Estructuras

Sofía Beatriz Pérez
Daniel Agustín Rosso

sperez@iua.edu.ar
drosso@iua.edu.ar

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Informática II - Clase número 4 - Ciclo lectivo 2022

Agenda

Disclaimer

Los siguientes slides tienen el objetivo de dar soporte al dictado de la asignatura. De ninguna manera pueden sustituir los apuntes tomados en clases y/o la asistencia a las mismas.
Es importante mencionar que todos este material se encuentra en un proceso de mejora continua.
Si encuentra bugs, errores de ortografía o redacción, por favor repórtelo a sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: ▸ infoI_IUA_GitLab

Introducción I

Definición

Una estructura es una colección de variables referenciadas bajo un mismo nombre. Las variables que forman a la estructura, es decir, las que estan dentro de ellas, son conocidas como **miembros, campos o elementos.**.

```
1  struct personal_data
2  {    char name[30];
3       char apellido[30];
4       int  dni;
5       char direccion[40];
6       char ciudad[20];
7       int  codigo_postal;
8  };
```

Introducción II

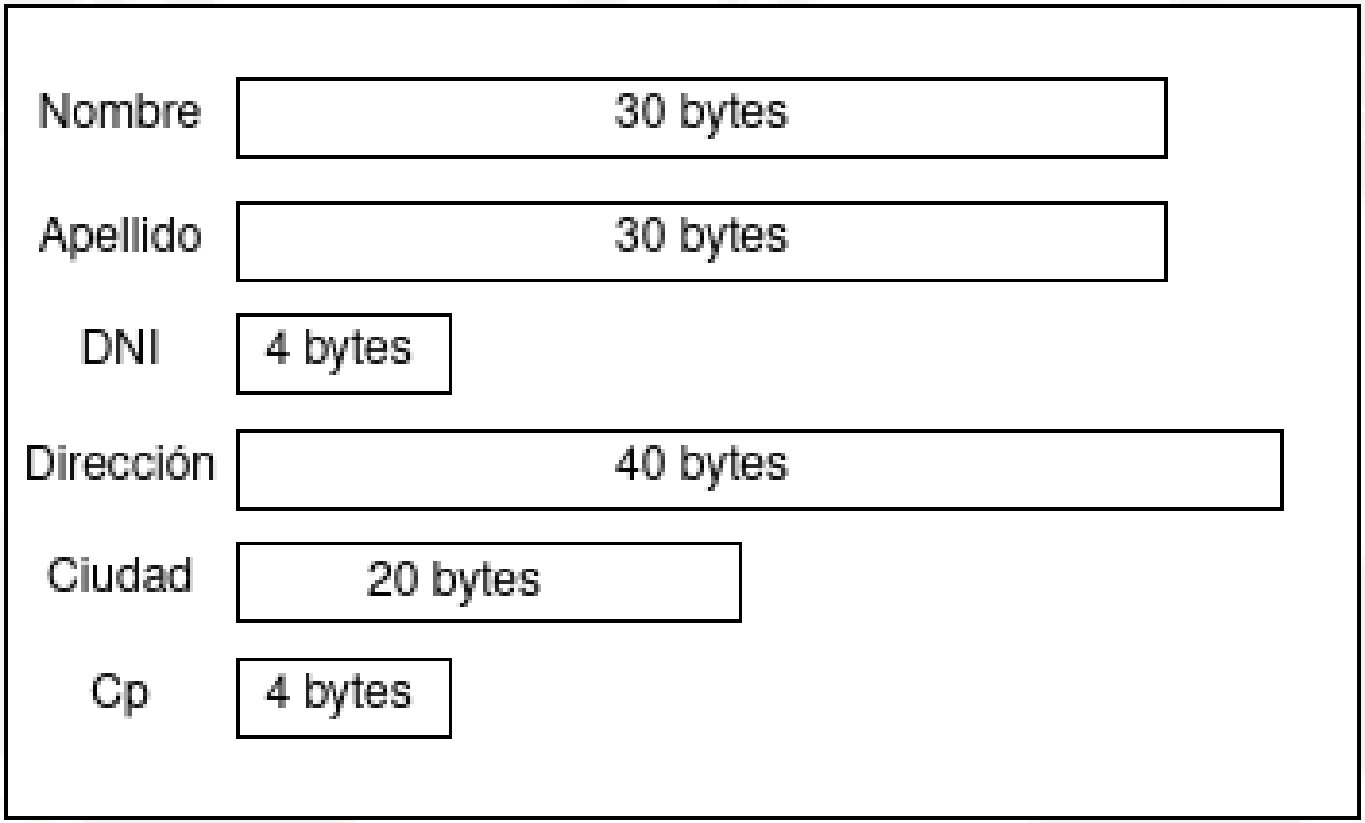

Figure: Miembros de una estructura.

