Dev Patel

COMP IV Sec 203: Project Portfolio

Spring 2022

**Contents**

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

**PS1** Linear Feedback Shift Register and Image Encoding

**PS2** N-Body Simulation

**PS3** Recursive Graphics

**PS4** StringSound

**PS5** DNA Alignment

**PS6** Random Writer

**PS7** Kronos Time Parsing

Time to complete:

**PS0: Hello World with SFML**

The Assignment

This was our first assignment in Computing IV project. The purpose of this assignment was to get set up what Linux platform we wanted to use for the semester; and test out the SFML library. We had to get Linux running through VM Machine or natively and run the sample code to test out SFML. Our job was to extend the code and play around with SFML and do something interesting. Setting up WSL and running a few sudo-install commands was the easy part. Now I’m ready to see what I can do with SFML.

Key Concepts

There weren’t much fancy concepts used in this assignment as it is our first one. It was the simple stuff variables, loops, etc.… The main concept is to play around with SFML’s different classes. That’s where I used what SFML had to offer, Images, Sprites, Test, Texture, and Keyboard. I loaded an image into a Texture, and a sprite must be created to use the Texture you gave it. I created a source code that used SFML’s sprite class to output an image on a SFML window. One thing I added was that you can use the arrow keys to move the sprite around up/down/left/right the screen.

What I learned

In this assignment I learned a few things about SFML. To start of; I learned how

to use SFML at a basic level; how to display images, even moving them around using SFML’s keyboard class, and how to keep a window open while doing these things. I also learned how to use rotate() to angle my image. Aside from SFML, I learned a lot about Virtual Machines and WSL. I found it neat how you can have another minicomputer inside your computer. One thing I found fascinating is how you can control how much space of your computer you want to give it, and what it has access to. I learned that virtual machines can be a really good tool for dealing with internet viruses, spyware, and malware, as resetting it takes a second. But it doesn't protect you from tracking. You need a VPN for that matter.

**Screenshots:**Graphical user interface, application

Description automatically generated

**Source code for PS0 Hello World**

Makefile:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system  DEPS = main.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** Test  **Test:** main.o  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f Test |

Main.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | #include<iostream>  #include <SFML/Graphics.hpp>  **int** **main**() {  sf::RenderWindow window(sf::VideoMode(**600**, **600**), "works!");  window.setFramerateLimit(**120**);  **float** dt;  sf::Clock dt\_clock;  sf::Texture texture;  **if**(!texture.loadFromFile("sprite.png")) {  std::cout << "not found" << std::endl;  **return** **0**;  }  **const** **float** moveSpeed = **100.f**;  sf::Vector2f velocity;  sf::Sprite sprite;  sprite.setTexture(texture);  sprite.scale(sf::Vector2f(**0.1**, **0.1**));  // rotate sprite  sprite.rotate(**15**);  **while** (window.isOpen()) {  dt = dt\_clock.restart().asSeconds();  sf::Event event;  **while** (window.pollEvent(event)) {  **if** (event.type == sf::Event::Closed)  window.close();  }  velocity.y = **0.f**;  velocity.x = **0.f**;  // sprite responds to arrow keys  **if** (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {  velocity.y += -moveSpeed \* dt;  }  **if** (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {  velocity.y += moveSpeed \* dt;  }  **if** (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {  velocity.x += -moveSpeed \* dt;  }  **if** (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {  velocity.x += moveSpeed \* dt;  }  sprite.move(velocity);  // collision  **if** (sprite.getPosition().x < **0.f**) {  sprite.setPosition(**0.f**, sprite.getPosition().y);  }  **if** (sprite.getPosition().y < **0.f**) {  sprite.setPosition(sprite.getPosition().x, **0.f**);  }  **if** (sprite.getPosition().x + sprite.getGlobalBounds().width > **600**) {  sprite.setPosition(**600** - sprite.getGlobalBounds().width,  sprite.getPosition().y);  }  **if** (sprite.getPosition().y + sprite.getGlobalBounds().height > **600**) {  sprite.setPosition(sprite.getPosition().x, **600** -  sprite.getGlobalBounds().height);  }  window.clear();  window.draw(sprite);  window.display();  }  **return** **0**;  } |

**PS1 LFSR and Image Encoding**

The Assignment

This assignment required us to implement a linear feedback shift register, m that produces pseudo-random bits, and then use it to implement a simple form of encryption for digital pictures. This type of register shifts all bits left one position, and then XOR’s the left most bit and the seed bit to fill the empty space on the far right after the shift left. Our implantation in a class FibLFSR. To test the class and implementations, we use Boost test framework. Using the class, we are tasked with creating a program that reads in a photo from the command line and then outputs the same image but encrypted/encoded. The FibLFSR class is used to encode the image by left shifting all the bits in the image, encoding it using XOR. And we use SFML window to output the image and save it to a new file. The code also goes both ways, decrypting an encrypted image.

Key Concepts

Using Boost test framework to test our FibLFSR class. For the registered bits I grabbed tap positions 15,13,12,10 using .at(). I then turned each see into an integer using if/else statements. And now that I have it in integer format, I performed the arithmetic. And that is why I choose to do it with strings because I understand the string library more. I then used .append() to shifts the bits one position to the left and replace the vacated bit. I continued to use the same strategy for the generate function. I create an empty string and add each bit as an integer after calling step() k amount of times. Next, convert the contents of the bit set to an unsigned long, long integer, and return that value. The Boost test frameworks was used to test our FibLFSR class, using Boost’s to test the step/generate methods. We also used SFML objects such as sprites, texture, images to read in the file, encode the image and output encrypted image to and SFML window and a text file.

What I learned

In this assignment I took away a lot about using testing in C++. In the past to check my code I’ve done numerous things like; compiling and running and writing out each element in a print statement for example to make sure each part worked properly, and the computer was properly interpreting what I needed it to do and commenting out every time I do error checking. Unit tests are a game changer. Using Boost tests made thing simple. I know where my code is messing up. I want to continue using Boost test framework in the future for new projects especially ones that get complicated. Overloading the << operator wasn’t anything new, but it was nice to brush up on it. Overall, this assignment was fun to work on, learning that each pixel can be manipulated to get a desired or in this case hidden image.

**Screenshots:**

Image->Encrypted image

Chart, pie chart

Description automatically generated

Encrypted Image -> Decrypted ImageChart

Description automatically generated with medium confidence

**Source Code for PS1 Image Encoding**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  DEPS = FibLFSR.h PhotoMagic.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** PhotoMagic  **PhotoMagic:** FibLFSR.o PhotoMagic.o  $(CC) $(CFLAGS) -o $@ $^ -lboost\_unit\_test\_framework  -lsfml-graphics -lsfml-window -lsfml-system  **clean:**  rm -f \*.o  rm -f PhotoMagic |

