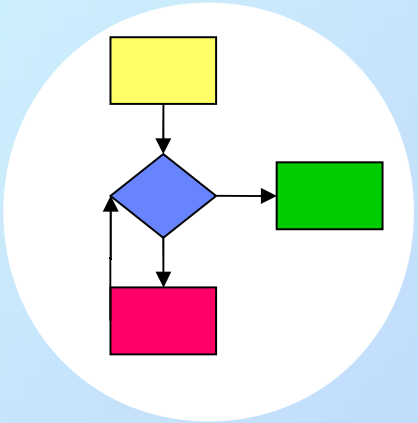
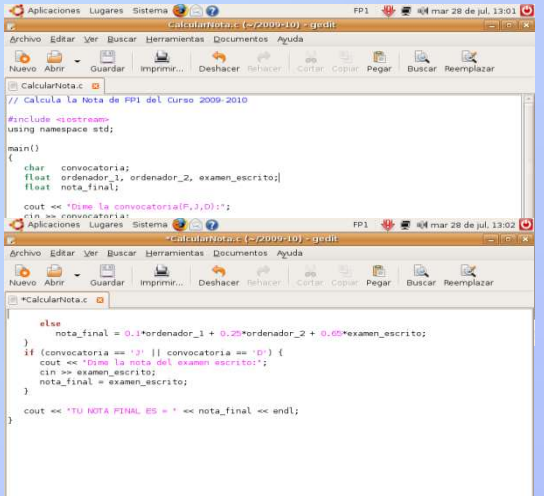


Programming 1

Lecture 7

Structured data types. Structs (Records)



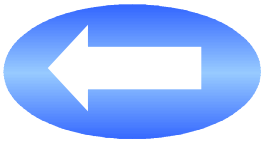
Dpnt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial

Objectives

- Understand the concept of struct (record) data type
- Learn how to define and use complex data structures, nesting structured data types: arrays of structs
- Manage, read and display struct data types in C language

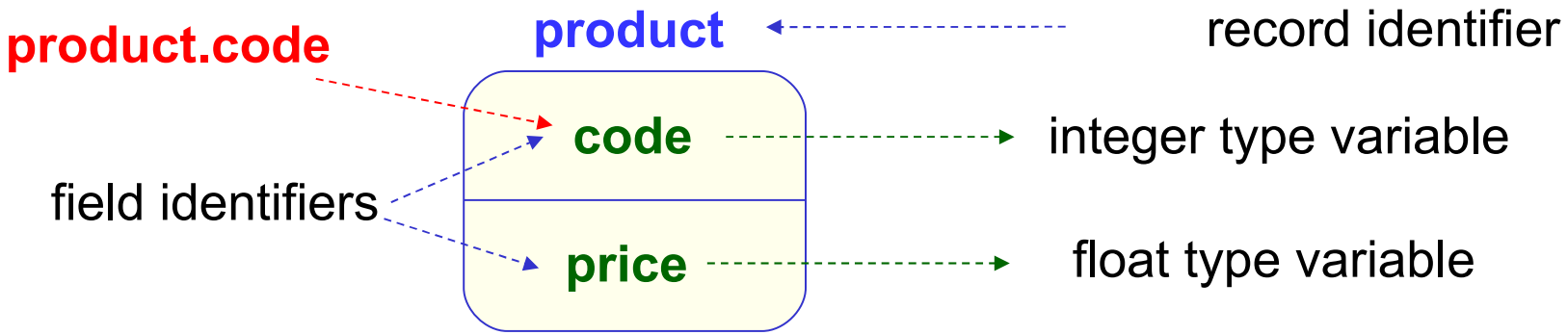
Topics

1. **Struct data type**
2. Arrays of structs
3. Examples
4. Information sources



The struct or record type

- The **struct type** (C language) or **record type** (other languages) is a data structure to store a finite collection of **heterogeneous** elements (the data may belong to different data types)
  - A struct usually represents a set of attributes of an entity
- Every element in the record is called **field**
  - To refer to an element in a record, use the record identifier, followed by a dot '.' and the field identifier
- Example: record with two elements