Introducción III

De forma general:

```
1  struct nombre_estructura
2  {
3      tipo nombre_miembro;
4      tipo nombre_miembro;
5      tipo nombre_miembro;
6      tipo nombre_miembro;
7      tipo nombre_miembro;
8      tipo nombre_miembro;
9      tipo nombre_miembro;
10 };
```

Es importante aclarar que hasta este punto, no se han creado nuevas variables. Sólo se ha definido el tipo de dato de una estructura (tipo de dato compuesto).

Declaración de una estructura

Para declarar una variable de tipo estructura, se debe proseguir de la siguiente manera:
En C:

```
1 struct personal_data daniel;
2 struct nombre_estructura var1;
```

En C++ una vez que una estructura se ha definido, se pueden declarar variables de ese tipo sólo referenciando al nombre de la estructura, es decir no es necesario preceder a la declaración con la palabra struct:

```
1 personal_data daniel;
2 nombre_estructura var1;
```

Acceso a miembros

Individualmente los miembros de una estrcutra son accedidos utilizando el operador (.)

```
1  daniel.dni=36053357;
2  daniel.codigo_postal=5010;
3
4  for (int ii;ii<10;i++)
5  {
6      scanf(" %c",&daniel.nombre[ii]);
7  }
8
9  var1.miembro_de_la_estructura;
```

Ejemplo

```c
#include <stdio.h>
struct personal_data{
    char nombre[30];
    char apellido[30];
    int  dni;
    char direccion[40];
    char ciudad[20];
    int  cp;};
int main()
{   struct personal_data daniel={"Daniel","Rosso",
    36053357,"Pelegrini 193", "Alta Gracia",5186};

    printf("Nombre:\t %s\n",daniel.nombre);
    printf("DNI:\t %d\n",daniel.dni);

    return(0);}
```

¿Qué tamaño tiene una estructura?: el operador sizeof I

Como se mencionó anteriormente, una estructura es una colección de variables, agrupadas bajo el mismo nombre:

```
1  #include <stdio.h>
2  struct personal_data
3  {
4      char nombre[30];
5      char apellido[30];
6      int  dni;
7      char direccion[40];
8      char ciudad[20];
9      int  cp;
10 };
11
12
13
```

¿Qué tamaño tiene una estructura?: el operador sizeof II

```
14  int main()
15  {
16    struct personal_data daniel={"Daniel","Rosso",
17    36053357,"Pelegrini 193", "Alta Gracia",5186};
18
19
20    printf("Size de la estructura %ld bytes \n",
21    sizeof(daniel));
22
23
24    return(0);
25  }
```

Arreglos de estructuras I

Uno de los usos mas comunes de las estructuras es utilizarlas dentro de un **arreglo de estructuras.**

```
1  struct measurement
2  {
3      float temp;
4      float hum;
5      int id;
6  }
7
8  struct measurement mesurement_array[10];
```

Arreglos de estructuras II

```
1  #include <stdio.h>
2  struct measurement
3  {
4      float temp;
5      float hum;
6      int id;
7  };
8
9  int main()
10 {
11     struct measurement m_array[3];
12
13
14
15
```

Arreglos de estructuras III

```
16      for(int ii=0;ii<3;ii++)
17      {   printf("Ingrese la temperatura %d\n",ii);
18          scanf("%f",&m_array[ii].temp);
19          printf("Ingrese la humedad %d\n",ii);
20          scanf("%f",&m_array[ii].hum);
21          m_array[ii].id=ii;
22      }
23
24      for(int ii=0;ii<3;ii++)
25      {
26          printf("Temperatura \t %f\n",m_array[ii].temp);
27          printf("Humedad \t %f\n",m_array[ii].hum);
28          printf("Id \t \t %d\n",m_array[ii].id);
29      }
30  }
```

Estructuras y funciones I

Cuando una estructura es pasada a una función, este pasaje es realizado **POR VALOR**. Esto significa que cualquier cambio realizado por la función en el contenido de la estructura, no sufrirá efecto. [1]

```
1  #include <stdio.h>
2  struct measurement
3  {
4      float temp;
5      float hum;
6      int id;
7  };
8
9  /*Prototipo de la funcion*/
10 void print_members(struct measurement);
```

Estructuras y funciones II

```
11
12
13  int main()
14  {
15      struct measurement m;
16      printf("Ingrese la temperatura\n");
17      scanf("%f",&m.temp);
18      printf("Ingrese la humedad\n");
19      scanf("%f",&m.hum);
20     m.id=10;
21      /*Llamada a la funcion*/
22      print_members(m);
23
24  }
25
```

