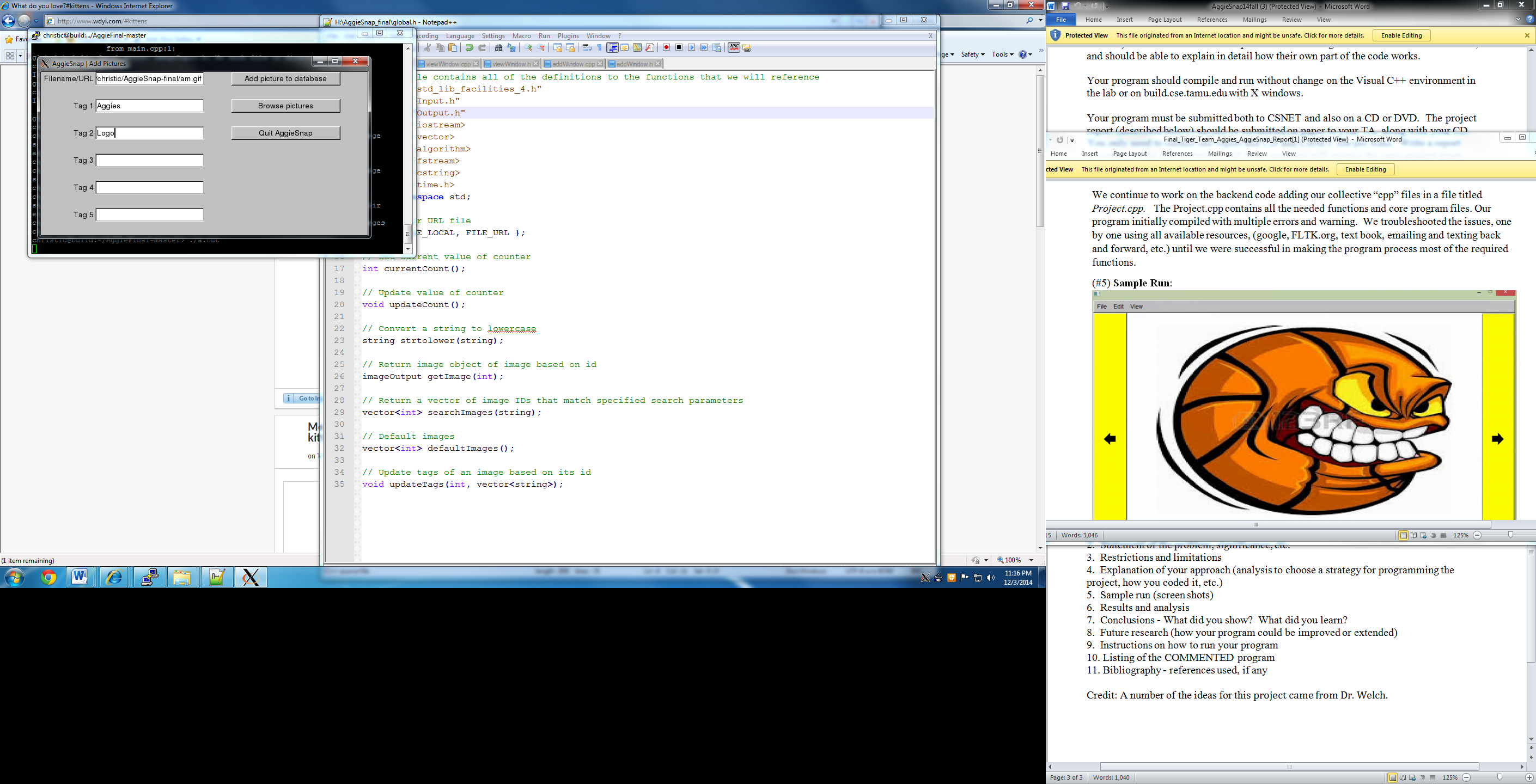
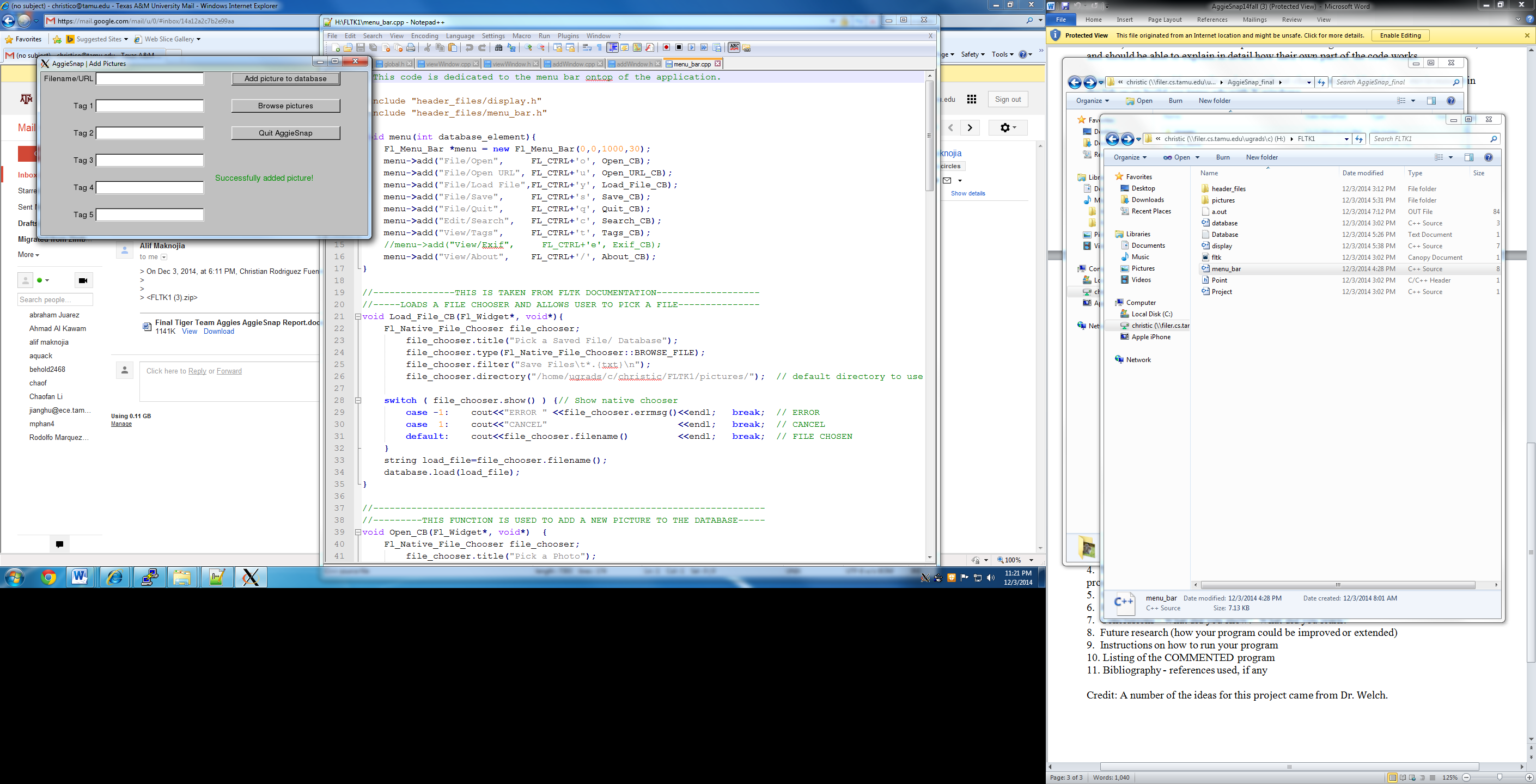
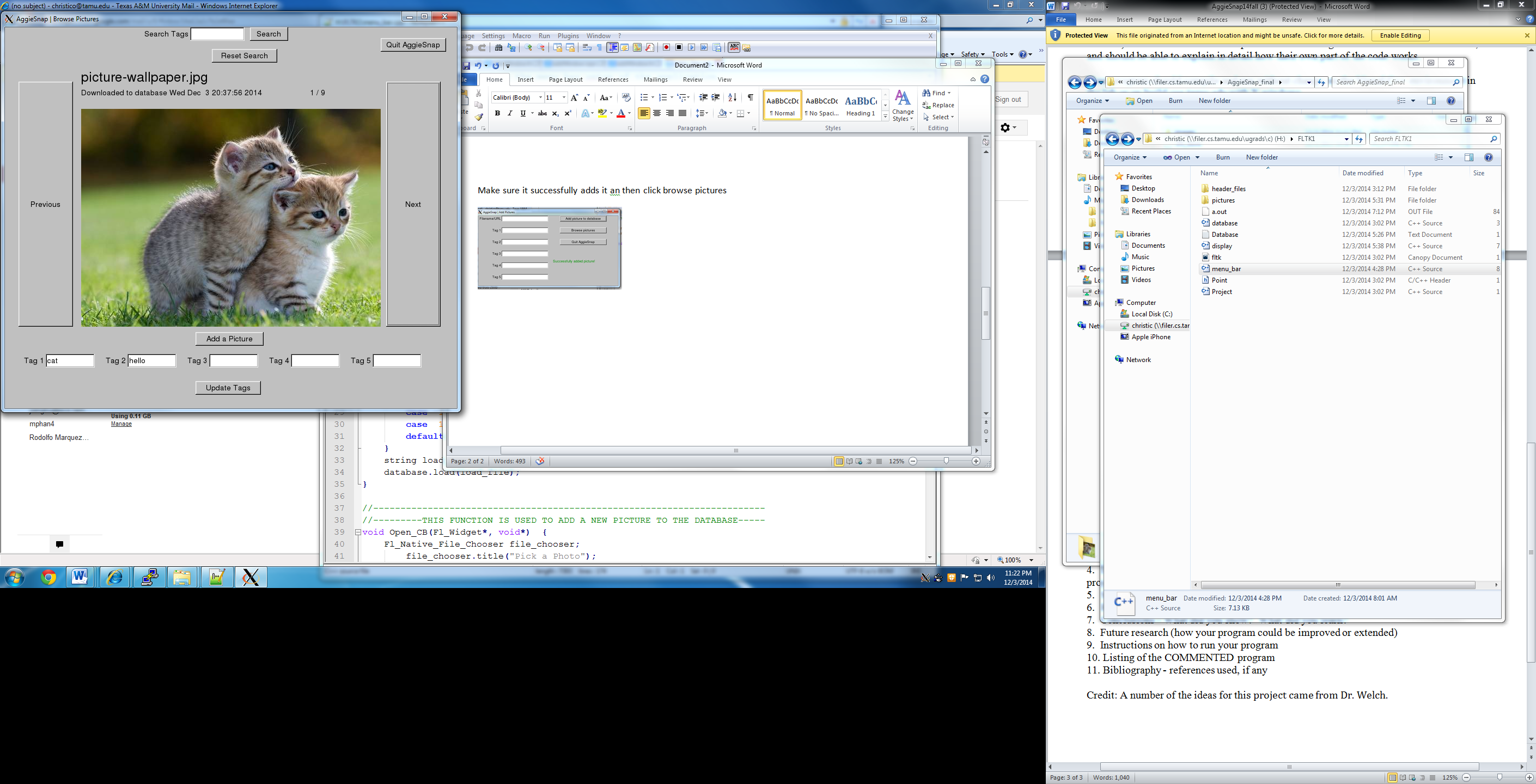
CSCE 121 section 501