FibLFSR.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include<string>  #include<iostream>  #include<cstdlib>  #include <stdio.h>  #include <stdlib.h>  #include <bitset>  #include <SFML/System.hpp>  #include <SFML/Window.hpp>  #include <SFML/Graphics.hpp>  **using** **namespace** std;  **using** **namespace** sf;  **class** **FibLFSR** {  **public:**  FibLFSR();  FibLFSR(string seed) : seed1(seed) {}  string getSeed(**void**)**const** {  **return** seed1;  }  **int** step();  **int** **generate**(**int** k);    **private:**  string seed1;  };  FibLFSR.cpp |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | #include "FibLFSR.h"  //Dev Patel  **int** FibLFSR:: step() {  **char** bit15 = seed1.at(**0**); //extract bit 15  **char** bit13 = seed1.at(**2**); //extract bit 13  **char** bit12 = seed1.at(**3**); //extract bit 12  **char** bit10 = seed1.at(**5**); //extract bit 10  **int** first;  **int** x;  **if** (bit15 == '0') { // turn bit 15 string into int  first = **0**;  }  **else** {  first = **1**;  }  **if** (bit13 == '0') { // turn bit 13 string into int  x = **0**;  }  **else** {  x = **1**;  }  x = first ^ x; // xor bit 15 and bit 13  **if** (bit12 == '0') { // turn bit 12 string into int  first = **0**;  }  **else** {  first = **1**;  }  x = x ^ first; //xor result of bit(15^13) with bit 12  **if** (bit10 == '0') { // turn bit 10 string into int  first = **0**;  }  **else** {  first = **1**;  }  x = x ^ first; //xor previous result with bit 10  //shifts bit one position and replacing vacated bit with final result  //by:  seed1.append(to\_string(x)); //appending result on  seed1.erase(**0**,**1**); //erasing first bit  **return** x; //return value  }  **int** FibLFSR:: generate(**int** k){  string s = "";  **int** num;    **for**(**int** i = **0**; i < k; i++){ //run step k amount of time and inset  //into empty string  num = step();  s += to\_string(num);  }  **unsigned** **long** **long** value = bitset<**64**>(s).to\_ullong(); //convert binary  //to decimal using to\_ullong  **return** value;  }  ostream& **operator**<<(ostream& out, **const** FibLFSR& bit){  cout << bit.getSeed() << " "; //print seed  **return** out;  } | |

PhotoMagic.cpp (Main)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | #include "FibLFSR.h"  **void** **transform**(Image& picutre, FibLFSR\* l1);  **int** **main**(**int** argc, **char**\* argv[])  {  sf::Image image;  **if** (!image.loadFromFile(argv[**1**]))  **return** -**1**;  FibLFSR l1(argv[**3**]);  sf::Color p;  transform(image, &l1);  sf::Vector2u size = image.getSize();  sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "window1");  sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "window2");  sf::Texture texture;  texture.loadFromImage(image);  sf::Sprite sprite;  sprite.setTexture(texture);  sf::Image image1;  **if** (!image1.loadFromFile(argv[**1**])){  **return** -**1**;  }    sf::Texture texture1;  texture1.loadFromImage(image1);  sf::Sprite sprite2;  sprite2.setTexture(texture1);  **while** (window1.isOpen() && window2.isOpen())  {  sf::Event event;  **while** (window1.pollEvent(event))  {  **if** (event.type == sf::Event::Closed)  window1.close();  }  **while** (window2.pollEvent(event))  {  **if** (event.type == sf::Event::Closed)  window2.close();  }  window1.clear();  window1.draw(sprite2);  window1.display();  window2.clear();  window2.draw(sprite);  window2.display();  }  // fredm: saving a PNG segfaults for me, though it does properly  // write the file  **if** (!image.saveToFile(argv[**2**]))  **return** -**1**;  **return** **0**;  }  **void** **transform**(Image& picture, FibLFSR\* l1){  Color p;  **int** width;  width = picture.getSize().x;  **int** height;  height = picture.getSize().y;  **for** (**int** x = **0**; x<width; x++) {  **for** (**int** y = **0**; y< height; y++) {  p = picture.getPixel(x, y);  p.r = p.r ^ l1->generate(**100**);  p.g = p.g ^ l1->generate(**100**);  p.b = p.b ^ l1->generate(**100**);  picture.setPixel(x, y, p);  }  }  } |

**PS2 N-Body Simulation**

The Assignment

In this assignment we create a program that loads and displays a static universe. And then we will add the physics simulation and animate the display! Model the universe on a 2d plane using Newtons laws of gravity to make the simulation realistic. My job is to read in a file from standard I/0 and using that data that I read in to display the planets(sprites) and give them the info they need to be in the correct positions at the correct times. The code takes in two command lines, the total elapsed time for the simulation and how much you want to step the time by. “./NBody 157788000.0 25000.0 < planets.txt”. The finished product is a moving universe on an SFML window.

Key Concepts

Overall, this assignment uses many concepts. The first key step was to read in the data from the file. To do that < is used on the command line. In the source code I overloaded the >> to read in the data in the correct order. Another key concept was the overall physics used to correctly simulate the universe; Newton’s law of universal gravitation, newtons second law of motion. Using formulas for calculating force, net force, acceleration, change in velocity. These formula’s can be found on <https://courses.cs.duke.edu/compsci201/fall17/assign/nbody/>. In using these formulas, I was able to simulate the planets movement in correspondence with each other. The planets moved by updating their velocities and position using the physics calculations given.

What I learned

There are so many parts that went into this assignment, and I felt like there were so many things I didn’t yet. One big thing I learned to use is the draw method in our Celestial Body class. I had never used it before so getting to learn how use that to my advantage was exciting. Another thing I didn’t have too much knowledge in is overloading the >> though the function in this code is very light it still took me a few tries to get it properly read in the right data. I also learned how to display a running clock on the SFML window. Using the SFML libraries to display the planets was not an issue, it was trying to use coordinates that aren’t the same as the SFML window used to output the display. Another thing that I had little to no knowledge on is physics. So, because of this assignment I learned a little bit of physics and how to actually implement the equations into the sprites.

**Screenshots**A picture containing outdoor object, star, night sky

Description automatically generated

**Text

Description automatically generated**

**Source Code for PS2 N-Body Simulation**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system  DEPS = CelestialBody.cpp Universe.h    %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** NBody  **NBody:** CelestialBody.o main.o Universe.o  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f NBody |

CelestialBody.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | #pragma once  #include <fstream>  #include <iostream>  #include <string>  #include <SFML/System.hpp>  #include <SFML/Window.hpp>  #include <SFML/Graphics.hpp>  #include <memory>  **using** **namespace** std;  **using** **namespace** sf;  **class** **CelestialBody** : **public** Drawable{  **public:**  CelestialBody();  CelestialBody(**double** initX, **double** initY, **double** initVelX, **double** initVelY, **double** initMass, string initFile);  **void** **setx**(**double** initX){  x = initX;  }  **void** **sety**(**double** initY){  y = initY;  }  **void** **setVelX**(**double** initVelX){  intVelocityX = initVelX;  }  **void** **setVelY**(**double** initVelY){  intVelocityY = initVelY;  }  **void** **setMass**(**double** initMass){  mass = initMass;  }  **double** **getX**(**void**){  **return** x;  }  **double** **getY**(**void**){  **return** y;  }  **double** **getIntVelocityX**(**void**){  **return** intVelocityX;  }  **double** **getIntVelocityY**(**void**){  **return** intVelocityY;  }  **double** **getMass**(**void**){  **return** mass;  }  string **getFile**(**void**){  **return** fileName;  }  **friend** istream& **operator**>>(istream& in, CelestialBody &thing);  **void** **set**(**int** sizeX1, **int** sizeY1, **double** uRadius);  **private:**  **double** x;  **double** y;  **double** intVelocityX;  **double** intVelocityY;  **double** mass;  string fileName;  Sprite sprite1;  Texture texture1;  **virtual** **void** **draw**(sf::RenderTarget& target, sf::RenderStates states) **const**  {  target.draw(sprite1, states);  }  }; |

