Operator Overloading

Chapter 11

Objectives

You will be able to

- Add overloaded operators, such as +,-, *, and / to your classes.
- Understand and use *friend* functions.

Operator Overloading

- Recall *function* overloading
- We can write multiple functions with the same name providing they have different signatures.
- Operators are just an alternative notation for function calls.
  - Each operator has an equivalent method.
  - We can define overloaded versions in our own classes.

Why bother with this?

- Permits more user-friendly versions of methods.
- Use + to add two objects
    - Where the concept of addition make sense.
- Consider + for string contatenation in C++ vs. strcat() in C.

> Operator for Circles

If c1 and c2 are Circle objects,

```
c1 > c2
```

compiles as

```
c1.operator>(c2)
```

Add to Circle.h

```
bool operator>(const Circle& other) const;
```

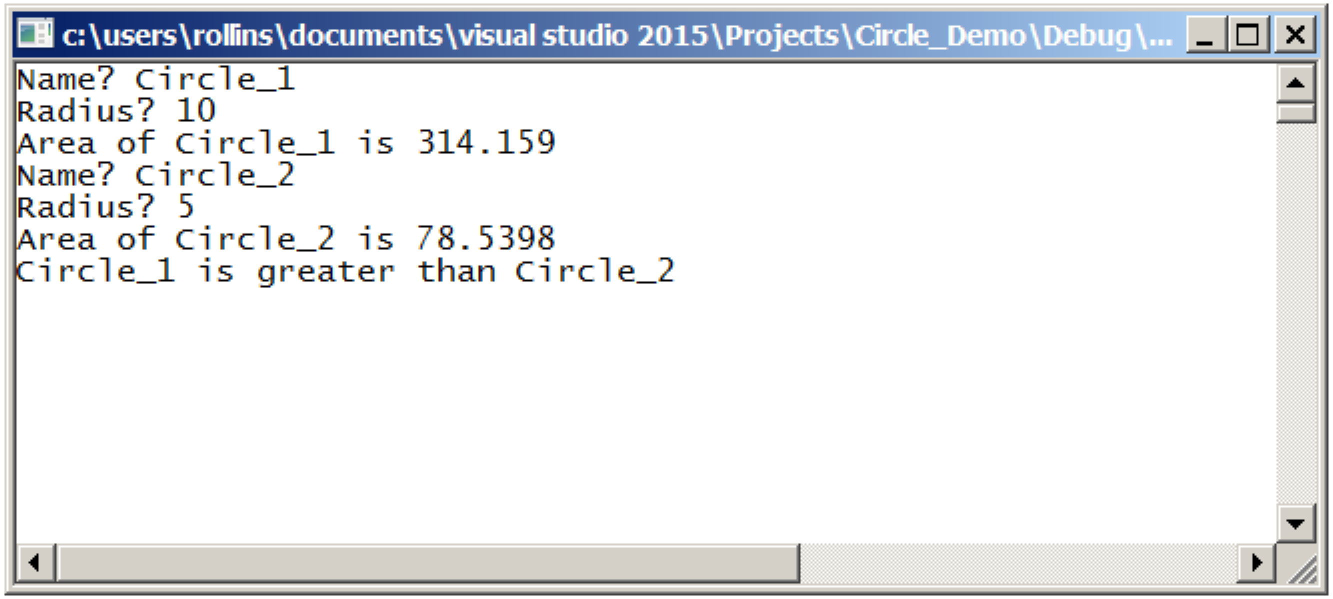
Add to Circle.cpp

```
bool Circle::operator>(const Circle& other) const
{
    return this->radius > other.radius;
}
```

Circles_Test.cpp

```
if (*c1 > *c2)
{
    cout << c1->Name() << " is greater than "
         << c2->Name() << endl;
}
else if (*c2 > *c1)
{
    cout << c2->Name() << " is greater than "
         << c1->Name() << endl;
}
else
{
    cout << c1->Name() << " and " << c2->Name()
         << " are the same size\n";
}
```

Works the Same


c:\users\rollins\documents\visual studio 2015\Projects\Circle_Demo\Debug\...

```
Name? Circle_1
Radius? 10
Area of Circle_1 is 314.159
Name? Circle_2
Radius? 5
Area of Circle_2 is 78.5398
Circle_1 is greater than Circle_2
```

Overloading the Insertion Operator

Let's look at overloading a different operator

- The "insertion" operator, <<

```
cout << myString;
```

- says to *insert* myString into the console output stream.