1. The team members are Christian Rodriguez, Brandon Glover, and Alif Maknojia. Christian Rodriguez did parts 1 and 2 which are getting file from local file and then the other is getting file from URL. Alif Maknojia Did parts 3 and 4 which are adding tags and saving image to a disk. Finally Brandon Glover did parts 5 and 6 which are to browse all pictures, tags and providing the next and previous button. Each member did a fair share of work with an equal amount of work. We implanted extra credit as a team.
2. The project was to design and to write a C++11/FLTK picture database program with a graphical user interface. The program must be able to open pictures of specific file formats whether from a URL, cellphone camera, user’s hard drive etc., and save the pictures with tags to the program’s database. The program has to be able to retrieve the pictures for display, browse with any combinations of tags in a user friendly manner. Most digital devices nowadays have the ability to take pictures. This couple with social media makes this project significant. As must users store pictures in the cloud, mobile devices or on their computers so our knowledge gained from this project is useful and realistic.
3. In this project we felt like we had little to no limitations. We had to change the FLTK library files, GUI interface a little to change and make our program to succeed.
4. We originally approach this project by dividing up the project tasks into models by project team member, once each individual completed their task, we combined the code; compile the code with hope for the desired end state. We quickly learned that approach put a higher burden on the team member who had to combine the completed individually task into one. That said, we adjusted and then collectively reviewed all the code focusing on it being built efficiently and reliably. We had drawings and sketches that made us visualize our future program. We first focused on display an image on FLTK and then we work our way up the parts. Using header files to display buttons and searches for the navigation. We then focused on making sure we can input tags into the images. After that we see if we could keep adding images and have a next and previous buttons. Then we cleaned up the display of the images and buttons for it to look nice and neat. After that we wanted to have it search for only one tag and what pictures it contain and that was tricky and took as a while to figure out on FLTK. After all of that we were able to tighten up some things and just fix some alignments.
5. First we get image from local file or url, and add picture to database.

