius**);**

21**|**

22**|** //add the body to the vector

23**|** celestialBodies**.**push\_back**(**temp**);**

24**|**

25**|** //used for testing

26**|** //std::cout << \*temp;

27**|** **}**

28**|** **}**

29**|**

30**|** //sets the radius if we need to.

31**|** void universe**::**set\_radius**(**float radius**){**

32**|** universeRadius **=** radius**;**

33**|** **return;**

34**|** **}**

35**|**

36**|** //uses an iterator to go through and set position of each celestialBody

37**|** void universe**::**set\_position**(**sf**::**Vector2u windowSize**){**

38**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

39**|** **(\*\***it**).**set\_position**(**universeRadius**,** windowSize**);**

40**|** **}**

41**|** **}**

42**|**

43**|**

44**|** //uses an iterator to go through and draw each celestialBody

45**|** void universe**::**draw**(**sf**::**RenderTarget**&** target**)** const**{**

46**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

47**|** target**.**draw**(\*\***it**);**

48**|** **}**

49**|** **}**

50**|**

51**|** //print function for testing.

52**|** void universe**::**print **(){**

53**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

54**|** std**::**cout **<<** **\*\***it**;**

55**|** **}**

56**|** **}**

57**|**

58**|**

#### Celestialbody.hpp

01**|** #ifndef CELESTIALBODY\_H

02**|** #define CELESTIALBODY\_H

03**|**

04**|** #include **<**iostream**>**

05**|** #include **<**string**>**

06**|** #include **<**vector**>**

07**|** #include **<**SFML**/**System**.**hpp**>**

08**|** #include **<**SFML**/**Window**.**hpp**>**

09**|** #include **<**SFML**/**Graphics**.**hpp**>**

10**|**

11**|**

12**|** // Constants for the window size.

13**|** const sf**::**Vector2u defaultWindowSize**(**1000**,** 1000**);**

14**|**

15**|** class celestialBody**:** public sf**::**Drawable**{**

16**|** public**:**

17**|**

18**|**

19**|** // Constructors

20**|** celestialBody**();**

21**|**

22**|** celestialBody**(**double posX**,** double posY**,** double velX**,** double velY**,** double mass**,** std**::**string file**);**

23**|**

24**|**

25**|** void set\_position**(**float radius**,** sf**::**Vector2u windowSize **=** defaultWindowSize**);** // Sets the planets positions will default to 1000x1000 screen size

26**|**

27**|** // Overridden operator >> for inputing from a file

28**|** friend std**::**istream**&** **operator>>** **(**std**::**istream **&**input**,** celestialBody **&**body**);**

29**|**

30**|** // Overriddden operator << for debugging

31**|** friend std**::**ostream**&** **operator<<** **(**std**::**ostream **&**output**,** celestialBody **&**body**);**

32**|**

33**|**

34**|**

35**|** private**:**

36**|**

37**|** // Draw method

38**|** void virtual draw**(**sf**::**RenderTarget**&** target**,** sf**::**RenderStates states**)** const**;**

39**|**

40**|** // Member variables

41**|** double positionX**,** positionY**,** velocityX**,** velocityY**,** celestialMass**;**

42**|** std**::**string fileName**;**

43**|**

44**|** // Image related objects

45**|** sf**::**Image celestialImage**;**

46**|** sf**::**Sprite celestialSprite**;**

47**|** sf**::**Texture celestialTexture**;**

48**|** **};**

49**|**

50**|** #endif //CELESTIALBODY\_H

#### Celestialbody.cpp

01**|** #include "CelestialBody.hpp"

02**|**

03**|** //default constructor doesn't need to do anything, since the >> operator will set everything

04**|** celestialBody**::**celestialBody**(){**

05**|** **return;**

06**|** **}**

07**|**

08**|** //sets values if given

09**|** celestialBody**::**celestialBody**(**double posX**,** double posY**,** double velX**,** double velY**,** double mass**,** std**::**string file**){**

10**|** positionX **=** posX**;**

11**|** positionY **=** posY**;**

12**|** velocityX **=** velX**;**

13**|** velocityY **=** velY**;**

14**|** celestialMass **=** mass**;**

15**|** fileName **=** file**;**

16**|**

17**|** **if(**celestialImage**.**loadFromFile**(**fileName**)){**

18**|** **return;**

19**|** **}**

20**|**

21**|** celestialTexture**.**loadFromImage**(**celestialImage**);**

22**|** celestialSprite**.**setTexture**(**celestialTexture**);**

23**|** **}**

24**|**

25**|**

26**|** void celestialBody**::**set\_position**(**float radius**,** sf**::**Vector2u windowSize**){** // Sets the planets positions based on window size, will default to 1000,1000.

27**|**

28**|** /\*

29| The math for this is obtain a ratio for the size of the screen. then multiplies it by distance from center of universe(which is half of the screen) to figure out where in relation to the center the celestialBody is. Then offsets both x and y based of coordinates of center of the universe.

30| \*/

31**|** positionX **=** **(** **(**positionX **/** radius**)** **\*** **(** windowSize**.**x **/** 2**)** **)** **+** **(** windowSize**.**x **/** 2**);**

32**|** positionY **=** **(** **(**positionY **/** radius**)** **\*** **(** windowSize**.**y **/** 2**)** **)** **+** **(** windowSize**.**y **/** 2**);**

33**|**

34**|** //apply position to sprite.

35**|** celestialSprite**.**setPosition**(**positionX**,** positionY**);**

36**|** **}**

37**|**

38**|** // Overridden operator >> for inputing from a file

39**|** std**::**istream**&** **operator>>** **(**std**::**istream **&**input**,** celestialBody **&**body**){**

40**|** //read in all the information

41**|** input **>>** body**.**positionX **>>** body**.**positionY **>>** body**.**velocityX **>>** body**.**velocityY **>>** body**.**celestialMass **>>** body**.**fileName**;**

42**|**

43**|** //set the images and positions, like the parameter constructor.

44**|** **if(!**body**.**celestialImage**.**loadFromFile**(**body**.**fileName**)){**

45**|** **throw** std**::**invalid\_argument**(**"no file for celestialImage"**);**

46**|** **}**

47**|**

48**|** body**.**celestialTexture**.**loadFromImage**(**body**.**celestialImage**);**

49**|** body**.**celestialSprite**.**setTexture**(**body**.**celestialTexture**);**

50**|**

51**|** **return** input**;**

52**|** **}**

53**|**

54**|** //overridden operator << for testing purposes.

55**|** std**::**ostream**&** **operator<<** **(**std**::**ostream **&**output**,** celestialBody **&**body**)**

56**|** **{**

57**|** // For debugging, output all the data stored in the object.

58**|** output **<<** "File name: " **<<** body**.**fileName **<<** std**::**endl **<<** "X position: " **<<** body**.**positionX **<<** std**::**endl **<<** "Y position: " **<<** body**.**positionY **<<** std**::**endl **<<** "X velocity: " **<<** body**.**velocityX **<<** std**::**endl **<<** "Y velocity: " **<<** body**.**velocityY **<<** std**::**endl **<<** "Mass: " **<<** body**.**celestialMass **<<** std**::**endl**;**

59**|**

60**|** **return** output**;**

61**|** **}**

62**|**

63**|** // Drawable method

64**|** void celestialBody**::**draw**(**sf**::**RenderTarget**&** target**,** sf**::**RenderStates states**)** const

65**|** **{**

66**|** // draw the image

67**|** target**.**draw**(**celestialSprite**);**

68**|** **}**

## PS2b: Nbody Simulation (Part B)

### Discussion:

In this portion of the project, I added motion to the universe that I had created in the previous portion of the project. I also added music to play in the background and a timer to show elapsed time. After the window is closed, the program also outputs the state of the Universe to the console. To simulate the motion, I had to create a function to calculate the positions of each Celestialbody and update them in the SFML window. I used Newton’s law of universal gravitation to calculate the new positions of each Celestialbody and then used the same ratio as in part A to figure out where the Celestialbodies should be placed on the screen. Additionally, the elapsed time in the simulation goes by steps instead of literally second by second. Each tic of the elapsed time is equivalent to the value of the step variable in seconds. This is known as the leapfrog finite difference approximation scheme. By using the leapfrog finite difference approximation scheme, we can relatively smoothly approximate the positions as time is “elapsed.”

### Screenshot:

### Code:

Note: Since PS2b builds off PS2a, reused files will not be included in the code section if they were not modified.

#### main.cpp

001**|** #include "Universe.hpp"

002**|** #include **<**SFML**/**Graphics**.**hpp**>**

003**|** #include **<**SFML**/**Audio**.**hpp**>**

004**|**

005**|** int main**(**int argc**,** char**\*** argv**[]){**

006**|**

007**|** std**::**string**::**size\_type sz**;** //for stod use

008**|**

009**|** //assign the parameters to doubles.

010**|** double T **=** std**::**stod**(**argv**[**1**],** **&**sz**);**

011**|** double deltaT **=** std**::**stod**(**argv**[**2**],** **&**sz**);**

012**|** double timeElapsed **=** 0**;**

013**|**

014**|** //std::cout << T << " " << deltaT; for testing

015**|** int numPlanets**;**

016**|** float radius**;**

017**|** std**::**cin **>>** numPlanets **>>** radius**;**

018**|**

019**|**

020**|** universe galaxy **=** universe**(**numPlanets**,** radius**);**

021**|**

022**|**

023**|** //galaxy.step(deltaT);

024**|** //std::cout << "HIHIHIHIHIHIHI" << std::endl;

025**|** //galaxy.step(deltaT);

026**|**

027**|** sf**::**RenderWindow window**(**sf**::**VideoMode**(**defaultWindowSize**.**x**,** defaultWindowSize**.**y**),** "The Galaxy"**);**

028**|**

029**|** window**.**setFramerateLimit**(**60**);**

030**|**

031**|** //create image for background

032**|** sf**::**Image background**;**

033**|**

034**|** //if load fails, throw error

035**|** **if(!**background**.**loadFromFile**(**"space.png"**)){**

036**|** **throw** std**::**invalid\_argument**(**"no file for celestialImage"**);**

037**|** **}**

038**|**

039**|** //creating texture for background

040**|** sf**::**Texture backgroundTexture**;**

041**|** backgroundTexture**.**loadFromImage**(**background**);**

042**|**

043**|** //creating sprite for background

044**|** sf**::**Sprite backgroundSprite**;**

045**|** backgroundSprite**.**setTexture**(**backgroundTexture**);**

046**|**

047**|** //create text.

048**|** sf**::**Font timeFont**;**

049**|** timeFont**.**loadFromFile**(**"arial.ttf"**);**

050**|**

051**|** sf**::**Text timeDisplay**;**

052**|** timeDisplay**.**setFont**(**timeFont**);**

053**|**

054**|** //make time legible

055**|** timeDisplay**.**setCharacterSize**(**14**);**

056**|** timeDisplay**.**setFillColor**(**sf**::**Color**::**White**);**

057**|**

058**|** //create music (which I did not make. I found it royalty free. see readme.)

