Libraries.h

```
/**
 * ******* Lasalle College Vancouver *******.
 *
 * Object Oriented Programming in C++ II
 * Week 2 - Templates and Introduction to STL Vectors
 * @author
 * Ivaldo Tributino de Sousa <ISousa@lasallecollegevancouver.com>
 */
#pragma once

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

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

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

//Algorithms library
#include<algorithm>

// Numerics library
#include <cmath
```

listType.h

```
template<class elemType>
class listType
{
public:
    bool isEmpty() const; //determine whether the list is empty
    bool isFull() const; // determine whether the list is full
    int getLength() const; // return the number of elements in the list.
    int getMaxSize() const; // return the maximum number of elements that can
be stored in the list.
    void sort(); // to sort the list
    void print() const; //Outputs the elements of the list.
    void insertAt(const elemType& item, int position); //Function to insert
item in the list at the location
    //specified by position.

    /**
      * Constructor
      * Creates an array of the size specified by the parameter listSize
      * the default array size is 50.
      */
    listType(int listSize = 50);

    /**
     * Destructor
     * Deletes all the elements of the list.
     * Postcondition: The array list is deleted.
     */

    ~listType();


private:
```

```
    int maxSize; // variable to store the maximum size

    int length; // variable to store the number of elements in the list

    elemType *list; //pointer to the array that holds the list elements.

};

template <class elemType>
bool listType<elemType>::isEmpty() const
{
    return (length == 0);
}

template <class elemType>
bool listType<elemType>::isFull() const
{
    return (length == maxSize);
}

template <class elemType>
int listType<elemType>::getLength() const
{
    return length;
}

template <class elemType>
int listType<elemType>::getMaxSize() const
{
    return maxSize;
}

//Constructor; the default array size is 50
template <class elemType>
listType<elemType>::listType(int listSize)
{
```

```
    maxSize = listSize;
    length = 0;
    list = new elemType[maxSize];
}

template <class elemType>
listType<elemType>::~listType() //destructor
{
    delete [] list;
}

template <class elemType>
void listType<elemType>::sort() //selection sort
{
    int i, j;
    int min;
    elemType temp;
    for (i = 0; i < length; i++)
    {
        min = i;
        for (j = i + 1; j < length; ++j)
            if (list[j] < list[min])
                min = j;
        temp = list[i];
        list[i] = list[min];
        list[min] = temp;
    }//end for
}//end sort

template <class elemType>
void listType<elemType>::print() const
{
    int i;
    for (i = 0; i < length; ++i)
        cout << list[i] << " ";
    cout << endl;
```

```
}//end print

template <class elemType>
void listType<elemType>::insertAt(const elemType& item,int position)
{
    assert(position >= 0 && position < maxSize);
    list[position] = item;
    length++;
}
```

Polygon.cpp

```
#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 &); //Overload the operator +

//Overload the stream insertion and extraction operators

friend ostream& operator<<(ostream&, const Polygon&);

friend istream& operator>>(istream&, Polygon&);

// friend Polygon operator+(const Polygon &obj1, const Polygon &obj2);
/**
  * Destructor: frees all memory associated with a given Polygon object.
  * Invoked by the system.
  */
virtual ~Polygon(); // Destructor

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

double area();


/**
  * 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){

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

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

  Polygon temp;

  temp.length_ = length_ + obj.length_;
  temp.numberSides_ = numberSides_ + obj.numberSides_;

  return temp;
}

ostream& operator<<(ostream& osObject, const Polygon& obj){
}

istream& operator>>(istream& isObject, Polygon& obj) {
}



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

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

string Polygon::shapeName(){

  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;
  };
}

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