Make sure it successfully adds it an then click browse pictures



Then we get to this window where you see the image displaying and can update tags at the bottom and click next or previous for other pictures. Also if you search at the top cat I will display all the pictures within the tag. To add another picture click add a picture below the cat picture and it will send you back to the previous window and then you can add other pictures.

1. Results: Throughout the entire process we design our program to be easy to use and to work well with the person using it. Everyone displays as expected and should be able to do every task given to us.
2. Working in a group and combining individual tasks to achieve a common purpose is challenging. This approach is very corporate like and provided us an awesome learning experience. This project also afforded us the opportunity to enhance our C++ and FLTK knowledge.
3. Future research: The only thing that our team could think of to improve our program is to have more color to the buttons and make everything look nicer than we have. We could probably add more features but we were not required so we did not, so we could improve in that sense.
4. To run our program first you go on putty and make sure you select the file then to compile the program we use g++4.7 –std=c++11 \*.cpp –lfltk –lfltk\_images to compile all cpp files, and to run it we make sure to have xming running and type ./a.out. To add a picture to be displayed we used the Filename/URL to add the picture. For local file we locate the picture and type the where is located for example /homes/ugrads/c/christic/AggieSnap/am.gif to get a fig local file, and add the corresponding tags you want for the picture and click add picture to database. For URL we just copy and paste the link into the box and type the tags and then add the picture to database. After we added the file we go to browse to go to the display images. We have a next and previous button to go to different pictures. To add tags to the picture we can update them by typing them at the bottom and clicking update tags. To search for a specific tags and what pictures it contains we go to the top of the window and type the tag wanted and click search and all the pictures of that tag will display. To reset the aggiesnap program we can click the reset search. And finally to quit the program top right button to quit the program.

**Global.cpp**

// Header and function definitions

#include "global.h"

// Get current value of counter

int currentCount() {

string line;

int count = 0;

ifstream current("counter");

if (current.is\_open()) {

getline(current, line);

count = stoi(line);

current.close();

}

return count;

}

// Update value of counter

void updateCount() {

// Get current value of counter

int count = currentCount();

// Update value of counter

++count;

// Write to file

ofstream update("counter");

if (update.is\_open()) {

update << count << endl;

update.close();

}

}

// Convert a string to lowercase

string strtolower(string str) {

for (auto i = 0; i < str.length(); ++i) {

str[i] = tolower(str[i]);

}

return str;

}

// Return image object of image based on id

imageOutput getImage(int id) {

string line;

ifstream db("db");

vector<string> data;

bool found = false;

imageOutput image;

// If file opened successfully

if (db) {

// Get each line of file

while (getline(db, line) && found == false) {

int offset = 0, pos = 0;

// While it keeps finding delimeters

while (line.find("^", offset) != string::npos) {

pos = line.find("^", offset);

// Check to see if id matches id being searched for. If not, break out of current search and look in next string

if (line.substr(offset, pos-offset) != to\_string(id) && found == false) {

break;

}

// Push data between delimiters back into vector

data.push\_back(line.substr(offset, pos-offset));

offset = pos+1;

found = true;

