In Java, you create new data types by creating classes.
- Classes have member variables and associated methods.
- You can control access, and inherit.

C has structs.
- Member variables only (no methods)
- No means of hiding information (public/private)

22-1: Structs example

typedef struct {
    char name[80];
    int id;
    char DOB[80];
} Person;

Notice:
- typedef - this declares a new type, which is a struct.
- The name of the new type is after the definition
- Ends with a semicolon.

22-2: Using a struct

Modifying a struct looks much like working with public member variables in Java.

```c
int main(void) {
    Person pers;
    strcpy(pers.name, "bob");
    pers.id = 12345;
    printf("%s %d", pers.name, pers.id);
}
```

22-3: Exercise 1

Write a Point struct.
- It should have x and y variables
- Write a main method that prompts the user for two points, then calculates the distance between them.
- \( d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \)

22-4: Structs as parameters

Like everything else in C, structs are passed by value.

```c
void setName(Person p, char *n) {
    strcpy(p.name, n);
}

int main(void) {
    Person pers;
    strcpy(pers.name, "bob");
    printf("%s", pers.name);
    setName(pers, "richard");
    printf("%s", pers.name);
}
```
22-5: Passing structs by reference

Like other data types, we can pass a pointer to a struct:

```c
void setName(Person *p, char *n) {
    ...
}
```

```c
int main(void) {
    Person pers;
    strcpy(pers.name, "bob");
    printf("%s", pers.name);
    setName(&pers, "richard");
    printf("%s", pers.name);
}
```

22-6: Passing structs by reference

How do we refer to the fields of a pointer to a struct?

- `*p.name` won't work
  - `.` has higher precedence than `*`
- we could do `(*p).name`, but that's awkward.
  - `strcpy(("p).name, "bob")`;

22-7: Passing structs by reference

C has a special operator to deal with this problem: `->`

- denotes fields of pointers to structs.

```c
void setName(Person *p, char *n) {
    strcpy(p->name, n);
}
```

22-8: Exercise 2

Make a file called Point.h that contains the definition of your Point struct.

- Inside Point.c, define setters and getters for the Point's x and y variables.
  - They'll need to take a pointer to a Point as an argument
- Place your main method from exercise 1 in a file called distance.c and compile all of them together like so:
  - `gcc -o distance distance.c Point.c -lm`

22-9: Arrays of Structs

We can store structs in arrays, just like any other data type.

- We need to tell malloc to allocate enough memory for the appropriate number of structs.

```c
int nelements = 10;
Person *parray = (Person *)malloc(nelements * sizeof(Person));
```

22-10: Random numbers: a digression

How can we create random integers in C?

- The function `rand()` returns numbers between 0 and MAXINT.
- We can use modular arithmetic to reduce this range
  - `rand() % 10` returns numbers between 0 and 9.
- Addition can be used to shift the endpoints.
  - `rand() % 10 + 5` returns numbers between 5 and 14.
Random numbers: a digression

What if we want to generate random floating point numbers?

Say we want random floats between 0 and 10, with two decimal places?

We can generate random integers from a larger range, and then divide by the appropriate power of 10.

\[ \text{double randomDouble} = (\text{rand()} \% 1000) / 100.0; \]

/* this will give a decimal with two places of precision.

Random numbers: a digression

We can generalize this by using the pow() function.

\[ \text{pow(number, exponent)} \]

Exercise 3

Write a program that prompts the user for:
- A min
- a max
- a number of random numbers to generate.

and then generates that many floating point numbers between min and max.

Run your program multiple times. What do you notice?

Seeds

Your program should’ve generated the same numbers each time you provided the same parameters.

That’s not very random!

Computers are actually quite bad at doing truly random things.

Usually, we want them to produce predictable results.

Seeds

Random number generators actually generate a deterministic sequence of numbers.

If the sequence is long enough and well-distributed, it looks random from our point of view.

We call this a pseudorandom sequence.

Given a particular starting number, the generator produces a long string of numbers.

In the last program, we were always starting from the same point! (0)

Problem: How do we specify a starting point?
22-17: Seeds

- The starting point for a RNG is called a seed.
- We can set the seed with the function `srand(int s)`
- Where can we get a seed that will be different every time?

22-18: Seeds

- The starting point for a RNG is called a seed.
- Where can we get a seed that will be different every time?
- We can use the system clock.
- C provides us access to this through a function called `time`.
- `srand(time(NULL))`

22-19: Exercise 4

- Write a program that:
  - Prompts the user for a number of points.
  - Allocates an array with that many points in it.
  - Sets random values for the x and y values for each point using your getter and setter methods.
  - For each point in the array, find the point in the array that is closest to it and print out both points’ x and y values.