UNIT 6.
STRUCTURED DATA TYPES
PART 2: STRUCTURES IN C

Programming

Year 2017-2018

Industrial Technology Engineering

Paula de Toledo

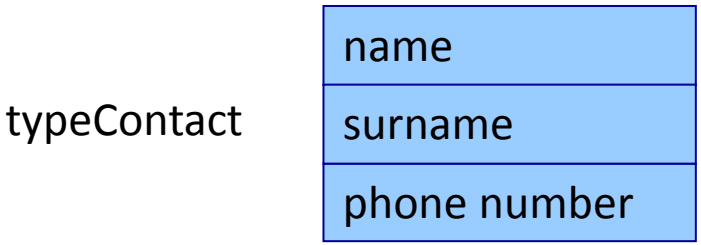
Universidad
Carlos III de Madrid
www.uc3m.es

Contents

6.2 STRUCTURES

Structures

- **User defined datatype** to combine data items of **different kinds**
- Structures are used to represent a record, grouping under the same name data items of the **same or different datatype** that are logically related
  - Example 1 : store a name, surname and telephone number in a structure named typeContact



  - Example 2: Structure to manage email accounts
    - Login (string), password(string), e-mail (string) and user id (int)

Declaring structures

- To declare a structure you need **two steps**
  - STEP 1. Declare the structure itself
    - It's a new type of data!!
  - STEP 2. Declare one (or more) variables of the new type defined in step 1

Declaring a structure

- Step 1: Declare the structure (the datatype)

```
struct name_structure {
        DataType_1 element_1;
        DataType_2 element_2;
        .....
        DataType_n element_N;
};
```

- Example:

```
struct typeAccount {
        char login [256];
        char password [256];
        char email [256];
        int userId;
};
```

Declaring a structure

- Step 1: Declare the structure (the datatype)
- More examples
  - structure to store personal information of a person

```
struct typePerson{
    char name[20];
    char surname [50];
    int age;
    float height;
};
```

  - structure to work with a point in a plane

```
struct typeCoordinates{
    float x;
    float y;
};
```

Declaring a structure

- STEP 2: Declaring a variable of the datatype

- template:

  struct structure_ name variable _ name;

- Examples

  struct typeCoordinates pointA;
  struct typePerson myNeighbour ;
  struct typeAccount myAccount;

Structure members

- Components of a structure are called **members**
- To access any member of a structure, we use the **member access** operator (.).
  - Different to arrays where elements are accessed using the index

    myAccount.login

    myAccount.password

  - myAccount is a variable of type “typeAccount ", a user defined datatype
  - (.) is the member access operator
  - login and password are the names of two members of the " typeAccount” structure

Assign values to structure members and structures

- Two options
  - Assign value to each structure member individually

```
myNeighbour.age = 22;
myNeighbour.height = 1.90;
strcpy (myNeighbour.name, "Juan");
```

  - Assigning one whole structure to another

```
myBoyFriend = myNeighbour;
```

Using structures - example

```
// Step 1. Declare the datatype
struct typePerson   {
      char name[20];
      char surname [50];
      int age;
      float height;

struct point3D {
    float x;
    float y;
    float z;
};

// Step 2. Declare the variable using the datatype
struct point3D pointA;
struct typePerson myNeighbour;
struct typePerson myBoyFriend;
```

Using structures

```
// Step 3. Use the datatype
puntoA.y =100;
strcpy(myNeighbour.name, "Pablo");
myNeighbour.age = 20;
myNeighbour.heigth = 1.90;

printf("%s \n", myNeighbour.name);
printf("%i \n", myNeighbour.age);
printf("%4.2f \n", vmyNeighbour.heigth);
```

Structure initialization

- We can initialize all members of a variable of type structure in the variable declaration
- Similar to vectors