}

}

}

// If it found a match, set all the values of the image object else return a null image object

if (found) {

image.setAttributes(stoi(data[0]), data[1], data[2], stoi(data[3]), stoi(data[4]), vector<string> {data[5], data[6], data[7], data[8], data[9]});

} else {

image.setAttributes(0, "", "", 0, 0, vector<string> {"", "", "", "", ""});

}

return image;

}

// Return a vector of image IDs that match specified search parameters

vector<int> searchImages(string text) {

// Init empty results vector;

vector<int> results;

// Don't allow empty tag searches

if (text == "") {

return results;

}

// Get all of the tags

vector<string> tags;

int offset = 0, pos = 0;

// While it keeps finding delimeters

while (text.find(",", offset) != string::npos) {

pos = text.find(",", offset);

tags.push\_back(text.substr(offset, pos-offset));

offset = pos+1;

}

tags.push\_back(text.substr(offset, text.size()-offset));

// Cycle through all of the images

for (auto i = 1; i < currentCount(); ++i) {

imageOutput check = getImage(i);

// Cycle through all of the tags

for (auto j = 0; j < tags.size(); ++j) {

// If any of the tags match, push back the image id

if (check.getTag(1) == tags[j] || check.getTag(2) == tags[j] || check.getTag(3) == tags[j] || check.getTag(4) == tags[j] || check.getTag(5) == tags[j]) {

results.push\_back(i);

}

}

}

// Remove duplicates if image contains two tags that were searched for

vector<int>::iterator it;

it = unique(results.begin(), results.end());

results.resize(distance(results.begin(), it));

return results;

}

// Default images to cycle through (all images)

vector<int> defaultImages() {

vector<int> results;

for (int a = 1; a < currentCount(); ++a) {

results.push\_back(a);

}

return results;

}

// Update the tags of specified image

void updateTags(int id, vector<string> tags) {

stringstream data;

for (auto i = 1; i < currentCount(); ++i) {

imageOutput tmp = getImage(i);

if (i == id) {

// Update tags

data << tmp.getID() << "^" << tmp.getName() << "^" << tmp.getExtension() << "^" << tmp.getDate() << "^" << tmp.getType() << "^" << tags[0] << "^" << tags[1] << "^" << tags[2] << "^" << tags[3] << "^" << tags[4] << "^\n";

} else {

// Reinput same data

// 1^siberian\_husky^jpg^1386016286^0^husky^sdaf^^^^

data << tmp.getID() << "^" << tmp.getName() << "^" << tmp.getExtension() << "^" << tmp.getDate() << "^" << tmp.getType() << "^" << tmp.getTag(1) << "^" << tmp.getTag(2) << "^" << tmp.getTag(3) << "^" << tmp.getTag(4) << "^" << tmp.getTag(5) << "^\n";

}

}

ofstream db("db");

if (db.is\_open()) {

db << data.rdbuf();

}

}

Global.h

// This file contains all of the definitions to the functions that we will reference

#include "std\_lib\_facilities\_4.h"

#include "Input.h"

#include "Output.h"

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

#include <cstring>

#include <time.h>

using namespace std;

// Local or URL file

enum { FILE\_LOCAL, FILE\_URL };

// Get current value of counter

int currentCount();

// Update value of counter

void updateCount();

// Convert a string to lowercase

string strtolower(string);

// Return image object of image based on id

imageOutput getImage(int);

// Return a vector of image IDs that match specified search parameters

vector<int> searchImages(string);

// Default images

vector<int> defaultImages();

// Update tags of an image based on its id

void updateTags(int, vector<string>);

viewWindow.cpp

#include "viewWindow.h"

// Buttons

void viewWindow::cb\_next(Address, Address pw) {

reference\_to<viewWindow>(pw).next();

}

void viewWindow::cb\_previous(Address, Address pw) {

reference\_to<viewWindow>(pw).previous();

}

void viewWindow::cb\_new\_pic(Address,Address pw) {

reference\_to<viewWindow>(pw).new\_pic();

}

void viewWindow::cb\_tag\_search(Address,Address pw) {

reference\_to<viewWindow>(pw).tag\_search();

}

void viewWindow::cb\_reset\_tag\_search(Address,Address pw) {

reference\_to<viewWindow>(pw).reset\_tag\_search();

}

void viewWindow::cb\_quit(Address,Address pw) {

reference\_to<viewWindow>(pw).quit();

}

void viewWindow::cb\_update\_tags(Address,Address pw) {

reference\_to<viewWindow>(pw).update\_tags();

}

// Next button

void viewWindow::next() {

// If there are pictures to cycle through

if (cycleVals.size() != 0) {

// Increase picture index by one

++index;

// Start from beginning

if (index>cycleVals.size()-1) {

index = 0;

}

// Redraw update image and image info

redrawUpdate();

next\_button\_pushed = true;

}

}

// Previous button

void viewWindow::previous() {

// If there are pictures to cycle through

if (cycleVals.size() != 0) {

// Decrease picture index by one

--index;

// If out of range, start from end of cycleVals

if (index == -1) {

index = cycleVals.size()-1;

}

// Redraw update image and image info

redrawUpdate();

previous\_button\_pushed = true;

}

}

// Add pictures

void viewWindow::new\_pic() {

// Go to add picture screen

hide();

new\_button\_pushed = true;

}

// Filter pictures by tag

void viewWindow::tag\_search() {

tags = strtolower(search.get\_string());

cycleVals = searchImages(tags);

index = 0;