Examples: records

Address	
street	char array
post code	char array
city	char array

Book	
author	char array
title	char array
borrowed	boolean

Date	
day	integer
month	integer
year	integer

Employee data	
name	char array
social security number	char array
salary	float
address	record
birth date	record

Definition of structs (records) in C language

- To declare a variable of struct type, this type must be previously defined. There are several ways:

```
typedef struct struct_name {
      field_type1 field_name1;
      field_type2 field_name2;
     ...
};
```

  - **struct_name**: name for the defined struct. Any valid identifier
  - **field_type**: type for the corresponding struct field
  - **field_name**: name for the corresponding struct field. There can be as many fields as needed

Example: struct declaration

```
typedef struct TProduct {
      int    code;
      float price;
};

TProduct  p1, p2;
```

p1 and p2 are variables of struct type *TProduct*

```
typedef struct TBook {
      bool borrowed;
      char author [30];
      char title [50];
} ;

TBook book1, book2;
```

Nested structured data types: a char array inside a struct

book1 and book2 are variables of struct type *TBook*

```
typedef struct TNif{
      int number;
      char letter;
};
```

```
typedef struct TClient{
      TNif nif;
      char name [30];
};

TClient client1;
```

Nested structs

Struct initialization and access

- To initialize a struct, all its fields must be initialized, having access to each one
- To **access** a struct field, the **operator '.'** is used

```
struct_name.field_name;
```

  - **struct_name**: struct identifier
  - **field_name**: struct field name

- Example: accessing the code of product p1: **p1.code;**
- Example: accessing the book book1 author's initial letter: **book1.author [0];**
- Example: accessing the NIF letter of client client1: **client1.nif.letter;**

Example: struct initialization and access

```
typedef struct TProduct {
      int    code;
      float price;
} ;

TProduct  p1, p2;
```

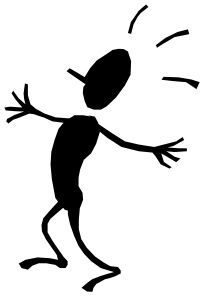
```
p1.code = 3;
p1.price = 34.8;
p2 = p1;
```

Struct assignment is allowed in C language

```
Typedef struct TBook {
      bool borrowed;
      char author [30];
      char title [50];
      TDate borrow_date;
} ;

TBook book1, book2;
```

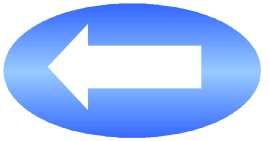
The structs can be nested, that is, a struct field can be of another struct type



```
strcpy(book1.author ,"Quevedo");
cin.getline(book1.title, 50-1);
book1.borrowed = true;
book1.borrow_date.day = 16;
book1.borrow_date.month = 11;
book1.borrow_date.year = 2010;
```

