Structs in C

→ we started out by considering primitive data types (int, float, double, char)

→ we looked at a more complex data type → arrays → lists of primitive types

→ we considered pointers → addresses of primitive types, or the elements of arrays.

Question → what if we want to create a data structure that is a mix of different types?

Answer → Enter C structs !

(Note: structs were introduced into C in an attempt to progress towards

object-oriented programming!)

When we write things like:

```
int i;
double x;
char a;
```

the words "int", "double", and "char" are keywords that are recognized by the compiler ~> English "go set up a place in memory for an object of type (int, double, char, ...)"

Together, these keywords are known as **typedef** statements.

What we want to do, now, is create a new typedef, that ... to our mixed structure.

Courses pass

Example: Simple Car

just like int, double, ...

SimpleCar a ;

(this would create a struct, in memory, of type SimpleCar)

What variables might we like?

total # of miles driven → int miles ;

age in years → double age ;

make → char make [20] ;

model → char model [20] ;

Implementation:

typedef struct SimpleCar_struct {

int miles ;

} SimpleCar_type

defines SimpleCar

```
    double age;
    char make[20];
    char model[20];
};
```

SimpleCar;

⇨ SimpleCar a;

instantiates (creates) a SimpleCar Object in memory!

```
a.miles = 97;
a.age = 0.2;
strcpy(a.make, "Toyota");
strcpy(a.model, "Camry");
```

Aside: Algebra

→ from Arabic ~ al gebra ~ "the method"

→ A set of rules for how to perform

addition, subtraction, mult., div.,
exponents, abs. values, ...
for that type of number.
(algebra of reals ≠ algebra of vectors...)

→ in C, there is already code that specifies the algebra of int, double, char, ...

BUT: No code exists for our newly created type (SimpleCar)

We must provide this!

What sort of things should we provide?

(+, −, ×, ÷) don't even have meanings for cars, do they? Our "algebra" contains different operations / functions!

1. Initialization → we should always provide this.

InitCar()

→ takes no parameters, returns a object of type SimpleCar.

Ex. `SimpleCar a = InitCar();`

takes no parameters

`SimpleCar InitCar() {`

returns a SimpleCar

```
SimpleCar newCar;
newCar.miles = 0;
newCar.age = 0.0;
strcpy(newCar.make, "Toyota");
strcpy(newCar.model, "Camry");

return newCar;
}
```

---

2. Setter and Getter Methods.

→ It is used to provide methods to both SeA

and get

every variable in the

structure.

→ we don't want the users manipulating the structures themselves... If we change the structure, that should be transparent to the user!!

Ex.

int GetOdometer(SimpleCar car);

(we pass a car, it returns that car's mileage)

SimpleCar SetOdometer(SimpleCar car, int mileage);

(we pass a car, and a mileage, and it sets that variable)

Then returns the updated object!!

Ex 2:

```
char *  GetMake( SimpleCar car );
```

↳ returns a pointer to a char (first character of string)

```
SimpleCar  SetMake( SimpleCar car,
                    char * make );
```

↑ returns a new SimpleCar object with updated make.

↑ pass in a pointer to a char

---

3. Complex Methods (which use setter and getter methods!)

```
SimpleCar Drive( int dist, SimpleCar car );
```

↑ returns updated car

↑ drives this many miles

↑ this car

SimpleCar Reverse (int dist, SimpleCar car);

void HonkHorn (SimpleCar car);
(just a print statement)

void Report (SimpleCar car);
(more print statements about
internal variables)

Summary:

① structs allow us to make complex data objects!

② We have to provide all of the code for all of the algebra of the new structure!

-> initializators
-> getter methods

→ setter / ge...

→ complex functions

③ When we pass a struct as the argument of a function, it makes a copy of that struct within the function !!!

→ if we really want to modify the struct in the function, we need to remember this !

Super important !!

mystruct = myfunction(mystruct, ...)

④ Convention :

(i) put the struct definition and the function prototypes in a header file :

SimpleCar.h

(ii) put the code for

Setter, getter, init, and other functions in a separate C file:

SimpleCar.c

(iii) <u>Include</u> the header file in <u>both</u> main.c and SimpleCar.C.

<u>Code structure</u>:

SimpleCar.h
SimpleCar.c
main.c

BIG QUESTION: How do we compile these files into a single executable???

Next Class: Makefile Tutorial

Next [illegible]