How to expose interface and hide implementation in C++?

Let's say we have a class Point defined here.

```
class Point {

  public:
         Point(double=0.0, double=0.0); // Default constructor
       Point(const Point&);           // Copy constructor
       ~Point();              // Destructor
       Point& operator=(const Point&);  //assignment operator
       double getX() const;
       double getY() const;
       string toString() const;

  protected:
       double _x, _y;
}
```

****

```
Point p0 = Point();
Point p1(21.34);
Point p2(2.1, 4.3);
Point p3(p1);
Point p4(1,1);
```

Which are valid declarations for Point?
****

Let's implement the methods of Point class.

```
Point::Point(double x, double y) {
  _x = x, _y = y;
}

Point::Point(const Point& p) {
  _x = p.getX(), _y = p._y; // _y = p.getY();
}

Point::~Point() { }

Point& Point::operator=(const Point& p) {
 _x = p.getX();
 _y = p.getY();
 return this;
}

double Point::getX() const { return _x; }
double Point::getY() const { return _y; }

string Point::toString() const {
  ostringstream output;
  output << "(" << _x << "," << _y << ")";
  return output.str();
}
```

Typically, for a small test program its fine to have all of the above code in a single source code file, say, Point.cpp. But this exposes your implementation details to outsiders in case you want them to use Point class.

Better approach::

Put class interface in Point.h

Put class implementation in Point.cpp

For the outside parties, you may just provide binary for Point.cpp and Point.h, so that they can use your Point class.

Note::

1. As the Point.h contains the class interface only, Point.cpp will need to includ this file in its source to begin with else Point.cpp will not compile.

2. Take care of header file inclusion in appropriate file(s)

3. Write a routine to test the class & its methods.

==========
****
I want to add operator << to Point class. Where to add?
****
==========

Modify Point.h

Add: friend function ostream& operator<<(ostream&, const Point&);

Include its implementation in Point.cpp as

```
ostream& operator<<(ostream& ostr, const Point& point) {
          return ostr << point.toString();
}
```

Compile Point.cpp and driver.cpp

Remember -c option to create intermediate .o file.

==========
Please refer to ::
container_set.cpp
container_map.cpp
==========

On Day1, we had discussed Stack.

Application of Stack:: Lab

1. Implement Stack class using arrays.
2. Solve Towers of Hanoi problem using recursion and later using Stack as a data structure and without recursion.

Note: Towers of Hanoi consist of three vertical pegs and a sequence of n disks with holes in their centers. The radii of these disks are in arithmetic progression and are mounted on Peg A. Objective of is to move all the disks to Peg C. While moving following needs to be adhered to.

No disks may be above a smaller disk on the same Peg.
You can move only one disk at any given point of time.

Illustrate for 2 disks and 3 disks. Generalize algo. Implement it.

3. Reverse Polish Notation calculator (postfix)
   operator is placed at the end of operands.

   4*(5+6) (usually called infix)

   4 5 6 + *

   Postfix is easier to implment in the computing(calculators,etc)

   Write a program to convert infix expression into postfix.
   Develop RPN calculator.

4. Using Stack as a DS implement a method to check whether a given string is a palindrome.

   Palindrome string reads same as backwards and forwards.
   E.g., mom, kayak, anna, radar, refer, ullu

==========