// If no matches found, display no matches found image or else display first image of matches

if (cycleVals.size() == 0) {

// Detach previous image, name, data, and saved\_tags

detach(\*img);

detach(\*name);

detach(\*data);

detach(\*changes\_saved);

img = new Image(Point(300,275), "nomatches.jpg");

name = new Text(Point(140,100), "");

name->set\_font\_size(25);

data = new Text(Point(140,125), "");

tag1.put("");

tag2.put("");

tag3.put("");

tag4.put("");

tag5.put("");

// Attach image, name, and data

attach(\*img);

attach(\*name);

attach(\*data);

} else {

// Redraw update image and image info

redrawUpdate();

}

search\_button\_pushed = true;

// Redraw GUI

Fl::redraw();

}

// Reset tag search input and cycleVals

void viewWindow::reset\_tag\_search() {

// Clear search box

search.put("");

// Cycle through default images

cycleVals = defaultImages();

// If user has images in DB and hasn't pushed previous or next yet, display the 1st image in the database

if (currentCount() != 1) {

// Redraw update image and image info

redrawUpdate();

} else {

// Detach previous image, name, data, and saved\_tags

detach(\*img);

detach(\*name);

detach(\*data);

detach(\*changes\_saved);

// Set null/default values of image, name, data, and tags

img = new Image(Point(300,275), "default.jpg");

name = new Text(Point(140,100), "");

name->set\_font\_size(25);

data = new Text(Point(140,125), "");

tag1.put("");

tag2.put("");

tag3.put("");

tag4.put("");

tag5.put("");

// Attach image, name, and data

attach(\*img);

attach(\*name);

attach(\*data);

}

search\_button\_pushed = false;

// Redraw GUI

Fl::redraw();

}

// Quit window

void viewWindow::quit() {

exit(EXIT\_SUCCESS);

}

// Update tags

void viewWindow::update\_tags() {

if (cycleVals.size() != 0) {

detach(\*changes\_saved);

// Update tags

updateTags(index+1, {tag1.get\_string(), tag2.get\_string(), tag3.get\_string(), tag4.get\_string(), tag5.get\_string()});

update\_tags\_button\_pushed = true;

changes\_saved = new Text(Point(335,640),"Changes Saved Successfully");

changes\_saved->set\_font\_size(15);

changes\_saved->set\_color(Color::dark\_green);

attach(\*changes\_saved);

// Redraw GUI

Fl::redraw();

}

}

// Listener boolean values

bool viewWindow::wait\_for\_button\_new\_pic() {

show();

new\_button\_pushed = false;

Fl::run();

return new\_button\_pushed;

}

bool viewWindow::wait\_for\_button\_next\_pic() {

next\_button\_pushed = false;

return next\_button\_pushed;

}

bool viewWindow::wait\_for\_button\_previous\_pic() {

previous\_button\_pushed = false;

return previous\_button\_pushed;

}

bool viewWindow::wait\_for\_reset\_search\_button() {

reset\_search\_button\_pushed = false;

return reset\_search\_button\_pushed;

}

bool viewWindow::wait\_for\_button\_update\_tags() {

update\_tags\_button\_pushed = false;

return update\_tags\_button\_pushed;

}

// Redraw and update text values when changing to different image view

void viewWindow::redrawUpdate() {

// Detach previous image, name, data, and saved\_tags

detach(\*img);

detach(\*name);

detach(\*data);

detach(\*changes\_saved);

// Get next image's info

imgObj = getImage(cycleVals[index]);

imgLoc = "images/" + to\_string(imgObj.getID()) + "." + imgObj.getExtension();

img = new Image(Point(140,150), imgLoc);

// Crop image

img->set\_mask(Point(0,0),550,400);

// Get image name

name = new Text(Point(140,100), imgObj.getShortName() + "." + imgObj.getExtension());

name->set\_font\_size(25);

// Get image date and show whether it was added or downloaded

time\_t date = imgObj.getDate();

if (imgObj.getType() == FILE\_LOCAL) {

datastring = "Added to database " + to\_string(asctime(gmtime(&date))) + "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t " + to\_string(index+1) + " / " + to\_string(cycleVals.size());

} else {

datastring = "Downloaded to database " + to\_string(asctime(gmtime(&date))) + "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t " + to\_string(index+1) + " / " + to\_string(cycleVals.size());

}

data = new Text(Point(140,125), datastring);

// Update tag information

tag1.put(to\_string(imgObj.getTag(1)));

tag2.put(to\_string(imgObj.getTag(2)));

tag3.put(to\_string(imgObj.getTag(3)));

tag4.put(to\_string(imgObj.getTag(4)));

tag5.put(to\_string(imgObj.getTag(5)));

// Attach image, name, and data

attach(\*img);

attach(\*name);

attach(\*data);

// Redraw GUI

Fl::redraw();

}

**viewWindow.h**

#ifndef VIEWWINDOW\_H\_

#define VIEWWINDOW\_H\_

#include "window.h"

#include "graph.h"

#include "gui.h"

#include "global.h"

using namespace Graph\_lib;

struct viewWindow : Graph\_lib::Window {

public:

viewWindow() :