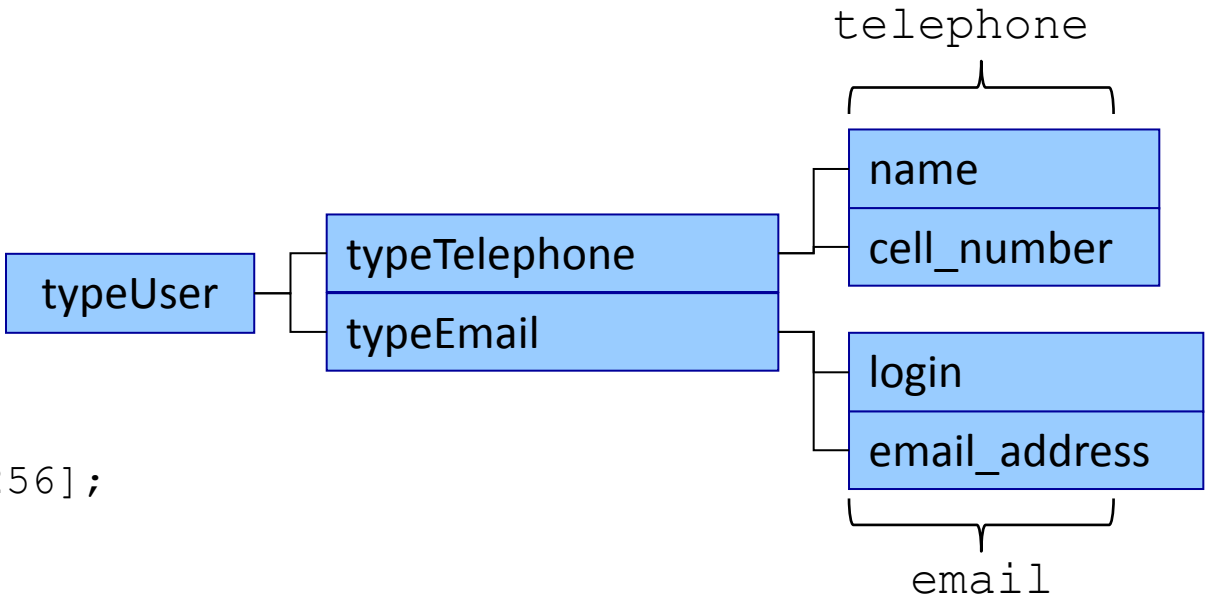
```
struct tipoCoordenadas {
        float x;
        float y;
    };
struct Example{
    char letra;
    long entero;
    char palabra[20];
  };
struct Point3D point1 = {2.1, 3.4, 9.8};
struct Example Example1 = {'a', 23, "Hola"};
```

6.2.3 NESTED STRUCTURES

Nested structures

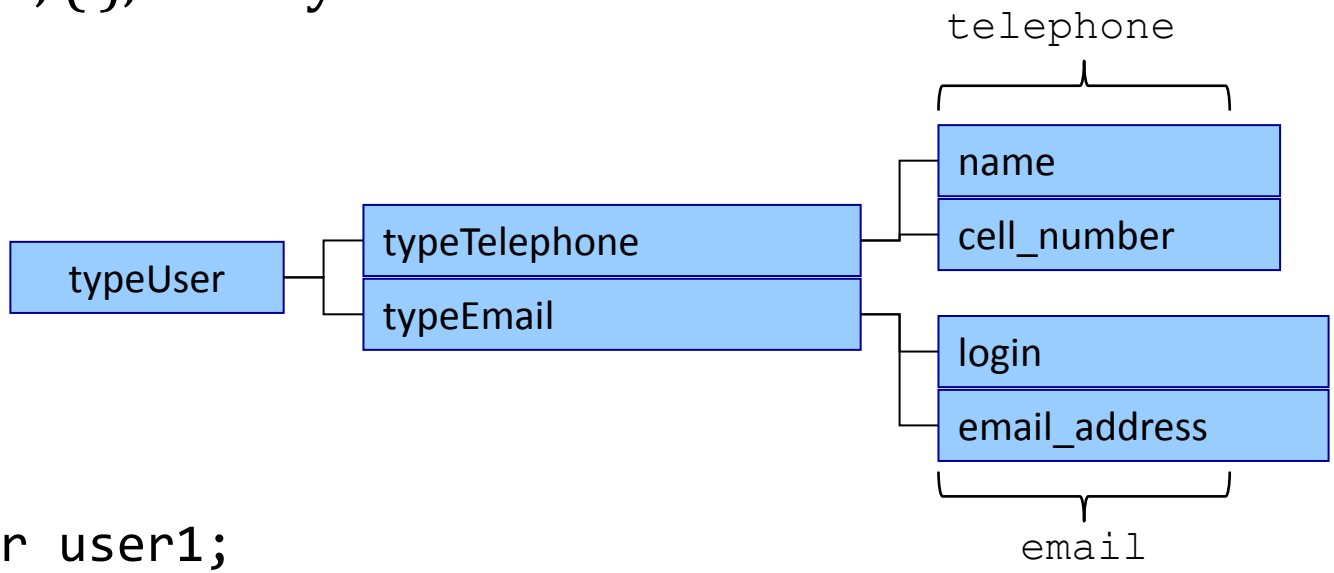
- Members of a structure can be variables of any data type, either simple (int, float, char, pointer.), or structured (array, another structure)
- A structure inside another structure is called a **nested structure**

```
struct typeTelephone{
  char name[256];
  long cell_number;
};
struct typeEmail{
   char login[256];
   char email_address [256];
};
struct typeUser{
   struct typeTelContact telephone;
   struct typeEmail email;
};
```