059**|** sf**::**Music music**;**

060**|** **if(!**music**.**openFromFile**(**"Joey Pecoraro - Your Favorite Place.flac"**)){**

061**|** **throw** std**::**invalid\_argument**(**"no file for music"**);**

062**|** **}**

063**|**

064**|** //play music. This may cause some delay due to needing to open it.

065**|** music**.**play**();**

066**|** //std::cout << "music playing: " << (music.getStatus() == sf::SoundSource::Status::Playing) <<std::endl; for testing if music is actually playing since i can't hear it off of the uml server.

067**|**

068**|** **while** **(**window**.**isOpen**()){**

069**|** // Process events

070**|** sf**::**Event event**;**

071**|** **while** **(**window**.**pollEvent**(**event**)){**

072**|** // Close window: exit

073**|** **if** **(**event**.**type **==** sf**::**Event**::**Closed**){**

074**|** window**.**close**();**

075**|** **}**

076**|** **}**

077**|** window**.**clear**();**

078**|**

079**|**

080**|** //draw background

081**|** window**.**draw**(**backgroundSprite**);**

082**|**

083**|**

084**|**

085**|** //function in universe class to draw all celestialBodies in universe object uses target.draw() in the function.

086**|** galaxy**.**draw**(**window**);**

087**|**

088**|** //check to make sure elapsed time does not go over T.

089**|** **if(**timeElapsed**+** deltaT **<=** T**){**

090**|** galaxy**.**step**(**deltaT**);**

091**|** galaxy**.**set\_position**(**window**.**getSize**());**

092**|** timeElapsed **+=** deltaT**;**

093**|** timeDisplay**.**setString**(**"Elapsed Time: " **+** std**::**to\_string**(**timeElapsed**));**

094**|** **}**

095**|**

096**|** //draw time Display onto window.

097**|** window**.**draw**(**timeDisplay**);**

098**|**

099**|** //std::cout << ++count<< std::endl;

100**|** window**.**display**();**

101**|** **}**

102**|**

103**|** galaxy**.**print**();** //prints final state of the universe.

104**|**

105**|** **return** 0**;**

106**|** **}**

#### Universe.hpp

01**|** #ifndef UNIVERSE\_H

02**|** #define UNIVERSE\_H

03**|**

04**|** #include "CelestialBody.hpp"

05**|** #include **<**vector**>**

06**|** #include **<**memory**>**

07**|** #include **<**math**.**h**>**

08**|**

09**|** class universe**{**

10**|** public**:**

11**|** //default constructor

12**|** universe**();**

13**|**

14**|** //parameter constructor

15**|** universe**(**float radius**);**

16**|**

17**|** //constructor for taking in input from txt doc.

18**|** universe**(**int numOfCelestialBodies**,** float radius**);**

19**|**

20**|** //set radius if needed

21**|** void set\_radius**(**float radius**);**

22**|**

23**|** //get radius if needed

24**|** double get\_radius**();**

25**|**

26**|** //calculate positions based on window size.

27**|** void set\_position**(**sf**::**Vector2u windowSize **=** defaultWindowSize**);**

28**|**

29**|** // Draw method for universe

30**|** void virtual draw**(**sf**::**RenderTarget**&** target**)** const**;**

31**|**

32**|** //for testing

33**|** void print **();**

34**|**

35**|** void step**(**double deltaT**);**

36**|**

37**|** void calculate\_velocities**(**double deltaT**);**

38**|**

39**|** private**:**

40**|** double universeRadius**;**

41**|**

42**|** //using a vector of shared\_ptr to hold celestialBodies

43**|** std**::**vector**<**std**::**shared\_ptr**<**celestialBody**>>** celestialBodies**;**

44**|** **};**

45**|**

46**|** #endif //UNIVERSE\_H

#### Universe.cpp

001**|** #include "Universe.hpp"

002**|**

003**|**

004**|** universe**::**universe**(){**

005**|** **return;**

006**|** **}**

007**|**

008**|** universe**::**universe**(**float radius**){**

009**|** universeRadius **=** radius**;**

010**|** **}**

011**|**

012**|** //constructor for taking in input from txt doc.

013**|** universe**::**universe**(**int numOfCelestialBodies**,** float radius**){**

014**|**

015**|** universeRadius **=** radius**;**

016**|**

017**|** //for loop to collect data for the number of celestial bodies.

018**|** **for(**int i**=**0**;** i**<**numOfCelestialBodies**;** i**++){**

019**|** std**::**shared\_ptr**<**celestialBody**>** temp **=** std**::**make\_shared**<**celestialBody**>** **();**

020**|** std**::**cin **>>** **\***temp**;**

021**|**

022**|** //calculate initial position for all bodies as we loop through

023**|** temp**->**set\_position**(**radius**);**

024**|**

025**|** //add the body to the vector

026**|** celestialBodies**.**push\_back**(**temp**);**

027**|**

028**|** //used for testing

029**|** //std::cout << \*temp;

030**|** **}**

031**|** **}**

032**|**

033**|** //sets the radius if we need to.

034**|** void universe**::**set\_radius**(**float radius**){**

035**|** universeRadius **=** radius**;**

036**|** **return;**

037**|** **}**

038**|**

039**|** //gets radius

040**|** double universe**::**get\_radius**(){**

041**|** **return** universeRadius**;**

042**|** **}**

043**|**

044**|** //uses an iterator to go through and set position of each celestialBody

045**|** void universe**::**set\_position**(**sf**::**Vector2u windowSize**){**

046**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

047**|** **(\*\***it**).**set\_position**(**universeRadius**,** windowSize**);**

048**|** **}**

049**|** **}**

050**|**

051**|**

052**|** //uses an iterator to go through and draw each celestialBody

053**|** void universe**::**draw**(**sf**::**RenderTarget**&** target**)** const**{**

054**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

055**|** target**.**draw**(\*\***it**);**

056**|** **}**

057**|** **}**

058**|**

059**|** //print function for testing.

060**|** void universe**::**print **(){**

061**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

062**|** std**::**cout **<<** **\*\***it**;**

063**|** **}**

064**|** **}**

065**|**

066**|** //changes position of celestialBodies depending on velocity and time step(deltaT).

067**|** void universe**::**step**(**double deltaT**){**

068**|** **this->**calculate\_velocities**(**deltaT**);**

069**|** // std::cout << "NEW PHASE" << std::endl;

070**|**

071**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

072**|** **(\*\***it**).**set\_positionX**((\*\***it**).**get\_positionX**()** **+** **((\*\***it**).**get\_velocityX**()** **\*** deltaT**));**

073**|** **(\*\***it**).**set\_positionY**((\*\***it**).**get\_positionY**()** **+** **((\*\***it**).**get\_velocityY**()** **\*** deltaT**));**

074**|** **}**

075**|** // this->print();

076**|** **}**

077**|**

078**|** void universe**::**calculate\_velocities**(**double deltaT**){**

079**|** const double gravity **=** 6.67e-11**;**

080**|** //nested loops to be able to compare two celestialBodies together. second loop only goes to first iterator because all calculations can be negated and applied to other celestialBody. This way we can go through less of the nested loop but still hit everything

081**|** **for(**auto it **=** celestialBodies**.**begin**();** it **!=** celestialBodies**.**end**();** **++**it**){**

082**|** **for(**auto it2 **=** celestialBodies**.**rbegin**();** **\***it2 **!=** **\***it**;** **++**it2**){**

083**|**

084**|** double distX **=** **(\***it2**)->**get\_positionX**()** **-** **(\***it**)->**get\_positionX**();**

085**|** double distY **=** **(\***it2**)->**get\_positionY**()** **-** **(\***it**)->**get\_positionY**();**

086**|** //std::cout <<distX << distY;

087**|** double distSQ **=** pow**(**distX**,**2**)** **+** pow**(**distY**,**2**);** //calculate r^2

088**|** double dist **=** sqrt**(**distSQ**);** //calculate r

089**|** double force **=** **(**gravity**\*** **(\*\***it**).**get\_mass**()** **\*** **(\*\***it2**).**get\_mass**())/**distSQ**;** //calulate force (g\*m1\*m2)/r

090**|** double forceX **=** force**\*** **(**distX**/**dist**);** //force ratio for x

091**|** double forceY **=** force**\*** **(**distY**/**dist**);** //force ratio for y

092**|** double accelXit **=** forceX**/(\*\***it**).**get\_mass**();** //a = F/m

093**|** double accelYit **=** forceY**/(\*\***it**).**get\_mass**();** //a = f/m

094**|** double velocityXitChange **=** accelXit **\*** deltaT**;** //v = deltaT\*a

095**|** double velocityYitChange **=** accelYit **\*** deltaT**;** //v = deltaT\*a

096**|** double accelXit2 **=** **-**1**\***forceX**/(\*\***it2**).**get\_mass**();** //negate for it2 because we have distances based on it, not it2

097**|** double accelYit2 **=** **-**1**\***forceY**/(\*\***it2**).**get\_mass**();** //negate for it2 because we have distances based on it, not it2

098**|** double velocityXit2Change **=** accelXit2 **\*** deltaT**;** //already negated as accel was negated, so dont need to negate again.

099**|** double velocityYit2Change **=** accelYit2 **\*** deltaT**;** //already negated as accel was negated, so dont need to negate again.

100**|** //std::cout << velocityXitChange << velocityYitChange<< velocityXit2Change<<velocityYit2Change;

101**|** **(\*\***it**).**set\_velocityX**((\*\***it**).**get\_velocityX**()** **+** velocityXitChange**);** //set X position for it

102**|** **(\*\***it**).**set\_velocityY**((\*\***it**).**get\_velocityY**()** **+** velocityYitChange**);** //set Y position for it

103**|** **(\*\***it2**).**set\_velocityX**((\*\***it2**).**get\_velocityX**()** **+** velocityXit2Change**);** //set X position for it2

104**|** **(\*\***it2**).**set\_velocityY**((\*\***it2**).**get\_velocityY**()** **+** velocityYit2Change**);** //set Y position for it2

105**|** **}**

106**|** **}**

107**|** **}**

108**|**

#### Celestialbody.hpp

01**|** #ifndef CELESTIALBODY\_H

02**|** #define CELESTIALBODY\_H

03**|** #include **<**iostream**>**

04**|** #include **<**string**>**

05**|** #include **<**vector**>**

06**|** #include **<**SFML**/**System**.**hpp**>**

07**|** #include **<**SFML**/**Window**.**hpp**>**

08**|** #include **<**SFML**/**Graphics**.**hpp**>**

09**|**

10**|**

11**|** // Constants for the window size.

12**|** const sf**::**Vector2u defaultWindowSize**(**1000**,** 1000**);**

13**|**

14**|** class celestialBody**:** public sf**::**Drawable**{**

15**|** public**:**

16**|**

17**|**

18**|** // Constructors

19**|** celestialBody**();**

20**|**

21**|** celestialBody**(**double posX**,** double posY**,** double velX**,** double velY**,** double mass**,** std**::**string file**);**

22**|**

23**|**

24**|** void set\_position**(**float radius**,** sf**::**Vector2u windowSize **=** defaultWindowSize**);** // Sets the planets positions will default to 1000x1000 screen size

25**|**

26**|** // Overridden operator >> for inputing from a file

27**|** friend std**::**istream**&** **operator>>** **(**std**::**istream **&**input**,** celestialBody **&**body**);**

28**|**

29**|** // Overriddden operator << for debugging

30**|** friend std**::**ostream**&** **operator<<** **(**std**::**ostream **&**output**,** celestialBody **&**body**);**

31**|**

32**|** //getters and setters for X velocity

33**|** void set\_velocityX**(**double Xvelocity**);**

34**|**

35**|** double get\_velocityX**();**

36**|**

37**|** //getters and setters for Y velocity

38**|** void set\_velocityY**(**double Yvelocity**);**

39**|**

40**|** double get\_velocityY**();**

41**|**

42**|** //getters and setters for X position

43**|** void set\_positionX**(**double Xposition**);**

44**|**

45**|** double get\_positionX**();**

46**|**

47**|** //getters and setters for Y position

48**|** void set\_positionY**(**double Yposition**);**

49**|**

50**|** double get\_positionY**();**

51**|**

52**|** //getter for mass

53**|** double get\_mass**();**

54**|** private**:**

55**|**

56**|** // Draw method

57**|** void virtual draw**(**sf**::**RenderTarget**&** target**,** sf**::**RenderStates states**)** const**;**

58**|**

59**|** // Member variables

60**|** double positionX**,** positionY**,** velocityX**,** velocityY**,** celestialMass**;**

61**|** std**::**string fileName**;**

62**|**

63**|** // Image related objects

64**|** sf**::**Image celestialImage**;**

65**|** sf**::**Sprite celestialSprite**;**

66**|** sf**::**Texture celestialTexture**;**

67**|** **};**

68**|**

69**|** #endif //CELESTIALBODY\_H

#### Celestialbody.cpp

001**|** #include "CelestialBody.hpp"

002**|**

003**|** //default constructor doesn't need to do anything, since the >> operator will set everything

004**|** celestialBody**::**celestialBody**(){**

005**|** **return;**

006**|** **}**

007**|**

008**|** //sets values if given

009**|** celestialBody**::**celestialBody**(**double posX**,** double posY**,** double velX**,** double velY**,** double mass**,** std**::**string file**){**

010**|** positionX **=** posX**;**

011**|** positionY **=** posY**;**

012**|** velocityX **=** velX**;**

013**|** velocityY **=** velY**;**

014**|** celestialMass **=** mass**;**

015**|** fileName **=** file**;**

016**|**

017**|** **if(**celestialImage**.**loadFromFile**(**fileName**)){**

018**|** **return;**

019**|** **}**

020**|**

021**|** celestialTexture**.**loadFromImage**(**celestialImage**);**

022**|** celestialSprite**.**setTexture**(**celestialTexture**);**

023**|** **}**

024**|**

025**|**

026**|** void celestialBody**::**set\_position**(**float radius**,** sf**::**Vector2u windowSize**){** // Sets the planets positions based on window size, will default to 1000,1000.

027**|**

028**|** /\*

029| The math for this is obtain a ratio for the size of the screen. then multiplies it by distance from center of universe(which is half of the screen) to figure out where in relation to the center the celestialBody is. Then offsets both x and y based of coordinates of center of the universe.

030| \*/

031**|**

032**|** double posX **=** **(** **(**positionX **/** radius**)** **\*** **(** windowSize**.**x **/** 2**)** **)** **+** **(** windowSize**.**x **/** 2**);**

033**|** double posY **=** **(** **-(**positionY **/** radius**)** **\*** **(** windowSize**.**y **/** 2**)** **)** **+** **(** windowSize**.**y **/** 2**);**//negate to make it orbit counter-clockwise

034**|** /\*for testing

035| std::cout <<\*this<< "X pixel: " << posX << std::endl;

036| std::cout << "Y pixel: " << posY << std::endl;

037| std::cout << "x positoin : " << positionX << std::endl;

038| std::cout << "y position: " << positionY << std::endl;

039| std::cout << "radius " << radius << std::endl;

040| \*/

041**|** //apply position to sprite.

042**|** celestialSprite**.**setPosition**(**posX**,** posY**);**

043**|**

044**|** **}**

045**|**

046**|** //getters and setters for X velocity

047**|** void celestialBody**::**set\_velocityX**(**double Xvelocity**){**

048**|** **this->**velocityX **=** Xvelocity**;**

049**|** **}**

050**|**

051**|** double celestialBody**::**get\_velocityX**(){**

052**|** **return** **this->**velocityX**;**

053**|** **}**

054**|**

055**|** //getters and setters for Y velocity

056**|**

057**|** void celestialBody**::**set\_velocityY**(**double Yvelocity**){**

058**|** **this->**velocityY **=** Yvelocity**;**

059**|** **}**

060**|**

061**|** double celestialBody**::**get\_velocityY**(){**

062**|** **return** **this->**velocityY**;**

063**|** **}**

064**|**

065**|**

066**|**

067**|** //getters and setters for X position

068**|** void celestialBody**::**set\_positionX**(**double Xposition**){**

069**|** **this->**positionX **=** Xposition**;**

070**|** **}**

071**|**

072**|** double celestialBody**::**get\_positionX**(){**

073**|** **return** **this->**positionX**;**

074**|** **}**

075**|**

076**|** //getters and setters for Y position

077**|**

078**|** void celestialBody**::**set\_positionY**(**double Yposition**){**

079**|** **this->**positionY **=** Yposition**;**

080**|** **}**

081**|**

082**|**

083**|** double celestialBody**::**get\_positionY**(){**

084**|** **return** **this->**positionY**;**

085**|** **}**

086**|**

087**|** //getter for mass

088**|** double celestialBody**::**get\_mass**(){**

089**|** **return** **this->**celestialMass**;**

090**|** **}**

091**|**

092**|** // Overridden operator >> for inputing from a file

093**|** std**::**istream**&** **operator>>** **(**std**::**istream **&**input**,** celestialBody **&**body**){**

094**|** //read in all the information

095**|** input **>>** body**.**positionX **>>** body**.**positionY **>>** body**.**velocityX **>>** body**.**velocityY **>>** body**.**celestialMass **>>** body**.**fileName**;**

096**|**

097**|** //set the images and positions, like the parameter constructor.

098**|** **if(!**body**.**celestialImage**.**loadFromFile**(**body**.**fileName**)){**

099**|** **throw** std**::**invalid\_argument**(**"no file for celestialImage"**);**

100**|** **}**

101**|**

102**|** body**.**celestialTexture**.**loadFromImage**(**body**.**celestialImage**);**

103**|** body**.**celestialSprite**.**setTexture**(**body**.**celestialTexture**);**

104**|**

105**|** **return** input**;**

106**|** **}**

107**|**

108**|** //overridden operator << for testing purposes.

109**|** std**::**ostream**&** **operator<<** **(**std**::**ostream **&**output**,** celestialBody **&**body**)**

110**|** **{**

111**|** // For debugging, output all the data stored in the object.

112**|** output **<<** "File name: " **<<** body**.**fileName **<<** std**::**endl **<<** "X position: " **<<** body**.**positionX **<<** std**::**endl **<<** "Y position: " **<<** body**.**positionY **<<** std**::**endl **<<** "X velocity: " **<<** body**.**velocityX **<<** std**::**endl **<<** "Y velocity: " **<<** body**.**velocityY **<<** std**::**endl **<<** "Mass: " **<<** body**.**celestialMass **<<** std**::**endl**;**

113**|**

114**|** **return** output**;**

115**|** **}**

116**|**

117**|** // Drawable method

118**|** void celestialBody**::**draw**(**sf**::**RenderTarget**&** target**,** sf**::**RenderStates states**)** const

119**|** **{**

120**|** // draw the image

121**|** target**.**draw**(**celestialSprite**);**

122**|** **}**

123**|**

# PS3: Synthesizing a Plucked String

## PS3a: CircularBuffer implementation

### Discussion:

I created a ring buffer using a vector. The vector makes it convenient to enqueue and dequeue items up to the capacity. I kept tract of two indexes for the head and tail as well as values for the size and capacity. Then, I used some index management to make sure that the index wraps around when it goes above capacity. I also used boost testing to do unit tests and make sure that the program is running as intended.

### Console Output (Running Boost test):

Running 2 test cases...

\*\*\* No errors detected

### Code:

#### Makefile:

01| CC **=** g++

02| CFLAGS **=** -std=c++14 -c -g -Og -Wall -Werror -pedantic

03| OBJ **=** test.o CircularBuffer.o

04| LIBS **=**

05| EXE **=** PS3a

06|

07| all**:** PS3a

08|

09| PS3a**:** $(OBJ)

10| $(CC) $(OBJ) -o $(EXE) $(LIBS)

11|

12| CircularBuffer.o**:** CircularBuffer.cpp CircularBuffer.h

13| $(CC) $(CFLAGS) -o $@ $<

14|

15| test.o**:** test.cpp CircularBuffer.h

16| $(CC) $(CFLAGS) -o $@ $<

17|

18| clean**:**

19| rm \*.o

20| rm $(EXE)

21|

#### test.cpp:

01**|** #include **<**iostream**>**

02**|** #include **<**stdexcept**>**

03**|**

04**|** #include "CircularBuffer.h"

05**|**

06**|** #define BOOST\_TEST\_DYN\_LINK

07**|** #define BOOST\_TEST\_MODULE Main

08**|** #include **<**boost**/**test**/**included**/**unit\_test**.**hpp**>**

09**|**

10**|**

11**|** BOOST\_AUTO\_TEST\_CASE**(**constructors **)** **{**

12**|** // tests negative number in constructor. expects error thrown

13**|** BOOST\_REQUIRE\_THROW**(**CircularBuffer cTest**(-**5**),** std**::**invalid\_argument**);**

14**|** // tests 0 in constructor. expects error thrown

15**|** BOOST\_REQUIRE\_THROW**(**CircularBuffer cTest**(**0**),** std**::**invalid\_argument**);**

16**|** // tests 1 in constructor. expects error thrown

17**|** BOOST\_REQUIRE\_THROW**(**CircularBuffer cTest**(**1**),** std**::**invalid\_argument**);**

18**|**

19**|** // tests an expected input into the constructor. no error should be thrown

20**|** BOOST\_REQUIRE\_NO\_THROW**(**CircularBuffer cTest**(**10**));**

21**|** // tests an expected input into the constructor. no error should be thrown

22**|** BOOST\_REQUIRE\_NO\_THROW**(**CircularBuffer cTest**(**20**));**

23**|** // tests an expected input into the constructor. no error should be thrown

24**|** BOOST\_REQUIRE\_NO\_THROW**(**CircularBuffer cTest**(**37**));**

25**|** **}**

26**|**

27**|** BOOST\_AUTO\_TEST\_CASE**(**ring\_buffer\_modification **)** **{**

28**|** CircularBuffer test**(**10**);**

29**|** // check functions on empty buffer

30**|** BOOST\_REQUIRE\_THROW**(**test**.**peek**(),** std**::**runtime\_error**);**

31**|** BOOST\_REQUIRE\_THROW**(**test**.**dequeue**(),** std**::**runtime\_error**);**

32**|** BOOST\_REQUIRE**(**test**.**isEmpty**()** **==** **true);**

33**|** BOOST\_REQUIRE**(**test**.**isFull**()** **==** **false);**

34**|** BOOST\_REQUIRE**(**test**.**size**()** **==** 0**);**

35**|** // add 5 elements to buffer

36**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**5**));**

37**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**6**));**

38**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**7**));**

39**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**8**));**

40**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**9**));**

41**|**

42**|** // check peek and dequeue on buffer with elements. should return/pop head.

