COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment #1 -

Name: Andy Qin **Date**: October 5th, 2020

1. (4 pts) Design a class for a point in polar-coordinates (r, theta), r, radius, is in cm and theta is in degrees with the following member functions:

default constructor (0, 0)
getters: get_r and get_angle
setter: set_angle

please provide an example of a main function using the polar coordinates.

a) The Header file of polar_coord class

```
#ifndef ASS1_H     // Prevent duplicate definition
#define ASS1_H
#include <iostream>

namespace coen79_assignment1
{
  class polar_coord
  {
  private:
      double r, angle;

  public:
    polar_coord(double next_r = 0, double next_angle = 0);
    double get_r() const;
    double get_angle() const;
    void set_angle(double next_angle);
  };

```

b)The Implementation of polar_coord

```cpp
#include <iostream>
#include "assignment1.h"

using namespace std;

namespace coen79_assignment1
{

    polar_coord :: polar_coord(double next_r, double next_angle)
    {
      r = next_r;
      angle = next_angle;
    }
    double polar_coord :: get_r() const
    {
      return r;
    }
    double polar_coord :: get_angle() const
    {
      return angle;
    }
    void polar_coord :: set_angle(double next_angle)
    {
      angle = next_angle;
    }

}
```

c) Main Tester

```cpp
int main(int argc, const char* argv[])
{
  polar_coord p1(2, 30), p2, p3(6,0);

  cout << "Initial Values:" <<endl;
  cout << "p1:" <<endl;
  printPolarCoordInfo(p1);
  cout << "p2:" <<endl;
  printPolarCoordInfo(p2);
  cout << "p3:" <<endl;
  printPolarCoordInfo(p3);

  p1.set_angle(30);
  p2.set_angle(60);
  p3.set_angle(90);

  cout << "Modified Values:" <<endl;
  cout << "p1:" <<endl;
  printPolarCoordInfo(p1);
  toCartesian(p1);
  cout << "p2:" <<endl;
  printPolarCoordInfo(p2);
  toCartesian(p2);
  cout << "p3:" <<endl;
  printPolarCoordInfo(p3);
  toCartesian(p3);

return 0;
}
```

d) main functions added to test the class

```
//print out all the private varaibles of a polar point
void printPolarCoordInfo(polar_coord & p)
{
    cout << "Radius: " << p.get_r() << endl;
    cout << "Angle: " << p.get_angle() << endl;
    cout << endl;
}

//convert polar coordinate to cartesian coordinate
void toCartesian(polar_coord & p)
{
  double theta = p.get_angle() * M_PI / 180;
  cout << "In Cartesian:" << endl;
  cout << "X: " << p.get_r() * cos(theta) << endl;
  cout << "Y: " << p.get_r() * sin(theta) << endl;
  cout << endl;
}
```

2. (2 pts) What is the time complexity of `fun()`. Please show your proof.

```
int fun(int n)
{
 int count = 0;
 for (int i = n; i > 0; i /= 2)
 for (int j = 0; j < i; j++)
 count += 1;
 return count;
}
```

Answer: The run time for inner loop is about: $(4n+4(n/2)+4(n/4) + 4(n/8) + \ldots +4) \cong \mathbf{n}$

The run time for outer loop is about: $\mathbf{\log_2(n)}$ since i decrement by half every loop

The total run time = **n * log(n)**.

COEN 79

Object-Oriented Programming and Advanced Data Structures

3. (1 pt) Give a concise formula that gives the approximate number of digits in a positive integer. The integer is written in base 10.

Answer: int num_digit(int pos)

```
{
    return floor(log(pos)) + 1;
}
```

4. (2 pts) What are the differences between references and pointers?

Answer:

Pointers

- variables that hold memory addressed of their own on the stack
- Need to be dereferenced before accessing the memory
- Can be changed to point to another memory

References

- Variables that point to already declared variables data and share the memory space
- They are of the type "constant pointer", which means their values can not be manipulated. In other words, they can only be used to access one variable data.
- They are copies or instances of a variable and are temporary.

5. (3 pts) What are the three ways we can use items defined in a namespace. Include examples in your answer.

Answer:

1.Make all of the namespace available

- Ex: "**using namespace coen79_assignment1;**"

2.Make only a specific item in the namespace available

- Ex: "**using coen79_assignment :: polar_coord;**"

3.Use item of the choice from the namespace

- Ex: "**coen79_assignment1 :: item_name;**"

6. (2 pts) Discuss about the output of the following code. How will the result change if we replace struct with class?

```
1. struct Test {
2. int x;
3. };
4.
5. int main() {
6. Test t;
7. t.x = 20;
8. cout<t.x<endl;
9. return 0;
10. }
```

Answer: the code will not compile because a class declaration has stricter accessibility of the variables than a struct declaration. The variable x declared in a class will be private and will not be accessible to the outsider programs. So when the main.cpp tries to do "t.x", it can not do it because it does not have access to the variable x.

7. (2 pts) A The header of the `point` class is as follows:

```
1. class point
2. {
3. public:
4. // CONSTRUCTOR
5. point (double initial_x = 0.0, double initial_y = 0.0);
6.
7. // MODIFICATION MEMBER FUNCTIONS
8. void set_x (double& value);
9. void set_y (double& value);
10.
11. // CONST MEMBER FUNCTIONS
12. point operator+ (double& in) const;
13.
14. private:
15. double x; // x coordinate of this point
16. double y; // y coordinate of this point
17.
18. };
```

```
1. main() {
2. point myPoint1, myPoint2, myPoint3;
3. double shift = 8.5;
4. myPoint1 = shift + myPoint2;
5. myPoint3 = myPoint1.operator+ (shift);
6. myPoint1 = myPoint1 + shift;
7. }
```

▪ Which line of the following code results in an error? Explain why.

Answer:

Line 4 will not run. Since the function "operator+" is provoked like "(point).operator +(double),
It does not make sense to do "(double).operator+ (point)".
What line 4 does is putting a non-member into a member function.

▪ What's the solution?

Answer: we can add a non-member overloaded function "point operator+ (double & in, point & p) const", which allows non-member variables. To access private variables, we can add "friend" in the front.

8. (2 pts) What is the output of this code? Discuss your answer.

```
1. #include < iostream >
2. using namespace std;
3.
4. class CMyClass {
5. public:
6. static int m_i;
7. };
8.
9. int CMyClass::m_i = 0;
10.
11. CMyClass myObject1;
12. CMyClass myObject2;
13. CMyClass myObject3;
14.
15. int main() {
16. CMyClass::m_i = 2;
17. myObject1.m_i = 1;
18.
19. cout << myObject1.m_i << endl;
20. cout << myObject2.m_i << endl;
21.
22. myObject2.m_i = 3;
23. myObject3.m_i = 4;
24.
25. cout << myObject1.m_i << endl;
26. cout << myObject2.m_i << endl;
27. }
```

Answer: the output is:

1
1
4
4

Because m_i is static, all objects created will only have references to the same m_i variable
So when manipulating m_i, every objects' m_i changes