Estructuras y funciones III

```
26
27
28  void print_members(struct measurement m)
29  {
30      printf("Temperatura \t %f\n",m.temp);
31      printf("Humedad \t %f\n",m.hum);
32      printf("Id \t \t %d\n",m.id);
33  }
```

[1] A esta altura del año, se supone que el lector maneja a la perfección la diferencia entre paso por valor y paso por referencia ;):

Punteros a estructuras I

Tal como sucede con otro tipo de variables, C/C++ permiten trabajar con punteros a estructuras. Esto es particularmente útil por dos motivos principales:

- Pasar estructuras a funciones utilizando referencia: cuando el espacio requerido por una estructura es demasiado, aparecen penalidades de rendimiento en tiempo de ejecución cuando estas son enviadas a funciones utilizando el mecanismo por default, es decir por valor.
- Crear estructuras auto-referenciadas: se cubrirá esto en las clases siguientes.

Punteros a estructuras II

```
1  #include <stdio.h>
2  struct measurement
3  {
4      float temp;
5      float hum;
6      int id;
7  };
8  /*Prototipo de la funcion*/
9  void print_members(struct measurement *);
10
11
12
13
14
15
```

Punteros a estructuras III

```
16  void main()
17  {
18      struct measurement m;
19      struct measurement *p;
20
21      printf("Ingrese la temperatura\n");
22      scanf("%f",&m.temp);
23      printf("Ingrese la humedad\n");
24      scanf("%f",&m.hum);
25      m.id=10;
26      p=&m;
27      /*Llamada a la funcion*/
28      print_members(p);
29  }
```

Punteros a estructuras IV

Para acceder a miembros de una estructura usando un puntero a dicha estructura, se debe hacer uso del operador flecha ($->$). Este operador se utiliza en lugar del operador punto (.) cuando se está accediendo a una estructura utilizando un pzuntero a ella.

```
1  void print_members(struct measurement *m)
2  {
3      printf("Temperatura \t %f\n",m->temp);
4      printf("Humedad \t %f\n",m->hum);
5      printf("Id \t \t %d\n",m->id);
6  }
```

▸ Ver ejemplo en gitlab

Anidación de estructuras I

Una estructura anidada en C es una estructura dentro de la cual existe otra estructura como miembro de la primera.

```
1  #include <stdio.h>
2
3  struct timestamp
4  {
5      int day;
6      int month;
7      int hh;
8      int mm;
9  };
10
11
```

Anidación de estructuras II

```
12  struct measurement
13  {
14      float temp;
15      float hum;
16      int id;
17      struct timestamp time;
18  };
19
20
21
22
23
24
25
26
```

Anidación de estructuras III

```
27  void main()
28  {    struct measurement m;
29       printf("Ingrese la temperatura\n");
30       scanf("%f",&m.temp);
31       printf("Ingrese la humedad\n");
32       scanf("%f",&m.hum);
33      m.id=10;
34       printf("Ingrese el dia \n");
35       scanf("%d",&m.time.day);
36       printf("Ingrese el mes \n");
37       scanf("%d",&m.time.month);
38       printf("Ingrese la hora \n");
39       scanf("%d",&m.time.hh);
40       printf("Ingrese los minutos \n");
41       scanf("%d",&m.time.mm);
```

Anidación de estructuras IV

```
42      printf("Temperatura \t %f\n",m.temp);
43      printf("Humedad \t %f\n",m.hum);
44      printf("Id \t \t %d\n",m.id);
45      printf("Dia \t %d \t\n",m.time.day);
46      printf("Mes \t %d \t\n",m.time.month);
47  }
```

La estructura definida como timestamp, puede también ser utilizada de forma aislada.
Es imperante aclarar que se a mostrado sólo una anidación de dos niveles. Esto puede ser extendido mas niveles, pero se debe tener en cuenta que C soporta anidación de hasta 15 niveles.

Typedef con estructuras I

La palabra reservada **typedef** permite cerear alias para tipo de datos definidos anteriormente

```
1  #include <stdio.h>
2  struct timestamp
3  {
4      int day;
5      int month;
6      int hh;
7      int mm;
8  };
9
10
11
12
```

Typedef con estructuras II

```
13  struct measurement
14  {
15      float temp;
16      float hum;
17      int id;
18      struct timestamp time;
19  };
20
21  typedef struct timestamp timestamp_t;
22  typedef struct measurement measurement_t;
23
24
25
26
27
```

Typedef con estructuras III

```
28
29  int main()
30  {
31
32      measurement_t m1, m2, m3;
33      timestamp_t t1, t2, t3;
34
35      m1.temp=10;
36      m2.hum=19;
37
38    return(0);
39  }
```

¡Muchas gracias!

Consultas:

sperez@iua.edu.ar
drosso@iua.edu.ar