Topics

1. Struct data type
2. **Arrays of structs**
3. Examples
4. Information sources



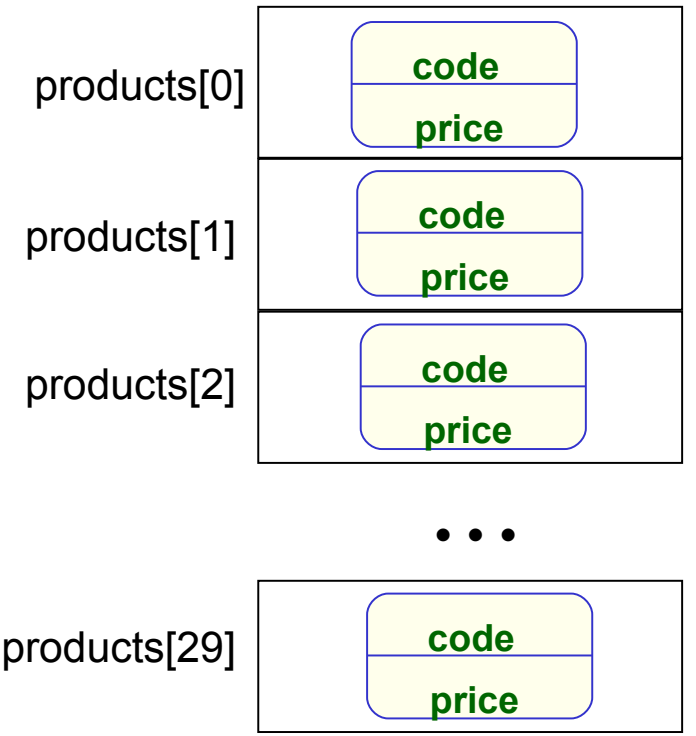
Arrays of structs

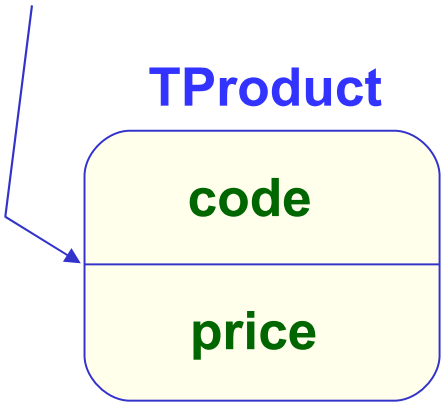
- In an array of structs each element in the array is a struct

```
typedef struct TProduct{
   int    code;
   float price;
} ;
```

```
typedef TProduct TProductList [30];
TProductList  products;
```

```
TProduct products [30];
```

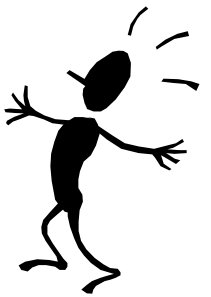




How would you access the third product code?
products[2].code

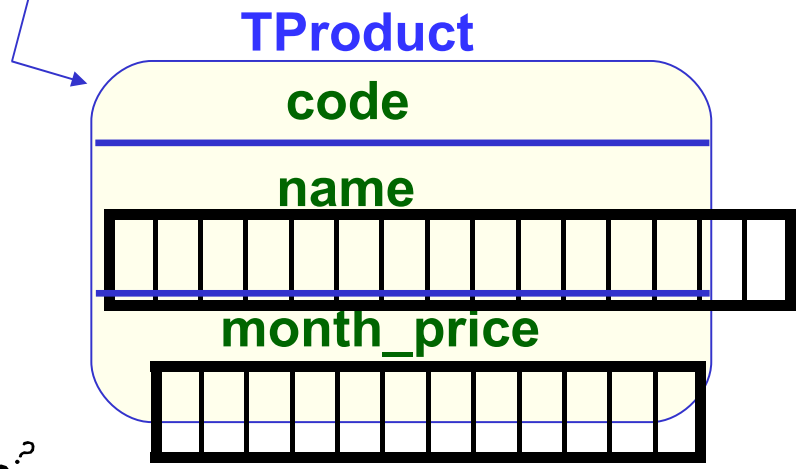
The structured data types can be nested (I)

