The Programming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.
   The goal of this assignment was to use linked lists, vectors, nodes and regex to extract and hold the name, email address and course of an html file of peer teachers and display them in an organized manner to the user in its entirety and by course number.

2. The description of data structures and algorithms used to solve the problem.

   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)
   $structNode\{$//Node Struct
   $stringname, address, course;$//Initialization of pointers to strings to hold name,address and course number.
   $Node()\{name = ""; address = ""; course = "";\}$//Constructor for strings of Node Class
   $\};$

   $std :: vector < std :: list < Node >> a;$//Initializes "a", a vector of a list of Nodes.
   $smatchname, address, course;$//Initializes three arrays for regex_search of name, address and course number.
   $std :: vector < string > namevec, addressvec, coursevec;$//Initializes vector of strings for name, address and course number.
   $std :: vector < int > linecounter;$//Initializes vector of integers to count the lines of the .html file.

   $std :: list < Node > r;$//Initializes new list of nodes.

   $Noden;$//Initializes new Node
   $n.name = namevec[i];$//Sets name object of Node to element at i in vector of names.
   $n.address = addressvec[i];$//Sets address object of Node to element at i in vector of addresses.
   $n.course = coursevec[j];$//Sets course object of Node to element at i in vector of objects.
   $r.push\_back(n);$//Places node in the last position of a list of nodes, "r".

   $regexpattern1("\backslash\backslash w + \backslash\backslash s\backslash\backslash w + (-)?\backslash\backslash w * (? =< /h3 >)");$//Initalizes a regex pattern for the names of peer teachers.
   $if(regex_search(line, name, pattern1))$//Uses regex_search function to return a boolean value corresponding to the regex pattern and holds the match in smatch "name".

   $std :: vector < std :: list < string >> s;$//Initializes a vector of a list of strings.
   $std :: list < string > list121, list111, list110, list221, list206, list313, list312, list315;$//Initializes 8 lists of strings.

   $s.push\_back(list110);$//Pushes back list of 110 peer teachers into vector s.

   $list < string > x = s.at(i);$//Initializes list of strings to vector "s" at i.

   $cout << "Name : " << x.back() << "\backslash n";$//Prints the value at the back of list x.
   $x.pop\_back();$//Deletes last element of list x.

(b) Write about the ADTs implementation in C++.
$structNode\{$//Initialization of new Struct with name Node
$stringname, address, course;$//Allocates memory for string variables to hold name, address and course number info in Node struct.
$Node()\{name = ''''; address = ''''; course = ''''; \}$//Initializes the string variables to "NULL" equivalent using empty strings to prevent compilation error.
$\};$

$std :: vector < std :: list < Node >> a;$//Allocates memory for a STD vector of an STD list of Nodes in variable "a".
$smatchname, address, course;$//Allocates memeory for three arrays for regex_search with variables names name, address and course to hold corresponding information.
$std :: vector < string > namevec, addressvec, coursevec;$//Allocates memory for a STD vector of strings to hold the name, course and address info when regex_search returns a bool value of true.
$std :: vector < int > linecounter;$//Allocates memory for a STD vector of integers to count lines in between course numbers to tell if the peer teacher helps with more than one course.

$std :: list < Node > r;$//Allocates memory for a new list of nodes in variable "r".

$Noden;$//Allocates memory for a new Node, "n".
$n.name = namevec[i];$//The object in the name variable of the node struct is set to the element at "i" in the vector "namevec".
$n.address = addressvec[i];$//The object in the address variable of the node struct is set to the element at "i" in the vector "address".
$n.course = coursevec[j];$//The object in the name variable of the node struct is set to the element at "i" in the vector "namevec".
$r.push\_back(n);$//Uses STD vector push_back operation to place the node "n" at the end of list "r".

$regexpattern1(''\backslash\backslash w + \backslash\backslash s\backslash\backslash w + (-)?\backslash\backslash w * (? =< /h3 >)'');$//The regex pattern scans through the line of the html document to see if any characters follow the formula of letters, a single white space, letters, an optional hyphen, letters and if it ends in a </h3>.
$if(regex_search(line, name, pattern1))$//The STD regex_search function is used to scan through the line of html code and returns true if the pattern matches the regex pattern listed in the function parameters. If the bool value is true the the characters matching the pattern are stored in an smatch array "name".