43**|** BOOST\_REQUIRE**(**test**.**size**()** **==** 5**);**

44**|** BOOST\_REQUIRE**(**test**.**peek**()** **==** 5**);**

45**|** BOOST\_REQUIRE**(**test**.**dequeue**()** **==** 5**);**

46**|**

47**|** // check to make sure size went down after dequeue and that isEmpty is false

48**|** BOOST\_REQUIRE**(**test**.**size**()** **==** 4**);**

49**|** BOOST\_REQUIRE**(**test**.**isEmpty**()** **==** **false);**

50**|** BOOST\_REQUIRE**(**test**.**isFull**()** **==** **false);**

51**|**

52**|** // add 2 more elements to buffer

53**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**10**));**

54**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**11**));**

55**|**

56**|** // double check peek and dequeue

57**|** BOOST\_REQUIRE**(**test**.**peek**()** **==** 6**);**

58**|** BOOST\_REQUIRE**(**test**.**dequeue**()** **==** 6**);**

59**|**

60**|** // add elements to put size to capcaity

61**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**12**));**

62**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**13**));**

63**|**

64**|** // Test the Size function some more

65**|** BOOST\_REQUIRE**(**test**.**size**()** **==** 7**);**

66**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**14**));**

67**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**15**));**

68**|** BOOST\_REQUIRE\_NO\_THROW**(**test**.**enqueue**(**16**));**

69**|**

70**|** // check isEmpty and isFull

71**|** BOOST\_REQUIRE**(**test**.**isEmpty**()** **==** **false);**

72**|** BOOST\_REQUIRE**(**test**.**isFull**()** **==** **true);**

73**|**

74**|** // 12 total elements added, but 2 popped off

75**|** // buffer is now full, so adding another should throw error

76**|** BOOST\_REQUIRE\_THROW**(**test**.**enqueue**(**17**),** std**::**runtime\_error**);**

77**|** **}**

78**|**

#### CircularBuffer.h

01**|** #ifndef CIRCULARBUFFER\_H

02**|** #define CIRCULARBUFFER\_H

03**|**

04**|** #include **<**stdint**.**h**>**

05**|** #include **<**vector**>**

06**|**

07**|** class CircularBuffer**{**

08**|** public**:**

09**|** // required constructor listed in pdf

10**|** CircularBuffer**(**int capacity**);**

11**|**

12**|** // required functions listed in pdf

13**|** int size**();**

14**|** bool isEmpty**();**

15**|** bool isFull**();**

16**|** void enqueue**(**int16\_t x**);**

17**|** int16\_t dequeue**();**

18**|** int16\_t peek**();**

19**|**

20**|** private**:**

21**|** // private member variables

22**|** int bufferTail**;**

23**|** int bufferHead**;**

24**|** int bufferCapacity**;**

25**|** int bufferSize**;**

26**|** // using vector to hold the ring buffer

27**|** std**::**vector**<**int16\_t**>** buffer**;**

28**|** **};**

29**|**

30**|** #endif //CIRCULARBUFFER\_H

#### CircularBuffer.cpp

01**|** #include "CircularBuffer.h" // NOLINT

02**|** #include **<**stdexcept**>**

03**|**

04**|** CircularBuffer**::**CircularBuffer**(**int capacity**)** **{**

05**|** // Have capacity <= 1 becuase algorithm requires 2 items.

06**|** **if** **(**capacity **<=** 1**)** **{**

07**|** **throw** std**::**invalid\_argument

08**|** **(**"CircularBuffer constructor: capacity must be greater than zero"**);**

09**|** **}**

10**|** bufferCapacity **=** capacity**;**

11**|** bufferSize **=** 0**;**

12**|** bufferHead **=** 0**;**

13**|** bufferTail **=** 0**;**

14**|** buffer**.**resize**(**capacity**);**

15**|** **}**

16**|**

17**|** // required functions in pdf

18**|** int CircularBuffer**::**size**()** **{**

19**|** **return** bufferSize**;**

20**|** **}**

21**|**

22**|** bool CircularBuffer**::**isEmpty**()** **{**

23**|** **if** **(**bufferSize **==** 0**)** **{**

24**|** **return** **true;**

25**|** **}**

26**|** **return** **false;**

27**|** **}**

28**|**

29**|** bool CircularBuffer**::**isFull**()** **{**

30**|** **if** **(**bufferSize **==** bufferCapacity**)** **{**

31**|** **return** **true;**

32**|** **}**

33**|** **return** **false;**

34**|** **}**

35**|**

36**|** void CircularBuffer**::**enqueue**(**int16\_t x**)** **{**

37**|** **if** **(**isFull**())** **{**

38**|** **throw** std**::**runtime\_error**(**"enqueue: can't enqueue into a full ring"**);**

39**|** **}**

40**|**

41**|** buffer**[**bufferTail**]** **=** x**;**

42**|**

43**|** // loops around if capacity is reached. indexes are 0 to capacity-1,

44**|** // so if bufferTail = capacity-1, then we just inserted into the last location

45**|** **if** **(**bufferTail **==** bufferCapacity**-**1**)** **{**

46**|** bufferTail **=** 0**;**

47**|** **}** **else** **{**

48**|** bufferTail**++;**

49**|** **}**

50**|** bufferSize**++;**

51**|** **}**

52**|**

53**|** int16\_t CircularBuffer**::**dequeue**()** **{**

54**|** **if** **(**isEmpty**())** **{**

55**|** **throw** std**::**runtime\_error**(**"dequeue: can't dequeue from an empty ring"**);**

56**|** **}**

57**|**

58**|** // this should make sure bufferHead never reaches out of index.

59**|** **if** **(**bufferHead **==** bufferCapacity**-**1**)** **{**

60**|** bufferHead **=** 0**;**

61**|** bufferSize**--;**

62**|** **return** buffer**[**bufferCapacity**-**1**];**

63**|** **}**

64**|**

65**|** // if we don't need to worry about bufferHead going out of index,

66**|** // just do things normally.

67**|** bufferSize**--;**

68**|** **return** buffer**[**bufferHead**++];**

69**|** **}**

70**|**

71**|** int16\_t CircularBuffer**::**peek**()** **{**

72**|** **if** **(**isEmpty**())** **{**

73**|** **throw** std**::**runtime\_error**(**"peek: can't peeke from an empty ring"**);**

74**|** **}**

75**|** // bufferHead should never get out of index, so we can just check it.

76**|** **return** buffer**[**bufferHead**];**

77**|** **}**

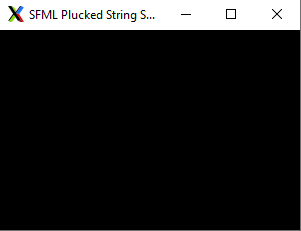
78**|**

## PS3b: StringSound Implementation and SFML audio output

### Discussion:

I utilized the CircularBuffer to simulate the frequency of a held note. The CircularBuffer is used to create the following white noise as a note is held but started at a specific frequency based on what key is pressed. I utilized SFML to create a window and to play the simulated note. To calculate the frequency of each key, I used the formula 440 × 2(i - 24) / 12. This was also the first project where I included a lambda expression, which was something new that I learned during this class.

### Screenshot:



Code:

Note: Since PS3b builds off PS3a, reused files will not be included in the code section if they were not modified.

#### Makefile

01| CC **=** g++

02| CFLAGS **=** -std=c++14 -c -g -Og -Wall -Werror -pedantic

03| OBJ **=** KSGuitarSim.o StringSound.o CircleBuffer.o

04| LIBS **=** -lsfml-graphics -lsfml-system -lsfml-window -lsfml-audio

05| EXE **=** KSGuitarSim

06|

07| all**:** KSGuitarSim

08|

09| KSGuitarSim**:** $(OBJ)

10| $(CC) $(OBJ) -o $(EXE) $(LIBS)

11|

12| KSGuitarSim.o**:** KSGuitarSim.cpp StringSound.h

13| $(CC) $(CFLAGS) -o $@ $<

14|

15| StringSound.o**:** StringSound.cpp StringSound.h CircleBuffer.h

16| $(CC) $(CFLAGS) -o $@ $<

17|

18| CircleBuffer.o**:** CircleBuffer.cpp CircleBuffer.h

19| $(CC) $(CFLAGS) -o $@ $<

20|

21| clean**:**

22| rm \*.o

23| rm $(EXE)

24|

#### KSGuitarSim.cpp

01**|** #include **<**SFML**/**Graphics**.**hpp**>**

02**|** #include **<**SFML**/**System**.**hpp**>**

03**|** #include **<**SFML**/**Audio**.**hpp**>**

04**|** #include **<**SFML**/**Window**.**hpp**>**

05**|**

06**|** #include **<**math**.**h**>**

07**|** #include **<**limits**.**h**>**

08**|**

09**|** #include **<**iostream**>**

10**|** #include **<**string**>**

11**|** #include **<**exception**>**

12**|** #include **<**stdexcept**>**

13**|** #include **<**vector**>**

14**|**

15**|** #include "CircleBuffer.h"

16**|** #include "StringSound.h"

17**|**

18**|**

19**|**

20**|** std**::**vector**<**sf**::**Int16**>** makeSamples**(**StringSound gs**,** bool dif**)** **{**

21**|** std**::**vector**<**sf**::**Int16**>** samples**;**

22**|**

23**|** gs**.**pluck**();**

24**|** int duration **=** 8**;** // seconds

25**|** int i**;**

26**|** **for** **(**i**=** 0**;** i **<** SAMPLES\_PER\_SEC **\*** duration**;** i**++)** **{**

27**|** gs**.**tic**();**

28**|** samples**.**push\_back**(**gs**.**sample**());**

29**|** **}**

30**|** **return** samples**;**

31**|** **}**

32**|**

33**|** int main**()** **{**

34**|** sf**::**RenderWindow window**(**sf**::**VideoMode**(**300**,** 200**),** "SFML Plucked String Sound Lite"**);**

35**|** sf**::**Event event**;**

36**|**

37**|** double freq**;**

38**|**

39**|** std**::**vector**<**std**::**vector**<**sf**::**Int16**>>** samples**(**37**);**

40**|** std**::**vector**<**sf**::**SoundBuffer**>** soundBuffers**(**37**);**

41**|** std**::**vector**<**sf**::**Sound**>** sounds**(**37**);**

42**|**

43**|** std**::**vector**<**std**::**vector**<**sf**::**Int16**>>** samples2**(**37**);**

44**|** std**::**vector**<**sf**::**SoundBuffer**>** soundBuffers2**(**37**);**

45**|** std**::**vector**<**sf**::**Sound**>** sounds2**(**37**);**

46**|**

47**|** // create a string with the keyboard keys

48**|** std**::**string keyboard **=** "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' "**;**

49**|**

50**|** //make frequencies for the keyboard.