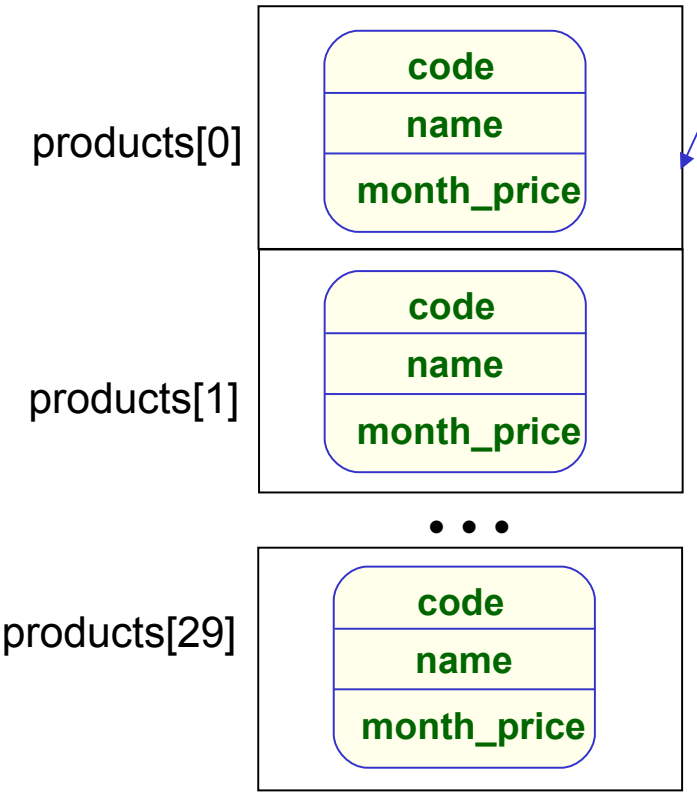
- A struct field can be an array itself



```
typedef struct TProduct {
    int    code;
    char name [15]
    float  month_price[12];
}
```

TProduct products [30];





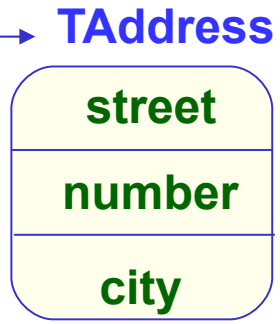
- How would you access the price in august of the fifth product?

The structured data types can be nested (II)

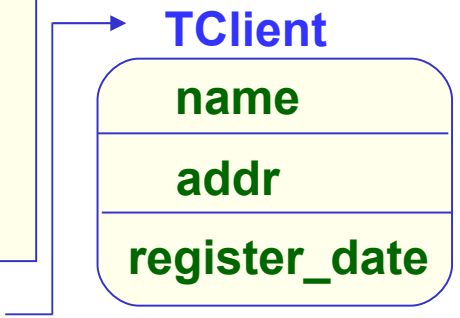
- A struct field can be a struct itself

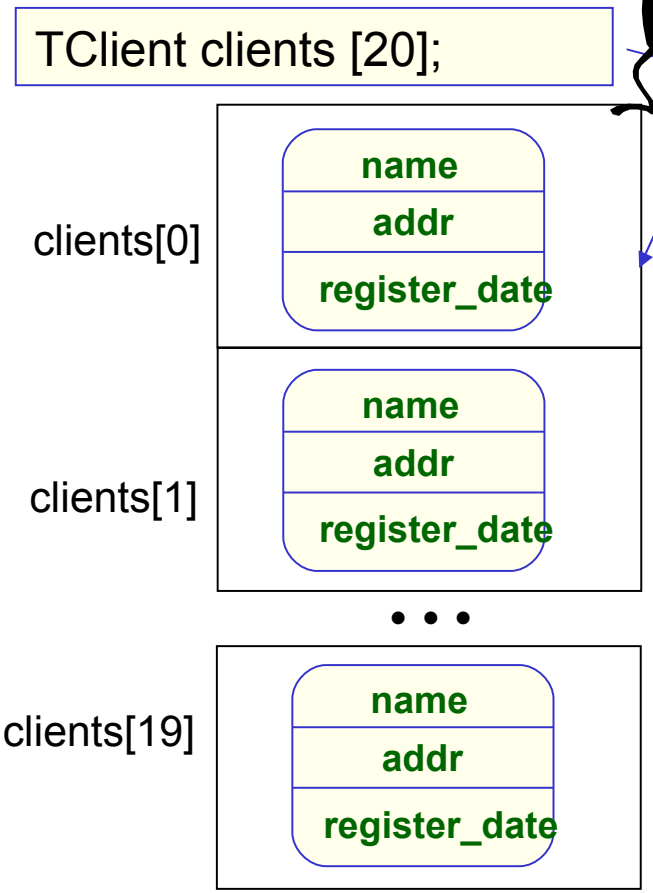
```
typedef char TString[15];

typedef struct TAddress {
    TString     street;
    int         number;
    TString     city;
} ;
```




```
typedef struct TClient {
   TString    name;
   TAdress    addr;
   TDate      register_date;
} ;
```





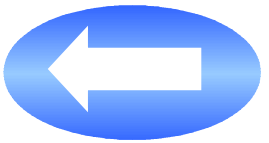
- How would you access the last client's city?

clients[19].addr.city

- And to the year of the first client's registration?