```
int main(){

    //************************************************
    //  -------- Initialize a vector in C++ --------
    //************************************************
    cout << "---- Initialize a vector in C++ ----" << endl;

    cout << "Create an empty vector: " << endl;
    vector<int> vect1;

    for(int x : vect1){
        cout << x << ",";
    }
    cout << "\n";

    cout << "Create a vector using the default constructor: " << endl;
    vector<int> vect2(10);
    for(int x : vect2){
        cout << x << ",";
    }
    cout << "\n";

    cout << "Create a vector of size n with all values as 10: " << endl;
    int n = 7;
    vector<int> vect3(n, 10);
    for (int x : vect3){
         cout << x << ",";
    }
    cout << "\n";

    cout << "Create a vector from another vector: " << endl;
    vector<int> vect4(vect3.begin()+2, vect3.end());
    for (int x : vect4){
```

```
        cout << x << ",";
    }
    cout << "\n";

    vect4.resize(10,3);
    for (int x : vect4){
        cout << x << ",";
    }
    cout << "\n";
    //************************************************
    //  -------- MANIPULATE THE DATA --------
    //************************************************
    cout << "---- MANIPULATE THE DATA ----" << endl;

    vector<int> vecPrime(5);

    for (int j = 2; j < 7; j++)
        vecPrime[j-2] = j;

    vecPrime.insert(vecPrime.end()-1,7);

    // To add elements to intPrimevecPrime, we can use the function push_back
as follows:
    vecPrime.push_back(11);
    vecPrime.push_back(13);
    vecPrime.push_back(17);
    vecPrime.push_back(19);

    vecPrime.resize(11, 23);

    for (int x : vecPrime){
        cout << x << ",";
    }
    cout << "\n";

    cout << "Erasing 4 and 6 " << endl;
```

```
    vecPrime.erase(vecPrime.begin()+2);
    vecPrime.erase(vecPrime.begin()+4);
    for (int x : vecPrime){
         cout << x << ",";
    }
    cout << "\n";

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

    vector<int>::iterator vecPrimeIter;

    cout <<  "Navigate through a vector using iterators" << endl;
    for (vecPrimeIter = vecPrime.begin() ; vecPrimeIter != vecPrime.end(); +
+vecPrimeIter)
        cout << *vecPrimeIter << " ";
    cout <<  '\n';


    //************************************************
    //  --------  Using copy function --------
    //************************************************
    cout <<  "--------  Using copy function --------" << endl;
    int primeArray [] = {2,3,5,7,11,13,17,19,23,29};
    vecPrime.resize(10);

    /*Recall that the array name, intArray, is, in fact, a pointer and
contains the base address of the array.
    Therefore, intArray points to the first component of the array, intArray
+ 1
    points to the second component of the array, and so on.*/

    copy(primeArray, primeArray + 10, vecPrime.begin());
```

```
    for (vecPrimeIter = vecPrime.begin(); vecPrimeIter != vecPrime.end();
vecPrimeIter++)
        cout << *vecPrimeIter << " ";
    cout << '\n';

    // Now consider the statement:
    copy(vecPrime.rbegin() + 2, vecPrime.rend(), vecPrime.rbegin());

    for (vecPrimeIter = vecPrime.begin(); vecPrimeIter != vecPrime.end();
vecPrimeIter++)
        cout << *vecPrimeIter << " ";
    cout << '\n';

    //************************************************
    //  -- Now using ostream_iterator & copy function --
    //************************************************
    cout <<  "-- Now using ostream_iterator & copy function --" << endl;

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

    copy(primeArray, primeArray + 10, vecPrime.begin());
    copy(vecPrime.rbegin() + 2, vecPrime.rend(), vecPrime.rbegin());
    copy(vecPrime.begin(), vecPrime.end(), screen);
    cout << '\n';

    cout << "-- Now let's create a vector from a text file and print its
elements. --" << endl;

    ifstream fileIn("francis.txt");

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

```
    string data;

    // istream_iterator< string > is(fileIn);
    // istream_iterator< string > eof;

    vector< string > text;

    while (fileIn >> data) // loop until no more input or error
    {
        // and remember operator>> can only extract string without spaces!
        text.push_back(data);
    }

    // copy( is, eof, back_inserter(text));
    std::ostream_iterator<string> os(cout, " ");
    copy( text.begin(), text.end(), os);




    return 0;
}
```