```c
struct rectangle {
    int width;
    int height;
};

// INITALIZING STRUCT FIELDS

struct rectangle my_rect = {1,2}; // struct fields can be intialized immediately

// ACCESS STRUCT FIELDS
    // dot notation

my_rect.width; // returns 1
my_rect.height; // returns 20

// typedefs can be assigned to structs for convenience (and also can be done during struct definition)

typedef struct rectangle rect; // this is valid

int area(rect r) {
    return r.width * r.height;
}

typedef struct {
    int width;
    int height;
} rect; // this is also valid
```

Operators

```c
// ---------- OPERATOR ----------

// ARITHMETIC

int i1 = 1, i2 = 2; // valid shorthand for multiple declaration
float f1 = 1.0, f2 = 2.0; // same here as well

i1 + i2; // addition
i1 - i2; // subtraction
i1 * i2; // multiplication
i1 / i2; // division, though in this case evaluates to 0.5 is truncated towards 0
11 % 3; // modulo, be careful when arguments are negative though

int j = 0;
int s = j++; // increment by 1 operator, returns j then increments it
int z = ++j; // increment by 1 operator, increments j then returns it
int e = j--; // decrement by 1 operator, decrements j then returns it
int f = --j; // decrement by 1 operator, decrements j then returns it

(float)i1/i2; // evaluates to 0.5f since we need to cast at least one integer to a float to get a floating-point result
i1/(double)i2; // does the same for doubles
f1 / f2; // evaluates to 0.5 since both are floats here so fulfills the above requirement of at least one operand being a float

// COMPARISON

3 == 2; // complete equality in value and type, returns 0
```