Topics

1. Struct data type
2. Arrays of structs
3. **Examples**
4. Information sources

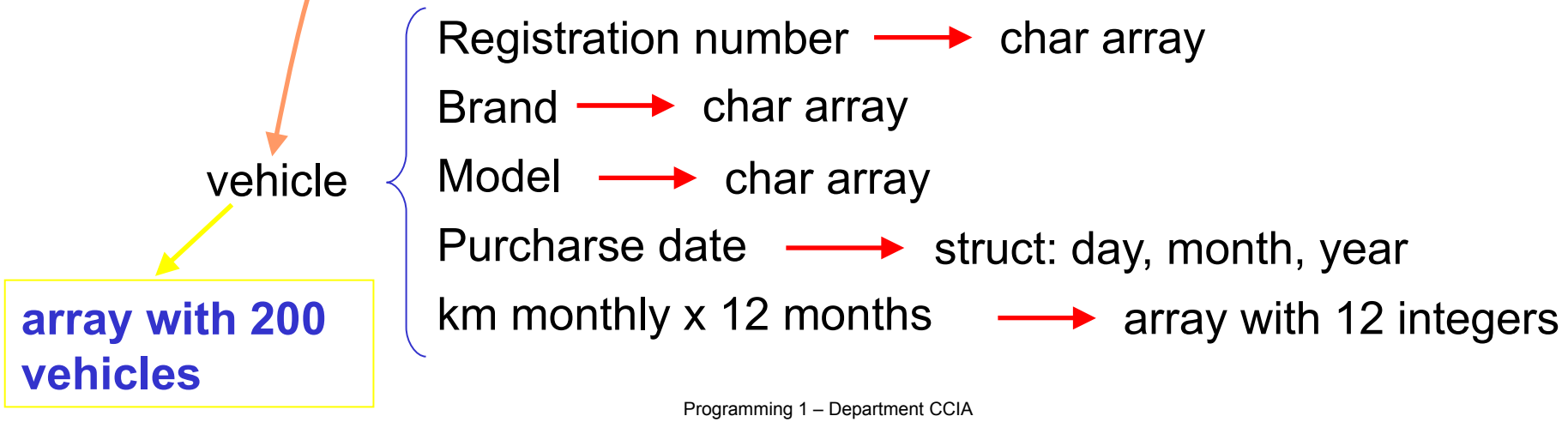


Example 1

- Define the necessary data structures to store and process the following information:
  - A rent-a-car company wants to manage information about its car fleet (up to 200 vehicles): registration number, brand, model, purchase date and the amount of kilometres done each month during a year. The aim is to obtain vehicles which do the highest average amount of kilometres during a year (there can only be one car or several cars with the same average)

Example 1: Data structures

- Define the necessary data structures to store and process the following information:
  - A rent-a-car company wants to manage information about its car fleet (up to **200** vehicles): **registration number**, **brand**, **model**, **purchase date** and the **amount of kilometres** done each month during a year. The aim is to obtain vehicles which do the highest average amount of kilometres during a year (there can only be one car o several cars with the same average)

