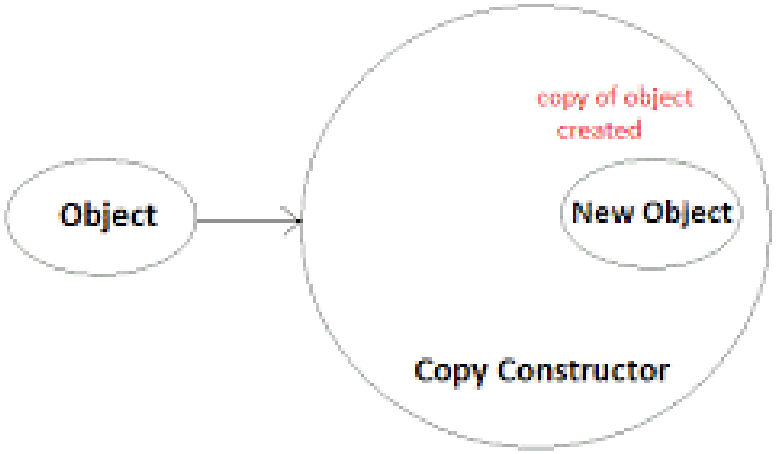
C/C++ Programming: C++ interm. (3/3)



Course **EE3093** – Lecturer: Dr F. Verdicchio

Any question?

Relevant topics to ask questions:

- C++ Objects:
  - Constructors; destructor; static members.
  - Copy constructor; overloading assignment operator; s
  - Constant reference arguments; “const” member function; “this” pointer

Operators applied to objects

For "simple" variables the behaviour of basic operators, such as " = " , " + " is straightforward (mostly – remember pointers?!) :

```
int x, y = 5;
x = y +2;
```

But what happens when those are applied to objects?

Let's start with " = "

(1) Initialize one (previously instantiated) object with content of another

```
testobj1=testobj0;
```

Solved by overloading ***operator =***

(2) Instantiate and initialize one object with content of another

```
rectangle testobj1=testobj0;
```

Solved by implementing a ***copy constructor***

Operators applied to objects

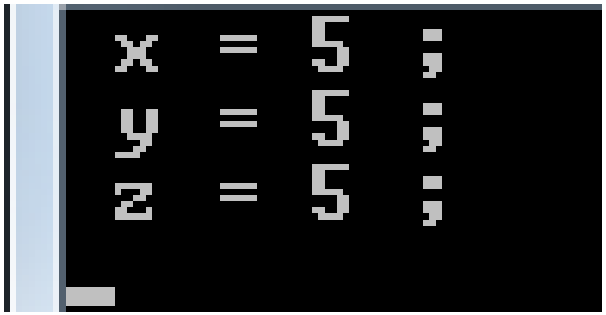
Note how this works (for integers):

```
int x, y, z = 5;
x = (y = z);
cout << " x = "<< x << " ;" << endl;
cout << " y = "<< y << " ;" << endl;
cout << " z = "<< z << " ;" << endl;
```

Operators applied to objects

Note how this works (for integers):

```
int x, y, z = 5;
x = (y = z);
cout << " x = "<< x << " ;" << endl;
cout << " y = "<< y << " ;" << endl;
cout << " z = "<< z << " ;" << endl;
```



Operators applied to objects

Note how this works (for integers):

int x, y, z = 5;

x = (y = z);

What happens here:

(a) **y set equal to z;**

(b) **the value of the assignment (y = z) is returned**;

(c) **x set equal** to this value **(x = y = z);**

Operators applied to objects

… and now for rectangles as well:

```
rectangle testobj0, testobj1, testobj2;
testobj2.inputSides(5,5);
testobj0=(testobj1=testobj2);
cout << " testobj0:"<< endl;
testobj0.printRectangleInfo();
cout << " testobj1:"<< endl;
testobj1.printRectangleInfo();
cout << " testobj2:"<< endl;
testobj2.printRectangleInfo();
```

Operators applied to objects

… and now for rectangles as well:

```
rectangle testobj0, testobj1, testobj2;
testobj2.inputSides(5,5);
testobj0=(testobj1=testobj2);
cout << " testobj0:"<< endl;
testobj0.printRectangleInfo();
cout << " testobj1:"<< endl;
testobj1.printRectangleInfo();
cout << " testobj2:"<< endl;
testobj2.printRectangleInfo();
```