$std :: vector < std :: list < string >> s;$//Allocates memory for a STD vector of an STD list of strings in a variable "s".
$std :: list < string > list121, list111, list110, list221, list206, list313, list312, list315;$//Allocates memory for an STD list of strings in the variables listed.

$s.push\_back(list110);$//Use of the STD vector push_back function to place a list of strings at the end of the vector of lists of strings, "s".

$list < string > x = s.at(i);$//Allocates memory for a list of strings with a variable "x" of the list of strings in the vector "s" at location "i". This line uses the STD vector function at() to extract the object at the location of "i" in the vector "s".

$cout << ''Name : '' << x.back() << ''\backslash n'';$//The cout function is used in pairing with the STD list function back() to display the last element of the list "x" on the console.
$x.pop\_back();$//The STD list function pop_back is used to deallocate the memory of the last element of the list "x", thus deleting the element.

(c) Describe algorithms used to solve the problem.
The algorithms in this assignment are centered around regex and STD vectors and lists.

Regex has a function called regex_search to find character literals that follow a given pattern within a string and place it in an array if matched.

STD vectors and lists are able to hold values of multiple types in a sequence with the ability to add or remove values from the sequence.

(d) Analyze the algorithms according to assignment requirements.
The regex functions are used to search for a specific pattern pertaining to the name, address or course number for each line within the html file, if found it places the pertaining value in a node.

The vectors were used to hold the data of the lists within each, and the lists held the data of the nodes within each element.

3. A C++ organization and implementation of the problem solution

   (a) Provide a list and description of classes or interfaces used by a program such as classes used to implement the data structures or exceptions.
   Struct Node: Holds three string values pertaining to the name, address and course of a peer teacher.
   STD vector: Standard vector used for holding Nodes, Lists, Strings, Integers and combinations of those listed.
   STD list: Standard vector used for holding Nodes and Strings.

   (b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).
   The file does not use any .h files rather than the standard template directory. A .h file was not needed since the Node struct was relatively simple, containing only three string variables.

   (c) Provide features of the C++ programming paradigms like Inheritance or Polymorphism in case of object oriented programming, or Templates in the case of generic programming used in your implementation.
   Templates-The program is heavily dependent upon templating. The STD list and vector are used in multiple occurances to hold different types of variables. For example, the first half of the assignment focused on creating a vector of lists of nodes to hold the information of each peer teacher according to their name and course number. The second focused on creating a vector of lists of strings to hold the names of the peer teachers for each course. Without templating each vector and list would have to be constructed and initialized with all types of variables that could be used individually.
   Polymorphism-Since vectors and lists are able to hold different forms of data, it is inferred that polymorphism is applicable to the code since the vector and list classes morph to the data type the vector or list is initialized with.

4. A user guide description how to navigate your program with the instructions how to:

   (a) compile the program: specify the directory and file names, etc.
   Begin by changing the directory to Kaiser-Alexander-A3 using the cd keyword in putty. When inside the directory, complile using g++ by typing into the console "g++ Kaiser-Alexander-A3Pt2.cpp".

   (b) run the program: specify the name of an executable file.
   After compilation, simply use ./a.out to run the program in the console.

5. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).
   The program requires that the Peer-TeacherHTMLONLY.html file is in the same directory as the .cpp file or else an error will be thrown. The program will output onto the console when run.

   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.
   The only input required is the .html file listed above and the user to compile and run the program. No other user input is required.

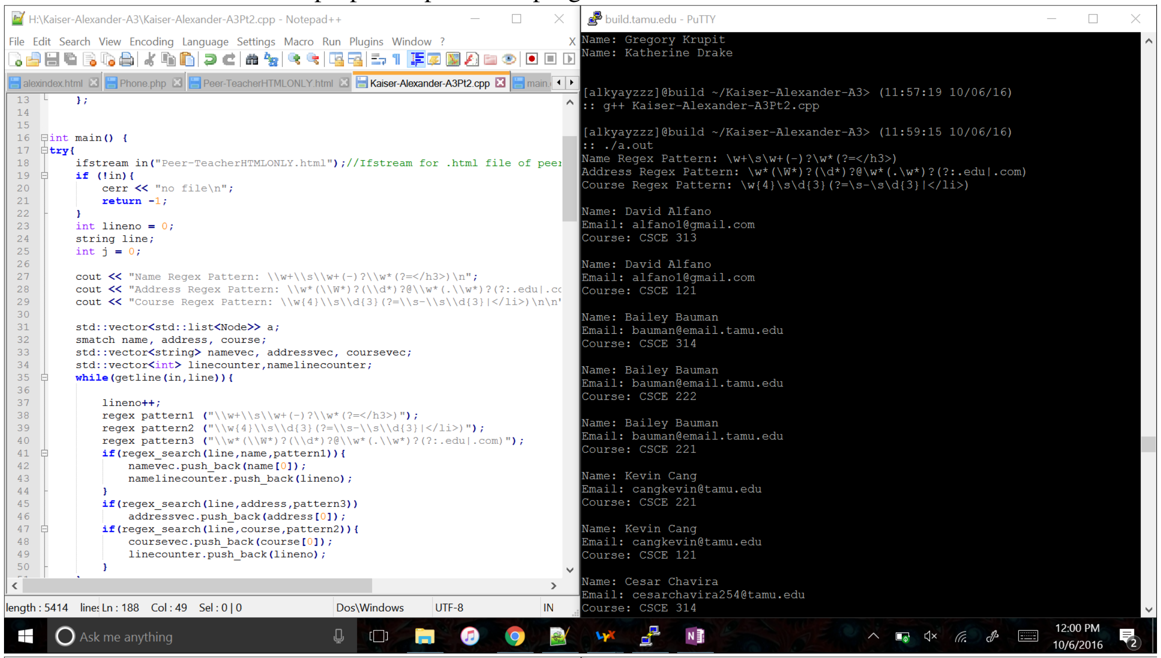
   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)
   The program will crash if the PeerTeacherHTMLONLY.html file is not in the same directory as the .cpp program or if the exact file name is not listed in the ifstream. If the user were to change any of the values of the for loops or regex patterns, the program would likely produce a segmenation fault and crash due to the fact that the amount of items that the program is designed to iterate wouled be changed.

6. Provide types of exceptions and their purpose in your program.

   (a) logical exceptions (such as deletion of an item from an empty container, etc.).
   Logical exceptions are prevented using if statements. For example, if there is no file in the ifstream or if the file is not in the correct directory, the program is terminated and "No file" is output onto the console. In order to prevent working with an empty list or vector, if statements are used in the program to check to see if the list or vector is empty. This prevents attempting to manipulate a nonexistent variable.