CelestialBody.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #include "Universe.h"  CelestialBody::CelestialBody(**double** initX, **double** initY, **double** initVelX, **double** initVelY, **double** initMass, string initFile){  x = initX;  y = initY;  intVelocityY = initVelY;  intVelocityX = initVelX;  mass = initMass;  fileName = initFile;  Image image;  image.loadFromFile(fileName);  texture1.loadFromImage(image);  sprite1.setTexture(texture1);  }  istream& **operator**>> (istream& in, CelestialBody &thing) {  in >> thing.x >> thing.y >> thing.intVelocityX >> thing.intVelocityY >> thing.mass >> thing.fileName;  **return** in;  }  **void** CelestialBody::set(**int** sizeX1, **int** sizeY1, **double** uRadius){  **double** tempX = (sizeX1 / **2**) + ((**this**->x / uRadius)/**2**) \* sizeX1;  **double** tempY = (sizeY1 / **2**) - ((**this**->y / uRadius)/**2**) \* sizeY1;    **this**->sprite1.setPosition(tempX, tempY);  } |

Universe.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | #pragma once  #include "CelestialBody.h"  #include <utility>  **class** **Universe**{  **public:**  **friend** **const** istream& **operator**>> (istream& in, Universe &thing);  **friend** ostream& **operator**<< (ostream &output, Universe &cBody);  **int** **getNumPlanets**(){  **return** numPlanets;  }  **double** **getNumRadius**(){  **return** radius;  }    CelestialBody **getBody**(**int** i){  **return** \*(planet[i]);  }    **void** **setXval**(**double** initX){  xVal = initX;  }  **void** **setYval**(**double** initY){  yVal = initY;  }  **void** **scale**(**int** sizeX, **int** sizeY);  **void** **step**(**double** stepTime);  pair<**double**,**double**>calcPlanetForce(shared\_ptr<CelestialBody> currentPlanet);  **private:**  vector<shared\_ptr<CelestialBody>> planet;  **double** radius;  **int** numPlanets;  **double** seconds;  **double** xVal;  **double** yVal;  }; |

Universe.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | #include "Universe.h"  #include <cmath>  #include <utility>  **const** **double** G = (**6.67**\*pow(**10.0**,-**11.0**));  **const** istream& **operator**>> (istream& in, Universe &thing){  in >> thing.numPlanets >> thing.radius;  // CelestialBody object;  **double** x,y,velX,velY,mass;  string file;  **for**(**int** i = **0**; i < thing.numPlanets; i++){  in >> x >> y >> velX >> velY >> mass >> file;  thing.planet.push\_back(shared\_ptr<CelestialBody>(**new** CelestialBody(x, y, velX, velY, mass, file)));  }  **return** in;  }  ostream& **operator**<< (ostream &output, Universe &cBody){  output << cBody.getNumPlanets() << endl;  output << cBody.getNumRadius() << endl;    **for**(**int** i = **0**; i < cBody.getNumPlanets(); i++){  output << cBody.planet[i]->getX() <<"**\t**" << cBody.planet[i]->getY() <<"**\t**" << cBody.planet[i]->getIntVelocityX() <<"**\t**" << cBody.planet[i]->getIntVelocityY() <<"**\t**" << cBody.planet[i]->getMass() << "**\t**" << cBody.planet[i]->getFile() << endl;  }  **return** output;  }  **void** Universe::scale(**int** sizeX, **int** sizeY){  **for**(**int** i=**0**; i < getNumPlanets(); i++){  **this**->planet[i]->set(sizeX,sizeY,radius);  }  }  **void** Universe::step(**const** **double** stepTime){  vector<pair<**double**, **double**>> forces;  **double** aX = **0.0**,aY = **0.0**;  **double** vX = **0.0**,vY = **0.0**;  **double** pX = **0.0**, pY = **0.0**;  **for** (**int** i = **0**; i < numPlanets; i++) {  forces.push\_back(calcPlanetForce(planet[i]));  }  **for**(**int** k = **0**; k < numPlanets; k++){  aX = forces[k].first/planet[k]->getMass();  aY = forces[k].second/planet[k]->getMass();  vX = planet[k]->getIntVelocityX() + (stepTime\*aX);  vY = planet[k]->getIntVelocityY() + (stepTime\*aY);  pX = planet[k]->getX() + (stepTime\*vX);  pY = planet[k]->getY() + (stepTime\*vY);  planet[k]->setVelX(vX);  planet[k]->setVelY(vY);    planet[k]->setx(pX);  planet[k]->sety(pY);  planet[k]->set(xVal, yVal, radius);  }    }  pair<**double**,**double**>Universe::calcPlanetForce(shared\_ptr<CelestialBody> currentPlanet) {  **double** deltaX = **0.0**,deltaY = **0.0**;  **double** rad = **0.0**,rad2 = **0.0**;  **double** f = **0.0**;  **double** fX = **0.0**, fY = **0.0**;  **for** (**int** i = **0**; i < numPlanets; i++) {  shared\_ptr<CelestialBody> comparePlanet = planet[i];  **if** (currentPlanet != comparePlanet) {  deltaX = (comparePlanet->getX() - currentPlanet->getX());  deltaY = (comparePlanet->getY() - currentPlanet->getY());  rad = sqrt(pow(deltaX, **2.0**) + pow(deltaY, **2.0**));  rad2 = pow(deltaX, **2.0**) + pow(deltaY, **2.0**);  f = (G\*currentPlanet->getMass()\*comparePlanet->getMass())/rad2;  fX += (f \* (deltaX/rad));  fY += (f \* (deltaY/rad));  }  }  **return** pair<**double**, **double**> (fX, fY);  } |

Main.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | #include <iostream>  #include <string>  #include <stdlib.h>  #include <math.h>  #include <SFML/System.hpp>  #include <SFML/Window.hpp>  #include <SFML/Graphics.hpp>  #include "CelestialBody.h"  #include "Universe.h"  **using** **namespace** std;  **int** **main**(**int** argc, **char** \*argv[]){  sf::Image image;  Universe uni;  cin >> uni;  **if** (!image.loadFromFile("starfield.jpg")){  **return** -**1**;  }  sf::Vector2u size = image.getSize();  uni.setXval(size.x);  uni.setYval(size.y);  sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "window1");  sf::Texture texture;  texture.loadFromImage(image);  **double** time = stod(argv[**1**]);  **double** deltaTime = stod(argv[**2**]);  **double** cTime = **0.0**;  sf::Sprite sprite;  sprite.setTexture(texture);  uni.scale(size.x,size.y);  window1.setFramerateLimit(**120**);  sf::Font font;  font.loadFromFile("Roboto-Regular.ttf");  sf::Text text(std::to\_string(cTime), font);  text.setCharacterSize(**30**);  text.setFillColor(sf::Color::White);  **while** (window1.isOpen())  {  sf::Event event;  **while** (window1.pollEvent(event))  {  **if** (event.type == sf::Event::Closed)  window1.close();  }  window1.clear();  window1.draw(sprite);  text.setString(std::to\_string(cTime));  window1.draw(text);  **if**(cTime < time){  uni.step(deltaTime);  cTime += deltaTime;  }  **else** **if**(cTime >= time){  cout << uni << endl;  window1.close();  }    **for**(**int** i =**0**; i < uni.getNumPlanets(); i++){  window1.draw(uni.getBody(i));  }  window1.display();  }    **return** **0**;  } |

**PS3 Recursive Graphics**

The Assignment

In this assignment, our job is to implement a variation of the Sierpinski triangle. The Polish mathematician Wacław Sierpiński described the pattern in 1915. The main idea behind this assignment is to use recursion to create the triangle. We are tasked with creating a program that shall take two command-line arguments L and N. L is the length of the side of the base equilateral triangle (double), N is the depth of the recursion (int). The program draws one 1 triangle as the base. Recursion then draws a triangle at each point of the triangle half the length of the triangle its attached to, and so on – in effect drawing triangles around each other recursively.