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 5
Rectangle side B is: 5
Rectangle area is: 25
Rectangle perimeter is: 20
 testobj1:
Rectangle ID is: 1
Rectangle side A is: 5
Rectangle side B is: 5
Rectangle area is: 25
Rectangle perimeter is: 20
 testobj2:
Rectangle ID is: 2
Rectangle side A is: 5
Rectangle side B is: 5
Rectangle area is: 25
Rectangle perimeter is: 20
```

Operators applied to objects

… and now for rectangles as well:

testobj0=(testobj1=testobj2);

What happens here:

(a) **testobj1 set** to have **equal** values **to testobj2;**

```
rectangle& operator = (const rectangle& other)
{
    this->copyFrom(other);
    return *this;
}
```

Operators applied to objects

… and now for rectangles as well:

testobj0=(testobj1=testobj2);

What happens here:

(a) **testobj1 set** to have **equal** values **to testobj2;**

(b) the assignment returns **testobj1**;

```
rectangle& operator = (const rectangle& other)
 {
     this->copyFrom(other);
     return *this;
 }
```

Operators applied to objects

… and now for rectangles as well:

testobj0=(testobj1=testobj2);

What happens here:

(a) **testobj1 set** to have **equal** values **to testobj2;**

(b) the assignment returns **testobj1**;

(c) **testobj0 set** to have **equal** values **to testobj1;**

```
rectangle& operator = (const rectangle& other)
{
    this->copyFrom(other);
    return *this;
}
```

Any question?

Operators applied to objects

Now consider the following for “simple” variables :

```
int x, y = 5;
x += y;
```

How do we extend this to objects?

Increment one (previously instantiated) object with content of another

```
testobj1 += testobj0;
```

define ***operator +=***

Operators applied to objects

Now consider the following for “simple” variables :

int x=3; int y = 5;

x += y;

How do we extend this to objects?

Increment one (previously instantiated) object with content of another

testobj1 += testobj0;

define ***operator +=***

via a member function *addSidesFrom()*

Operators applied to objects: example

```
void test_more_operators()
{
    rectangle testobj0, testobj1;
    testobj0.inputSides(6,10);
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    testobj1.inputSides(50,20);
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    testobj0+=testobj1;
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void addSidesFrom(const rectangle& other)
{
    // if this object is not initialized:
    // just copy the other object (if initialized)
    if( !( isInitialized() ) )
        copyFrom(other);
    else
        if( other.isInitialized() )
        {   // both this and the other object are initialized:
            //copy the current sides (prior to reset)
            double tempA_this=getSide(1); double tempB_this=getSide(2);
            double tempA_other=other.getSide(1); double tempB_other=other.getSide(2);
            resetRectangle();// then add the two sides
            inputSides(tempA_this + tempA_other, tempB_this + tempB_other);
        }
}
rectangle& operator += (const rectangle& other)
{
    // add the other object to the curren one;
    addSidesFrom(other);
    // return the reference to the current object;
    return *this;
}
```

Operators applied to objects: example

```
void addSidesFrom(const rectangle& other)
{
    // if this object is not initialized:
    // just copy the other object (if initialized)
    if( !( isInitialized() ) )
        copyFrom(other);
    else
        if( other.isInitialized() )
        {   // both this and the other object are initialized:
            //copy the current sides (prior to reset)
            double tempA_this=getSide(1); double tempB_this=getSide(2);
            double tempA_other=other.getSide(1); double tempB_other=other.getSide(2);
            resetRectangle();// then add the two sides
            inputSides(tempA_this + tempA_other, tempB_this + tempB_other);
        }
}
rectangle& operator += (const rectangle& other)
{
    // add the other object to the curren one;
    addSidesFrom(other);
    // return the reference to the current object;
    return *this;
}
```

Operators applied to objects: example

```
void addSidesFrom(const rectangle& other)
{
    // if this object is not initialized:
    // just copy the other object (if initialized)
    if( !( isInitialized() ) )
        copyFrom(other);
    else
        if( other.isInitialized() )
        {   // both this and the other object are initialized:
            //copy the current sides (prior to reset)
            double tempA_this=getSide(1); double tempB_this=getSide(2);
            double tempA_other=other.getSide(1); double tempB_other=other.getSide(2);
            resetRectangle();// then add the two sides
            inputSides(tempA_this + tempA_other, tempB_this + tempB_other);
        }
}
rectangle& operator += (const rectangle& other)
{
    // add the other object to the curren one;
    addSidesFrom(other);
    // return the reference to the current object;
    return *this;
}
```

Operators applied to objects: example

```
void addSidesFrom(const rectangle& other)
{
    // if this object is not initialized:
    // just copy the other object (if initialized)
    if( !( isInitialized() ) )
        copyFrom(other);
    else
        if( other.isInitialized() )
        {   // both this and the other object are initialized:
            //copy the current sides (prior to reset)
            double tempA_this=getSide(1); double tempB_this=getSide(2);
            double tempA_other=other.getSide(1); double tempB_other=other.getSide(2);
            resetRectangle();// then add the two sides
            inputSides(tempA_this + tempA_other, tempB_this + tempB_other);
        }
}
rectangle& operator += (const rectangle& other)
{
    // add the other object to the curren one;
    addSidesFrom(other);
    // return the reference to the current object;
    return *this;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
    rectangle testobj0, testobj1;
    testobj0.inputSides(6,10);
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    testobj1.inputSides(50,20);
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    testobj0+=testobj1;
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
    rectangle testobj0, testobj1;
    testobj0.inputSides(6,10);
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
```

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 6
Rectangle side B is: 10
Rectangle area is: 60
Rectangle perimeter is: 32
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 1
```

```
    testobj0+=testobj1;
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
```

```
 testobj1:
Rectangle ID is: 1
Rectangle side A is: 50
Rectangle side B is: 20
Rectangle area is: 1000
Rectangle perimeter is: 140
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 2
```

```
    testobj1.inputSides(50,20);
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
```

O example

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 6
Rectangle side B is: 10
Rectangle area is: 60
Rectangle perimeter is: 32
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 1

 testobj1:
Rectangle ID is: 1
Rectangle side A is: 50
Rectangle side B is: 20
Rectangle area is: 1000
Rectangle perimeter is: 140
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 2

 testobj0:
Rectangle ID is: 0
Rectangle side A is: 56
Rectangle side B is: 30
Rectangle area is: 1680
Rectangle perimeter is: 172
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 2
```

```
void
{
    testobj0+=testobj1;
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects

Next the following for “simple” variables :

int x, y, z;

y = 10; z = 5;

x = y + z;

How do we extend this to objects?

Initialize one (previously instantiated) object with the sum of two nothers

testobj2 = testobj0 + testobj1 ;

define ***operator +***

Operators applied to objects: example

```
void test_more_operators()
{
    rectangle::printRectangleCount();

    rectangle testobj0, testobj1;
    testobj0.inputSides(6,10);
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    testobj1.inputSides(50,20);
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    rectangle testobj2 = testobj0 + testobj1;
    cout << " testobj2:"<< endl;
    testobj2.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
rectangle operator + (const rectangle& other)
 {
     // temp variable is created copying content from this object
     rectangle temp(*this);
     // temp is "incremented" with the other object
     temp += other;
     // return (by value) temp
     return temp;
 }
```

Operators applied to objects: example

```
void test_more_operators()
{
    rectangle::printRectangleCount();

    rectangle testobj0, testobj1;
    testobj0.inputSides(6,10);
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
```

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 6
Rectangle side B is: 10
Rectangle area is: 60
Rectangle perimeter is: 32
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 1
```

```
    testobj2.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
```

```
 testobj1:
Rectangle ID is: 1
Rectangle side A is: 50
Rectangle side B is: 20
Rectangle area is: 1000
Rectangle perimeter is: 140
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 2
 TOT currently Active Rectangles: 2
 TOT currently Initialized Rectangles: 2
```

```
    testobj1.inputSides(50,20);
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    rectangle testobj2 = testobj0 + testobj1;
    cout << " testobj2:"<< endl;
    testobj2.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
```