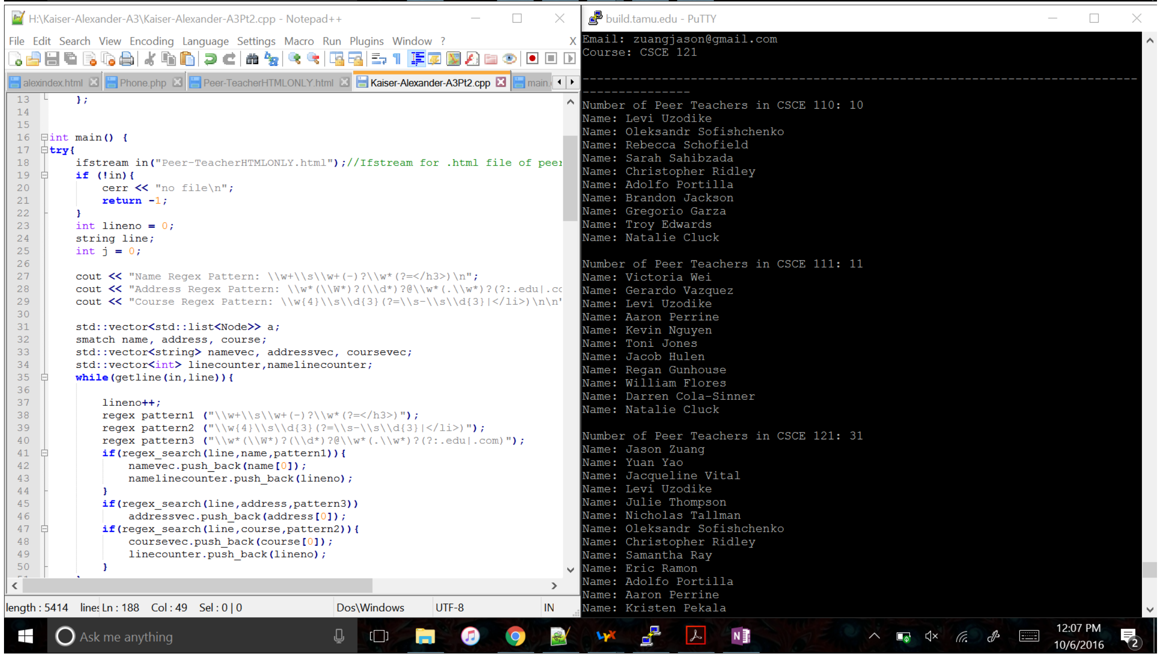
   (b) runtime exception (such as division by 0, etc.)
   To prevent runtime exception, a try-catch loop was implemented in the main function. When a runtime exception is encountered (such as a segmentation fault) the program is terminated completely and a reason is given as to why the termination occurred.

7. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output.
   The assignment does provide capability of assigning random values, thus there is no screenshot for random inputs.
   The screenshots below show proper output of the program.



```cpp
    };


int main() {
try{
    ifstream in("Peer-TeacherHTMLONLY.html");//Ifstream for .html file of peer
    if (!in){
        cerr << "no file\n";
        return -1;
    }
    int lineno = 0;
    string line;
    int j = 0;

    cout << "Name Regex Pattern: \\w+\\s\\w+(-)?\\w*(?=</h3>)\n";
    cout << "Address Regex Pattern: \\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.cc
    cout << "Course Regex Pattern: \\w{4}\\s\\d{3}(?=\\s-\\s\\d{3}|</li>)\n\n'

    std::vector<std::list<Node>> a;
    smatch name, address, course;
    std::vector<string> namevec, addressvec, coursevec;
    std::vector<int> linecounter,namelinecounter;
    while(getline(in,line)){

        lineno++;
        regex pattern1 ("\\w+\\s\\w+(-)?\\w*(?=</h3>)");
        regex pattern2 ("\\w{4}\\s\\d{3}(?=\\s-\\s\\d{3}|</li>)");
        regex pattern3 ("\\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.com)");
        if(regex_search(line,name,pattern1)){
            namevec.push_back(name[0]);
            namelinecounter.push_back(lineno);
        }
        if(regex_search(line,address,pattern3))
            addressvec.push_back(address[0]);
        if(regex_search(line,course,pattern2)){
            coursevec.push_back(course[0]);
            linecounter.push_back(lineno);
        }
```

