```
/**
 * ******* Lasalle College Vancouver *******.
 *
 * Object Oriented Programming in C++ II
 * Week 4
 * @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.
#include <cstdio>
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>
```

## Polygon.h

```
#pragma once

#include "libraries.h"

class Polygon {

  private: // Private members:

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

  protected: // Protected mebers:
    string solidName;

  public:  // Public members:
    /**
      * Creates a numberSides sided Polygon.
      */
    Polygon(int numberSides);

    Polygon(const Polygon & obj); // Custom Copy constructor

    Polygon & operator=(const Polygon & obj);  // Custom assignment operator;

    ~Polygon();

    /**
      * Function Call Operator () Overloading:
      */
    double operator()(float lenght) const;

    bool operator<(const Polygon & obj);

    bool operator>(const Polygon & obj);
    /**
    * Return the polygon name by its number of sides.
    */
    string shapeName() const;

    /**
      * Gets and Sets
      */

    unsigned int getNumberSides() const;

    void setNumberSides(unsigned int numberOfSides);
};
```

## Polygon.cpp

```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(int numberSides){
  (numberSides > 2)? numberSides_ = numberSides : numberSides_ = 3;
  cout << "Constructor Invoked"  << endl;
}

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

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

Polygon::~Polygon() {
    std::cout << "Polygon destroyed" << std::endl;
}
// function to overload the operator
double Polygon :: operator()(float length) const{
  double perimeter = numberSides_*length;
  double apothem = (length)/(2*tan(PI/numberSides_));
  return perimeter*apothem/2;

}
```

```
bool Polygon :: operator <(const Polygon & obj){}

bool Polygon :: operator >(const Polygon & obj){}

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;
}

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

void  Polygon :: setNumberSides(unsigned int numberOfSides){
  numberSides_ = numberOfSides;
}
```

# ourElements.h

```
#include "libraries.h"

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

// Alias-declaration - In C++11
using BinaryPredicate = bool(*)(int, int); // or  = std::function<bool(int,
int)>;

struct absValue
{
    double operator()(double f) {
        return f > 0 ? f : -f;
    }
};

class PrintName {
public:
    void operator()(const Polygon & elem){
        cout << elem.shapeName() << " ";
    }
};

int add3(int number){
  return 3 + number;
}

template<typename T>
void SelectionSort(int *array, int size, T bp)
{
    // Step through each element of the array
    for (int startIndex = 0 ; startIndex < (size - 1); ++startIndex)
    {
        // smallestIndex is the index of the smallest element we've
encountered so far.
        int smallestIndex = startIndex;

        // Look for smallest element remaining in the array (starting at
startIndex+1)
        for (int currentIndex(startIndex + 1 ); currentIndex < size; +
+currentIndex)
        {
            // If the current element is smaller than our previously found
smallest
            if (bp(array[smallestIndex], array[currentIndex])) // COMPARISON
DONE HERE
            {
                // This is the new smallest number for this iteration
                smallestIndex = currentIndex;
            }
        }

        // Swap our start element with our smallest element
        std::swap(array[startIndex], array[smallestIndex]);
    }
}

bool ascendingZ5(int x, int y)
{
    return x%5 > y%5;
}

bool descendingZ5(int x, int y)
{
    return x%5 < y%5;
}
```

```
struct RingAscendin
{
    int quotient;
    RingAscendin(int q) : quotient(q){}

    bool operator()(int x, int y){
        return (x%quotient) > (y%quotient);
    }
};
```

## main.cpp

```cpp
#include "Polygon.h"
#include "ourElements.h"

int main() {
  //************************************************
  //           -------- Area Matrix --------
  //************************************************
  cout << "-------- Area Matrix --------" << endl;
  Polygon triangle(3);
  Polygon square(4);
  Polygon pentagon(5);
  Polygon hexagon(6);
  Polygon heptagon(7);

  vector<Polygon> polys; // for the fastest implementation use pointers. vector<Polygon*> polys
  polys.reserve(5);

  polys.push_back(triangle);
  polys.push_back(square);
  polys.push_back(pentagon);
  polys.push_back(hexagon);
  polys.push_back(heptagon);

  for(Polygon const & poly : polys){
    for(int i =1; i<6; i++){
        cout <<  poly(2*i) << " | ";
    }
    cout << endl;
  }

  cout << "- object behaves like a function" << endl;
  cout << triangle(20/pow(3,0.25)) << endl;
  cout << square(10) << endl;
  cout << hexagon(20/pow(3,0.25)) << endl;
```

```
//************************************************
//       -------- Example: absValue --------
//************************************************
cout << "-------- absValue --------" << endl;
absValue absObj;

cout << -PI << endl;
cout << absObj(-PI) << endl;
cout << absObj(PI) << endl;

//************************************************
//       -------- Example: PrintName --------
//************************************************
cout << "-------- PrintName --------" << endl;

PrintName print;

// The for_each function applied a specific function to each member of a range:
for_each(polys.begin(), polys.end(), print);
cout << endl;

// Call print(Polygon) is equivalent to print.operator()(Polygon).
for(auto iter = polys.begin(); iter!=polys.end(); ++iter){
  print.operator()(*iter);
}
cout << '\n';

//************************************************
//       -------- Function Pointer --------
//************************************************
cout << "-------- Function Pointer --------" << endl;

// Tell C++ to interpret function addd3 as a void pointer
cout << reinterpret_cast<void*>(add3) << endl;

// add3_ptr is a pointer to function add3
```

```
  int (*add3_ptr)(int) = &add3; // or (*add3_ptr)(int)(&add3);

/* The above line is equivalent of following two
  int (*add3_ptr)(int);
  add3_ptr = &fun;
*/

// Invoking add3() using add3_ptr
cout << (*add3_ptr)(10) << endl;
cout << add3_ptr(10) << endl;

int a[] = {1, 14, 30, 52, 63};

SelectionSort(a, 4, ascendingZ5);

for(int i = 0; i < 4; ++i){
  cout << a[i] << " ";
}

SelectionSort(a, 4, descendingZ5);
cout << '\n';
for(int i = 0; i < 4; ++i){
  cout << a[i] << " ";
}
cout << '\n';

//************************************************
//       -------- Lambda Function --------
//************************************************
cout << "-------- Lambda Function --------" << endl;

for_each(polys.begin(), polys.end(), [](const Polygon& elem)
{
  cout << elem.shapeName() << " ";
});
cout << '\n';
```

```
  int a1[] = {101, 17, 56, 18, 99};
  SelectionSort(a1, 5, [](int x ,int y)->bool{ return x%8<y%8;});

  for(int i = 0; i < 5; ++i){
    cout << a1[i] << " ";
  }
  cout << '\n';

  cout << "-------- std::function --------" << endl;
  // Polymorphic Function Wrapper std::function (can be used in order to
store lambda expressions.)

  int n = 3;
  std::function<bool(int, int)> f;
  f = [n](int x ,int y)->bool{ return (x%n)>(y%n);};
  std::cout << f(9, 11) << endl; // 0 so false

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

  int angles[] = {189, 2910, 2640, 1216, 430};
  n = 360;
  SelectionSort(angles, 5, [n](int x ,int y)->bool
  {
    return (x%n)>(y%n);

  });
  for(int i = 0; i < 5; ++i){
    cout << angles[i] << " ";
  }
 return 0;
}
```