Overloading the Insertion Operator

- << works with all native C++ types.
  - Overloaded definitions for all native types are included in <iostream>

- What about our own classes?
  - Would like to be able to write

```
cout << my_circle << endl;
```

- If want it to work as expected, we have to provide a new overload of the << operator for that class:

```
void operator<<(ostream& os, const Circle& c);
```

- Cannot be a member function.   Why?

Friend Methods

- A class can declare a non-member function as a *friend*.
  - Function has the same access to class members as a member method.
  - The function is normally defined in the same cpp file as the member functions.
  - Effectively part of the interface published by the class.
  - Read about this in Chapter 11.

Circle.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;
class Circle
{
    private:
         double radius;
         string name;
     public:
          Circle(string Name, double Radius);
          …
          bool operator>(const Circle& other) const;

          friend void operator<<(ostream& os, const Circle& c);
};
```

Add to Circle.cpp

```
void operator<<(ostream& os, const Circle& c)
{
    os << c.name << " Radius " << c.radius;
}
```

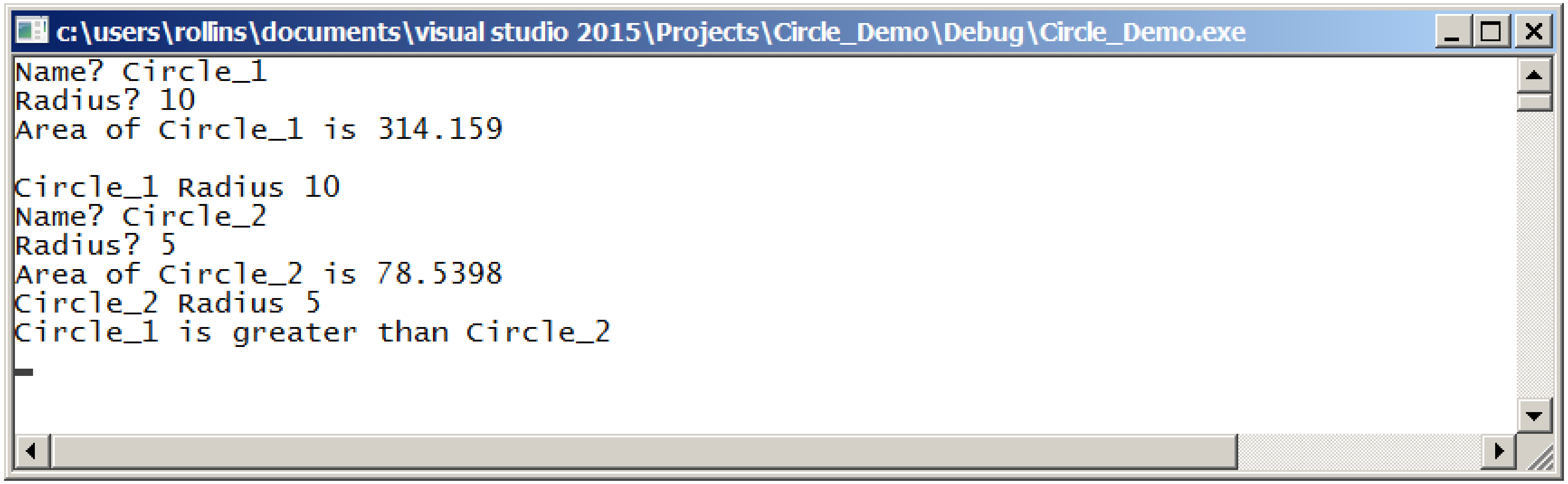
Note: NOT a member of class Circle.

No Circle::

In Circle_Test.cpp

```
int main()
{
    Circle* c1 = Create_Circle();
    double c1_area = c1->Area();
    cout << "Area of " << c1->Name() << " is " << c1_area << endl;
    cout << endl;
    cout << *c1;
    cout << endl;

