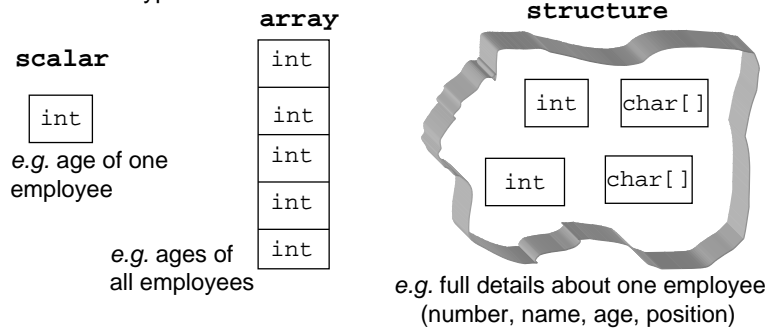


Structures

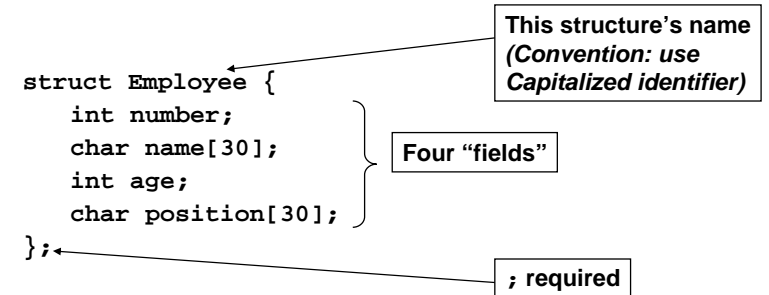
- A **scalar** holds a **single value** of a **single type**
- An **array** holds **several values** of a **single type**
- It is sometimes convenient to be able to group together items of information which are of **different types**
 - Example: an employee's **number**, **name**, **age** and **position** are logically related, but these pieces of data are of different types and so are not suitable for storage in an array
- A **structure** allows the programmer to group together data items of different types



1

Defining Structures

- To define a structure we must create a **structure template**:



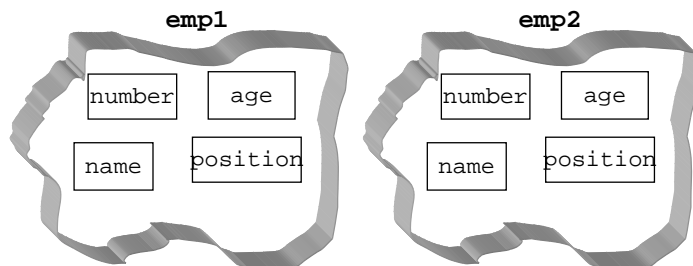
- **Does not allocate any memory** or define a new variable
- Just **defines a new datatype**, called **Employee**

2

- Now that we've defined the new datatype, we can define variables to be of type **Employee** (just as we can define variables to be of type **int**, **char**, **float**, ...):

```
struct Employee emp1, emp2;
```

- This declares two variables, **emp1** and **emp2**, to be of type **Employee**

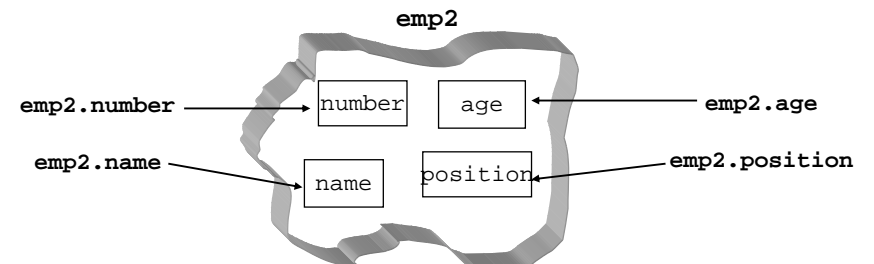


- Each of these structure variables has four fields: **number**, **name**, **age** and **position**

3

- In order to access structure fields, use the **field selection operator**

```
emp1.number = 4321;
emp1.age = 34;
strcpy(emp1.name, "John Smith");
printf("what is %d's position:", emp1.number);
scanf("%s", emp1.position);
```
- Each member can be treated just like any other value of that datatype: can use string functions on **name** and **position**, can use **age** and **number** for arithmetic, etc.
 - Similar to arrays: if **a[]** is an array of **floats**, then **a[i]** can be treated just like a "normal" **float** for any valid subscript **i**



4

- If we wanted to print out the contents of this structure we could do it as follows:

```
printf("employee number: %d\n", emp2.number);
printf("\tname: %s\n", emp2.name); /* \t means tab */
printf("\tage: %d\n", emp2.age);
printf("\tposition: %s\n", emp2.position);
```

- Often a good idea to define a function to print out structures of this type (saves you rewriting this over and over; makes your program easier to understand, improve, debug, ...):

```
void printEmployee (struct Employee e){
    printf("employee number: %d\n", e.number);
    printf("\tname: %s\n", e.name);
    printf("\tage: %d\n", e.age);
    printf("\tposition: %s\n", e.position);
}
```