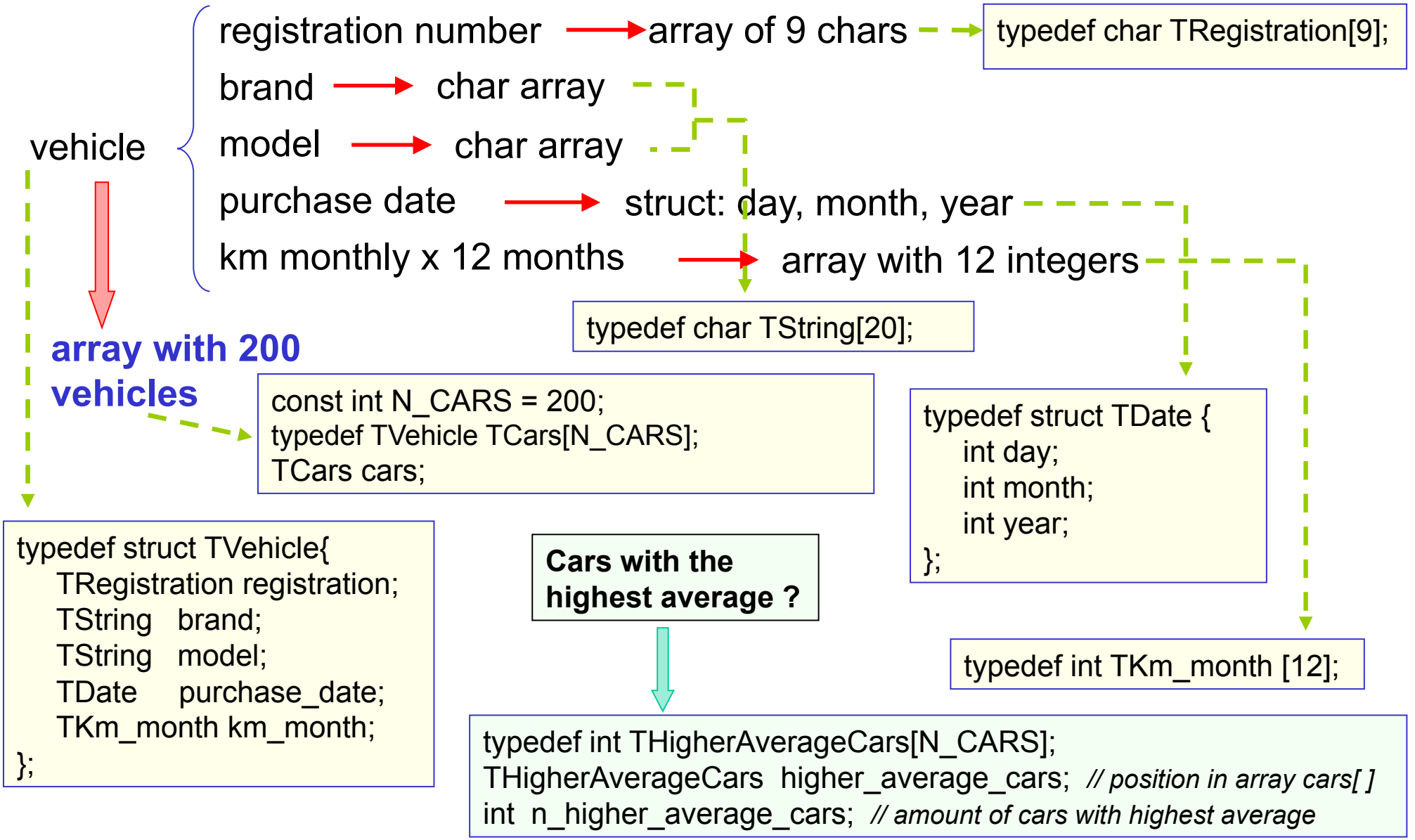


Example 1: Our aim is ...

- Define the necessary data structures to store and process the following information:
  - A rent-a-car company wants to manage information about its car fleet (up to 200 vehicles): registration number, brand, model, purchase date and the amount of kilometres done each month during a year. The aim is to obtain vehicles which do the highest average amount of kilometres during a year (there can only be one car o several cars with the same average)

array with the index of the vehicles with the highest average (200)

Example 1: Data design

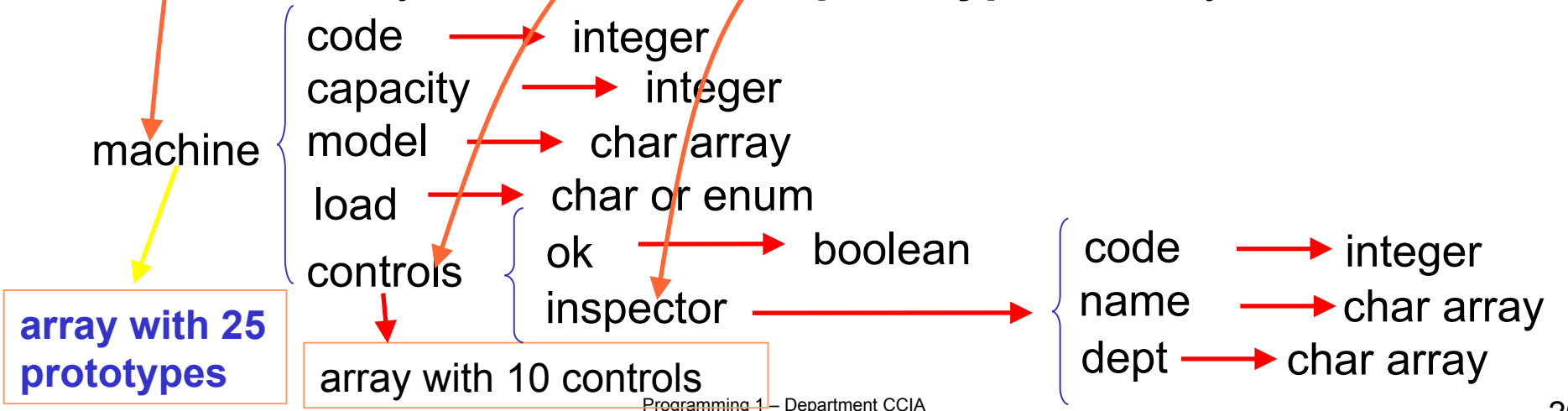


Example 2

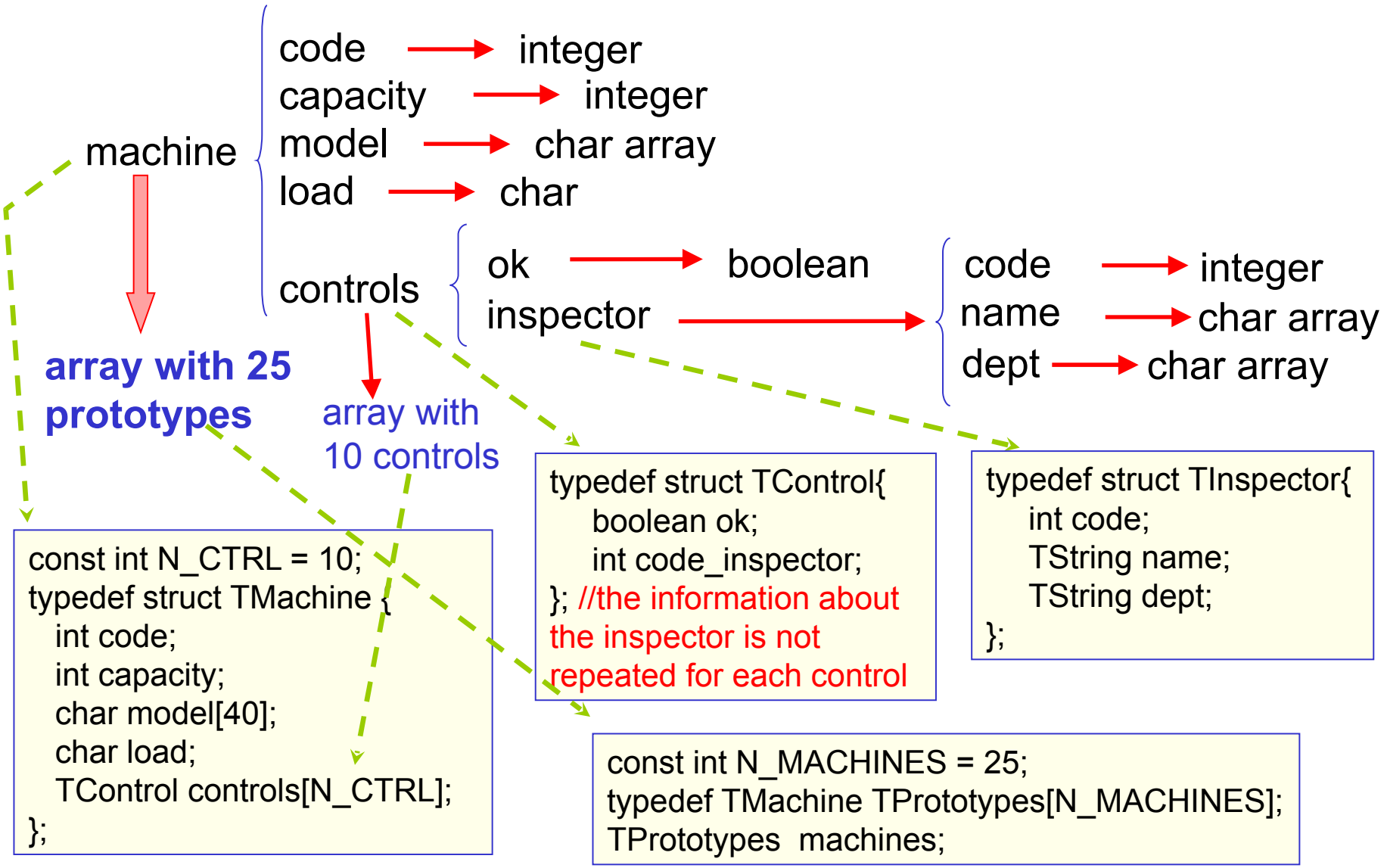
- Define the necessary data structures to store and process the following information:
  - In a washing machine factory a computerized quality control wants to be established to check its prototypes. Each machine has a numerical code and a set of features: capacity (in kg), model, loading type (top/front) and the result of 10 control tests. Each control test can only have two possible results: accepted or rejected. Moreover, we need to know the inspector for each control test. We have the following information for each inspector: numerical code, name, department where he/she belongs to. The factory manufactures 25 prototypes each year.

Example 2: Revise the text