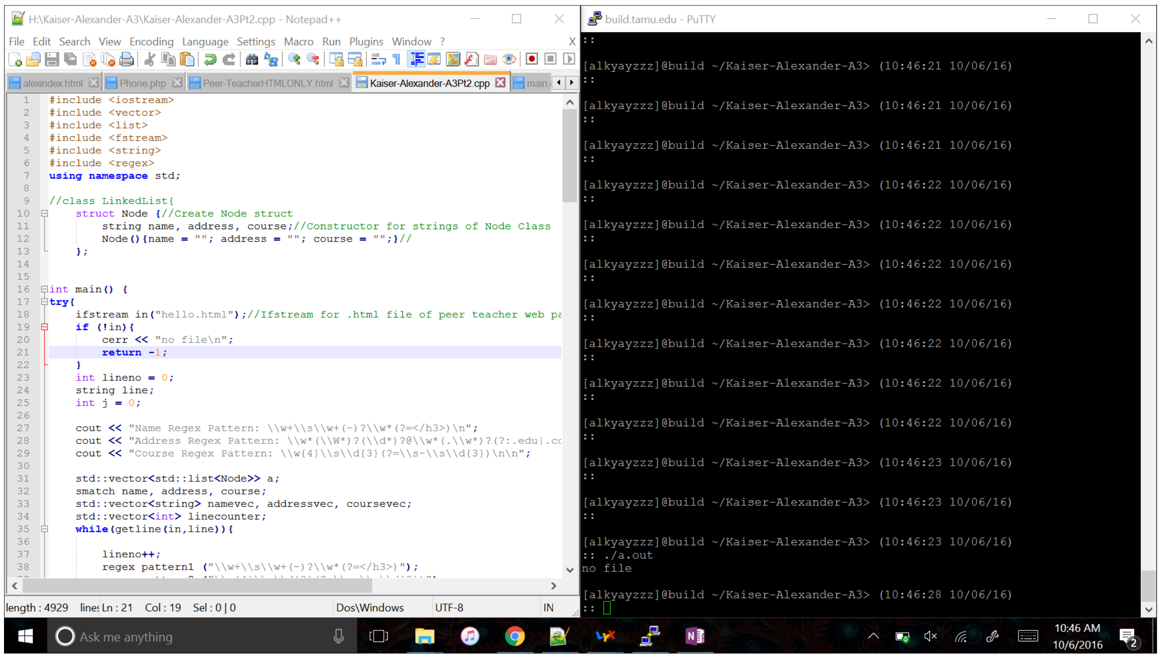
```
Name: Gregory Krupit
Name: Katherine Drake


[alkyayzzz]@build ~/Kaiser-Alexander-A3> (11:57:19 10/06/16)
:: g++ Kaiser-Alexander-A3Pt2.cpp

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (11:59:15 10/06/16)
:: ./a.out
Name Regex Pattern: \w+\s\w+(-)?\w*(?=</h3>)
Address Regex Pattern: \w*(\W*)?(\d*)?@\w*(.\w*)?(?:.edu|.com)
Course Regex Pattern: \w{4}\s\d{3}(?=\s-\s\d{3}|</li>)

Name: David Alfano
Email: alfano1@gmail.com
Course: CSCE 313

Name: David Alfano
Email: alfano1@gmail.com
Course: CSCE 121

Name: Bailey Bauman
Email: bauman@email.tamu.edu
Course: CSCE 314

Name: Bailey Bauman
Email: bauman@email.tamu.edu
Course: CSCE 222

Name: Bailey Bauman
Email: bauman@email.tamu.edu
Course: CSCE 221

Name: Kevin Cang
Email: cangkevin@tamu.edu
Course: CSCE 221

Name: Kevin Cang
Email: cangkevin@tamu.edu
Course: CSCE 121

Name: Cesar Chavira
Email: cesarchavira254@tamu.edu
Course: CSCE 314
```



```cpp
    };


int main() {
try{
    ifstream in("Peer-TeacherHTMLONLY.html");//Ifstream for .html file of peer
    if (!in){
        cerr << "no file\n";
        return -1;
    }
    int lineno = 0;
    string line;
    int j = 0;

    cout << "Name Regex Pattern: \\w+\\s\\w+(-)?\\w*(?=</h3>)\n";
    cout << "Address Regex Pattern: \\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.cc
    cout << "Course Regex Pattern: \\w{4}\\s\\d{3}(?=\\s-\\s\\d{3}|</li>)\n\n'

    std::vector<std::list<Node>> a;
    smatch name, address, course;
    std::vector<string> namevec, addressvec, coursevec;
    std::vector<int> linecounter,namelinecounter;
    while(getline(in,line)){

        lineno++;
        regex pattern1 ("\\w+\\s\\w+(-)?\\w*(?=</h3>)");
        regex pattern2 ("\\w{4}\\s\\d{3}(?=\\s-\\s\\d{3}|</li>)");
        regex pattern3 ("\\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.com)");
        if(regex_search(line,name,pattern1)){
            namevec.push_back(name[0]);
            namelinecounter.push_back(lineno);
        }
        if(regex_search(line,address,pattern3))
            addressvec.push_back(address[0]);
        if(regex_search(line,course,pattern2)){
            coursevec.push_back(course[0]);
            linecounter.push_back(lineno);
        }
```

```
Email: zuangjason@gmail.com
Course: CSCE 121

------------------------------------------------------------------------------
--------------
Number of Peer Teachers in CSCE 110: 10
Name: Levi Uzodike
Name: Oleksandr Sofishchenko
Name: Rebecca Schofield
Name: Sarah Sahibzada
Name: Christopher Ridley
Name: Adolfo Portilla
Name: Brandon Jackson
Name: Gregorio Garza
Name: Troy Edwards
Name: Natalie Cluck

Number of Peer Teachers in CSCE 111: 11
Name: Victoria Wei
Name: Gerardo Vazquez
Name: Levi Uzodike
Name: Aaron Perrine
Name: Kevin Nguyen
Name: Toni Jones
Name: Jacob Hulen
Name: Regan Gunhouse
Name: William Flores
Name: Darren Cola-Sinner
Name: Natalie Cluck

Number of Peer Teachers in CSCE 121: 31
Name: Jason Zuang
Name: Yuan Yao
Name: Jacqueline Vital
Name: Levi Uzodike
Name: Julie Thompson
Name: Nicholas Tallman
Name: Oleksandr Sofishchenko
Name: Christopher Ridley
Name: Samantha Ray
Name: Eric Ramon
Name: Adolfo Portilla
Name: Aaron Perrine
Name: Kristen Pekala
```

