Structures

Derrick

Concepts

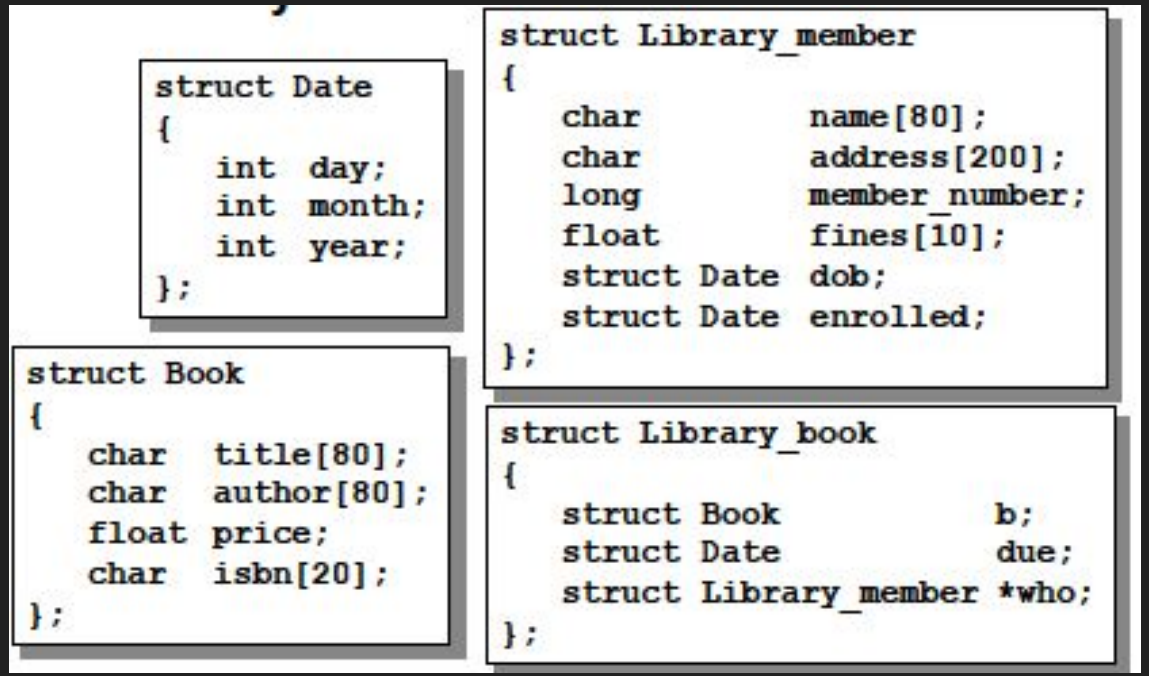
- A ***structure*** is a collection of one or more variables grouped together under a single variable.
- The variables in a structure are called ***members*** and may have any type, including arrays or other structures.

Steps

- Set-up template (blueprint) to tell the compiler how to build the structure
- Use the template to create as many instances of the structure as needed
- Access the members of an instance as needed

Setting up the template

- Structure templates are created by using the `struct` keyword.



```
struct Date
{
    int  day;
    int  month;
    int  year;
};
```

```
struct Library_member
{
    char         name[80];
    char         address[200];
    long         member_number;
    float        fines[10];
    struct Date  dob;
    struct Date  enrolled;
};
```

```
struct Book
{
    char   title[80];
    char   author[80];
    float  price;
    char   isbn[20];
};
```

