10: C – Structs and Unions

Intro

A struct is a collection of different types of data. It's like an (OOP) object.

A union is used to create a space in memory that can store data of different types at different times. E.g. a union of a `char` and an `int` can hold either a `char` or an `int`. You need to remember what's in there to interpret the type correctly.

Structs

There may be gaps between elements in a structure.

Structures represent user-defined data types.

Most commonly a `typedef` name is defined:

```
typedef struct {
    […]
}
```

Initialising Structures

You can use array initialisation syntax:

```
VITALSTAT vs = {"Joe Smith", 0982459180, 3, 5, 1975};
```

Referencing Members

Two methods, depending on whether you have the structure or a pointer to it:

```
/* With the structure */
vs.month = 3;
vs.day = 15;
```

```c
/* With a pointer */
pvs->month = 3;
pvs->day = 15;
```

The arrow operator is shorthand for dereferencing the pointer and using the dot operator:

```c
(*pvs).day = 15;
```

Self-Referencing Structs

You can't have instances of a structure inside itself, but you can have pointers to instances. This allows you to create e.g. linked lists or trees.

Relevance to Project

Say you have a series of pins, and each pin has a zone. You can define structs for each pin which define the type of the zone.