    Circle* c2 = Create_Circle();
    double c2_area = c2->Area();
    cout << "Area of " << c2->Name() << " is " << c2_area << endl;
    cout << *c2;
    cout << endl;
```

Program Running


c:\users\rollins\documents\visual studio 2015\Projects\Circle_Demo\Debug\Circle_Demo.exe

```
Name? Circle_1
Radius? 10
Area of Circle_1 is 314.159

Circle_1 Radius 10
Name? Circle_2
Radius? 5
Area of Circle_2 is 78.5398
Circle_2 Radius 5
Circle_1 is greater than Circle_2
```

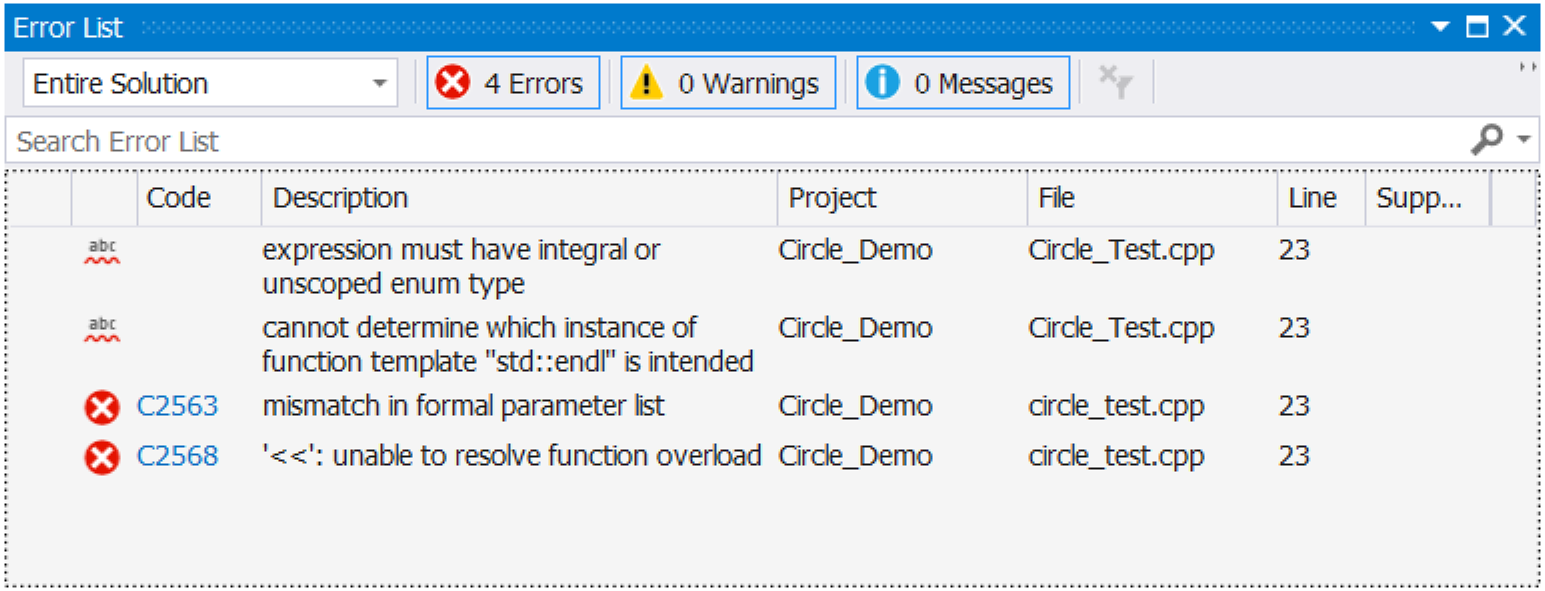
But there is a flaw

Suppose we want to put more output after c1

```
int main()
{
    Circle* c1 = Create_Circle();
    double c1_area = c1->Area();
    cout << "Area of " << c1->Name() << " is " << c1_area << endl;
    cout << endl;
    cout << *c1 << endl;
```

This doesn't work. Why?

Compile Time Errors



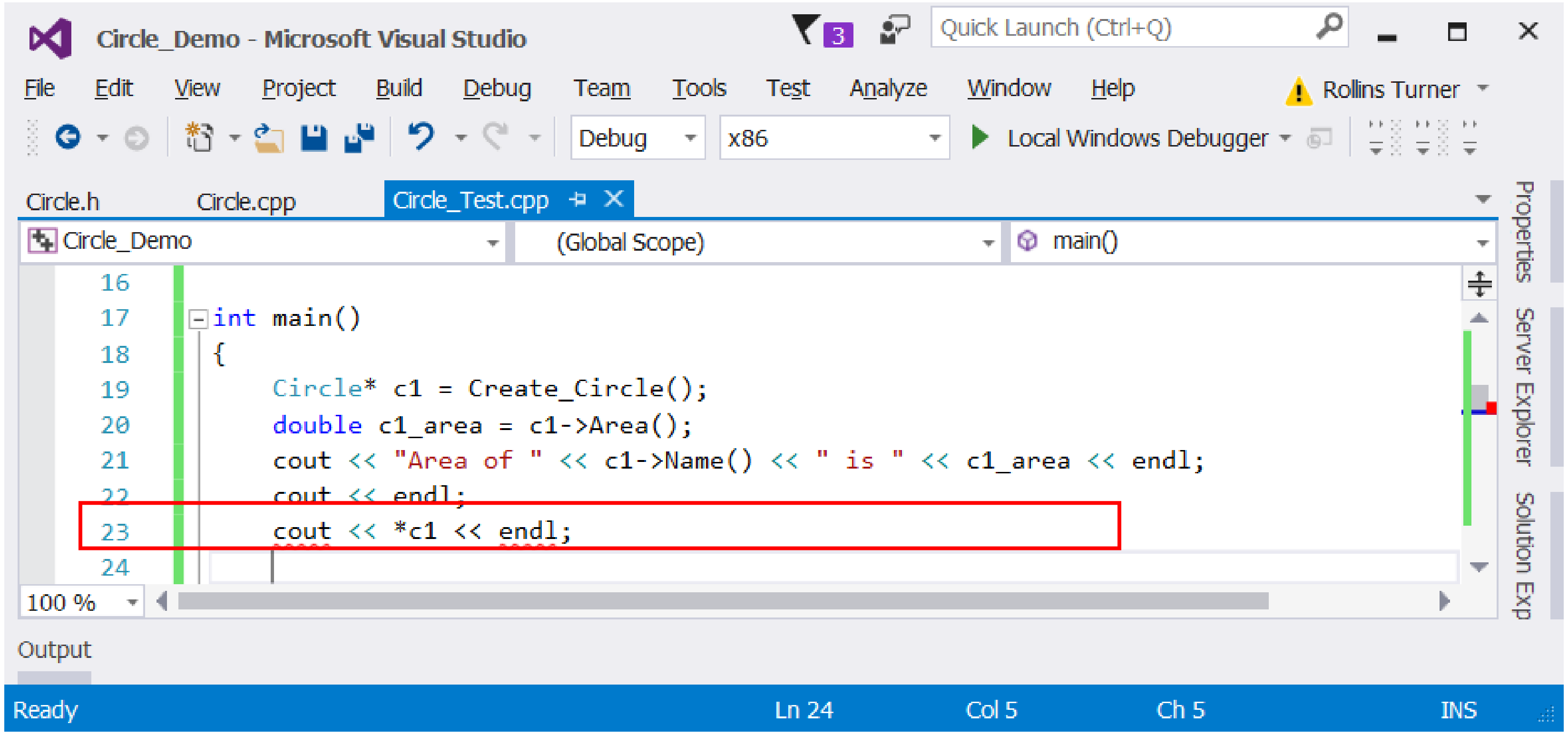
Error List

Entire Solution | 4 Errors | 0 Warnings | 0 Messages

Search Error List

	Code	Description	Project	File	Line	Supp...
		expression must have integral or unscoped enum type	Circle_Demo	Circle_Test.cpp	23	
		cannot determine which instance of function template "std::endl" is intended	Circle_Demo	Circle_Test.cpp	23	
	C2563	mismatch in formal parameter list	Circle_Demo	circle_test.cpp	23	
	C2568	'<<': unable to resolve function overload	Circle_Demo	circle_test.cpp	23	

The error line



