Day 11 Implementing Structures, Unions, and TypeDefs

structure- collection of one or more variables grouped under a single name. can be of different data types. can hold arrays, pointers, and other structures

variable within structure called **member**

```
struct tag {
members;
} instance1, instance2;
```

tag name of structure template, use instances to declare a version of that template

member operator- (.) used to refer to a member of a instance

to copy info from one instance to another: instance1 = instance2. same thing as instance1.x = instance2.x; instance1.y = instance2.y

if you just define without instance can declare later with:

```
struct tag instancename;
```

Ex: struct time { int hours; int minutes; } time_of_birth = { 8, 45 };

using structures that are more complex

```
struct rectable {
    struct coord topleft;
    struct coord bottomrt;
} mybox;
```

struct coord just x and y previously defined- now describing a shape

to change the coordinates of x and y of top left would need to member incept:

```
mybox.topleft.x = 0
mybox.topleft.y = 10
```

structures that contain arrays

```
record.x[2] = 100
record.y[1] = 'x'
```

pointer to first element of array y: record.y

Arrays of Structures

if you already have structure type entry- to declare array of entry structures:

```
struct entry list[1000]
```

```
list[0] has:
list[0].fname
list[0].lname

list[1] has:
list[1].fname
list[1].lname
....
```

assign one array element to another:

```
list[1] = list[5]
```

move data between members:

```
strcpy(list[1].phone, list[5].phone);
```

Initializing Structures

initialize and declare instance:

```
struct sale {
    struct customer buyer;
    char item[20];
    float amount;
} mySale = { "Acme", "left hand widget", 1000.00 };

struct sale 1990[100] {
{ fill in, for , index[0] }
{ fill in, for, index[1]  }
```

Structures and Pointers

can use pointers as structure members, can also declare pointers to structures

Declare

```
struct data {
    int *value;
} first;
```

Initialize

```
first.value = &cost;
```

Creating Pointers to Structures

used when passing a structure as an argument to a function

```
1. struct part \*p_part;
2. struct part gizmo;
3. p_part = \&gizmo;
```

must declare a pointer to the structure tag, then link pointer to a particular instance

```
(*p_part).number = 100;
```

could also use the **indirect membership operator**

```
p_part->member;
```

Pointers and Arrays of Structures

```
struct part data[100];
struct part *p_part;
p_part = &data[0] OR p_part = data;
```

points at first structure in array of structures

can use normal pointer arithmetic to move through array of structures

can pass structures to functions using instance or pointer to structure
if pass pointer to structure as argument, must use -> to access members

Unions

only one member of union can be used at a time. all members of union occupy same address of memory

instances can hold one variable type OR another variable type

only one member can be initialized at a time

pg. 279 listing 11.8 uses the type variable in structure as almost a sub instance to assign information to the union:

```
struct tag {
    char type;
    union shared_tag{
        char c;
        char i;
```

```
    } shared;
} tag_instance;
```

Typedef

can use typedef when declaring structure so don't have to always use struct: