Appendix

```
/**
 * ******* Lasalle College Vancouver *******.
 *
 * Object Oriented Programming in C++ II
 * Week 3 - List, iterators and algorithms
 * @author
 * Ivaldo Tributino de Sousa <ISousa@lasallecollegevancouver.com>
 */
#pragma once

// Input/output library
#include <iostream>
#include <fstream> // Input/output stream class to operate on files.
using std :: cout;
using std :: endl;
using std::ifstream;

// Containers library
#include<vector>
#include <list>
using std :: vector;
using std :: list;

// Strings library
#include <string>
using std :: string;
using std :: to_string;

//Algorithms library
#include<algorithm>

// Iterators library
#include <iterator>

// Numerics library
#include <cmath>
```

linkedList.h

```
#pragma once

template<class T>
struct Node
{
    const T & data;
    Node *next;

    Node(const T & data);
    ~Node();
};

template <class T>
Node<T> :: Node(const T & data) : data(data), next(nullptr) { };

template <class T>
Node<T> :: ~Node<T>()
{
    std::cout << "Node_";
};

template <class T>
class linkedList
{
private:
    int count_;         //variable to store the number of elements in the list
    Node<T> *head_;   //pointer to the first node of the list
    Node<T> *thru_;    //pointer to traverse the list

public:

    linkedList();
    const T & operator[](unsigned index);
    void insertAtFront(const T & data);
    void display();
```

```
    int length() const;
    void insertPosition(unsigned index, const T & data);
    void deleteNote(const T & data);
    ~linkedList() {
        thru_ = nullptr;
        Node<T> *temp;
        int index = 0;
        while (head_ != nullptr) {
            temp = head_;
            head_ = head_->next;
            delete temp;
            std::cout << index << " has been deleted" << std::endl;
            ++index;
        }
        count_ = 0;
    }
};

template <class T>
linkedList<T> :: linkedList() : count_(0), head_(nullptr), thru_(nullptr){ }

template <class T>
const T & linkedList<T>::operator[](unsigned index) {

    thru_ = head_;

    if(index < count_){
        while(index>0 && thru_->next != nullptr)
        {
            thru_ = thru_->next;
            index--;
        }

    }
    return thru_->data;
}
```

Appendix

```
template <class T>
void linkedList<T>::insertAtFront(const T & data) {
  // Create a new Node on the heap:
  Node<T> *node = new Node<T>(data);
  ++count_;

  // Set the new node's next pointer point the current
  // head of the List:
  node->next = head_;

  // Set the List's head pointer to be the new node:
  head_ = node;
}

template <class T>
void linkedList<T> :: display(){

    thru_ = head_;

    while (thru_!= nullptr){
        cout << thru_->data << " ";
        thru_ = thru_->next;
    }
    cout << endl;
}

template <class T>
int linkedList<T> :: length() const{
    return count_;
}

template <class T>
void linkedList<T> :: insertPosition(unsigned index, const T & data){}
template <class T>
void linkedList<T> :: deleteNote(const T & data){}
```

Polygon.h

```
#pragma once

#include "libraries.h"

class Polygon {

  private: // Private members:

    // Data Members (underscore indicates a private member variable)
    double length_;
    unsigned int numberSides_;

  public:  // Public members:
    /**
      * Creates a triangle with one side measuring 1.
      */
    Polygon(); // Custom default constructor

    /**
      * Create a polygon using the following parameters:
      * @param numberSides.
      * @param length.
      */
    Polygon(double length, unsigned int numberSides); // Custom Constructor

    /**
      * Copy constructor: creates a new Polygon from another.
      * @param obj polygon to be copied.
      */
    Polygon(const Polygon & obj); // Custom Copy constructor

    /**
```

```
    * Assignment operator for setting two Polygon equal to one another.
    * @param obj Polygon to copy into the current Polygon.
    * @return The current image for assignment chaining.
    */
  Polygon & operator=(const Polygon & obj);  // Custom assignment operator;

  Polygon operator+(const Polygon & obj); // Operators + Overloading

  /**
    * Destructor: frees all memory associated with a given Polygon object.
    * Invoked by the system.
    */
  virtual ~Polygon(); // Destructor

   /**
    * Override Functions
    */
  string shapeName() const;

  double area() const;


  /**
    * Gets and sets
    */

  void setlength(double length);

  void setNumberSides(unsigned int numberSides) ;

  double getlength() const;

  unsigned int getNumberSides() const;

};
```

Polygon.cpp

```
#include "Polygon.h"

// #define Allows the programmer to give a name to a constant value before
the program is compiled
#define PI 3.14159265


Polygon :: Polygon(){
  length_ = 1;
  numberSides_ = 3;
  std::cout << "Default Constructor Invoked"  << std::endl;
}

Polygon :: Polygon(double length, unsigned int numberSides){
  length_ = (length>0)? length: 1;
  numberSides_ = (numberSides>2)? numberSides : 3;
  std::cout << "Constructor Invoked"  << std::endl;
}

Polygon :: Polygon(const Polygon & obj){
  length_ = obj.length_;
  std::cout << "Copy Constructor Invoked"  << std::endl;
}

Polygon & Polygon :: operator=(const Polygon & obj){
  length_ = obj.length_;
  numberSides_ = obj.numberSides_;
  std::cout << "Assignment operator invoked"  << std::endl;
  return *this; // dereferenced pointer
}

Polygon Polygon :: operator+(const Polygon & obj){
```

Appendix

```
  Polygon A;
  A.length_ = this->length_ + obj.length_;
  A.numberSides_ = this->numberSides_ + obj.numberSides_;
  return A;
}

Polygon::~Polygon() {
    std::cout << "Polygon destroyed" << std::endl;
}

double Polygon:: area() const{
  double perimeter = numberSides_*length_;
  double apothem = (length_)/(2*tan(PI/numberSides_));
  return perimeter*apothem/2;
}

string Polygon::shapeName() const {

  string arrayName[6] = {"triangle" , "square", "pentagon",
  "hexagon", "heptagon", "octagon"};

  string name = (numberSides_<9)? arrayName[numberSides_-3]:
to_string(numberSides_)+"_polygon";

  return name;
}

void Polygon::setlength(double length) {
  if (length>0){
    length_ = length;
  }
  else{
    std::cout << "Please, set a value greater than 0" << std::endl;
  };
}
```

Appendix

```
void Polygon :: setNumberSides(unsigned int numberSides) {

  if (numberSides>2){
    numberSides_ = numberSides;
  }
  else{
    std::cout << "Please, only set values above 2." << std::endl;
  };

}

double Polygon ::getlength() const {
  return length_ ;
}

unsigned int Polygon :: getNumberSides() const {
  return numberSides_;
}
```

main.cpp

```
bool is_all_upper(const string& str);

int main(){
    //*************************************************
    //            -------- Array --------
    //*************************************************
    cout << "-------- Array --------" << endl;

    int a[] = {1,2,3,4,5,6,7,8,9,0};

    for(int i=0; i < 10; i++){
        cout << "Memory address of data " << a[i] << " is: " << &a[i]<<endl;
    }

    //*************************************************
    //            -------- Vector --------
    //*************************************************
    cout << "-------- Vector --------" << endl;

    vector<int> v(10);
    copy(a, a+10, v.begin());

    cout << "Vectors are assigned memory in blocks of contiguous locations"
<< endl;
    for(int i=0; i < 10; i++){
        cout << "Memory address of data " << v[i] << " is: " << &v[i]<<endl;
    }

    v.erase(v.begin()+4);

    cout << "What will be the address of the elements after erose one of
them?" << endl;
```

Appendix

```
for(int i=0; i < 10; i++){
    cout << "Memory address of data " << v[i] << " is: " << &v[i]<<endl;
}

//************************************************
//            -------- Node<T> --------
//************************************************
cout << "-------- Node<T> --------" << endl;
{
Node<int> n_1(20);
Node<int>* n_2 = new Node<int>(10);
n_1.next = n_2;
cout << n_1.data << endl;
cout << n_2 << endl;
cout << n_2->data << endl;
cout << n_1.next->data << endl;
delete n_2;
}
cout << '\n';
//************************************************
//            -------- LINKEDLIST --------
//************************************************
cout << "-------- LINKEDLIST of integers --------" << endl;
{
linkedList<int> list;
list.insertAtFront(10); //l[4]
list.insertAtFront(13); //l[3]
list.insertAtFront(16); //l[2]
list.insertAtFront(19); //l[1]
list.insertAtFront(21); //l[0]

list.display();

std::cout << "list[0] = " << list[0] << std::endl;
```

```
list.insertPosition(0, 45);
list.insertPosition(2, 100);
list.insertPosition(100,888);
list.insertPosition(100,888);
list.insertPosition(100,888);
list.insertPosition(0, 888);
list.display();

list.deleteNote(888);
list.display();
}

cout << "-------- LINKEDLIST of Polygons --------" << endl;

{
    linkedList<Polygon> polyList;
    Polygon triangle;
    Polygon square(1,4);
    Polygon pentagon(1,5);

    polyList.insertAtFront(triangle);
    polyList.insertAtFront(square);
    polyList.insertAtFront(pentagon);

    for(int i = 0; i<3; i++){
        cout << polyList[i].shapeName() << endl;
    }

}

//************************************************
//           -------- LIST --------
//************************************************
cout << "-- List Container Example from our Textbook --" << endl;

list<int> intList1, intList2, intList3, intList4;
```