51**|** **for(**int i**=**0**;** i**<**37**;** i**++){**

52**|** // Use formula in pdf to calculate frequency

53**|** freq **=** 440**\***pow**(**2**,((**i**-**24**)/**12.0**));**

54**|** StringSound temp **=** StringSound**(**freq**);**

55**|** // use the StringSound to fill the samples vector in that index

56**|** samples**[**i**]** **=** makeSamples**(**temp**,** **false);**

57**|**

58**|** **if(!**soundBuffers**[**i**].**loadFromSamples**(&**samples**[**i**][**0**],** samples**[**i**].**size**(),** 2**,** 44100**)){**

59**|** **throw** std**::**runtime\_error**(**"Could not load soundBuffers from samples"**);**

60**|** **}**

61**|** sounds**[**i**].**setBuffer**(**soundBuffers**[**i**]);**

62**|** **}**

63**|**

64**|** **while** **(**window**.**isOpen**())** **{**

65**|** **while** **(**window**.**pollEvent**(**event**))** **{**

66**|** //check if event is unicode character (because our entire keyboard consists of unicode characters)

67**|** //better than doing a switch for each key

68**|** **if(**event**.**type **==** sf**::**Event**::**TextEntered**){**

69**|** // ASCII is a subset of unicode, only need to deal with ASCII chars

70**|** **if(**event**.**text**.**unicode **<** 128**){**

71**|** // convert unicode to ascii

72**|** char key **=** **(**char**)** **(**event**.**text**.**unicode**);**

73**|**

74**|** **for(**int i **=** 0**;** i**<**37**;** i**++){**

75**|** **if(**keyboard**[**i**]** **==** key**){**

76**|** sounds**[**i**].**play**();**

77**|** **break;**

78**|** **}**

79**|** **}**

80**|** **}**

81**|** **}**

82**|** // Close window: exit

83**|** **if** **(**event**.**type **==** sf**::**Event**::**Closed**)**

84**|** window**.**close**();**

85**|** **}**

86**|** window**.**clear**();**

87**|** window**.**display**();**

88**|** **}**

89**|** **return** 0**;**

90**|** **}**

91**|**

#### StringSound.h

01**|** #ifndef STRINGSOUND\_H

02**|** #define STRINGSOUND\_H

03**|**

04**|** #include **<**SFML**/**Graphics**.**hpp**>**

05**|** #include **<**SFML**/**System**.**hpp**>**

06**|** #include **<**SFML**/**Audio**.**hpp**>**

07**|** #include **<**SFML**/**Window**.**hpp**>**

08**|** #include **<**vector**>**

09**|** #include **<**memory**>**

10**|** #include "CircleBuffer.h"

11**|**

12**|** #define SAMPLES\_PER\_SEC 44100

13**|** #define CONCERT\_A 220.0

14**|**

15**|** class StringSound**{**

16**|** public**:**

17**|** StringSound**(**double frequency**);**

18**|** StringSound**(**std**::**vector**<**sf**::**Int16**>** init**);**

19**|** void pluck**();**

20**|** void tic**();**

21**|** sf**::**Int16 sample**();**

22**|** int time**();**

23**|** int getBufferSize**();**

24**|** **~**StringSound**();**

25**|** private**:**

26**|** int ticsPassed**;**

27**|** CircleBuffer**\*** buffer**;**

28**|** **};**

29**|**

30**|** #endif //STRINGSOUND\_H

31**|**

#### StringSound.cpp

01**|** #include "StringSound.h"

02**|** #include **<**cmath**>**

03**|** #include **<**random**>**

04**|** #include **<**exception**>**

05**|**

06**|** StringSound**::**StringSound**(**double frequency**){**

07**|** ticsPassed **=** 0**;**

08**|** // use lambda expression so we don't have to calculate capacity,

09**|** // assign it to a variable, then use it for the constructor.

10**|** //Add exception for when samples/freq <=1

11**|** **if(**frequency **>=** 44100 **||** frequency **<=**0**){**

12**|** **throw** std**::**invalid\_argument

13**|** **(**"Frequency must be between 1 and 44099"**);**

14**|** **}**

15**|** //int value = ceil(SAMPLES\_PER\_SEC/frequency);

16**|** buffer **=** **new** CircleBuffer**([](**int samples**,** int freq**){return** ceil**(**samples**/**freq**);}(**SAMPLES\_PER\_SEC**,** frequency**));**

17**|** **}**

18**|**

19**|** StringSound**::**StringSound**(**std**::**vector**<**sf**::**Int16**>** init**){**

20**|** buffer **=** **new** CircleBuffer**(**init**.**size**());**

21**|** **for** **(**auto it **=** init**.**begin**();** it **<** init**.**end**();** it**++){**

22**|** buffer**->**enqueue**((**int16\_t**)** **\***it**);**

23**|** **}**

24**|**

25**|** ticsPassed **=** 0**;**

26**|** **}**

27**|**

28**|** void StringSound**::**pluck**(){**

29**|** buffer**->**empty**();**

30**|**

31**|** //make random number generator

32**|** std**::**mt19937 mt**(**1729**);**

33**|** std**::**uniform\_int\_distribution**<**int16\_t**>** dist**(-**32768**,**32767**);**

34**|** **for(**int i **=**0**;** i **<** buffer**->**capacity**();** i**++){**

35**|** buffer**->**enqueue**(**dist**(**mt**));**

36**|** **}**

37**|** **}**

38**|**

39**|** void StringSound**::**tic**(){**

40**|** /\*

41| // check if buffer size is too small for dequeue + peek

42| if(buffer->size()<2){

43| throw std::length\_error("Buffer is too small, can't tic");

44| }

45| \*/

46**|** // dequeue the first value

47**|** int16\_t first **=** buffer**->**dequeue**();**

48**|**

49**|** // get second value for Karplus-Strong, but don't dequeue

50**|** int16\_t second **=** buffer**->**peek**();**

51**|**

52**|** // using 0.498 instead of decay factor of 0.996 because i took the 1/2 from the average (of first and second) and multiplied it to the 0.996

53**|** buffer**->**enqueue**((**sf**::**Int16**)** **(**0.498**\*(**first**+**second**)));**

54**|** ticsPassed**++;**

55**|** **}**

56**|**

57**|**

58**|** sf**::**Int16 StringSound**::**sample**(){**

59**|** // check for exception if buffer is too empty

60**|** **if(**buffer**->**isEmpty**()){**

61**|** **throw** std**::**length\_error**(**"Buffer is too small, can't peek for sample"**);**

62**|** **}**

63**|**

64**|** **return** **(**sf**::**Int16**)** buffer**->**peek**();**

65**|** **}**

66**|**

67**|** int StringSound**::**time**(){**

68**|** **return** ticsPassed**;**

69**|** **}**

70**|**

71**|** //made to facilitate making of frequency chirp

72**|** int StringSound**::**getBufferSize**(){**

73**|** **return** buffer**->**size**();**

74**|** **}**

75**|**

76**|** StringSound**::~**StringSound**(){**

77**|** //delete buffer;

78**|** **}**

79**|**

80**|**

# PS4: DNA Sequence Alignment

### Discussion:

This project introduced the idea of dynamic programming. I had already known about space and time complexity from previous computer science courses, but this was also a good refresher. I used nested vectors to make a 2D matrix so that could align the strings in the matrix and then calculate the cost of each step. My program first fills out the matrix from the bottom right corner and then to print out the optimal path, it traverses the matrix from the upper left corner. The algorithm that I used specifically was the Needleman-Wunsch method.

### Console Output when run on example10.txt:

Edit distance = 7

A T 1

A A 0

C - 2

A A 0

G G 0

T G 1

T T 0

A - 2

C C 0

C A 1

Execution time is 0.000909seconds

Edit distance = 7

### Code:

#### main.cpp

01**|** #include **<**iostream**>**

02**|** #include **<**string**>**

03**|** #include **<**SFML**/**System**.**hpp**>**

04**|** #include "ED.hpp"

05**|**

06**|** int main**(**int argv**,** char**\*\*** argc**)** **{**

07**|** sf**::**Clock clock**;**

08**|** sf**::**Time t**;**

09**|**

10**|** std**::**string string1**,** string2**;**

11**|** std**::**cin **>>** string1 **>>** string2**;**

12**|** ED test**(**string1**,** string2**);**

13**|** std**::**cout **<<** "Edit distance = " **<<** test**.**OptDistance**()** **<<** "\n"**;**

14**|** std**::**cout **<<** test**.**Alignment**();**

15**|**

16**|** t **=** clock**.**getElapsedTime**();**

17**|** std**::**cout **<<** "Execution time is " **<<** t**.**asSeconds**()** **<<** "seconds \n"**;**

18**|** // Edit distance is reprinted for ease of looking at output

19**|** std**::**cout **<<** "Edit distance = " **<<** test**.**OptDistance**()** **<<** "\n"**;**

20**|** **return** 0**;**

21**|** **}**

22**|**

#### ED.hpp

01**|** #ifndef ED\_H

02**|** #define ED\_H

03**|**

04**|** #include **<**vector**>**

05**|** #include **<**string**>**

06**|**

07**|** class ED**{**

08**|** public**:**

09**|** ED**(**std**::**string stringOne**,** std**::**string stringTwo**);**

10**|** static int penalty**(**char a**,** char b**);**

11**|** static int min**(**int a**,** int b**,** int c**);**

12**|** int OptDistance**();**

13**|** std**::**string Alignment**();**

14**|** void print**();**

15**|** private**:**

16**|** // 2D vector matrix, so I can use the i,j like coordinates

17**|** std**::**vector**<**std**::**vector**<**int**>>** matrix**;**

18**|** std**::**string string1**;**

19**|** std**::**string string2**;**

20**|**

21**|** **};**

22**|**

23**|** #endif //ED\_H

#### ED.cpp

001**|** #include "ED.hpp"

002**|** #include **<**iostream**>**

003**|** #include **<**exception**>**

004**|** #include **<**sstream**>**

005**|**

006**|** ED**::**ED**(**std**::**string stringOne**,** std**::**string stringTwo**)** **{**

007**|** // store strings

008**|** string1 **=** stringOne**;**

009**|** string2 **=** stringTwo**;**

010**|**

011**|** // make matrix size of string1.length+1 x string2.length+1

012**|** **for** **(**int i **=** 0**;** i **<=** **static\_cast<**int**>(**string1**.**length**());** i**++)** **{**

013**|** std**::**vector**<**int**>** temp**;**

014**|** temp**.**resize**(**string2**.**length**()+**1**);**

015**|** matrix**.**push\_back**(**temp**);**

016**|** **}**

017**|** **}**

018**|**

019**|** int ED**::**penalty**(**char a**,** char b**)** **{**

020**|** **return** a **==** b **?** 0 **:** 1**;**

021**|** **}**

022**|**

023**|** // return smallest value

024**|** int ED**::**min**(**int a**,** int b**,** int c**)** **{**

025**|** // if a is smallest or equal to smalles

026**|** **if** **(**a **<=** b **&&** a **<=** c**)** **{**

027**|** **return** a**;**

028**|** **}** **else** **if** **(**b **<=** c**)** **{** // since a is not smallest, check if b <= c

029**|** **return** b**;**

030**|** **}** **else** **{** // return c because it must be smallest

031**|** **return** c**;**

032**|** **}**

033**|** **}**

034**|**

035**|** // fill out matrix for distances

036**|** int ED**::**OptDistance**()** **{**

037**|** int m **=** string1**.**length**();**

038**|** int n **=** string2**.**length**();**

039**|**

040**|** // fill in right column

041**|** **for** **(**int i **=** 0**;** i **<=** m**;** i**++)** **{**

042**|** matrix**[**i**][**n**]** **=** 2**\*** **(**m**-**i**);**

043**|** **}**

044**|**

045**|** // don't have to do when j=n becuase it was set in the previous for loop

046**|** **for** **(**int j **=** 0**;** j **<** n**;** j**++)** **{**

047**|** matrix**[**m**][**j**]** **=** 2**\*(**n**-**j**);**

048**|** **}**

049**|**

050**|** // start at bottom and go up.

051**|** **for** **(**int i **=** m**-**1**;** i **>=** 0**;** i**--)** **{**

052**|** **for** **(**int j **=** n**-**1**;** j **>=** 0**;** j**--)** **{**

053**|** matrix**[**i**][**j**]** **=** min**(**matrix**[**i**+**1**][**j**+**1**]** **+** penalty**(**string1**[**i**],** // NOLINT added for false positive

054**|** string2**[**j**]),** matrix**[**i**+**1**][**j**]+**2**,** matrix**[**i**][**j**+**1**]+**2**);**

055**|** **}**

056**|** **}**

057**|**

058**|** // return the optimal edit distance

059**|** **return** matrix**[**0**][**0**];**

060**|** **}**

061**|**

062**|** std**::**string ED**::**Alignment**()** **{**

063**|** std**::**stringstream returnString**;**

064**|** int i **=** 0**;**

065**|** int j **=** 0**;**

066**|** int m **=** string1**.**length**();**

067**|** int n **=** string2**.**length**();**

068**|** int right**,** diag**;**

069**|** int penaltyCost **=** 5**;**

070**|**

071**|** // made lambda expressions for each calculated case.

072**|** auto rightCase **=** **[this](**int x**,** int y**){return** matrix**[**x**+**1**][**y**]** **+** 2**;};**

073**|** auto diagCase **=** **[this](**int x**,** int y**,** int penaltyCost**){**

074**|** **return** matrix**[**x**+**1**][**y**+**1**]** **+** penaltyCost**;**

075**|** **};**

076**|** // loop to check when we hit bottom right corner

077**|** **while** **(**i **<** m **||** j **<** n**)** **{**

078**|** // make the right gap case

079**|** **try** **{**

080**|** // right = matrix[i + 1][j] + 2;

081**|** right **=** rightCase**(**i**,** j**);**

082**|** **}** **catch** **(**std**::**out\_of\_range e**)** **{**

083**|** right **=** **-**1**;**

084**|** **}**

085**|**

086**|** // make diagonal case

087**|** **try** **{**

088**|** penaltyCost **=** penalty**(**string1**[**i**],** string2**[**j**]);**

089**|** // diag = matrix[i+1][j+1] + penaltyCost;

090**|** diag **=** diagCase**(**i**,** j**,** penaltyCost**);**

091**|** **}** **catch** **(**std**::**out\_of\_range e**)** **{**

092**|** diag **=** **-**1**;**

093**|** **}**

094**|**

095**|** // check if diagonal was optimal

096**|** **if** **(**matrix**[**i**][**j**]** **==** diag**)** **{**

097**|** returnString **<<** string1**[**i**]** **<<** " " **<<** string2**[**j**]**

098**|** **<<** " " **<<** penaltyCost **<<** "\n"**;**

099**|** i**++;**

100**|** j**++;**

101**|** **}** **else** **if** **(**matrix**[**i**][**j**]** **==** right**)** **{** // check if right was optimal

102**|** returnString **<<** string1**[**i**]** **<<** " - 2\n"**;**

103**|** i**++;**

104**|** **}** **else** **{** // if diagonal and right weren't optimal, down gap must be

105**|** returnString **<<** "- " **<<** string2**[**j**]** **<<** " 2\n"**;**

106**|** j**++;**

107**|** **}**

108**|** **}**

109**|**

110**|** **return** returnString**.**str**();**

111**|** **}**

112**|**

113**|** void ED**::**print**()** **{**

114**|** **for** **(**int i **=** 0**;** i **<** **static\_cast<**int**>(**matrix**.**size**());** i**++)** **{**

115**|** **for** **(**int j **=** 0**;** j **<** **static\_cast<**int**>(**matrix**[**0**].**size**());** j**++)** **{**

116**|** std**::**cout **<<** matrix**[**i**][**j**]** **<<** " "**;**

117**|** **}**

118**|** std**::**cout **<<** std**::**endl**;**

119**|** **}**

120**|** **}**

121**|**

# PS5: Markov Model of Natural Language

### Discussion:

In this project, I made a functioning Markov model based on a given string. I had not heard of a Markov model and it seems useful as a predictive algorithm. In order to organize all of the kgrams from the resulting model, I used a map to hold the kgrams and their frequencies. The map was set up so that the kgrams themselves were the keys and the frequencies were the stored value for that respective key. I think it is really interesting as a starter concept and would definitely like to see how we make predictive models for things like artificial intelligence.

### Console Output:

#### When running TextGenerator:

./TextGenerator 2 11 < input17.txt

gagagaggcga

#### When running Test:

Running 6 test cases...

\*\*\* No errors detected

### Code:

#### Makefile

01| CC **=** g++

02| CFLAGS **=** -std=c++11 -c -g -Og -Wall -Werror -pedantic -Iheader

03| OBJ **=** TextGenerator.o MModel.o

04| LIBS **=**

05| EXE **=** TextGenerator

06|

07| all**:** TextGenerator Test

08|

09| TextGenerator**:** $(OBJ)

10| $(CC) $(OBJ) -o $(EXE) $(LIBS)

11|

12| Test**:** test.o MModel.o

13| $(CC) test.o MModel.o -o Test

14|

15| MModel.o**:** MModel.cpp header/MModel.h

16| $(CC) $(CFLAGS) -o $@ $<

17|

18| test.o**:** test.cpp header/MModel.h

19| $(CC) $(CFLAGS) -o $@ $<

20|

21| TextGenerator.o**:** TextGenerator.cpp header/MModel.h

22| $(CC) $(CFLAGS) -o $@ $<

23|

24| clean**:**

25| rm \*.o

26| rm $(EXE) Test

27|

#### TextGenerator.cpp

01**|** #include "header/MModel.h"

02**|**

03**|** int main**(**int argc**,** char**\*\*** argv**)** **{**

04**|** // Check for number of args

05**|** **if** **(**argc **!=** 3**)** **{**

06**|** std**::**cout **<<** "There is an incorrect number of args. "

07**|** "There should be 3 args.\n"**;**

08**|** **return** **-**1**;**

09**|** **}**

10**|**

11**|** // convert args to ints

12**|** int k **=** atoi**(**argv**[**1**]);**

13**|** int l **=** atoi**(**argv**[**2**]);**

14**|**

15**|** std**::**string input**;**

16**|** std**::**string current**;**

17**|**

18**|** // take in entirety of string

19**|** **while** **(**std**::**cin **>>** current**)** **{**

20**|** input **+=** current**;**

21**|** current **=** ""**;**

22**|** **}**

23**|**

24**|** // make MModel

25**|** MModel model**(**input**,** k**);**

26**|**

27**|** // create Kgram of length K from first K chars

28**|** current **=** input**.**substr**(**0**,** k**);**

29**|**

30**|** // print generated string of length l starting with kgram

31**|** std**::**cout **<<** model**.**generate**(**current**,** l**)** **<<** "\n"**;**

32**|** **}**

33**|**

#### test.cpp

001**|** #define BOOST\_TEST\_DYN\_LINK

002**|** #define BOOST\_TEST\_MODULE Main

003**|** #include **<**boost**/**test**/**included**/**unit\_test**.**hpp**>**

004**|**

005**|** #include "header/MModel.h"

006**|**

007**|** BOOST\_AUTO\_TEST\_CASE**(**constructor**)** **{**

008**|** // tests an expected input into the constructor. no error should be thrown

009**|** BOOST\_REQUIRE\_NO\_THROW**(**MModel cTest**(**"gagggagaggcgagaaa"**,** 0**));**

010**|**

011**|** // tests an expected input into the constructor. no error should be thrown

012**|** BOOST\_REQUIRE\_NO\_THROW**(**MModel cTest**(**"gagggagaggcgagaaa"**,** 3**));**

013**|**

014**|** // tests an expected input into the constructor. no error should be thrown

015**|** BOOST\_REQUIRE\_NO\_THROW**(**MModel cTest**(**"gagggagaggcgagaaa"**,** 5**));**

016**|**

017**|** // tests an expected input into the constructor. no error should be thrown

018**|** BOOST\_REQUIRE\_NO\_THROW**(**MModel cTest**(**"gagggagaggcgagaaa"**,** 7**));**

019**|** **}**

020**|**

021**|** BOOST\_AUTO\_TEST\_CASE**(**Korder**)** **{**

022**|** // creating MModel to run freq on with kgram 0

023**|** MModel cTest**(**"gagggagaggcgagaaa"**,** 0**);**

024**|**

025**|** // ensure kOrder() is returning correct value

026**|** BOOST\_REQUIRE**(**cTest**.**kOrder**()** **==** 0**);**

027**|**

028**|** // creating MModel to run freq on with kgram 3

029**|** MModel cTest2**(**"gagggagaggcgagaaa"**,** 3**);**

030**|**

031**|** // ensure kOrder() is returning correct value

032**|** BOOST\_REQUIRE**(**cTest2**.**kOrder**()** **==** 3**);**

033**|** **}**

034**|**

035**|** BOOST\_AUTO\_TEST\_CASE**(**freq**)** **{**

036**|** // creating MModel to run freq on with kgram 0

037**|** MModel cTest**(**"gagggagaggcgagaaa"**,** 0**);**

038**|**

039**|** // test calling freq on cTest with a correct kgram

040**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest**.**freq**(**""**));**

041**|**

042**|** // test calling freq on cTest with an incorrect kgram

043**|** BOOST\_REQUIRE\_THROW**(**cTest**.**freq**(**"a"**),** std**::**runtime\_error**);**

044**|**

045**|** // double check value returned by freq

046**|** BOOST\_REQUIRE**(**cTest**.**freq**(**""**)** **==** 17**);**

047**|**

048**|** // test calling freq on cTest with a correct kgram

049**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest**.**freq**(**""**,** 'a'**));**

050**|**

051**|** // test calling freq on cTest with an incorrect kgram

052**|** BOOST\_REQUIRE\_THROW**(**cTest**.**freq**(**"a"**,** 'a'**),** std**::**runtime\_error**);**

