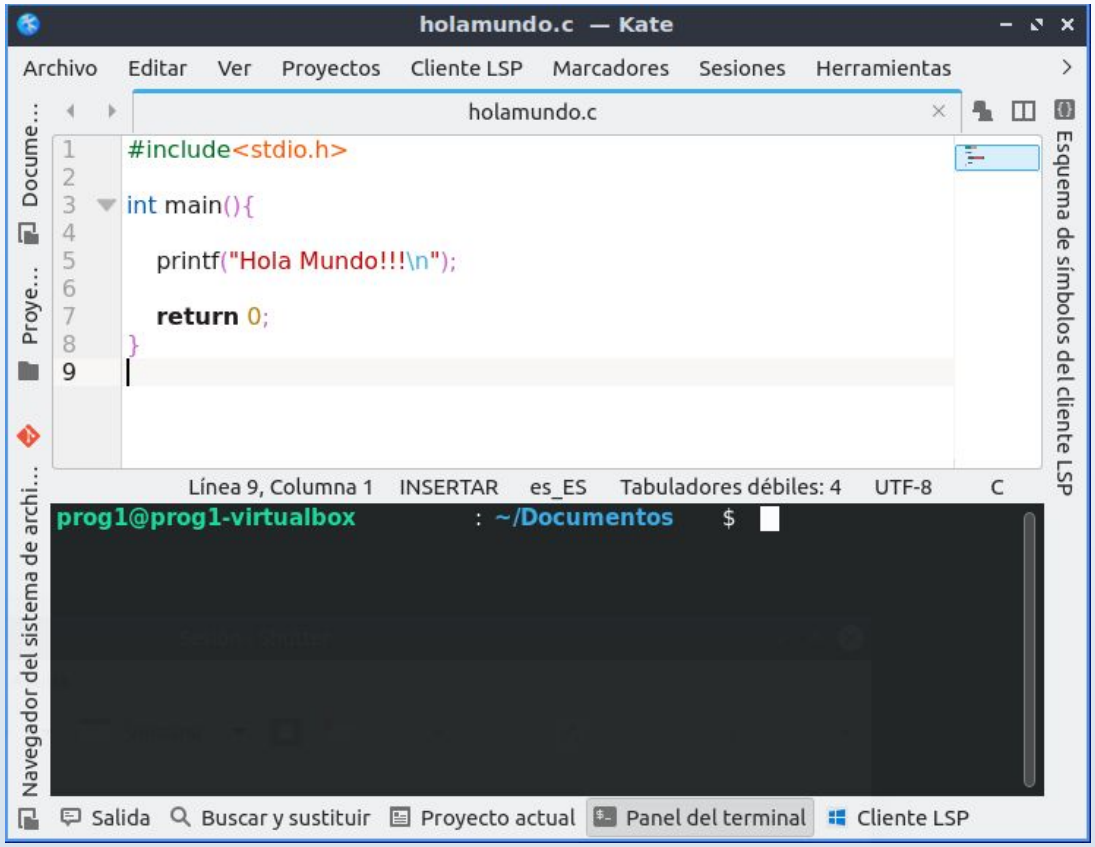

Programming 1

Lesson 7. Structured data types: structures

Degree in Computer Engineering

Index

1. Declaration of data types: `typedef`
2. The `struct` data type
3. Arrays of structs

1. Declaration of data types: `typedef`

- In the C language, custom data types can be defined using the reserved word **`typedef`**.
- It is useful to create new data types to improve the readability of programs.
- Syntax:

```
typedef declaration;
```

where *`declaration`* has the form of a variable declaration, except that the variable created will actually be a new data type.

1. Declaration of data types: `typedef`

Examples of the creation of new data types using `typedef`

- Creation of a new **`point2D`** data type:

```
typedef int point2D[2];
```

Variables of the new data type **`point2D`** can now be created:

```
point2D point;

point[0] = 3;
point[1] = -1;
```

Which is equivalent to:

```
int point[2];

point[0] = 3;
point[1] = -1;
```

1. Declaration of data types: `typedef`

Examples of the creation of new data types using `typedef`

```
// Declaration of data types
typedef int   TVector[20];
typedef float TMarks[50];
typedef char  TString[30];
typedef int   TMatrix[3][3];

// Declaration of variables
TMarks  P1_marks, P2_marks;
TString student1_name, student2_name;
TMatrix matrix1, matrix2;
```