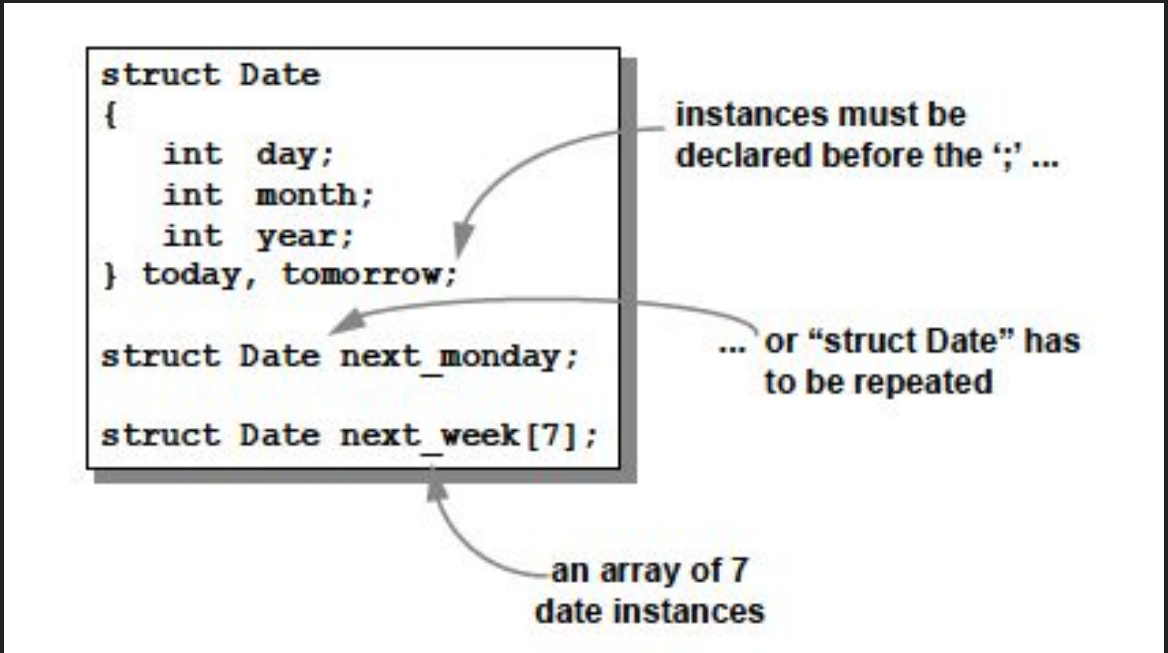
```
struct Library_book
{
    struct Book           b;
    struct Date           due;
    struct Library_member *who;
};
```

Structures vs Arrays

- Structures can have different data types as **members** whereas arrays have the same data type **elements**.
- Creating structure templates do not allocate any memory space whereas arrays allocate memory space based on a fixed size.
- Structures are values types whereas arrays are pointers.

Creating instances

- Having created the template, an instance (or instances) of the structure may be declared as:

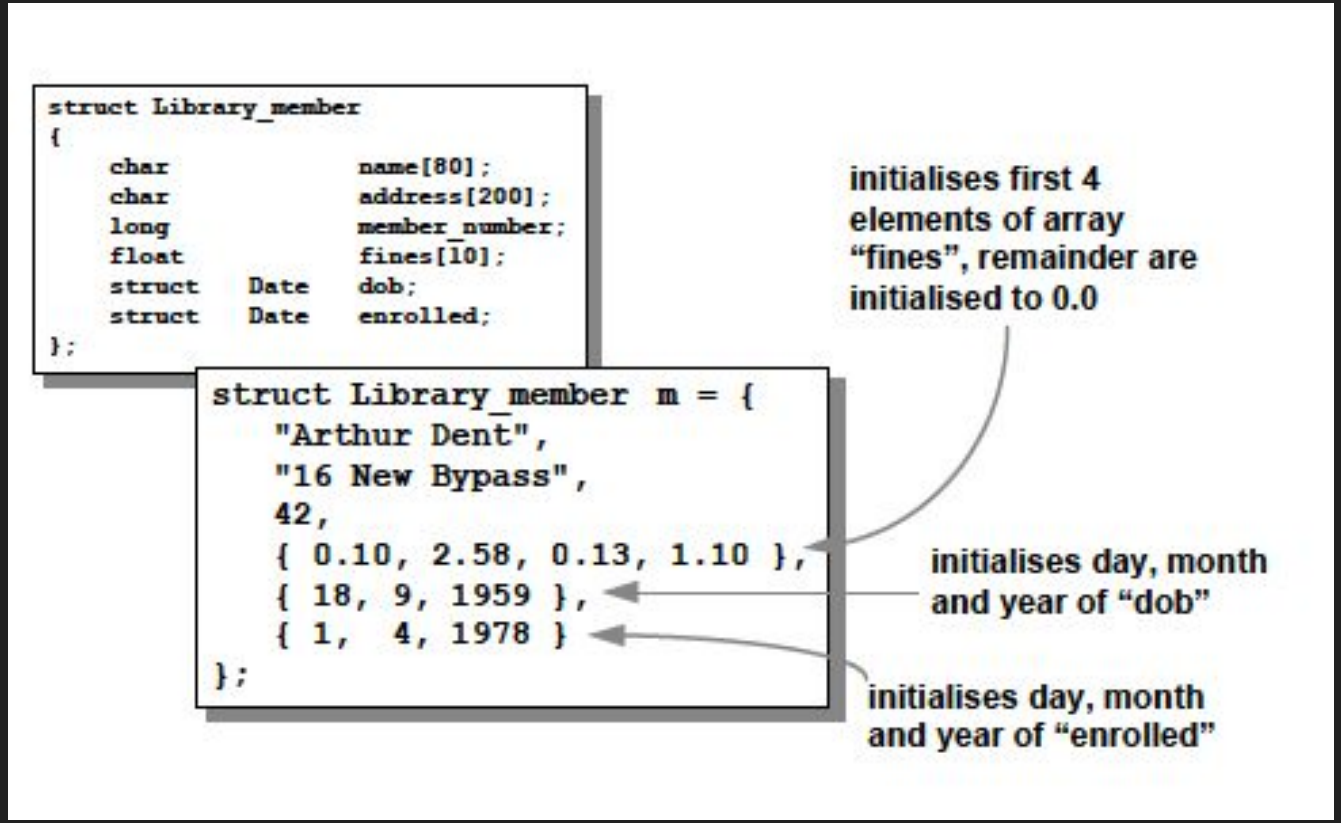


Initializing instances

- Structure instances may be initialized using braces (as with arrays)

```
struct Date next_monday;

next_monday.day = 13;

next_monday.month = 8;

next_monday.year = 2018;

// or

next_monday = { 13, 8, 2018 };
```

Structures within structures



Accessing members

- Members are accessed using the instance name, “.” and the member name.

```
struct Library_member m;

m.name;

m.fines[0];

m.dob.month;
```

Structure == value type

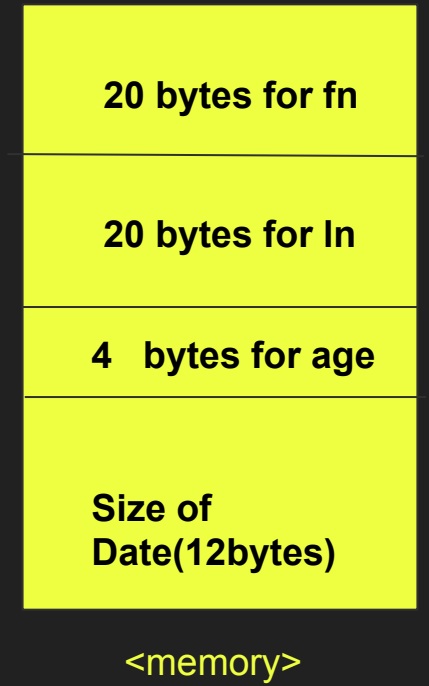
- Structures are value type. (not reference type)
- You can also create a pointer to a structure.
  - `struct Library_member *pt;`

Instances can be assigned

- Two structure instances may be assigned to one another via “=”
- All the members of the instance are ***copied*** (including `arrays` or other `structs`)
  - `struct Library_member tmp;`
  - `tmp = m;`
- !Remember! It is not possible to assign arrays in C
  - `int a[10];`
  - `int b[10];`
  - `a = b; /* a is a constant pointer */`

Visual Model (struct)

```
struct User {
    char        firstname[20];
    char        lastname[20];
    int         age;
    struct Date dob;
}
```



Passing Instances to Functions

- An instance of a structure may be passed to a function by value or by pointer
- Pass by value becomes less and less efficient as the structure size increases
- Pass by pointer remains efficient regardless of the structure size

```
void by_value(struct Library_member);

void by_ref(struct Library_member *);

by_value(m); /* compiler writes 300+ bytes onto the stack */

by_ref(&m); /* compiler writes a pointer (8 bytes) */
```

Pointers to Struct

- Passing pointers to struct instances is more efficient
- Dealing with a pointer to an instance of struct is not so straightforward!

```
void display_name(struct Library_member *p)

{

    printf(“name = %s\n”, (*p).name);

}
```

Why (*p).name?

- The messy syntax is need because “.” has higher precedence than “*”, thus:
  - `*p.name`
  - Means what `p.name` points to
- Since pointers and structs are being used frequently there is a new operator
  - `p->name == (*p).name`

Using p->name

```
void display_name(struct Library_member *p)
{
    printf(“name = %s\n”, p->name);
}
```

Pass by Reference

- Although pass by reference is more efficient, the function can alter the structure
- Use a pointer to a constant structure instead

```
void display_name(struct Library_member *p)

void display_name(const struct Library_member *pt)

/* pt is a read-only pointer */
```

Returning Structure instances

- Structure instances may be returned by value from functions
- This can be as inefficient as with pass by value (sometimes convenient)
- `struct Point { int x; int y; };`

```
struct Point add(struct Point p1, struct Point p2) {
    struct Point result;
    result.x = p1.x + p2.x;
    result.y = p1.y + p2.y;
    return result;
}
```

Returning a pointer to a struct (Be careful)

```
struct Point *add(struct Point p1, struct Point p2) {
    struct Point result;
    result.x = p1.x + p2.x;
    result.y = p1.y + p2.y;
    return &result;
}
```