Key Concepts

Of course, the key idea to this program is recursion. Another key data structure I used is Vector2f, to manipulate the x and y positions of the centers of my triangles and triangle's vertices. To draw this Sierpinski triangle I set all my original points for each vertex of the triangle in my draw function. I then create a constructor to find the original points of the triangle given the center of the screen. Next, in my recursive fTree() function I call my object and pass in the given length by command line and the center and draw. I then call fTree() within fTree() to get the position of the 3 new points of each triangle given their new centers in reference to the triangle.

What I Learned

I learned more about recursion. I’ve dealt with recursion before in previous classes, but I always found them so difficult but after working on this assignment the concept is a bit clearer. Before I wasn’t aware I could use recursion with objects to go back and draw them out. After learning how to draw just an object with recursion implementing the idea to the triangle was easier. I really like the output of this project, and I learned a different way to use recursion then how I used to do it before. I also, learned about cpplint in this assignment. I think it is a neat tool. It really helped me correct my poor coding habits and overall made my code look neat and very easily readable.

**Screenshots:A picture containing text

Description automatically generated**

**Source Code for PS3 Recursive Graphics**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system  DEPS = Triangle.h TFractal.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** TFractal  **TFractal:** Triangle.o TFractal.o  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f TFractal |

Triangle.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #ifndef TFRACT // NOLINT  #define TFRACT  #include <iostream>  #include <string>  #include <vector>  #include <memory>  #include <SFML/System.hpp>  #include <SFML/Window.hpp>  #include <SFML/Graphics.hpp>  #endif // NOLINT  **class** **Tri**: **public** sf::Drawable {  **public:**  Tri(**double** sLength, sf::Vector2f center) {  left.x = center.x - sLength/**2**;  left.y = center.y - sLength/**2**;  right.x = center.x + sLength/**2**;  right.y = center.y - sLength/**2**;  bottom.x = center.x;  bottom.y = center.y + sLength/**2**;  }  sf::Vector2f newCenterLeft(**double** sLength);  sf::Vector2f newCenterRight(**double** sLength);  sf::Vector2f newCenterBottom(**double** sLength);  **private:**  sf::Vector2f left;  sf::Vector2f right;  sf::Vector2f bottom;  **virtual** **void** draw(sf::RenderTarget& target, sf::RenderStates states) **const**; // NOLINT  }; |

Triangle.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #include "Triangle.h"  #include <vector>  #include <cmath>  **void** Tri::draw(sf::RenderTarget& target, sf::RenderStates states) **const** {  sf::ConvexShape tri;  tri.setPointCount(**3**);  tri.setPoint(**0**, left);  tri.setPoint(**1**, right);  tri.setPoint(**2**, bottom);  tri.setFillColor(sf::Color::Transparent);  tri.setOutlineColor(sf::Color::Red);  tri.setOutlineThickness(**2**);  tri.setOrigin(**0**, **0**);  target.draw(tri);  }  sf::Vector2f Tri::newCenterLeft(**double** sLength) {  **double** height = sLength/**2** \* **0.5** \* sqrt(**3**);  **return** sf::Vector2f(left.x, left.y - (height/**2**));  }  sf::Vector2f Tri::newCenterRight(**double** sLength) {  **double** height = sLength/**2** \* **0.5** \* sqrt(**3**);  **return** sf::Vector2f(right.x + (sLength/**4**), right.y + (height/**2**));  }  sf::Vector2f Tri::newCenterBottom(**double** sLength) {  **double** height = sLength/**2** \* **0.5** \* sqrt(**3**);  **return** sf::Vector2f(bottom.x - (sLength/**4**), bottom.y + (height/**2**));  } |

TFractal.cpp (Main)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | #include <iostream>  #include <SFML/System.hpp>  #include <SFML/Window.hpp>  #include <SFML/Graphics.hpp>  #include "Triangle.h"  **void** **fTree**(**double** length, **int** depth, sf::RenderWindow &window,  sf::Vector2f center); // NOLINT  **int** **main**(**int** argc, **char** \*argv[]) {  **double** length = atoi(argv[**1**]);  **int** depth = atoi(argv[**2**]);  sf::RenderWindow window(sf::VideoMode(length\***4**, length\***4**),  "TRIANGLE");  window.setFramerateLimit(**60**);  sf::Vector2f center(window.getSize().x/**2**, window.getSize().y/**2**);  **while** (window.isOpen()) {  sf::Event event;  **while** (window.pollEvent(event)) {  **if** (event.type == sf::Event::Closed)  window.close();  }  window.clear();  fTree(length, depth, window, center);  window.display();  }  **return** **0**;  }  **void** **fTree**(**double** length, **int** depth, sf::RenderWindow &window,  sf::Vector2f center) { // NOLINT  Tri triangle(length, center);  window.draw(triangle);  **if** (depth > **0**) {  fTree(length/**2**, depth - **1**, window,  triangle.newCenterLeft(length));  fTree(length/**2**, depth - **1**, window,  triangle.newCenterRight(length));  fTree(length/**2**, depth - **1**, window,  triangle.newCenterBottom(length));  } **else** {  **return**;  }  } |

**PS4 Circular Buffer**

The Assignment

In this assignment we implement a circular buffer, are fixed-size buffers that work as if the memory is contiguous & circular in nature. As memory is generated and consumed, data does not need to be reshuffled – rather, the head/tail pointers are adjusted. When data is added, the head pointer advances. When data is consumed, the tail pointer advances. If you reach the end of the buffer, the pointers simply wrap around to the beginning. The buffer is implemented by exceptions and try/catch blocks, we used the RingBuffer to create a model of a Guitar, by implementing the Karplus-Strong algorithm to simulate the plucking of a guitar string. We were tasked with implementing a few methods to simulate the playing of a guitar – such as pluck, tic, sample, etc. The keys are programmed in the main to respond to each key being pressed.

Key Concepts

The concepts this assignment revolved around is the Circular buffer and the Karplus-Strong algorithm, which how you simulate the guitar being played. The Karplus-Strong algorithm works by modeling frequencies, and it takes the first two values, averages them, and then multiplies the result by the energy decay factor, which in our case was .996. Combining this with the Circular buffer allows us to better simulate the sounds of a guitar being plucked. Another key concept I used in this assignment is C++ exceptions for error handling. This I how made sure my StringSound class was properly implemented.