```
int main()
{
    Circle* c1 = Create_Circle();
    double c1_area = c1->Area();
    cout << "Area of " << c1->Name() << " is " << c1_area << endl;
    cout << endl;
    cout << *c1 << endl;

```

The << operator requires an ostream object on the left side.

Our operator << is a void method.

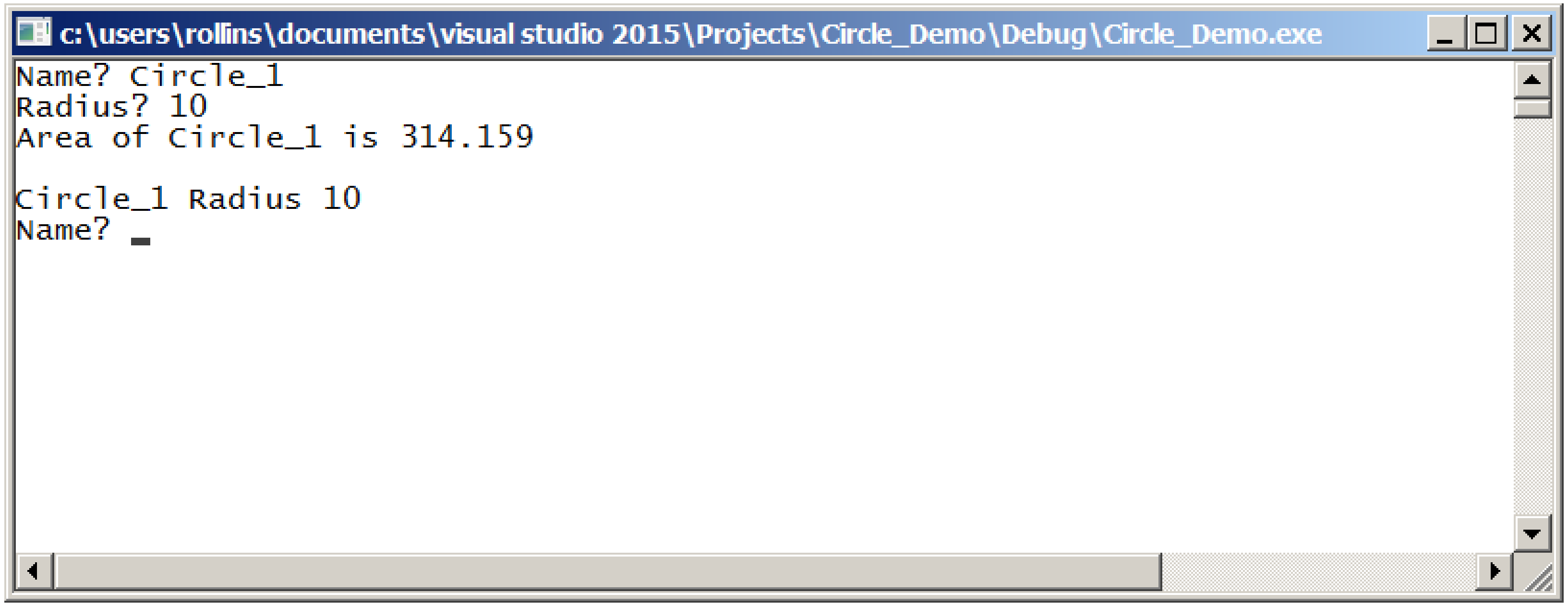
Correction in Circle.h

```
friend **ostream&** operator<<(ostream& os, const Circle& c);
```

Correction in Circle.cpp

```
ostream& operator<<(ostream& os, const Circle& c)
{
    os << c.name << " Radius " << c.radius;
    return os;
}
```

Now compiles and works


c:\users\rollins\documents\visual studio 2015\Projects\Circle_Demo\Debug\Circle_Demo.exe