053**|**

054**|** // double check value returned by freq

055**|** BOOST\_REQUIRE**(**cTest**.**freq**(**""**,** 'a'**)** **==** 7**);**

056**|**

057**|** // double check value returned by freq

058**|** BOOST\_REQUIRE**(**cTest**.**freq**(**""**,** 'c'**)** **==** 1**);**

059**|**

060**|** // double check value returned by freq

061**|** BOOST\_REQUIRE**(**cTest**.**freq**(**""**,** 'g'**)** **==** 9**);**

062**|**

063**|** // double check value returned by freq when char not in input

064**|** BOOST\_REQUIRE**(**cTest**.**freq**(**""**,** 'z'**)** **==** 0**);**

065**|**

066**|** // creating MModel to run freq on with kgram 3

067**|** MModel cTest2**(**"gagggagaggcgagaaa"**,** 3**);**

068**|**

069**|** // test calling freq on cTest with an incorrect kgram

070**|** BOOST\_REQUIRE\_THROW**(**cTest2**.**freq**(**"a"**),** std**::**runtime\_error**);**

071**|**

072**|** // double check value returned by freq

073**|** BOOST\_REQUIRE**(**cTest2**.**freq**(**"aaa"**)** **==** 1**);**

074**|**

075**|** // double check value returned by freq

076**|** BOOST\_REQUIRE**(**cTest2**.**freq**(**"gag"**)** **==** 4**);**

077**|**

078**|** // test calling freq on cTest2 with an incorrect kgram

079**|** BOOST\_REQUIRE\_THROW**(**cTest2**.**freq**(**"a"**,** 'a'**),** std**::**runtime\_error**);**

080**|**

081**|** // test calling freq on cTest2 with a correct kgram

082**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest2**.**freq**(**"aaa"**,** 'a'**));**

083**|**

084**|** // double check value returned by freq when next char not in input

085**|** BOOST\_REQUIRE**(**cTest2**.**freq**(**"aaa"**,** 'a'**)** **==** 0**);**

086**|**

087**|** // double check value returned by freq

088**|** BOOST\_REQUIRE**(**cTest2**.**freq**(**"gcg"**,** 'a'**)** **==** 1**);**

089**|** **}**

090**|**

091**|** BOOST\_AUTO\_TEST\_CASE**(**kRand**)** **{**

092**|** // creating MModel to run freq on with kgram 0

093**|** MModel cTest**(**"gagggagaggcgagaaa"**,** 0**);**

094**|**

095**|** // test calling kRand on cTest with an incorrect kgram

096**|** BOOST\_REQUIRE\_THROW**(**cTest**.**kRand**(**"a"**),** std**::**runtime\_error**);**

097**|**

098**|** // test calling kRand on cTest with a correct kgram

099**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest**.**kRand**(**""**));**

100**|**

101**|** // creating MModel to run freq on with kgram 3

102**|** MModel cTest2**(**"gagggagaggcgagaaa"**,** 3**);**

103**|**

104**|** // test calling kRand on cTest2 with an incorrect kgram

105**|** BOOST\_REQUIRE\_THROW**(**cTest2**.**kRand**(**"a"**),** std**::**runtime\_error**);**

106**|**

107**|** // test calling kRand on cTest2 with an correct kgram not in input

108**|** BOOST\_REQUIRE\_THROW**(**cTest2**.**kRand**(**"ccc"**),** std**::**runtime\_error**);**

109**|**

110**|** // test calling kRand on cTest2 with a correct kgram

111**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest2**.**kRand**(**"aaa"**));**

112**|** **}**

113**|**

114**|** BOOST\_AUTO\_TEST\_CASE**(**generate**)** **{**

115**|** // creating MModel to run freq on with kgram 0

116**|** MModel cTest**(**"gagggagaggcgagaaa"**,** 0**);**

117**|**

118**|** // test calling generate on cTest with an incorrect kgram

119**|** BOOST\_REQUIRE\_THROW**(**cTest**.**generate**(**"a"**,** 5**),** std**::**runtime\_error**);**

120**|**

121**|** // test calling generate on cTest with a correct kgram

122**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest**.**generate**(**""**,** 5**));**

123**|**

124**|** // test calling generate on cTest with a correct kgram and

125**|** // checking length of return

126**|** BOOST\_REQUIRE**(**cTest**.**generate**(**""**,** 5**).**length**()** **==** 5**);**

127**|**

128**|** // test calling generate on cTest with a correct kgram and

129**|** // checking length of return

130**|** BOOST\_REQUIRE**(**cTest**.**generate**(**""**,** 7**).**length**()** **==** 7**);**

131**|**

132**|** // creating MModel to run freq on with kgram 3

133**|** MModel cTest2**(**"gagggagaggcgagaaa"**,** 3**);**

134**|**

135**|** // test calling generate on cTest2 with an incorrect kgram

136**|** BOOST\_REQUIRE\_THROW**(**cTest2**.**generate**(**"a"**,** 5**),** std**::**runtime\_error**);**

137**|**

138**|** // test calling generate on cTest with a correct kgram

139**|** BOOST\_REQUIRE\_NO\_THROW**(**cTest2**.**generate**(**"ggg"**,** 5**));**

140**|**

141**|** // test calling generate on cTest with a correct kgram and

142**|** // checking length of return

143**|** BOOST\_REQUIRE**(**cTest2**.**generate**(**"aag"**,** 5**).**length**()** **==** 5**);**

144**|**

145**|** // test calling generate on cTest with a correct kgram and

146**|** // checking length of return

147**|** BOOST\_REQUIRE**(**cTest2**.**generate**(**"gcg"**,** 7**).**length**()** **==** 7**);**

148**|** **}**

149**|**

150**|** BOOST\_AUTO\_TEST\_CASE**(**overloaded\_function**)** **{**

151**|** // creating MModel to run freq on with kgram 0

152**|** MModel cTest**(**"gagggagaggcgagaaa"**,** 1**);**

153**|**

154**|** // redirects the constructors overloaded << to the boost test stream.

155**|** boost**::**test\_tools**::**output\_test\_stream output**;**

156**|**

157**|** output **<<** cTest**;**

158**|**

159**|** // test output is equal to expected input

160**|** BOOST\_REQUIRE**(**output**.**is\_equal**(**"Original text: gagggagaggcgagaaa"

161**|** "\nOrder: 1"

162**|** "\nAlphabet: gac\n"

163**|** "Markov Map: \n"

164**|** "Kgram: a Frequency: 7\n"

165**|** "Kgram+1: aa Frequency: 2\n"

166**|** "Kgram+1: ag Frequency: 5\n"

167**|** "Kgram: c Frequency: 1\n"

168**|** "Kgram+1: cg Frequency: 1\n"

169**|** "Kgram: g Frequency: 9\n"

170**|** "Kgram+1: ga Frequency: 5\n"

171**|** "Kgram+1: gc Frequency: 1\n"

172**|** "Kgram+1: gg Frequency: 3\n"

173**|** **));**

174**|** **}**

175**|**

#### MModel.h

01**|** #ifndef MMODEL\_H

02**|** #define MMODEL\_H

03**|**

04**|** #include **<**string**>**

05**|** #include **<**iostream**>**

06**|** #include **<**map**>**

07**|**

08**|** class MModel**{**

09**|** public**:**

10**|** MModel **(**std**::**string text**,** int k**);**

11**|** int kOrder**();**

12**|** int freq**(**std**::**string kgram**);**

13**|** int freq**(**std**::**string kgram**,** char c**);**

14**|** char kRand**(**std**::** string kgram**);**

15**|** std**::**string generate**(**std**::**string kgram**,** int L**);**

16**|** friend std**::**ostream**&** **operator<<** **(**std**::**ostream**&** os**,** MModel model**);**

17**|** private**:**

18**|** std**::**map**<**std**::**string**,** int**>** kGramMap**;**

19**|** std**::**string inputText**;**

20**|** std**::**string alphabet**;**

21**|** int order**;**

22**|**

23**|** **};**

24**|**

25**|** #endif // MMODEL\_H

#### MModel.cpp

001**|** #include **<**ctime**>**

002**|** #include **<**cstdlib**>**

003**|** #include "header/MModel.h"

004**|**

005**|**

006**|** MModel**::**MModel**(**std**::**string text**,** int k**)** **{**

007**|** srand**(**time**(NULL));**

008**|** order **=** k**;**

009**|** inputText **=** text**;**

010**|** int textLength **=** **(**unsigned**)** inputText**.**length**();**

011**|**

012**|** // generate alphabet

013**|** **for** **(**int i **=** 0**;** i **<** textLength**;** i**++)** **{**

014**|** **if** **(**alphabet**.**find**(**inputText**[**i**])** **==** std**::**string**::**npos**)** **{**

015**|** alphabet **+=** inputText**[**i**];**

016**|** **}**

017**|** **}**

018**|**

019**|** // create lambda expression to add key to kgram map

020**|** auto addToKGramMap **=** **[this](**std**::**string key**)** **{**

021**|** **if** **(**kGramMap**.**find**(**key**)** **==** kGramMap**.**end**())** **{**

022**|** kGramMap**[**key**]** **=** 1**;**

023**|** **}** **else** **{**

024**|** kGramMap**[**key**]** **+=** 1**;**

025**|** **}**

026**|** **};**

027**|**

028**|** // generate map

029**|** **for** **(**int i **=** 0**;** i **<** textLength**;** i**++)** **{**

030**|** std**::**string temp**;**

031**|**

032**|** // make kgram

033**|** **for** **(**int j **=** i**;** j **<** i **+** k**;** j**++)** **{**

034**|** temp **+=** inputText**[**j **%** textLength**];**

035**|** **}**

036**|**

037**|** // add kgram to kGramMap

038**|** addToKGramMap**(**temp**);**

039**|**

040**|** // generate k+1

041**|** temp **+=** inputText**[(**i **+** k**)** **%** textLength**];**

042**|**

043**|** // add kgram+1 to kGramMap

044**|** addToKGramMap**(**temp**);**

045**|** **}**

046**|** **}**

047**|**

048**|** // return order

049**|** int MModel**::**kOrder**()** **{**

050**|** **return** order**;**

051**|** **}**

052**|**

053**|** // return frequency of kgram

054**|** int MModel**::**freq**(**std**::**string kgram**)** **{**

055**|** **if** **(**kgram**.**length**()** **!=** **(**unsigned**)** order**)** **{**

056**|** **throw** std**::**runtime\_error**(**"kgram is not size k"**);**

057**|** **}**

058**|**

059**|** **if** **(**order **==** 0**)** **{**

060**|** **return** inputText**.**length**();**

061**|** **}**

062**|**

063**|** **return** kGramMap**[**kgram**];**

064**|** **}**

065**|**

066**|** // return frequency of c after kgram

067**|** int MModel**::**freq**(**std**::**string kgram**,** char c**)** **{**

068**|** **if** **(**kgram**.**length**()** **!=** **(**unsigned**)** order**)** **{**

069**|** **throw** std**::**runtime\_error**(**"kgram is not size k"**);**

070**|** **}**

071**|** **if** **(**order **==** 0**)** **{**

072**|** std**::**string s**;**

073**|** s**+=**c**;**

074**|** **return** kGramMap**[**s**];**

075**|** **}**

076**|** **return** kGramMap**[**kgram **+** c**];**

077**|** **}**

078**|**

079**|** // return char that could come after kgram

080**|** char MModel**::**kRand**(**std**::** string kgram**)** **{**

081**|** **if** **(**kgram**.**length**()** **!=** **(**unsigned**)** order**)** **{**