What I learned

There was a lot I took away from assignment. To start I learned to better use exceptions. I had somewhat used them in Computing III but haven’t looked at them since so this helped shake some of the rust off. I learned that exceptions are perfect to keep the project from doing the wrong cases and it adds some security to my code. Also, learning about Karplus-Strong was interesting to see that develop into the guitar strings. It was difficult to implement the algorithm with the buffer as my buffer wouldn’t update correctly for a long time until I switched to using a vector as my data structure for my buffer. I’ve been using vectors for a long time, so I had more experience on how-to better use it. And then assigning the proper keys to the right frequencies was the last part.

**Screenshots**

No screenshot. When you run code use keyboard to play sounds

**Source Code for PS4 Circular Buffer**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio  DEPS = CircularBuffer.h StringSound.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** CircularBuffer  **KSGuitarSim:** CircularBuffer.o StringSound.o KSGuitar.o test.o  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f KSGuitarSim |

CircularBuffer.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #pragma once  #ifndef BUFF // NOLINT  #define BUFF  #include <stdint.h>  #include <iostream>  #include <vector>  #include <memory>  #include <stdexcept>  #include <SFML/Graphics.hpp>  #include <SFML/System.hpp>  #include <SFML/Audio.hpp>  #include <SFML/Window.hpp>  #endif // NOLINT  **class** **CircularBuffer**{  **public:**  CircularBuffer(**size\_t** capacity); // create an empty ring buffer,  //with given max capacity  **size\_t** **size**(); // return number of items currently in the buffer  **bool** **isEmpty**(); // is the buffer empty (size equals zero)?  **bool** **isFull**(); // is the buffer full (size equals capacity)?  **void** **enqueue**(**int16\_t** x); // add item x to the end  **int16\_t** **dequeue**(); // delete and return item from the front  **int16\_t** **peek**(); // return (but do not delete) item from the front  **private:**  **size\_t** capacity1; // my array capacity  **size\_t** size1; // my array size  std::vector<**int16\_t**> ringB; // my ring buffer  }; |

CircularBuffer.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | #include "CircularBuffer.h"  CircularBuffer::CircularBuffer(**size\_t** capacity) { // takes in cap  // size to set the vector to  ringB.reserve(capacity); // only way to give vector capacity we want  size1 = **0**;  capacity1 = capacity;  **if** (capacity < **1**) {  **throw**  std::invalid\_argument("CircularBufferconstructor: capacity  must be greater than zero.");  // capacity cannot be less than 1  }  }  **size\_t** CircularBuffer::size() {  **return** size1; // return number of items in buffer  }  **bool** CircularBuffer::isEmpty() {  **if** (size1 > **0**) { // if the size is greater than zero the buffer  **return** false; // is not empty  } **else** {  **return** true; // if the size is not greater than zero the buffer  } // is empty  }  **bool** CircularBuffer::isFull() {  **if** (size1 < capacity1) { // if the size of the buffer is  **return** false; // less than the capacity the buffer is not full  } **else** {  **return** true; // if the size of the buffer is  } // equal to or greater than the capacity the buffer is full  }  **void** CircularBuffer::enqueue(**int16\_t** x) { // takes in item  **if** (isFull()) { // cannot add item to buffer if it is full  **throw** std::runtime\_error("enqueue: can't enqueue to a full ring.");  }  ringB.push\_back(x); // push back add x to the end of the buffer  size1++; // increase size by 1  }  **int16\_t** CircularBuffer::dequeue() {  **if** (isEmpty()) {  **throw**  std::runtime\_error("dequeue: can't dequeue to an empty ring.");  // cannot remove item if there is no item(buffer is empty)  }  **int16\_t** temp = ringB.front(); // hold the front value for later return  ringB.erase(ringB.begin()); // erase that front value  size1--; // decrease size by 1  **return** temp; // return that front value  }  **int16\_t** CircularBuffer::peek() {  **if** (isEmpty()) {  **throw**  std::runtime\_error("peek: cant peek to an empty ring.");  // cannot return item if there is no item(buffer is empty) to return  }  **return** ringB.front(); // return front value  } |

KSGuitarSim.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83 | #include <limits.h>  #include "CircularBuffer.h"  #include "StringSound.h"  #include <cmath>  #include <iostream>  #include <string>  #include <exception>  #include <stdexcept>  #include <vector>  #include <SFML/Graphics.hpp>  #include <SFML/System.hpp>  #include <SFML/Audio.hpp>  #include <SFML/Window.hpp>  #define CONCERT\_A 220.0  #define SAMPLES\_PER\_SEC 44100  // const double FACTOR = 0.996;  **const** std::string KEY = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";  // NOLINT  std::vector<sf::Int16> makeSamples(StringSound& gs) { // NOLINT  std::vector<sf::Int16> samples;  gs.pluck();  **int** duration = **8**; // seconds  **int** i;  **for** (i= **0**; i < SAMPLES\_PER\_SEC \* duration; i++) {  gs.tic();  samples.push\_back(gs.sample());  }  **return** samples;  }  **int** main() {  sf::RenderWindow window(sf::VideoMode(**300**, **200**), "SFML Plucked String  Sound Lite");  sf::Event event;  **double** freq;  std::vector<sf::Int16> samples;  std::vector<std::vector<sf::Int16>> first;  std::vector<sf::SoundBuffer> sec;  std::vector<sf::Sound> third;  **for** (**int** i = **0**; i < **37**; i++) {  freq = **440.0** \* pow(**2.0**, (i-**24.0**)/**12.0**);  StringSound **gs1**(freq);  first.push\_back(makeSamples(gs1));  }  **for** (**int** i = **0**; i < **37**; i++) {  sf::SoundBuffer buf1;  **if** (!buf1.loadFromSamples(&first[i][**0**], first[i].size(), **2**,  SAMPLES\_PER\_SEC)) {  **throw** std::runtime\_error("failed to load from samples");  }  sec.push\_back(buf1);  }  **for** (**int** i = **0**; i < **37**; i++) {  sf::Sound sound1;  sound1.setBuffer(sec[i]);  third.push\_back(sound1);  }  **while** (window.isOpen()) {  **while** (window.pollEvent(event)) {  **if** (event.type == sf::Event::Closed) {  window.close();  }  **if** (event.type == sf::Event::TextEntered) {  **unsigned** **int** temp = KEY.find(event.text.unicode);  **if** (temp < KEY.size()) {  third[temp].play();  }  }  window.clear();  window.display();  }  }  **return** **0**;  } |

**PS5 DNA Alignment**

The Assignment

In this assignment we wrote a program to compute the optimal sequence alignment of two DNA strings, using dynamic programming. we measured the similarity of two genetic sequences by their edit distance, a concept first introduced in the context of coding theory, but which is now widely used in spell checking, speech recognition, plagiarism detection, file revisioning, and computational linguistics. We align the two sequences, but we are permitted to insert gaps in either sequence. We pay a penalty for each gap that we insert and for each pair of characters that mismatch in the final alignment.