```
 ====> Element copied via constructor
 ====> Element copied via constructor
 testobj2:
Rectangle ID is: 3
Rectangle side A is: 56
Rectangle side B is: 30
Rectangle area is: 1680
Rectangle perimeter is: 172
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 4
 TOT currently Active Rectangles: 3
 TOT currently Initialized Rectangles: 3
```

```
    cout << " testobj1:" << endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    rectangle testobj2 = testobj0 + testobj1;
    cout << " testobj2:"<< endl;
    testobj2.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
```

```
 ====> Element copied via constructor
 ====> Element copied via constructor
 testobj2:
Rectangle ID is: 3
Rectangle side A is: 56
Rectangle side B is: 30
Rectangle area is: 1680
Rectangle perimeter is: 172
Total numbers for Rectangle instantiations:
 TOT instatntiated Rectangles (currently active or not): 4
 TOT currently Active Rectangles: 3
 TOT currently Initialized Rectangles: 3
```

```
    cout << " testobj1:" << endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    rectangle testobj2 = testobj0 + testobj1;
    cout << " testobj2:"<< endl;
    testobj2.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
void test_more_operators()
{
```

```
====> Element copied via constructor
====> Element copied via constructor
testobj2:
Rectangle ID is: 3
Rectangle side A is: 56
Rectangle side B is: 30
Rectangle area is: 1680
Rectangle perimeter is: 172
Total numbers for Rectangle instantiations:
TOT instatntiated Rectangles (currently active or not): 4
TOT currently Active Rectangles: 3
TOT currently Initialized Rectangles: 3
```

?

```
    cout << " testobj1:" << endl;
    testobj1.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;

    rectangle testobj2 = testobj0 + testobj1;
    cout << " testobj2:"<< endl;
    testobj2.printRectangleInfo();
    rectangle::printRectangleCount();
    cout<<endl;
}
```

Operators applied to objects: example

```
rectangle operator + (const rectangle& other)
{
    // temp variable is created copying content from this object
    rectangle temp(*this);
    // temp is "incremented" with the other object
    temp += other;
    // return (by value) temp
    return temp;
}
```

Any question?

Operators applied to objects

Finally consider the following for “simple” variables :

```
int x = 5;
int y = 3;
if(y < x)
   cout << “y smaller than x” << endl;
```

How do we extend this to objects?

Compare one object with another

testobj1 < testobj0;

define ***operator <***

via a member function *bool isAvgSideSmallerThan()*

Just for the sake of this example

Operators applied to objects: example

```
bool isAvgSideSmallerThan(const rectangle& other)
{
    bool result = false;
    // both objects need to be initialized to compare
    if( isInitialized() && other.isInitialized() )
    {
        double this_avg_side=(getSide(1) + getSide(2))/2;
        double other_avg_side=(other.getSide(1) + other.getSide(2))/2;
        if(this_avg_side < other_avg_side)
            result = true;
    }
    return result;
}
bool operator < (const rectangle& other) {return isAvgSideSmallerThan(other);}
```

Operators applied to objects: example

```
void test_more_operators_2()
{
    rectangle testobj0, testobj1;
    testobj0.inputSides(10,1);
    testobj1.inputSides(12,0.5);
    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    cout<<endl;
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    cout<<endl;
    if(testobj0<testobj1)
        cout << " testobj0 smaller than testobj1"<< endl;
    else
        cout << " testobj0 NOT smaller than testobj1"<< endl;
    cout<<endl;
}
```

Operators applied to objects: example

H:\fverdiccABDN\UniABDN\MyCourses\EE...

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 10
Rectangle side B is: 1
Rectangle area is: 10
Rectangle perimeter is: 22

 testobj1:
Rectangle ID is: 1
Rectangle side A is: 12
Rectangle side B is: 0.5
Rectangle area is: 6
Rectangle perimeter is: 25

 testobj0 smaller than testobj1
```

Any question?

Overloading Functions

In Cpp a member function, say *myFunction(int input)*, can be overloaded, i.e. we can implement a different version of the function that keeps the same name but accepts different input arguments *myFunction(double input1, int input2)*.

Overloading Functions

In Cpp a member function, say *myFunction(int input)*, can be overloaded, i.e. we can implement a different version of the function that keeps the same name but accepts different input arguments *myFunction(double input1, int input2)*.

A useful example of overloading is that of the **constructor**, where we can introduce a constructor with arguments, to create and initialize a rectangle (with given values for the sides) in one instruction:

Overloading the Constructor

.cpp | RectangleExample.h | PolygonArrayEample.h

ctangle | rectangle(c

```
    // initialization flag
    bool init_flag;
    // functions
    void computeArea() { ... }
    void computePerimeter() { ... }
    void set_init_flag(bool setval) { ... }
    void basicInitialization() { ... }
public:
    // constructor
    rectangle() {basicInitialization();}
    // overloaded constructor
    rectangle(double in_sideA, double in_sideB)
    {
        basicInitialization();
        inputSides(in_sideA, in_sideB);
    }
    // destructor
    ~rectangle() { ... }
```

Overloading the Constructor

```
void test_more_operators_3()
{
    rectangle testobj0;
    testobj0.inputSides(10,1);
    rectangle testobj1(6,10);

    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    cout<<endl;
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    cout<<endl;
}
```

Overloading the Constructor

```
void test_more_operators_3()
{
    rectangle testobj0;
    testobj0.inputSides(10,1);
    rectangle testobj1(6,10);

    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    cout<<endl;
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    cout<<endl;
}
```

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 10
Rectangle side B is: 1
Rectangle area is: 10
Rectangle perimeter is: 22

 testobj1:
Rectangle ID is: 1
Rectangle side A is: 6
Rectangle side B is: 10
Rectangle area is: 60
Rectangle perimeter is: 32
```

Overloading Operators

Like all member functions, operators can be overloaded; for example we can overload the operator “ < ” for the class rectangle;

Overloading Operators

Like all member functions, operators can be overloaded; for example we can overload the operator “ < ” for the class rectangle;

As usual, we first implement a member function (public or protected) as an intermediate step.
In this case we **overload** an **existing** (public) **member function**.

Overloading Operators

```
bool isAvgSideSmallerThan(const double& threshold) const
{
    bool result = false;
    // object needs to be initialized to compare
    if( isInitialized() )
    {
        double this_avg_side=(getSide(1) + getSide(2))/2;
        if(this_avg_side < threshold)
            result = true;
    }
    return result;
}
```

Overloading Operators

```
bool isAvgSideSmallerThan(const double& threshold) const
{
    bool result = false;
    // object needs to be initialized to compare
    if( isInitialized() )
    {
        double this_avg_side=(getSide(1) + getSide(2))/2;
        if(this_avg_side < threshold)
            result = true;
    }
    return result;
}
bool operator < (const double& threshold) const {return isAvgSideSmallerThan(threshold);}
```

Overloading Operators

```
void test_more_operators_3()
{
    rectangle testobj0;
    testobj0.inputSides(10,1);
    rectangle testobj1(6,10);

    cout << " testobj0:"<< endl;
    testobj0.printRectangleInfo();
    cout<<endl;
    cout << " testobj1:"<< endl;
    testobj1.printRectangleInfo();
    cout<<endl;

    double avgSideThreshold=5.8;

    if(testobj0 < avgSideThreshold)
        cout << " testobj0 avg side smaller than "<< avgSideThreshold << endl;
    else
        cout << " testobj0 avg side NOT smaller than "<< avgSideThreshold << endl;

    if(testobj1 < avgSideThreshold)
        cout << " testobj1 avg side smaller than "<< avgSideThreshold << endl;
    else
        cout << " testobj1 avg side NOT smaller than "<< avgSideThreshold << endl;

}
```

Overloading Operators

H:\fverdiccABDN\UniABDN\MyCourses\EE3093\Le...

```
 testobj0:
Rectangle ID is: 0
Rectangle side A is: 10
Rectangle side B is: 1
Rectangle area is: 10
Rectangle perimeter is: 22

 testobj1:
Rectangle ID is: 1
Rectangle side A is: 6
Rectangle side B is: 10
Rectangle area is: 60
Rectangle perimeter is: 32

 testobj0 avg side smaller than 5.8
 testobj1 avg side NOT smaller than 5.8
```

Any question?