The `const` Mysterium

The keyword `const` can be used mostly everywhere

```
static const Rectangle* const createNew(const Rectangle* const rect){
 rect->x = 0; // Error: const Rectangle
 rect = &Rectangle(); // Error: Rectangle* const
 ...
}

int main(){
 Rectangle rect1;
 const Rectangle* const rect2 = Rectangle::createNew(&rect1);
 rect2->x = 0; // Error: const Rectangle

 Rectangle rect3;
 rect2 = &rect3; // Error: Rectangle* const
}
```

Read from left-to-right

The `const` Mysterium

1. What does `const int* p` mean?

   p is a pointer to a constant int, p can't change the value of the int it's pointing to

2. What's the difference between `const int* p` (1), `int* const p` (2) and `const int* const p` (3)?

   (2) p is a constant pointer to an int

   (3) p is a constant pointer to a constant int

3. What does `const int& x` mean?

   x is a reference to a constant int

Reference vs Pointer

- What is a reference? `int& r = i;`

An alias (an alternate name) for an object, In other words, the reference of an object is not a pointer, nor a copy, but the object itself (see call-by-reference)

Implementation detail:

- underneath it's all a reference to the object, calling `r++;` increments the value, `r` is kind of a macro like `(*i)`

- What's the difference between a reference and a pointer?
- What does `int& const x` mean?