```
Name? Circle_1
Radius? 10
Area of Circle_1 is 314.159

Circle_1 Radius 10
Name? _
```

An Alternative

- It didn't *have to be* a friend.
- If the << operator used accessor functions it would not need to be a friend.
- Move declaration outside class definition and remove “friend”.

Circle.h

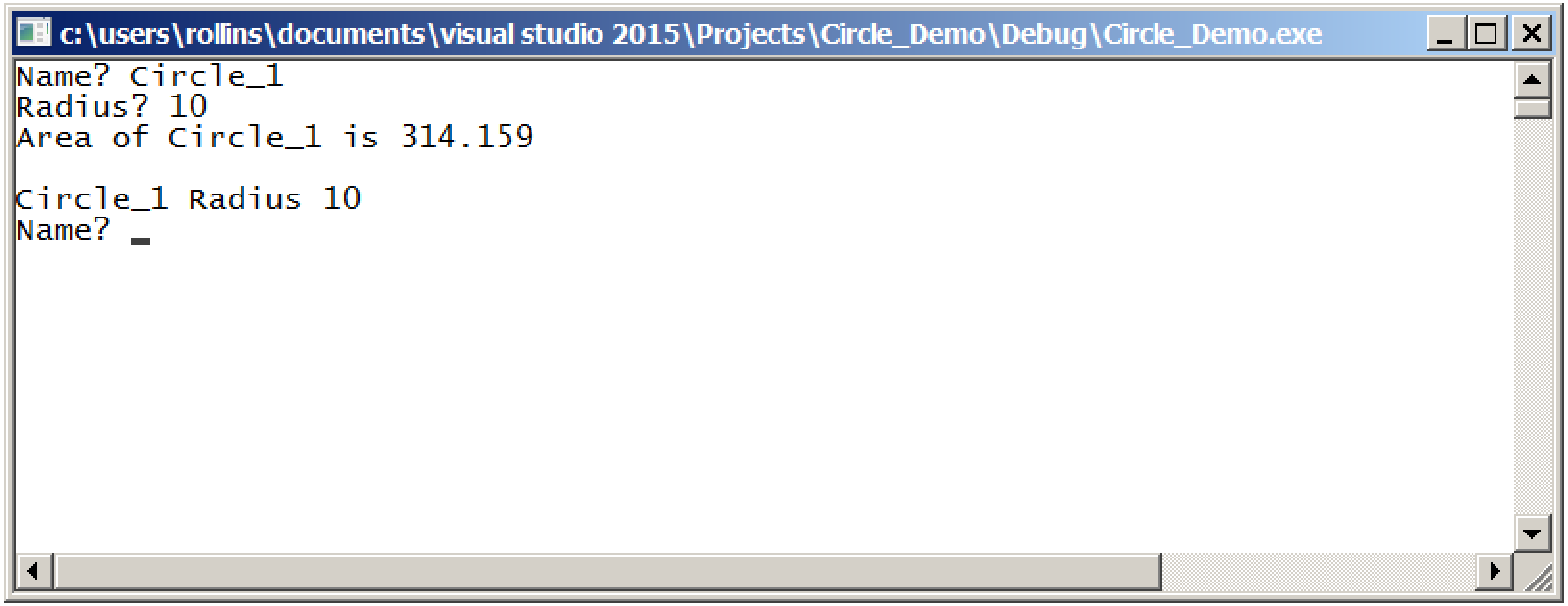
```
class Circle
{
    ...
    double Radius() const {return radius;};
    const char* Name() const {return name;};
    ...
};

ostream& operator<<(ostream& os, const Circle& c);
```

Circle.cpp

Now uses accessor methods rather than directly accessing member variables of class Circle.

```
ostream& operator<<(ostream& os, const Circle& c)
{
    os << c.Name() << " Radius " << c.Radius();
    return os;
}
```

Works the Same



c:\users\rollins\documents\visual studio 2015\Projects\Circle_Demo\Debug\Circle_Demo.exe

```
Name? Circle_1
Radius? 10
Area of Circle_1 is 314.159

Circle_1 Radius 10
Name? _
```

Summary

- Overloaded insertion operator, operator<<, should be defined with the class, but cannot be a member.
  - Could be a friend.
  - Could use accessor methods.