Window(Point(0,0),830,700,"AggieSnap | Browse Pictures"),

new\_button\_pushed(false),

next\_button\_pushed(false),

previous\_button\_pushed(false),

search\_button\_pushed(false),

reset\_search\_button\_pushed(false),

update\_tags\_button\_pushed(false),

cycleVals(defaultImages()),

imgObj(),

imgLoc(),

//outbox

tag1(Point(75, 600), 90, 25, "Tag 1"),

tag2(Point(225, 600), 90, 25, "Tag 2"),

tag3(Point(375, 600), 90, 25, "Tag 3"),

tag4(Point(525, 600), 90, 25, "Tag 4"),

tag5(Point(675, 600), 90, 25, "Tag 5"),

//inbox

search(Point(340,0), 100, 25, "Search Tags"),

// Buttons

button\_next(Point(700, 100), 100, 450, "Next", cb\_next),

button\_previous(Point(25, 100), 100, 450, "Previous", cb\_previous),

button\_new\_pic(Point(350, 560), 125, 25, "Add a Picture", cb\_new\_pic),

button\_reset\_search(Point(380,40),120,25,"Reset Search",cb\_reset\_tag\_search),

button\_tag\_search(Point(450, 0), 70, 25, "Search", cb\_tag\_search),

button\_quit(Point(690,20),120,25,"Quit AggieSnap",cb\_quit),

button\_update\_tags(Point(350,650),120,25,"Update Tags",cb\_update\_tags) {

// Attach buttons

attach(button\_next);

attach(button\_previous);

attach(button\_new\_pic);

attach(button\_tag\_search);

attach(button\_reset\_search);

attach(button\_quit);

attach(button\_update\_tags);

// Attach outboxes/inboxes

attach(tag1);

attach(tag2);

attach(tag3);

attach(tag4);

attach(tag5);

attach(search);

// If user has images in DB and hasn't pushed previous or next yet, display the 1st image in the database

if ((!next\_button\_pushed || !previous\_button\_pushed) && currentCount() != 1) {

imgObj = getImage(1);

imgLoc = "images/1." + imgObj.getExtension();

img = new Image(Point(140,150), imgLoc);

// Crop image

img->set\_mask(Point(0,0),550,400);

// Get image name

name = new Text(Point(140,100), imgObj.getShortName() + "." + imgObj.getExtension());

name->set\_font\_size(25);

// Get image date and show whether it was added or downloaded

time\_t date = imgObj.getDate();

if (imgObj.getType() == FILE\_LOCAL) {

datastring = "Added to database " + to\_string(asctime(gmtime(&date))) + "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t 1 / " + to\_string(cycleVals.size());

} else {

datastring = "Downloaded to database " + to\_string(asctime(gmtime(&date))) + "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t 1 / " + to\_string(cycleVals.size());

}

data = new Text(Point(140,125), datastring);

// Update tag information

tag1.put(to\_string(imgObj.getTag(1)));

tag2.put(to\_string(imgObj.getTag(2)));

tag3.put(to\_string(imgObj.getTag(3)));

tag4.put(to\_string(imgObj.getTag(4)));

tag5.put(to\_string(imgObj.getTag(5)));

// Attach image, name, and data

attach(\*img);

attach(\*name);

attach(\*data);

// If user has no images in DB

} else {

// Set null/default values of image, name, data, and tags

img = new Image(Point(300,275), "default.jpg");

name = new Text(Point(140,100), "");

name->set\_font\_size(25);

data = new Text(Point(140,125), "");

tag1.put("");

tag2.put("");

tag3.put("");

tag4.put("");

tag5.put("");

// Attach image, name, and data

attach(\*img);

attach(\*name);

attach(\*data);

}

}

bool wait\_for\_button\_new\_pic();

bool wait\_for\_button\_next\_pic();

bool wait\_for\_button\_previous\_pic();

bool wait\_for\_reset\_search\_button();

bool wait\_for\_button\_update\_tags();

int index = 0;

private:

// Call back functions for buttons

static void cb\_next(Address, Address);

static void cb\_previous(Address, Address);

static void cb\_new\_pic(Address, Address);

static void cb\_tag\_search(Address, Address);

static void cb\_reset\_tag\_search(Address, Address);

static void cb\_quit(Address, Address);

static void cb\_update\_tags(Address, Address);

// Respective functions for call backs

void next();

void previous();

void new\_pic();

void tag\_search();

void reset\_tag\_search();

void redrawUpdate();

void quit();

void update\_tags();

In\_box search;

In\_box tag1, tag2, tag3, tag4, tag5;

Button button\_next, button\_previous, button\_new\_pic, button\_tag\_search, button\_reset\_search, button\_quit, button\_update\_tags;

string tags;

imageOutput imgObj;

Image \*img;

string imgLoc, datastring;

Text \*name, \*data, \*changes\_saved;

vector<int> cycleVals;

bool new\_button\_pushed;

bool next\_button\_pushed;

bool previous\_button\_pushed;

bool search\_button\_pushed;

bool reset\_search\_button\_pushed;

bool update\_tags\_button\_pushed;

};

#endif

**addWindow.cpp**

#include "addWindow.h"

#include "Input.h"

// Button callback functions ////////

void addWindow::cb\_add\_pic(Address,Address pw)

{

reference\_to<addWindow>(pw).add\_pic() ;

}

void addWindow::cb\_view\_pic(Address,Address pw)

{

reference\_to<addWindow>(pw).view\_pic() ;

}

void addWindow::cb\_quit(Address,Address pw)

{

reference\_to<addWindow>(pw).quit() ;

}

/////////////////////////////////////

// Button callback actions ////////

// Add picture to db

void addWindow::add\_pic()

{

// Image object is created

imageInput image;

// Send image values to acceptInput function for processing and record any errors

error = image.acceptInput(Url.get\_string(), vector<string> {tag1.get\_string(), tag2.get\_string(), tag3.get\_string(), tag4.get\_string(), tag5.get\_string()});

// Display error codes

displayError(error);

}

// Display error message depending on code received

void addWindow::displayError(int errorNum) {

string msgContent;

// Detach previous error message

detach(\*msg);

if (errorNum == 0) {

msg = new Text(Point(320,200),"Successfully added picture!");

msg->set\_color(Color::dark\_green);

msg->set\_font\_size(15);

// Reset input fields

Url.put("");

tag1.put("");

tag2.put("");

tag3.put("");

tag4.put("");

tag5.put("");

} else {

if (errorNum == 1) {

msg = new Text(Point(320, 200), "Invalid image extension, please try again!");

} else if (errorNum == 2) {

msg = new Text(Point(320, 200), "Invalid Url, try again!");

} else if (errorNum == 3) {

msg = new Text(Point(320, 200), "Invalid Local image, try again!");

} else if (errorNum == 4) {

msg = new Text(Point(320, 200), "Unknown Error, please try again!");

}

msg->set\_color(Color::red);

msg->set\_font\_size(15);

}

// Attach and redraw

attach(\*msg);

Fl::redraw();

}

// Done adding pictures button

void addWindow::view\_pic() {

hide();

view\_button\_pushed = true;

}

// Wait until user hits browse pictures

bool addWindow::wait\_for\_button\_view\_pic()

{

show();

view\_button\_pushed = false;

Fl::run();

return view\_button\_pushed;

}

// Quit AggieSnap

void addWindow::quit() {

exit(EXIT\_SUCCESS);

}

**addWindow.h**

#ifndef ADDWINDOW\_H\_

#define ADDWINDOW\_H\_

#include "std\_lib\_facilities\_4.h"

#include "window.h"

#include "graph.h"

#include "gui.h"

using namespace Graph\_lib;

struct addWindow : Graph\_lib::Window {

public:

addWindow():

Window(Point(0,0),600,300,"AggieSnap | Add Pictures"),

view\_button\_pushed(false),

error(4),

Url(Point(100,0),200,25,"Filename/URL"),

tag1(Point(100,50),200,25,"Tag 1"),

tag2(Point(100,100),200,25,"Tag 2"),

tag3(Point(100,150),200,25,"Tag 3"),

tag4(Point(100,200),200,25,"Tag 4"),

tag5(Point(100,250),200,25,"Tag 5"),

button\_add\_pic(Point(350,0),200,25,"Add picture to database",cb\_add\_pic),

button\_view\_pic(Point(350,50),200,25,"Browse pictures",cb\_view\_pic),

button\_quit(Point(350,100),200,25,"Quit AggieSnap",cb\_quit)

{

// Attach buttons

attach(button\_add\_pic);

attach(button\_view\_pic);

attach(button\_quit);

// Attach in boxes

attach(Url);

attach(tag1);

attach(tag2);

attach(tag3);

attach(tag4);

attach(tag5);

// Default empty message

msg = new Text(Point(40,180),"");

attach(\*msg);

}

bool wait\_for\_button\_view\_pic();

private:

static void cb\_add\_pic(Address,Address);

static void cb\_view\_pic(Address,Address);

static void cb\_quit(Address,Address);

void displayError(int);

void add\_pic();

void view\_pic();

void quit();

In\_box Url,tag1,tag2,tag3,tag4,tag5;

Button button\_add\_pic,button\_view\_pic,button\_quit;

string output, first\_tag, second\_tag, third\_tag, fourth\_tag, fifth\_tag;

bool view\_button\_pushed;

Text \*msg;

int error;

};

#endif

**Input**

class imageInput {

int type = 0, id = 0;

string loc, name, extension;

vector<string> tag;

public:

int acceptInput(string, vector<string>);

void printInfo();

void addEntry();

// Get functions

string getLoc() { return loc; }

string getName() { return name; }

string getExtension() { return extension; }

string getTag(int num) { return tag[(num-1)]; }

int getID() { return id; }

int getType() { return type; }

private:

bool validExtension(string);

void setAttributes(string);

int saveImage();

};

**Input.cpp**

// Header and function definitions

#include "global.h"

// Asks the user for an input file and accept it if extension is valid.

int imageInput::acceptInput(string loc, vector<string> tag) {

// No error so far

int error = 0;

// Copy location and tags from parameter to object's location

this->loc = loc;

this->tag = tag;

// Determines if file is a local file or a URL

if ((loc.substr(0, 7) == "http://" || loc.substr(0, 8) == "https://") || loc.substr(0, 4) == "www.") {

type = FILE\_URL;

} else {

// Local file

type = FILE\_LOCAL;

}

// Invalid extension

if (!validExtension(loc)) {

return 1;

}

// Set the value of the imageInput's name and extension

setAttributes(loc);

// Download / copy file from URL or local

error = saveImage();

// If there was an error while copying / downloading image

if (error != 0) {

return error;

}

// Input new entry into image database

addEntry();

return 0;

}

// Print image info

void imageInput::printInfo() {

cout << "------\nlocation:\t" << loc << "\nname:\t" << name << "\nextension:\t" << extension << "\ntype:\t" << type << "\n------" << endl;

}

// Add image to database

void imageInput::addEntry() {

// Update value of counter

ofstream db("db", ios::app);

if (db.is\_open()) {

// 1^siberian\_husky^jpg^1386016286^0^husky^sdaf^^^^

db << getID() << "^" << getName() << "^" << getExtension() << "^" << time(0)-(3600\*6) << "^" << getType() << "^" << strtolower(getTag(1)) << "^" << strtolower(getTag(2)) << "^" << strtolower(getTag(3)) << "^" << strtolower(getTag(4)) << "^" << strtolower(getTag(5)) << "^\n";