- Note: `printEmployee()` has to know what an `Employee` structure is...

```
#include <stdio.h>
struct Employee {
    int number;
    char name[30];
    int age;
    char position[30];
};
void printEmployee (struct Employee e){
    printf("employee number: %d\n", e.number);
    printf("\tname: %s\n", e.name);
    printf("\tage: %d\n", e.age);
    printf("\tposition: %s\n", e.position);
}
main(){
    struct Employee emp;
    emp.number = 4321;
    emp.age = 34;
    strcpy(emp.name, "John Smith");
    printf("what is %d's position:", emp.number);
    scanf("%s", emp.position);
    printEmployee(emp);
}
```

Produces the screen output:
 what is 4321's position:lecturer
 employee number: 4321
 name: John Smith
 age: 34
 position: lecturer

Note: this is an example of a *call-by-value* function call

```
#include <stdio.h>
struct Employee {
    int number;
    char name[30];
    int age;
    char position[30];
};
void zapEmployee (struct Employee *pe){
    (*pe).number=-1;
    strcpy((*pe).name, "invalid");
    (*pe).age=-1;
    strcpy((*pe).position, "invalid");
}
void printEmployee (struct Employee e){
    printf("employee number: %d\n", e.number);
    printf("\tname: %s\n", e.name);
    printf("\tage: %d\n", e.age);
    printf("\tposition: %s\n", e.position);
}
main(){
    struct Employee emp;
    emp.number = 4321;
    emp.age = 34;
    strcpy(emp.name, "John Smith");
    printf("what is %d's position:", emp.number);
    scanf("%s", emp.position);
    printEmployee(emp);
    zapEmployee(&emp);
    printEmployee(emp);
}
```

Produces the screen output:
 what is 4321's position:lecturer
 employee number: 4321
 name: John Smith
 age: 34
 position: lecturer
 employee number: -1
 name: invalid
 age: -1
 position: invalid

`pe` is a *pointer* to a structure of type `Employee`

Note: this is an example of a *call-by-address* function call

Pointers to Structures: A shorthand

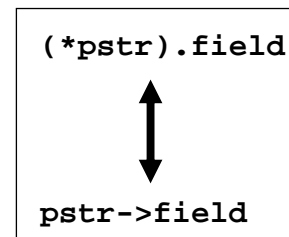
Expressions like this are used a lot:

`(*pe).number`

C has a shorthand notation for this:

`pe->number`

Or more generally, if `pstr` is a pointer to a structure, and `field` is one of that structure's fields, then these two mean the same thing:



EXAMPLE:

```
void zapEmployee (struct Employee *pe){
    pe->number = -1;
    strcpy(pe->name, "invalid");
    pe->age = -1;
    strcpy(pe->position, "invalid");
}
```

Nested Structures

- Quite complex structures can be put together by **nesting structures**: using a structure for a field of another structure. For example, suppose we defined a structure template as follows:

```
struct Address {
    int number;
    char street[20];
    char city[20];
};
```

- We could then incorporate this into our **Employee** structure template as follows:

```
struct Employee {
    int number;
    char name[30];
    int age;
    char position[30];
    struct Address addr;
};
```

9

Nested Structures (contd.)

- We can then declare a variable of this type:
`struct Employee emp;`
- We could then assign values such as:
`emp.number = 22; /* sets the Employee field */`
`emp.addr.number = 56; /* sets the Address field */`
`strcpy(emp.addr.street, "Elm Street");`

an **Address**
structure
that structure's
street field

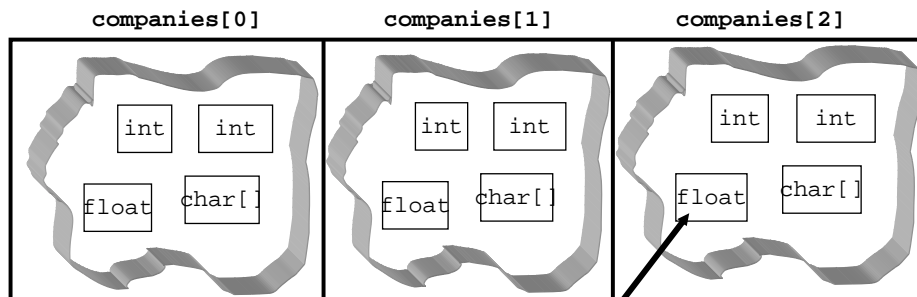
- Note that both **Employee** and **Address** have a field called **number**. These are different -- the structure will have two fields, one for each. This is OK -- there is no ambiguity, because you must include the appropriate "." to tell the compiler which field you intend.

10

Arrays of structures

```
struct Company {
    int sales;
    int profit;
    char name[30];
    float PERatio;
};
struct Company companies[3];
```

This declares a 3-element array called **companies**, each of whose elements is a **Company** structure.



11