Today

- Quiz 1
- Member function signatures

Assignments

- HW1 due
- HW2 out (due next Thurs)
Consider the following class definition

```cpp
class Rectangle
{
public:
    Rectangle();
    Rectangle(int l, int w);
    int area() const;
private:
    int length;
    int width;
};
```

Assume we want to add a `less_than` member function ...

- takes a rectangle object \( (\text{rhs}) \)
- returns true if the current object is “less than” the given \( (\text{rhs}) \) object
- where “less than” means a smaller area

(a). What are possible signature for `less_than`?

- Option 1: `bool less_than(Rectangle rhs) const;` ... copies \( \text{rhs} \)
- Option 2: `bool less_than(const Rectangle& rhs) const;`
- Which is “better” ... i.e., more efficient for parameter passing and why?

(b). What is a possible implementation for `less_than`?

```cpp
bool Rectangle::less_than(const Rectangle& rhs) const
{
    return area() < rhs.area();
}
```
Now assume we want to add a lesser_of member function ...

- takes a rectangle object (rhs)
- returns the current object if it is less than or equal to the rhs object
- returns the rhs object otherwise

(a). What are possible signatures for this function?

\[
\text{Rectangle} \ \text{lesser} \text{e}_{\text{of}}(\text{const Rectangle}& \ \text{rhs}) \ \text{const};
\]
\[
\text{Rectangle}& \ \text{lesser} \text{e}_{\text{of}}(\text{const Rectangle}& \ \text{rhs}) \ \text{const};
\]
- Which is better and why?

(b). What is a possible implementation?

\[
\text{Rectangle} \ & \ \text{Rectangle}::\text{lesser} \text{e}_{\text{of}}(\text{const Rectangle}& \ \text{rhs}) \ \text{const}
\{
\text{if} \ (\text{rhs}.\text{less} \text{e}_{\text{than}}(*\text{this}))
\text{\quad return rhs;}
\text{\quad return } *\text{this};
\}
\]

Note that without pass by reference (input param and return value) ...

Rectange \ r1(10, 10);
Rectangle \ r2(10, 20);
Rectange \ r = r1.\text{smaller} \text{e}_{\text{of}}(\text{r2});

- results in (potentially) 3 copies made!
- whereas with pass by reference, only one copy made (to assign to \ r)
- compiler may “optimize away” the last copy
When should we not return by reference?

When we would be returning a temporary variable ...

```cpp
Rectangle& double_length()
{
    Rectangle tmp = *this; // reuse copy constructor
    tmp.length *= 2;
    return tmp; // Q: why is this a bad idea?
}
```

- what happens to `tmp` after function exits?
- to make function const: `const Rectangle& double_length()` const

Also when we would be returning internal state ...

```cpp
int& get_length()
{
    return length; // Q: why is this a bad idea?
}
```

- what happens if the object is deleted/deallocated?
- to make function const: `const int& get_length()` const

We’ll talk more about signatures later ...