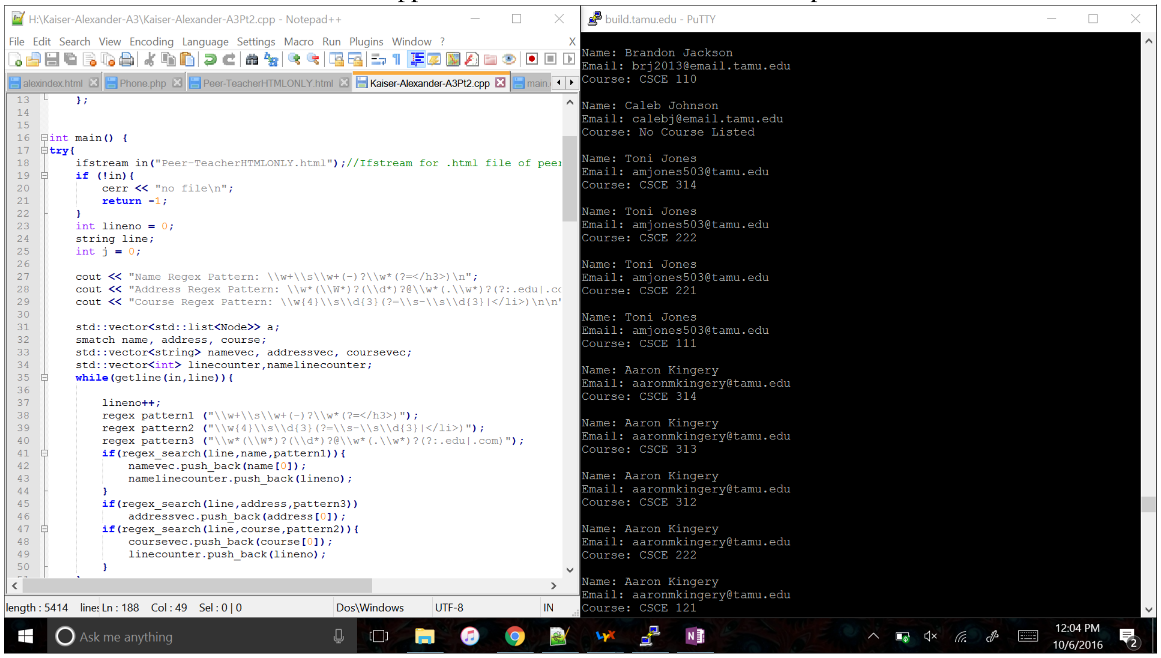
The screenshot below shows the output when there is not a file in the ifstream.



```
#include <iostream>
#include <vector>
#include <list>
#include <fstream>
#include <string>
#include <regex>
using namespace std;

//class LinkedList{
    struct Node {//Create Node struct
        string name, address, course;//Constructor for strings of Node Class
        Node(){name = ""; address = ""; course = "";}//
    };


int main() {
try{
    ifstream in("hello.html");//Ifstream for .html file of peer teacher web pa
    if (!in){
        cerr << "no file\n";
        return -1;
    }
    int lineno = 0;
    string line;
    int j = 0;

    cout << "Name Regex Pattern: \\w+\\s\\w+(-)?\\w*(?=</h3>)\n";
    cout << "Address Regex Pattern: \\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.cc
    cout << "Course Regex Pattern: \\w{4}\\s\\d{3}(?=\\s-\\s\\d{3})\n\n";

    std::vector<std::list<Node>> a;
    smatch name, address, course;
    std::vector<string> namevec, addressvec, coursevec;
    std::vector<int> linecounter;
    while(getline(in,line)){

        lineno++;
        regex pattern1 ("\\w+\\s\\w+(-)?\\w*(?=</h3>)");
```

```
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:21 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:21 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:21 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:22 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:23 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:23 10/06/16)
::

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:23 10/06/16)
:: ./a.out
no file

[alkyayzzz]@build ~/Kaiser-Alexander-A3> (10:46:28 10/06/16)
::
```