In Programming 1, by convention, we will name new data types with the prefix **T**, to indicate that it is a user data type, and the following (first) letter of the name of the new data type goes in capital letters

Which is equivalent to:

```
float P1_marks[50];
float P2_marks[50];
char  student_name1[30];
char  student_name2[30];
int   matrix1[3][3];
int   matrix2[3][3];
```

2. The `struct` data type

- The **`struct` data type** is a data structure, also known as **record**, that stores a finite collection of elements, not necessarily of the same data type, that are related to each other.
- A structure usually groups together attributes (properties) of an entity, e.g. a person, a vehicle etc.
- Each one of the elements (attributes) of a structure is called a **field**.

2. The `struct` data type

Examples of structures:

Address	
street name	array of chars
ZIP code	array of chars
city	array of chars

Book	
author	array of chars
title	array of chars
lent	boolean

Date	
day	integer
month	integer
year	integer

Employee	
name	array of chars
SSN	array of chars
salary	float
address	struct
date of birth	struct

Note: It is convenient to identify the structures (records) and their attributes (fields) well before defining them in the programming language.

2. The `struct` data type

Declaration of a *record* (data structure) in C

- To define a new *record* data type, the reserved word `struct` must be used together with the identifier (name) assigned to it and the set of attributes (fields) of the record delimited by curly brackets (`{}`):

```
struct [name]{
    type_field1 name_field1;
    type_field2 name_field2;
    ...
}[var1, var2, ...] ; // the semicolon is mandatory
```

The fields of the record are defined with the same syntax as the variables

  - `name`: name of the defined record type. It can be any valid identifier and can be omitted if the structure is only used to declare variables (*var1*, *var2*, ...) of that type.
  - `type_fieldX`: type of each field in the record.
  - `name_fieldX`: name of each field in the record. There can be as many fields as necessary.
  - *var1*, *var2*, …: Variables can be declared.

2. The `struct` data type

Examples of record declarations:

```
struct TProduct {
  int   code;
  float price;
} p1, p2;
```

p1 and **p2** are variables of the data type (record) **struct TProduct**

```
struct TControl{
  int code;
  bool passed[10];
};

...

struct TControl c1;
```

c1 is a variable of the data type (record) **struct TControl**. This record has an array of booleans as a field.

Since the structure has not been declared as a data type, to declare variables of this record type, you have to use **struct TControl**

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

```
typedef struct {
  TDni nif;
  char name[30];
} TMember;

...

TMember member1;
```

By using **typedef** we declare a new data type **TDni** which we can then use to declare variables of that type, even within other records (**record nesting**).

Access to the fields of a record in C

- To access the fields of a record, the dot operator "**.**" must be used.

- Syntax:

```
record_identifier.field_name
```

Example:

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

...

TDni myDni;
```

```
// Inicialization of the variable myDni
myDni.number = 12345678;
myDni.letter = 'A';
```

record identifier

operator "**.**" to access fields in the record

identifier of the field in the record to be accessed

Assignment of records in C

- The assignment operator "=" works in a similar way as it does between simple data types.

Example:

```c
#include<stdio.h>
#include<string.h> // to be able to use the strcpy() function

#define MAX_LEN 30

typedef char TString[MAX_LEN];

typedef struct {
  int number;
  char letter;
} TDni;

typedef struct {
  TCadena name;
  int age;
  TDni dni;
} TPerson;
```

```c
int main(){
  TPerson p1, p2;

  strcpy(p1.name, "Juan Pérez");
  p1.age = 43;
  p1.dni.number = 12345678; // access to data in nested
  p1.dni.letter = 'A';      // structures (records)

  p2 = p1;

  ...

  return 0;
}
```

After this sentence, both variables have the same information in their fields.

3. Arrays of records

- They are declared like any other array by using the record data type as the base type of the array.
- A variable of the record data type is stored in each array position.
- To access the information of a particular record in the array, first access the array position and then access the particular field.

Example:

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

TProduct productList[100];
```

Once the data type of the record has been declared, the array of records is declared.

```
...
productList[8].code = 456;
productList[8].price = 30.49;
...
```

Example in which the ninth record of the array is accessed to assign values to its two fields.

3. Arrays of records

Example 1:

Define the data structures necessary to process the following information:

- A vehicle rental company wishes to manage the information about the vehicles it has (no more than 200). Specifically: number plate, brand, model, date of purchase, and monthly kilometres travelled for the whole year, to obtain the vehicles that travel the most kilometres on average per year (it could be just one or many with the same average).

3. Arrays of records

Example 1 (II): Possible data structures

- A vehicle rental company wishes to manage the information about the **vehicles** it has (no more than 200). Specifically: number plate, brand, model, date of purchase, and monthly kilometres travelled for the whole year, to obtain the vehicles that travel the most kilometres on average per year (it could be just one or many with the same average).

record: **vehicle**

An array with 200 records of type vehicle is needed.

Field	Type of data
number plate	string of characters
brand	string of characters
model	string of characters
date of purchase	record: day, month, year
kms/month x 12 months	array of 12 integer numbers

3. Arrays of records

<u>Example 1 (III)</u>: Possible data structures

- A vehicle rental company wishes to manage the information about the vehicles it has (no more than 200). Specifically: number plate, brand, model, date of purchase, and monthly kilometres travelled for the whole year, <u>to obtain the vehicles that travel the most kilometres on average per year</u> (it could be just one or many with the same average).

<u>Goal</u>: **To obtain an array with the indexes of the vehicles with the highest average (at most, it will be all of them, i.e., 200).**

3. Arrays of records

Example 1 (IV): Design of the data

```
#define NUM_CARS 200

typedef char TString[30];
typedef char TNumberPlate[9];
typedef int  TKms_month[12];

typedef struct {
  int day;
  int month;
  int year;
} TDate;

typedef struct {
  TNumberPlate plate;
  TString      brand;
  TString      model;
  TDate        purch_date;
  TKms_month   kms_month;
} TVehicle;

typedef TVehicle TVehicleList[NUM_CARS];
TVehicleList vehicles;
```

Record structure for date

Record structure for vehicle

Structures to store the list of vehicles

```
typedef int TVehiclesMostKms[NUM_CARS];

// array of positions in the vehicle array
TVehiclesMostKms vehicles_most_kms;

// number of vehicles with the highest average
int num_vehicles_most_kms;
```

Structures to store the list of all indexes of the cars with the highest average number of km per year

3. Arrays of records

Example 2:

Define the data structures necessary to process the following information:

- A washing machine manufacturing plant wants to establish software-based quality control of its prototypes. Each appliance is characterized by a numerical code and a series of characteristics: capacity (in kilos), model, type of load (top/front), and the result of the 10 controls to which it has been subjected. The control has only two possibilities: it has passed or failed. In addition, knowing which inspector has carried out each check is necessary. An inspector can carry out several controls on the same equipment. The following information is available for each inspector: code number, name, and department to which they belong. The plant manufactures 25 prototypes per year.

Example 2 (II): Possible data structures

- A **washing machine** manufacturing plant wants to establish software-based quality control of its prototypes. Each appliance is characterized by a numerical code and a series of characteristics: capacity (in kilos), model, type of load (top/front), and the result of the 10 **controls** to which it has been subjected. The control has only two possibilities: it has passed or failed. In addition, knowing which **inspector** has carried out each check is necessary. An inspector can carry out several controls on the same equipment. The following information is available for each inspector: numerical code, name, and department to which they belong. The plant manufactures 25 prototypes per year.

record: **washing machine**

An array with 25 records of type washing machine is needed.

Field			Type of data
code			integer number
capacity			integer number
model			string of characters
load			character or enumerated
controls	ok		boolean
	inspector	code	integer
		name	string
		department	string

3. Arrays of records

Example 2 (III): Design of the data

```
#define NUM_MACHINES 25
#define NUM_CONTROLS 10

typedef char TString[30];

typedef struct {
  bool ok;
  int  codInspector;
} TControl;

typedef struct {
  int     code;
  TString name;
  TString department;
} TInspector;
```

Structure of the record for control

Structure of the record for inspector

Structure of the record for washing machine

```
typedef struct {
  int      code;
  int      capacity;
  TString  model;
  char     load; // t - top; f - front
  TControl controls[NUM_CONTROLS];
} TMachine;

typedef TMachine TMachineList[NUM_MACHINES];
TMachineList machineList;
```

Structures for storing the list of washing machines