}

}

// Checks the file for a valid extension and also updates value of object's extension

bool imageInput::validExtension(string file) {

int pos = 1;

string section, ext;

vector <string>exts = {"jpg", "gif", "jpeg", "bm", "bmp", "pbm", "pgm", "png", "ppm", "xbm", "xpm"};

// Step back character by charcter till period is detected to get extension

while (1) {

try {

section = file.substr(file.length()-pos, pos);

// Record filename's extension

if (section.substr(0, 1) == ".") {

ext = section.substr(1, section.length() - 1);

break;

}

pos++;

} catch (exception &e) {

return 0;

}

}

// Check if valid extension, uppercase/lowercase

for (int i = 0; i < exts.size(); ++i) {

if (strtolower(ext) == exts[i]) {

return 1;

}

}

return 0;

}

// Sets the values of the imageInput's name and extension

void imageInput::setAttributes(string file) {

// Init. vars

int pos = 1;

string section, ext, name;

bool ext\_found = false;

// Step backwards one character at a time until reach the first period.

while (1) {

try {

section = file.substr(file.length()-pos, pos);

// Record filename's extension

if (section.substr(0, 1) == "." && ext\_found == false) {

ext = strtolower(section.substr(1, section.length() - 1));

// Set imageInput's extension

this->extension = ext;

// Update flag for extension found

ext\_found = true;

}

// Record filename

if (section.substr(0, 1) == "/" || pos == file.length()) {

if (getType() == FILE\_URL) {

name = section.substr(1, section.length()-ext.length()-2);

} else {

name = section.substr(0, section.length()-ext.length()-1);

}

// Remove 1st character if slash

if (name[0] == '/') {

name = name.substr(1, name.length()-1);

}

// Set imageInput's name

this->name = name;

break;

}

// Increment position if period not found

pos++;

} catch (exception &e) {

cerr << e.what() << endl;

exit (EXIT\_FAILURE);

}

}

// Set image ID

id = currentCount();

}

// Save the image

int imageInput::saveImage() {

string command;

int error = 0;

// If imageInput is a URL, download from internet

if (getType() == FILE\_URL) {

// Bypass 403 forbidden and misc. by faking self as browser.

command = "wget -U 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.6) Gecko/20070802 SeaMonkey/1.1.4' --no-check-certificate " + getLoc() + " -O images/" + to\_string(getID()) + "." + getExtension() + " >/dev/null 2>&1";

} else {

// Or else, copy local file

command = "cp " + getLoc() + " images/" + to\_string(getID()) + "." + getExtension() + " >/dev/null 2>&1";

}

// Execute command

error = system(command.c\_str());

// Error downloading from URL

if (error != 0 && getType() == FILE\_URL) {

// Remove file that is downloaded since it is invalid

command = "rm images/" + to\_string(getID()) + "." + getExtension();

system(command.c\_str());

return 2;

}

// Error copying local file

if (error != 0 && getType() == FILE\_LOCAL) {

return 3;

}

// Update counter

updateCount();

return 0;

}

**Main.cpp**

#include "viewWindow.h"

#include "addWindow.h"

#include "graph.h"

using namespace Graph\_lib;

int main() {

try {

int show\_window = 0; //0 = add pic, 1 = view pic

bool win\_next = false;

bool view\_win\_next = true;

// Infinite loop until exit via button or OS window close

while (win\_next || view\_win\_next) {

if (win\_next) {

show\_window = 1;

}

if (view\_win\_next) {

show\_window = 0;

}

win\_next = false;

view\_win\_next = false;

if (show\_window == 0) {

addWindow addWin;

addWin.show();

win\_next = addWin.wait\_for\_button\_view\_pic();

} else if (show\_window == 1) {

viewWindow viewWin;

viewWin.show();

view\_win\_next = viewWin.wait\_for\_button\_new\_pic();

}

}

return 0;

} catch (exception& e) {

cerr << "exception: " << e.what() << endl;

return 1;

} catch (...) {

cerr << "Some exception" << endl;

return 2;

}

}

**Output.cpp**

#include "global.h"

// Set all of the values of the imageOutput object

void imageOutput::setAttributes(int id, string name, string ext, int date, int type, vector<string> tag) {

this->id = id;

this->name = name;

this->extension = ext;

this->date = date;

this->type = type;

this->tag = tag;

}

// Print image info

void imageOutput::printInfo() {

cout << "------" << endl;

cout << "id:\t\t" << getID() << endl;

cout << "name:\t\t" << getName() << endl;

cout << "Extension:\t" << getExtension() << endl;

cout << "date:\t\t" << getDate() << endl;

cout << "tag 1:\t\t" << getTag(1) << endl;

cout << "tag 2:\t\t" << getTag(2) << endl;

cout << "tag 3:\t\t" << getTag(3) << endl;

cout << "tag 4:\t\t" << getTag(4) << endl;

cout << "tag 5:\t\t" << getTag(5) << endl;

cout << "------" << endl;

}

string imageOutput::getShortName() {

if (this->getName().length() > 35) {

return this->getName().substr(0, 35) + "...";

}

return this->getName();

}

**Output.h**

class imageOutput {

int type = 0, id = 0, date = 0;

string name, extension;

vector<string> tag;

public:

void setAttributes(int, string, string, int, int, vector<string>);

void printInfo();

// Get functions

string getName() { return name; }

string getShortName();

string getExtension() { return extension; }

string getTag(int num) { return tag[(num-1)]; }

int getID() { return id; }

int getDate() { return date; }

int getType() { return type; }

};

1. Bibliography-

Bjarne Stroustrup- Programming:Principles and Practice Using C++