Access to members in nested structures

- To access a member nested in a structure we use the member access operator, (.), as may times as needed




```
struct typeUser user1;
user1.telephone.cell_number=684567123;
printf ("%s", user1.telephone.name);
strcpy( user1.email.email_address, "abc@xyz.com");
printf ("%s", user1.email.email_address]);
```

6.2.3 Arrays as members of a structure

Arrays as members of a structure

- Members of a structure can be variables of any data type, either simple or structured (array, another structure)
  - Arrays, vectors and strings
    - To access one element of the array, use the index

```
struct typeStudent   {
      char name[20];
      char surname [50];
      int age;
      float height;
       float marks[20]; // marks in each subject
};
struct typeStudent student1;
student1.marks[0]=10.0;
student1.marks[1]=5.0;
student1.name[0]='J'; // change intial letter of name
```

6.2 ARRAYS OF STRUCTURES

Arrays of structures

- Vectors or matrixes where the elements are structures
- Specially useful to store and manage information
  - Very seldom we use structures alone, more typical to use a vector of structures
  - Example: vector of students to store data from a class
- Declaration
  - Template:
    - struct name_structure name_array [size];
    - The structure has to be declared beforehand
  - Example
    - struct typeStudent class [135];
- Use
  - class[1].age = 18;
  - class[1].height= 1.63;

Vector of structures. example 2

```
// Step 1. declare structure
struct typeItem {
      float price;
      int amount;
      char name[30];
};
```

	price	amount	name
products[0]	15.95	10	"cocacola"
products[1]	17.95	3	“fanta"
products[2]	30.95	1	“trina"
products[3]	27.95	12	“pepsi"

```
// Step 2. declare a vector of 20 elements
// where each element is a variable of the typeItem
struct typeItem products[100];

//access amount of  the third item in the list
      productos[2].amount =1;
```

Vector of structures: example 3

```
// Step 1. declare structure
struct tipoFecha{
  int dia;
  int mes;
  int anyo; //ñ char not allowed in C
};

// Step 2. declare a vector where each element is a structure
struct tipoFecha fechaNac[4];

//access vector element and member
  fechaNac[2].anyo =2010;
```

dia	mes	anyo

	dia	mes	anyo
fechaNac[0]	5	10	1998
fechaNac[1]	17	3	2001
fechaNac[2]	30	1	2003
fechaNac[3]	27	12	2010

Example 4: Vector of structures as member of a structure

```
struct Point2D {
    float x;
    float y;
};

struct Triangle {
    // members of the structure are other structures
    struct Point2D a;
    struct Point2D b;
    struct Point2D c;
};

struct Dodecahedron {
    // members of this structure are a vector of structures
    struct Point2D points[12];
};
```

6.2.3 STRUCTURES AS ARGUMENTS OF FUNCTIONS

Structures as arguments

- You can use structures as a function argument in the same way as any other variable