The screenshot below shows what happens when a course is not listed for a peer teacher.



```
    };


int main() {
try{
    ifstream in("Peer-TeacherHTMLONLY.html");//Ifstream for .html file of peer
    if (!in){
        cerr << "no file\n";
        return -1;
    }
    int lineno = 0;
    string line;
    int j = 0;

    cout << "Name Regex Pattern: \\w+\\s\\w+(-)?\\w*(?=</h3>)\n";
    cout << "Address Regex Pattern: \\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.cc
    cout << "Course Regex Pattern: \\w{4}\\s\\d{3}(?=\\s-\\s\\d{3}|</li>)\n\n'

    std::vector<std::list<Node>> a;
    smatch name, address, course;
    std::vector<string> namevec, addressvec, coursevec;
    std::vector<int> linecounter,namelinecounter;
    while(getline(in,line)){

        lineno++;
        regex pattern1 ("\\w+\\s\\w+(-)?\\w*(?=</h3>)");
        regex pattern2 ("\\w{4}\\s\\d{3}(?=\\s-\\s\\d{3}|</li>)");
        regex pattern3 ("\\w*(\\W*)?(\\d*)?@\\w*(.\\w*)?(?:.edu|.com)");
        if(regex_search(line,name,pattern1)){
            namevec.push_back(name[0]);
            namelinecounter.push_back(lineno);
        }
        if(regex_search(line,address,pattern3))
            addressvec.push_back(address[0]);
        if(regex_search(line,course,pattern2)){
            coursevec.push_back(course[0]);
            linecounter.push_back(lineno);
        }
```

```
Name: Brandon Jackson
Email: brj2013@email.tamu.edu
Course: CSCE 110

Name: Caleb Johnson
Email: calebj@email.tamu.edu
Course: No Course Listed

Name: Toni Jones
Email: amjones503@tamu.edu
Course: CSCE 314

Name: Toni Jones
Email: amjones503@tamu.edu
Course: CSCE 222

Name: Toni Jones
Email: amjones503@tamu.edu
Course: CSCE 221

Name: Toni Jones
Email: amjones503@tamu.edu
Course: CSCE 111

Name: Aaron Kingery
Email: aaronmkingery@tamu.edu
Course: CSCE 314

Name: Aaron Kingery
Email: aaronmkingery@tamu.edu
Course: CSCE 313

Name: Aaron Kingery
Email: aaronmkingery@tamu.edu
Course: CSCE 312

Name: Aaron Kingery
Email: aaronmkingery@tamu.edu
Course: CSCE 222

Name: Aaron Kingery
Email: aaronmkingery@tamu.edu
Course: CSCE 121
```