- Define the necessary data structures to store and process the following information:
  - In a washing machine factory a computerized quality control wants to be established to check its prototypes. Each machine has a **numerical code** and a set of features: **capacity** (in kg), **model**, **loading** type (top/front) and the result of **10 control tests**. Each control test can only have two possible results: accepted or rejected. Moreover, we need to know the **inspector** for each control test. We have the following information for each inspector: numerical **code**, **name**, **department** where he/she belongs to. The factory manufactures **25 prototypes** each year.

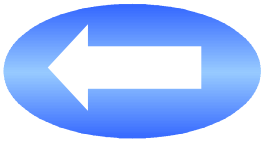


Example 2: Data design



Topics

1. Struct data type
2. Arrays of structs
3. Examples
4. **Information sources**



Information sources

Fundamentos de Programación
Jesús Carretero, Félix García, y otros
Thomson-Paraninfo 2007. ISBN: 978-84-9732-550-9

✓ Capítulo 9 (Apartados 9.1.1; 9.1.2; 9.3)

Problemas Resueltos de Programación en Lenguaje C
Félix García, Alejandro Calderón, y otros
Thomson (2002) ISBN: 84-9732-102-2

✓ Capítulo 2 (Apartados 7.1.1; 7.1.2; 7.2)

Resolución de Problemas con C++
Walter Savitch
Pearson Addison Wesley 2007. ISBN: 978-970-26-0806-6

✓ Capítulo 6 (Apartado 6.1)