- Structures have to be defined before any function that uses it
  - We recommend you define all structures before the main

- By default, structures are passed **by value**
  - As are int, float, char
  - When the value of a field of the structure is modified in a function, this change is not reflected in the parameter in the main
- Structures can also be passed **by reference**

- Structures can be returned with **return**

Passing structures by reference

- You can define **pointers to structures** in the same way as you define pointer to any other variable
- To pass the structure by reference
  - **Function header + declaration (formal parameter)**
    - The formal parameter is a pointer to the structure
    - To access the structure use indirection operator
      - Example: *product
    - To access a member of the structure (two options
      - Use member access operator (.)
        - (*product). price
      - Use -> operator
        - product->price is the same as (*product). price
  - **Call to the function (actual parameter**)
    - & preceding the parameter

Structures as parameters. Example 1

- Write a program to read a point's coordinates in a three dimensional space and find the distance from the point to the origin (0,0,0)

```
#include <stdio.h>
#include <math.h>
// structure declaration
struct typePoint {
   float x, y, z;
};
// prototypes
void readPoint (struct typePoint *p);
float (struct typePoint p);


int main(void){
    struct typePoint pto;
    readPoint (&pto);
    printf ("Distance from point to origin: %f\n", findDist(pto));
    system ("PAUSE");
  return 0;
}
```

Pass by REFERENCE *p
The function modifies the value

Pass by VALUE
The function doesn't modify the value

By REFERENCE
&pto

By VALUE
pto

```c
float findDist (struct typePoint p) {
    // find distance to origin
   return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}


void readPoint (struct typePoint *p){
     printf ("X?: "); scanf("%f", &(*p).x);
     printf ("Y?: "); scanf("%f", &(*p).y);
     printf ("Z?: "); scanf("%f", &(*p).z);
     return;
}

// readPoint with arrow operator
void readPoint(struct typePoint *p) {
         printf ("X?: "); scanf("%f", &p->x);
         printf ("Y?: "); scanf("%f", &p->y);
         printf ("Z?: "); scanf("%f", &p->z);
        return;
}
```

By VALUE: p
here * is the product, **nothing to do with pointers**

By REFERENCE: *p
To access structure members use (*p).z

& from scanf

p->z is the same as (*p).z

Structures as parameters. Example 2

Write a program defining a vector to store data regarding several products, checking if these products are fake and finding the total number of false products.

A product is fake if it's code starts with "UEX".

```
#include <stdio.h>
#include <string.h>
# define NPROD 4

// structure typeProduct
struct typeProduct {
     char name [15];
     char code[10];
     float price;
     int is_fake; // flag for fake products: 1 if fake, 0 if not
};

void checkProduct (struct typeProduct *p);
```

checkProduct modifes the value of the structure PASS BY REFERENCE
*p

```
void checkProduct (struct typeProduct *p) {
  // Function that takes as parameter a product
  // and modifies the value of member is_fake
  // depending on the product code
  // only products with codes starting with UEX are authentic

// initialize to false
(*p).is_fake = 1;

 //verify code
 if (((*p).code[0]=='U') && ((*p).code [1]=='E') && ((*p).code [2]=='X')) {
    (*p). is_fake = 0;
 }
 return;
}
```

By REFERENCE: *p
To access the structure members, use brackets
(*p).is_fake
(*p).code[2]

```c
int main(void){
  // vector with four products
  struct typeProduct prod[NPROD];
  int i, tot_fake=0;
  //assign values to the codes
  strcpy (prod[0].code, "UEX1002");
  strcpy (prod[1].code, "UEX2002");
  strcpy (prod[2].code, "UET3002");
  strcpy (prod[3].code, "UEZ1002");
 // ……..
  // check how many are fake
  for (i=0; i<NPROD; i++){
      checkProduct (&prod[i]);
      tot_fake= tot_fake + prod[i].is_fake;
  }
  printf ("There are %i fake products  \n", tot_fake);

  return 0;
}
```

By REFERENCE
&prod[i]

Universidad
Carlos III de Madrid
www.uc3m.es

UNIT 6.
STRUCTURED DATA TYPES

PART 2: STRUCTURES