Key Concepts

The main concept used in this assignment is known as Needleman-Wunsch method, which is a way of dynamic programming. The key idea of dynamic programming is to break up a large computational problem into smaller subproblems, store the answers to those smaller subproblems, and, eventually, use the stored answers to solve the original problem. This avoids recomputing the same quantity repeatedly. Instead of using recursion, use a nested loop that calculates opt[i][j] in the right order so that opt[i+1][j+1], opt[i+1][j], and opt[i][j+1] are all computed before we try to compute opt[i][j]. In this case, we used an NxM matrix to do so. This works by first calculating the easy edit distances – and then using those solutions to find the next round of edit distances, until you get at the solution in the [0][0] cell of the matrix.

What I learned

I learned more about valgrind more than anything in this assignment. I’ve only used a little bit in the past. It took me a while to get my valgrind to alone work. I also learned how to ostringstream to output the alignment.

**Screenshots**

**Text

Description automatically generated**

**Source Code for PS5 DNA Alignment**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system  DEPS = EDistance.h main.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** EDistance  **EDistance:** EDistance.o main.o  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f EDistance |

EDistance.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #pragma once  #include <stdint.h>  #include <iostream>  #include <string>  #include <sstream>  #include <algorithm>  #include <vector>  #include <memory>  #include <stdexcept>  #include <SFML/Graphics.hpp>  #include <SFML/System.hpp>  #include <SFML/Audio.hpp>  #include <SFML/Window.hpp>  **class** **ED** {  **public:**  ED();  ED(std::string \_s1, std::string \_s2);  **int** **penalty**(**char** a, **char** b);  **int** **min**(**int** a, **int** b, **int** c);  **int** **OptDistance**();  std::string Alignment();  **private:**  std::string s1; // NOLINT  std::string s2; // NOLINT  std::vector<std::vector<**int**>> \_M;  }; |

Edistance.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94 | #include "EDistance.h"  ED::ED(std::string \_s1, std::string \_s2) {  s1 = \_s1;  s2 = \_s2;  }  **int** ED::penalty(**char** a, **char** b) {  **if** (a == b) {  **return** **0**;  } **else** **if** (a != b) {  **return** **1**;  }  **return** -**1**;  }  **int** ED::min(**int** a, **int** b, **int** c) {  **if** (a < b && a < c) {  **return** a;  } **else** **if** (b < a && b < c) {  **return** b;  } **else** **if** (c < a && c < b) {  **return** c;  }  **return** a;  }  **int** ED::OptDistance() {  **for**(**int** i = **0**; (**unsigned**)i <= s2.size(); i++) {  std::vector<**int**> temp;  \_M.push\_back(temp);  **for** (**int** k = **0**; (**unsigned**)k <= s1.size(); k++) {  \_M.at(i).push\_back(**0**); // push 0's to fill vector  }  }  **for**(**int** i = **0**; (**unsigned**)i <= s2.size(); i++) { // bottom row  \_M[i][s1.size()] = **2** \* (s2.size() - i);  }  **for**(**int** k = **0**; (**unsigned**)k <= s1.size(); k++) { // side row  \_M[s2.size()][k] = **2** \* (s1.size() - k);  }  **for**(**int** i = s2.size() - **1**; i >= **0**; i--) {  **for** (**int** k = s1.size() - **1**; k >= **0**; k--) {  **int** o1 = \_M[i+**1**][k+**1**] + penalty(s1[k], s2[i]);  **int** o2 = \_M[i+**1**][k] + **2**;  **int** o3 = \_M[i][k+**1**] + **2**;  \_M[i][k] = min(o1, o2, o3);  }  }  **return** \_M[**0**][**0**];  }  std::string ED::Alignment() {  std::ostringstream return\_string;  **int** M = s2.length();  **int** N = s1.length();  **int** i = **0**, j = **0**;  **int** pen, opt1, opt2, opt3;  std::string retS;  **while** (i < M || j < N) { // run until we hit the far bottom right  try {  pen = penalty(s1[j], s2[i]);  opt1 = \_M.at(i+**1**).at(j+**1**) + pen;  }  **catch**(**const** std::out\_of\_range& error) {  opt1 = -**1**;  }  try {  opt2 = \_M.at(i+**1**).at(j) + **2**;  }**catch**(**const** std::out\_of\_range& error) {  opt2 = -**1**;  }  try {  opt3 = \_M.at(i).at(j+**1**) + **2**;  }**catch**(**const** std::out\_of\_range& error) {  opt3 = -**1**;  }  // Move diagonally  **if** (\_M[i][j] == opt1) {  return\_string << s1[j] << " " << s2[i] << " " << pen << "**\n**";  i++;  j++;  } **else** **if** (\_M[i][j] == opt2) { // down  return\_string << "- " << s2[i] << " 2**\n**";  i++;  } **else** **if** (\_M[i][j] == opt3) { // Move right  return\_string << s1[j] << " -" << " 2**\n**";  j++;  }  }  retS = return\_string.str();  **return** retS;  } |

Main.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include "EDistance.h"  **int** **main**(**int** argc, **const** **char**\* argv[]) {  sf::Clock clock;  sf::Time t;  std::string s1, s2;  std::cin >> s1 >> s2;  ED test(s1, s2);  std::cout << "Edit Distance = " << test.OptDistance() << std::endl;  std::cout << test.Alignment();  t = clock.getElapsedTime();  std::cout << "It took you this much time: " << t.asSeconds() << "seconds" << std::endl;  } |

**PS6 Random Writer**

The Assignment

In this assignment, I created a class that implements Markov chains to model English text. Markov chains are statistical model of text, which count the occurrences and sequences of characters in an English word / sentence. The RandWriter class uses several methods, such as freq() which returns the frequency of a character in a k-gram (fixed number of text), and randk() which returns a random character that follows a given k-gram. Using this RandWriter class allows us to generate pseudo-random text through the use of statistics – and specifically using the gen() method, which returns a string of random characters following a given k-gram.

Key Concepts

The key idea behind this assignment is to understand and implement Markov chains, which are statistical models of English text. To build this Markov Model class, I used a C++ map. the k-gram is the key to the map, and the int is the value, or number of occurrences of the given k-gram. We had several methods that were used to model the Markov chains. Freq() – two versions of this method, one for finding the frequency of k-grams and another for finding the frequency of a character following the given k-gram. Randk() – generates a random character that follows the given k-gram Gen() – generates a string of length T characters which simulates a trajectory through the given Markov chain. These methods make up the Text Writer in outputting the pseudo-random text. I also had a test file to check that the source code was using the correct frequencies on characters.

What I learned

I found out that many text prediction programs use Markov chains, which was interesting. Like on your phone or even when you’re typing an email, most use Markov chains to make a random yet educated guess to guess on what word you are trying to type. This is done by using the frequencies of characters the same way we did in this assignment.