Appendix

```
    std::ostream_iterator<int> screen(cout, " ");

    intList1.push_back(23);
    intList1.push_back(58);
    intList1.push_back(58);
    intList1.push_back(58);
    intList1.push_back(36);
    intList1.push_back(15);
    intList1.push_back(93);
    intList1.push_back(98);
    intList1.push_back(58);

    cout << "Line 135: intList1: ";
    copy(intList1.begin(), intList1.end(), screen);
    cout << endl;

    intList2 = intList1;

    cout << "Line 141: intList2: ";
    copy(intList2.begin(), intList2.end(), screen);
    cout << endl;

    intList1.unique();

    cout << "Line 147: After removing the consecutive " << "duplicates," <<
endl
    << " intList1: ";
    copy(intList1.begin(), intList1.end(), screen);
    cout << endl;

    intList2.sort();

    cout << "Line 154: After sorting, intList2: ";
    copy(intList2.begin(),intList2.end(),screen);
    cout << endl;
```

```
    intList3.push_back(13);
    intList3.push_back(25);
    intList3.push_back(23);
    intList3.push_back(198);
    intList3.push_back(136);

    cout << "Line 164: intList3: ";
    copy(intList3.begin(), intList3.end(), screen);
     cout << endl;

    intList3.sort();

    cout << "Line 170: After sorting, intList3: ";
    copy(intList3.begin(), intList3.end(), screen);
    cout << endl;

    intList2.merge(intList3);

    cout << "Line 176: After merging intList2 and " << "intList3, intList2: "
<< endl << " ";
    copy(intList2.begin(), intList2.end(), screen); cout << endl;


    //************************************************
    //           -------- Iterators --------
    //************************************************
    cout << "-------- Iterators --------" << endl;

    ifstream fileIn("francis.txt");

    if(!fileIn.is_open())
    {
        cout << "Failed to open file!\n";
        return 0;
    }
```

Appendix

```
    std::istream_iterator< string > is(fileIn);
    std::istream_iterator< string > eof;

    vector< string > text;
    copy( is, eof, back_inserter(text));

    std::ostream_iterator<string> os(cout, " ");
    copy( text.begin(), text.end(), os);
    cout << endl;

    // copy(std::istream_iterator< string >(fileIn), std::istream_iterator<
string >(),std::ostream_iterator<string> (cout, " "));

    cout << "- Now let's remove the character " << text.back() << endl;

    vector<string>::iterator lastElem;

    text.erase(remove(text.begin(), text.end(), "}"));

    copy(text.begin(), text.end(), os);
    cout << endl;

    cout << "- Finally, let replace the newline character with Full stop" <<
endl;

    text.at(text.size()-1) = "AMEN.";

    copy(text.begin(), text.end(), os);
    cout<< '\n';

    //************************************************
    //         -------- Algorithms --------
    //************************************************
    cout << "-------- Algorithms --------" << endl;
```

```
    cout << "- Using Find" << endl;

    vector<string>::iterator position;
    position = find(text.begin(), text.end(), "darkness,");
    copy(position, text.end(), os);
    cout<< '\n';

    cout << "- Using Find_if" << endl;

    cout<< "1 = true and 0 = false, Result: " <<is_all_upper(text.back()) <<
endl;
    position = find_if(text.begin(), text.end(), is_all_upper);
    copy(text.begin(), position, os);
    cout << '\n';

    cout << "- Using fill and fill_n" << endl;

    fill(text.end()-1,text.end(), "Amen." );
    copy(text.begin(), text.end(), os);
    cout << "\n";

    text.resize(text.size()+3);
    fill_n(text.rbegin(), 3, "!");
    copy(text.begin(), text.end(), os);
    cout << "\n";
    //************************************************
    //  -------- REMOVING DUPLICATES --------
    //************************************************
    cout << "-------- REMOVING DUPLICATES --------" << endl;
    vector<string> emails;
    ifstream isFile("mbox-short.txt");
    string str;
    while(getline(isFile, str)) // storing into str until the delimitation
character '\n' is find
    {
        if (str.find("From:") != string::npos)
```

```
        {
            int pos = str.find(" ");

            emails.push_back(str.substr(pos+1));
            cout << str.substr(pos+1) << endl;

        };
    }
    cout << "- After removing the duplication" << endl;

    std::sort(emails.begin(), emails.end());
    emails.erase(std::unique(emails.begin(),emails.end()), emails.end());

    vector<string> :: iterator itr;

    for(itr = emails.begin(); itr != emails.end(); ++itr){
        cout << *itr << endl;
    }



   return 0;
}
bool is_all_upper(const string& str){

    bool flag = true; // for verification

    if(str.length() == 1) flag = false;

    for (int i = 0; i < str.size(); i++) // till string length
        if (!isupper(str[i]) && str[i]!= '.') {
            flag = false;
            break;
        }

    return flag;
}
```