082**|** **throw** std**::**runtime\_error**(**"kgram is not size k"**);**

083**|** **}**

084**|** **if** **(**kGramMap**[**kgram**]** **==** 0**)** **{**

085**|** **throw** std**::**runtime\_error**(**"kgram is not an existing kgram"**);**

086**|** **}**

087**|**

088**|** // simulate frequency of next letter by adding more of them to string

089**|** std**::**string nextFrequency**;**

090**|** **for** **(**unsigned int i **=** 0**;** i **<** alphabet**.**length**();** i**++)** **{**

091**|** **for** **(**int j **=** 0**;** j **<** kGramMap**[**kgram **+** alphabet**[**i**]];** j**++)** **{**

092**|** nextFrequency **+=** alphabet**[**i**];**

093**|** **}**

094**|** **}**

095**|**

096**|** // return random char in nextFrequency

097**|** **return** nextFrequency**[**rand**()** **%** nextFrequency**.**size**()];**

098**|** **}**

099**|**

100**|** // generate a string of length L following Markov chain

101**|** std**::**string MModel**::**generate**(**std**::**string kgram**,** int L**)** **{**

102**|** **if** **(**kgram**.**length**()** **!=** **(**unsigned**)** order**)** **{**

103**|** **throw** std**::**runtime\_error**(**"kgram is not size k"**);**

104**|** **}**

105**|**

106**|** std**::**string generatedString **=** kgram**;**

107**|** std**::**string tempKGram **=** kgram**;**

108**|** **for** **(**int i **=** order**;** i **<** L **;** i**++)** **{**

109**|** // generate next char

110**|** char tempC **=** kRand**(**tempKGram**);**

111**|** // add next char to return string

112**|** generatedString **+=** tempC**;**

113**|**

114**|** // add next char to temkKGram

115**|** tempKGram **+=** tempC**;**

116**|** // delete first char in tempKGram

117**|** tempKGram**.**erase**(**0**,** 1**);**

118**|** **}**

119**|** **return** generatedString**;**

120**|** **}**

121**|**

122**|** // print model

123**|** std**::**ostream**&** **operator<<** **(**std**::**ostream**&** os**,** MModel model**)** **{**

124**|** os **<<** "Original text: " **<<** model**.**inputText**;**

125**|** os **<<** "\nOrder: " **<<** model**.**order **<<** "\nAlphabet: " **<<** model**.**alphabet**;**

126**|** os **<<** "\nMarkov Map: \n"**;**

127**|** unsigned int kOrder **=** **(**unsigned**)** model**.**kOrder**();**

128**|** **for** **(**auto const **&**it **:** model**.**kGramMap**)** **{**

129**|** **if** **(**it**.**first**.**length**()** **==** kOrder**)** **{**

130**|** os **<<** "Kgram: " **<<** it**.**first **<<** " Frequency: " **<<** it**.**second **<<** "\n"**;**

131**|** **}** **else** **{**

132**|** os **<<** "Kgram+1: " **<<** it**.**first **<<** " Frequency: " **<<** it**.**second **<<** "\n"**;**

133**|** **}**

134**|** **}**

135**|** **return** os**;**

136**|** **}**

137**|**

# PS6: Kronos Time Clock

### Discussion:

This project introduced me to regex. What I did in this project was to parse a log and figure out when the device attempted to boot and of the boot was successful. I used a regex to determine which lines were relevant to my goal, since they all had the same general format and were able to be grouped by a single regex for each task log entry. I then wrote the output related to my results in a file similar to the log file that I had parsed. If you know what the format of the data that you are looking for is, regex makes it much easier to filter out all the unnecessary noise that may accompany your important data. I am interested in building a web scraper using regex that will scrape the html code from webpages and isolate specific keywords that I may be interested in. I feel like a regex would be a good way to implement that if you can get the HTML to your C++ program.

### File Output (when run using device5\_intouch.log as the log file):

31063 (log.c.166) server started 2014-Jan-26 09:55:07 success elapsed time: 177000 ms

31274 (log.c.166) server started 2014-Jan-26 12:15:18 failure

31293 (log.c.166) server started 2014-Jan-26 14:02:39 success elapsed time: 165000 ms

32623 (log.c.166) server started 2014-Jan-27 12:27:55 failure

32641 (log.c.166) server started 2014-Jan-27 12:30:23 failure

32656 (log.c.166) server started 2014-Jan-27 12:32:51 failure

32674 (log.c.166) server started 2014-Jan-27 12:35:19 failure

32693 (log.c.166) server started 2014-Jan-27 14:02:38 success elapsed time: 163000 ms

33709 (log.c.166) server started 2014-Jan-28 12:44:17 failure

33725 (log.c.166) server started 2014-Jan-28 14:02:33 success elapsed time: 162000 ms

34594 (log.c.166) server started 2014-Jan-29 12:43:07 failure

34613 (log.c.166) server started 2014-Jan-29 14:02:35 success elapsed time: 164000 ms

37428 (log.c.166) server started 2014-Jan-30 12:43:05 failure

37447 (log.c.166) server started 2014-Jan-30 14:02:40 success elapsed time: 162000 ms

38258 (log.c.166) server started 2014-Jan-31 14:02:33 success elapsed time: 163000 ms

39150 (log.c.166) server started 2014-Feb-01 12:39:38 failure

39166 (log.c.166) server started 2014-Feb-01 12:42:07 failure

39182 (log.c.166) server started 2014-Feb-01 14:02:32 success elapsed time: 164000 ms

40288 (log.c.166) server started 2014-Feb-02 14:02:39 success elapsed time: 172000 ms

41615 (log.c.166) server started 2014-Feb-03 12:35:55 failure

41633 (log.c.166) server started 2014-Feb-03 12:38:22 failure

41648 (log.c.166) server started 2014-Feb-03 12:40:48 failure

41666 (log.c.166) server started 2014-Feb-03 12:43:17 failure

41684 (log.c.166) server started 2014-Feb-03 12:45:46 failure

41694 (log.c.166) server started 2014-Feb-03 14:02:34 success elapsed time: 164000 ms

### Code:

#### Makefile

01| CC **=** g++

02| CFLAGS **=** -std=c++11 -c -g -Og -Wall -Werror -pedantic -Iheader

03| OBJ **=** PS6.o

04| LIBS **=** -lboost\_regex -lboost\_date\_time

05| EXE **=** PS6

06|

07| all**:** PS6

08|

09| PS6**:** $(OBJ)

10| $(CC) $(OBJ) -o $(EXE) $(LIBS)

11|

12| PS6.o**:** PS6.cpp

13| $(CC) $(CFLAGS) -o $@ $<

14|

15|

16| clean**:**

17| rm \*.o

18| rm $(EXE)

19|

#### PS6.cpp

01**|** #include **<**iostream**>**

02**|** #include **<**string**>**

03**|** #include **<**fstream**>**

04**|** #include **<**boost**/**regex**.**hpp**>**

05**|** #include "boost/date\_time/gregorian/gregorian.hpp"

06**|** #include "boost/date\_time/posix\_time/posix\_time.hpp"

07**|**

08**|** **using** std**::**cout**;**

09**|** **using** std**::**cin**;**

10**|** **using** std**::**endl**;**

11**|** **using** std**::**string**;**

12**|**

13**|** **using** boost**::**gregorian**::**date**;**

14**|** **using** boost**::**gregorian**::**from\_simple\_string**;**

15**|** **using** boost**::**gregorian**::**date\_period**;**

16**|** **using** boost**::**gregorian**::**date\_duration**;**

17**|**

18**|** **using** boost**::**posix\_time**::**ptime**;**

19**|** **using** boost**::**posix\_time**::**time\_duration**;**

20**|** **using** boost**::**posix\_time**::**time\_from\_string**;**

21**|**

22**|** int main**(**int argc**,** char**\*** argv**[])** **{**

23**|** **if** **(**argc **!=**2**)** **{**

24**|** **throw** std**::**invalid\_argument**(**"There should only be 2 arguments!"**);**

25**|** **}**

26**|**

27**|** // open log file

28**|** std**::**fstream readLog**(**argv**[**1**],** std**::**fstream**::**in**);**

29**|** **if** **(!**readLog**.**is\_open**())** **{**

30**|** **throw** std**::**runtime\_error**(**"unable to open file"**);**

31**|** **}**

32**|**

33**|** // create report file

34**|** string reportName**(**string**(**argv**[**1**])** **+** ".rpt"**);**

35**|** std**::**fstream reportFile**(**reportName**.**c\_str**(),** std**::**fstream**::**out**);**

36**|**

37**|** // setup dateTime for regex

38**|** string dateTime**(**"([0-9]{4}-[0-9]{1,2}-[0-9]{1,2}) ([0-9]{2}:[0-9]{2}:[0-9]{2})"**);** // NOLINT

39**|**

40**|** // create lambda expression to automatically include

41**|** // dateTime at beginning of regex

42**|** auto make\_regex **=** **[**dateTime**](**string a**){**

43**|** **return** boost**::**regex**(**dateTime **+** a**);**

44**|** **};**

45**|**

46**|** // create regex

47**|** boost**::**regex boot **=** make\_regex**(**".\*(log.c.166).\*"**);**

48**|** boost**::**regex end **=** make\_regex**(**".\*oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080"**);** // NOLINT

49**|**

50**|** // hold matches

51**|** boost**::**smatch matches**;**

52**|**

53**|** // create string to store line in from file

54**|** string line**;**

55**|**

56**|** // create line number counter

57**|** int lineNumber **=** 1**;**

58**|**

59**|** // create boolean to remember if booting

60**|** bool isBooting **=** **false;**

61**|**

62**|** // create time variables for time calculation

63**|** ptime startTime**,** endTime**;**

64**|**

65**|** // go through file until EOF

66**|** **while** **(**getline**(**readLog**,** line**))** **{**

67**|** **if** **(**regex\_match**(**line**,** matches**,** boot**))** **{**

68**|** // set start time

69**|** startTime **=** time\_from\_string**(**matches**[**1**].**str**()** **+** " " **+** matches**[**2**].**str**());**

70**|**

71**|** // if already booting, we tried to start another boot so first one failed

72**|** **if** **(**isBooting**)** **{**

73**|** reportFile **<<** " failure\n"**;**

74**|** **}**

75**|** // print line number and start time.

76**|** reportFile **<<** lineNumber **<<** " (log.c.166) server started "**;**

77**|** reportFile **<<** startTime**;**

78**|** isBooting **=** **true;**

79**|**

80**|** **}** **else** **if** **(**regex\_match**(**line**,** matches**,** end**))** **{**

81**|** endTime **=** time\_from\_string**(**matches**[**1**].**str**()** **+** " " **+** matches**[**2**].**str**());**

82**|** // only print stuff if we are booting

83**|** **if** **(**isBooting**)** **{**

84**|** reportFile **<<** " success elapsed time: "**;**

85**|** reportFile **<<** **(**endTime **-** startTime**).**total\_milliseconds**()** **<<** " ms\n"**;**

86**|** isBooting **=** **false;**

87**|** **}**

88**|** **}**

89**|** // increment line number

90**|** **++**lineNumber**;**

91**|** **}**

92**|** **return** 0**;**

93**|** **}**

94**|**