**Screenshots**

**A screenshot of a computer

Description automatically generated**

**Source Code for PS6 Random Writer**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system  -lboost\_unit\_test\_framework -lboost\_regex -lboost\_date\_time  DEPS = RandWriter.h TextWriter.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -I/usr/local/Cellar/boost/**1.78.0**\_1/include  -I/usr/local/Cellar/sfml/**2.5.1**\_1/include -c $<  **all:** TextWriter  **TextWriter:** RandWriter.o TextWriter.o  $(CC) $(CFLAGS) -L/usr/local/Cellar/boost/**1.78.0**\_1/include  -L/usr/local/Cellar/sfml/**2.5.1**\_1/include -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f TextWriter |

RandWriter.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  #include <string>  #include <vector>  #include <map>  #include <algorithm>  #include <stdexcept>  #include <cstdlib>  #include <ctime>  #include <string>  #include <vector>  #include <map>  **class** **RandWriter** {  **public:**  RandWriter(std::string text, **int** k);  **int** order() **const**;  **int** **freq**(std::string kgram);  **int** **freq**(std::string kgram, **char** c);  **char** **randk**(std::string kgram);  std::string generate(std::string kgram, **int** T);  **friend** std::ostream& **operator**<<(std::ostream& os, RandWriter& mm);  **private:**  std::map <std::string, **int**> kgrams;  **int** inLine;  std::string word;  std::string init;  }; |

RandWriter.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127 | #include <iostream>  #include <string>  #include <vector>  #include <map>  #include <stdexcept>  #include <cstdlib>  #include <ctime>  #include "RandWriter.h"  RandWriter::RandWriter(std::string text, **int** k) {  srand(time(NULL));  inLine = k;  init = text;  **for** (**unsigned** i = **0**; i < text.size(); i++)  **if** (std::string::npos == word.find(text[i]))  word.push\_back(text[i]);  **for** (**unsigned** i = **0**; i < text.size(); i++) {  std::string temp;  std::string temp2;  **for** (**unsigned** j = i; j < i + k; j++)  **if** (j >= text.size())  temp.push\_back(text[j - text.size()]);  **else**  temp.push\_back(text[j]);  **if** (kgrams.end() == kgrams.find(temp))  kgrams[temp] = **1**;  **else**  kgrams[temp] += **1**;  **for** (**unsigned** j = **0**; j < word.size(); j++)  **if** (kgrams.end() == kgrams.find(temp + word[j]))  kgrams[temp + word[j]] = **0**;  **for** (**unsigned** j = i; j < i + k + **1**; j++)  **if** (j >= text.size())  temp2.push\_back(text[j - text.size()]);  **else**  temp2.push\_back(text[j]);  kgrams[temp2] += **1**;  }  }  **int** RandWriter::order() **const** {  **return** inLine;  }  **int** RandWriter::freq(std::string kgram) {  **if** (kgram.size() != (**unsigned**)inLine)  **throw** std::runtime\_error("kgram is not size k");  **if** (inLine == **0**)  **return** init.size();  **else**  **return** kgrams[kgram];  }  **int** RandWriter::freq(std::string kgram, **char** c) {  **if** (kgram.size() != (**unsigned**)inLine)  **throw** std::runtime\_error("kgram is not size k");  **if** (inLine == **0**) {  **int** count = **0**;  **for** (**unsigned** i = **0**; i < init.size(); i++)  **if** (init[i] == c)  count++;  **return** count;  } **else** {  **return** kgrams[kgram + c];  }  **return** **0**;  }  **char** RandWriter::randk(std::string kgram) {  **if** (kgram.length() != (**unsigned**)inLine) {  **throw** std::runtime\_error("Error - kgram not of  length **k** (randk)");  }  std::map<std::string, **int**>::iterator it;  it = kgrams.find(kgram);  **if** (it == kgrams.end()) {  **throw** std::runtime\_error("Error - Could not find  the given kgram! (randk)");  }  **int** kgram\_freq = freq(kgram);  **int** random\_value = rand() % kgram\_freq; //NOLINT  **double** test\_freq = **0**;  **double** random\_num = **static\_cast**<**double**>(random\_value) / kgram\_freq;  **double** last\_values = **0**;  **for** (**unsigned** **int** a = **0**; a < word.length(); a++) {  test\_freq = **static\_cast**<**double**>(freq(kgram, word[a])) / kgram\_freq;  **if** (random\_num < test\_freq + last\_values && test\_freq != **0**) {  **return** word[a];  }  last\_values += test\_freq;  }  **return** '-';  }  std::string RandWriter::generate(std::string kgram, **int** L) {  std::string s1 = kgram;  std::string s2 = kgram;  **char** rc;  **for** (**int** i = **0**; i < L - inLine; i++) {  rc = randk(s1);  s2.push\_back(rc);  s1.erase(s1.begin());  s1.push\_back(rc);  }  **return** s2;  }  std::ostream& **operator**<<(std::ostream& os, RandWriter& mm) {  std::map <std::string, **int**> temp = mm.kgrams;  // cppcheck-suppress StlMissingComparison  **for** (std::map<std::string, **int**>::iterator it = temp.begin();  it != temp.end(); ++it) {  os << it->first << " " << it->second << " => ";  **for** (**unsigned** i = **0**; i < mm.word.size(); i++) {  it++;  os << it->first << " " << it->second << " ";  }  os << std::endl;  }  **return** os;  } |

TextWriter.cpp (Main)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | #include <iostream>  #include <string>  #include <sstream>  #include <cstdlib>  #include "RandWriter.h"  **int** **main**(**int** argc, **const** **char**\* argv[]) {  **if** (argc != **3**) {  std::cout << "To Use ./TextWriter (K) (L) // L ≥ k**\n**";  **return** **0**;  }  std::string str\_k(argv[**1**]);  std::string **str\_t**(argv[**2**]);  **int** k = std::stoi(str\_k);  **int** t = std::stoi(**str\_t**);  std::string input = "NULL";  std::string current\_txt = "NULL";  **while** (std::cin >> current\_txt) {  input += " " + current\_txt;  current\_txt = "";  }  std::cout << "Original Text:**\n\n**";  **for** (**int** a = **0**; a < t; a++) {  std::cout << input[a];  **if** (input[a] == '.' || input[a] == '!') {  std::cout << "**\n**";  }  }  std::string output\_string = "";  RandWriter whatWord(input, k);  output\_string += "" + whatWord.generate(input.substr(**0**, k), t);  std::cout << "**\n\n**After Code:**\n\n**";  **for** (**int** a = **0**; a < t; a++) {  std::cout << output\_string[a];  **if** (output\_string[a] == '.' || output\_string[a] == '!') {  std::cout << "**\n**";  }  }  std::cout << "**\n**";  **return** **0**;  } |

test.cpp (test cases)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | #include "RandWriter.h"  #define BOOST\_TEST\_DYN\_LINK  #define BOOST\_TEST\_MODULE Main  #include <boost/test/unit\_test.hpp>  // testing constructor  BOOST\_AUTO\_TEST\_CASE(construct) {  BOOST\_REQUIRE\_NO\_THROW(RandWriter("gagggagaggcgagaaa", **2**));  }  // testing orderK  BOOST\_AUTO\_TEST\_CASE(orderK) {  RandWriter model0("gagggagaggcgagaaa", **0**);  RandWriter **model1**("gagggagaggcgagaaa", **1**);  RandWriter **model2**("gagggagaggcgagaaa", **2**);  BOOST\_REQUIRE(model0.order() == **0**);  BOOST\_REQUIRE(model1.order() == **1**);  BOOST\_REQUIRE(model2.order() == **2**);  }  // testing single freq  BOOST\_AUTO\_TEST\_CASE(Sfreq) {  RandWriter model("gagggagaggcgagaaa", **1**);  BOOST\_REQUIRE\_THROW(model.freq(""), std::runtime\_error);  BOOST\_REQUIRE\_THROW(model.freq("xx"), std::runtime\_error);  BOOST\_REQUIRE(model.freq("a") == **7**);  BOOST\_REQUIRE(model.freq("g") == **9**);  BOOST\_REQUIRE(model.freq("c") == **1**);  }  // testing double freq  BOOST\_AUTO\_TEST\_CASE(Dfreq) {  RandWriter model("gagggagaggcgagaaa", **1**);  BOOST\_REQUIRE(model.freq("a", 'a') == **2**);  BOOST\_REQUIRE(model.freq("a", 'c') == **0**);  BOOST\_REQUIRE(model.freq("a", 'g') == **5**);  BOOST\_REQUIRE(model.freq("c", 'a') == **0**);  BOOST\_REQUIRE(model.freq("c", 'c') == **0**);  BOOST\_REQUIRE(model.freq("c", 'g') == **1**);  BOOST\_REQUIRE(model.freq("g", 'a') == **5**);  BOOST\_REQUIRE(model.freq("g", 'c') == **1**);  BOOST\_REQUIRE(model.freq("g", 'g') == **3**);  }  // testing kRand  BOOST\_AUTO\_TEST\_CASE(kRand) {  RandWriter model("gagggagaggcgagaaa", **1**);  BOOST\_REQUIRE\_NO\_THROW(model.randk("a"));  BOOST\_REQUIRE\_NO\_THROW(model.randk("c"));  BOOST\_REQUIRE\_NO\_THROW(model.randk("g"));  BOOST\_REQUIRE\_THROW(model.randk("x"), std::runtime\_error);  BOOST\_REQUIRE\_THROW(model.randk("xx"), std::runtime\_error);  } |

**PS7 Kronos Time Parsing**

The Assignment

In this assignment, we analyze the Kronos InTouch time clock log by using regular expressions to parse the file and verify device boot up timing. Scan the complete log file and create a text file report chronologically describing each time the device was restarted. To parse the file, we used Boost’s regular expression library (regex), along with Boost’s date/time libraries. This allowed us to create an output file that verifies whether a time clock successfully booted or failed to boot.

Key Concepts

The main idea behind this assignment is to regular expressions, specifically using Boost’s regex library to match strings. The regex library was easy to use. Using boost::regex\_match or boost::regex\_search did the trick to find matches against the regular expressions.

To look for boots and date/time statements I used :

std::string s1 = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";  // NOLINT

     s1 +=   "([0-9]{2}):([0-9]{2}):([0-9]{2}): ";

     s1 +=   "\\(log.c.166\\) server started";

    boost::regex beginBoot(s1);

std::string s2 = "([0-9]{4})-([0-9]{2})-([0-9]{2}) ";  // NOLINT

     s2 +=   "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):";

     s2 +=   "INFO:";

     s2 +=   "oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080";

    boost::regex endBoot(s2);

What I learned

I think parsing is helpful, I also think what I learned from this assignment is going to play a part in a job I’m starting soon, in which I look at logs and scripts all the time. I learned a ton about parsing and its very practical. The date and time aspect of this assignment was hard. Date and time calculations were my hardest challenge. Creating an object of p-time for the data helped. And then turning all this data into strings to output into another file. Using stoi helped me with this bump in the road a ton. But I learned a lot about working dates and times, I will definitely be ready to do more in the future.

**Screenshots**

Text

Description automatically generated

**Output File**

A screenshot of a computer screen

Description automatically generated

**Source Code for PS7 Kronos Time Parsing**

Makefile

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | CC = g++  CFLAGS = -Wall -Werror -pedantic --std=c++**17**  LIBS = -lsfml-graphics -lsfml-window -lsfml-system  -lboost\_regex -lboost\_date\_time  DEPS = ps7.cpp  %.o: %.cpp $(DEPS)  $(CC) $(CFLAGS) -c $<  **all:** ps7  **ps7:** ps7.o  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)  **clean:**  rm -f \*.o  rm -f Test |

Ps7.cpp (Main)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | #include <iostream>  #include <fstream>  #include <string>  #include <boost/regex.hpp>  #include "boost/date\_time/gregorian/gregorian.hpp"  #include "boost/date\_time/posix\_time/posix\_time.hpp"  **using** boost::gregorian::date;  **using** boost::gregorian::from\_simple\_string;  **using** boost::gregorian::date\_period;  **using** boost::gregorian::date\_duration;  **using** boost::posix\_time::ptime;  **using** boost::posix\_time::time\_duration;  **int** **main**(**int** argc, **char**\* argv[]) {  std::string fileName = argv[**1**]; // NOLINT  std::ifstream lFile(fileName.c\_str());  std::ofstream reportFile(fileName + ".rpt", std::ostream::out);  date d;  ptime startT;  std::string line; // NOLINT  **bool** boot = false;  **int** ln = **1**;  **int** timeElapsed = **0**;  std::string s1 = "([0-9]{4})-([0-9]{2})-([0-9]{2}) "; // NOLINT  s1 += "([0-9]{2}):([0-9]{2}):([0-9]{2}): ";  s1 += "**\\**(log.c.166**\\**) server started";  boost::regex beginBoot(s1);  std::string s2 = "([0-9]{4})-([0-9]{2})-([0-9]{2}) "; // NOLINT  s2 += "([0-9]{2}):([0-9]{2}):([0-9]{2}).([0-9]{3}):";  s2 += "INFO:";  s2 += "oejs.AbstractConnector:Started  SelectChannelConnector@**0.0.0.0**:**9080**";  boost::regex endBoot(s2);  **if** (lFile.is\_open()) {  std::cout << "FiLe SuCcEsSfUlLy FoUnD" << std::endl;  **while** (std::getline(lFile, line)) {  boost::smatch sm;  **if** (regex\_search(line, sm, beginBoot)) {  date tempStartDate(stoi(sm[**1**]), stoi(sm[**2**]), stoi(sm[**3**]));  d = tempStartDate;  ptime **tempStartTime**(d, time\_duration(stoi(sm[**4**]), stoi(sm[**5**]),  stoi(sm[**6**])));  startT = tempStartTime;  **if** (boot) {  reportFile << " Incomplete boot **\n**" << std::endl;  boot = false;  }  reportFile << "==== Device boot ====**\n**"  << ln << "(" << fileName << "):"  << sm[**1**] << "-" << sm[**2**] << "-" << sm[**3**]  << " " << sm[**4**] << ":" << sm[**5**] << ":" << sm[**6**]  << " Boot Start" << std::endl;  boot = true;  }  **if** (regex\_match(line, sm, endBoot)) {  date tempEndDate(stoi(sm[**1**]), stoi(sm[**2**]), stoi(sm[**3**]));  ptime **endT**(tempEndDate, time\_duration(stoi(sm[**4**]),  stoi(sm[**5**]), stoi(sm[**6**])));  reportFile << ln << "(" << fileName << "):"  << sm[**1**] << "-" << sm[**2**] << "-" << sm[**3**]  << " " << sm[**4**] << ":" << sm[**5**] << ":" << sm[**6**]  << " Boot Completed" << std::endl;  time\_duration timeD = endT - startT;  timeElapsed = timeD.total\_milliseconds();  reportFile << "Boot Time: " << timeElapsed << "ms  \n" << std::endl;  boot = false;  }  ln++;  }  } **else** {  std::cout << "Error opening file" << std::endl;  }  reportFile.close();  lFile.close();  **return** **0**;  } |