Informática I

Claudio Paz

claudiojpaz@gmail.com

Septiembre 2019

Unidad 9

Estructuras y uniones en C. Campos de bit.

Repaso

Repaso

Los int, char, float con todos sus calificadores, junto con el tipo void y los punteros son conocidos como *tipos escalares*.

Repaso

Los int, char, float con todos sus calificadores, junto con el tipo void y los punteros son conocidos como *tipos escalares*.

Los arreglos vistos en la Unidad 5 forman parte de los *tipos agregados*.

Repaso

Los int, char, float con todos sus calificadores, junto con el tipo void y los punteros son conocidos como *tipos escalares*.

Los arreglos vistos en la Unidad 5 forman parte de los *tipos agregados*.

Los arreglos sirven para almacenar datos relacionados del mismo tipo bajo un mismo nombre.

Repaso

Los int, char, float con todos sus calificadores, junto con el tipo void y los punteros son conocidos como *tipos escalares*.

Los arreglos vistos en la Unidad 5 forman parte de los *tipos agregados*.

Los arreglos sirven para almacenar datos relacionados del mismo tipo bajo un mismo nombre.

Existe otra forma de datos de *tipo agregado*...

Estructuras

Estructuras

Las estructuras son tipos de datos *derivados* que agrupan datos relacionados que pueden ser de **distinto tipo**.

Estructuras

Las estructuras son tipos de datos *derivados* que agrupan datos relacionados que pueden ser de **distinto tipo**.

Ej. una estructura que tenga un entero y un char

Estructuras

Estructuras

Definición

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

- `struct` es la palabra reservada para indicar que se define una estructura

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

- dato es la *etiqueta* de la estructura

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

- entre llaves se definen los *miembros* de la estructura (la cantidad que se quiera)

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

- los *miembros* son variables de cualquier tipo, se definen con tipo y nombre, terminan en punto y coma (;)

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

- los *miembros* pueden ser de cualquier tipo, incluso arreglos, punteros u otras estructuras.

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

Estructuras

Definición

Ejemplo

```
struct dato {
  int a;
  char b;
};
```

- la *definición de la estructura* termina con punto y coma (;)

Definición

Definición

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1;

  return 0;
}
```

Inicialización

Inicialización

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3, 2};

  return 0;
}
```

Inicialización

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3, 2};

  return 0;
}
```

- Al igual que en los arreglos se inicializan entre llaves, donde los elementos se asignan en el orden que están definidos dentro de la estructura

Inicialización

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {.y=2};

  return 0;
}
```

Inicialización

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {.y=2};

  return 0;
}
```

- también se puede inicializar un elemento usando el operador punto y el miembro que corresponde. Se conocen como *inicializadores designados*.

Operadores

Precedencia de Operadores (Actualizada)

Operadores

Precedencia de Operadores (Actualizada)

Operador	Asociatividad
`() [] . ->`	Izq. a Der.
`+ - (tipo) ++ -- ! & *`	Der. a Izq.
`* / %`	Izq. a Der.
`+ -`	Izq. a Der.
`< <= > >=`	Izq. a Der.
`== !=`	Izq. a Der.
`&&`	Izq. a Der.
`||`	Izq. a Der.
`? :`	Der. a Izq.
`= += -= /= *= %=`	Der. a Izq.
`,`	Izq. a Der.

Acceso a los miembros de una estructura

Acceso a los miembros de una estructura

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3, 2};

  printf("(%.2f, %.2f)\n", p1.x, p1.y);

  return 0;
}
```

Acceso a los miembros de una estructura

Acceso a los miembros de una estructura

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1;

  p1.x = 3;
  p1.y = 2;

  printf("(%.2f, %.2f)\n", p1.x, p1.y);

  return 0;
}
```

Acceso a los miembros de una estructura

Acceso a los miembros de una estructura

```c
#include <stdio.h>

struct persona {
  int dni;
  char nombre[80];
  float altura;
  float peso;
};

int main (void) {
  struct persona emp = {27377780, "claudio J. Paz"};

  emp.nombre[0] = 'C';
  printf("Nombre: %s", emp.nombre);

  return 0;
}
```

Acceso a los miembros de una estructura

```c
#include <stdio.h>

struct persona {
  int dni;
  char nombre[80];
  float altura;
  float peso;
};

int main (void) {
  struct persona emp = {27377780, "claudio J. Paz"};

  emp.nombre[0] = 'C';
  printf("Nombre: %s", emp.nombre);

  return 0;
}
```

Acceso a los miembros de una estructura

```c
#include <stdio.h>

struct persona {
  int dni;
  char nombre[80];
  float altura;
  float peso;
};

int main (void) {
  struct persona emp = {27377780, "claudio J. Paz"};

  emp.nombre[0] = 'C';
  printf("Nombre: %s", emp.nombre);

  return 0;
}
```

A igual precedencia, con asociatividad desde la izq. primero opera el punto

Operaciones permitidas

Operaciones permitidas

Asignación

Operaciones permitidas

Asignación

Tomar dirección de memoria con &

Operaciones permitidas

Asignación

Tomar dirección de memoria con &

Desreferenciar con *

Operaciones permitidas

Asignación

Tomar dirección de memoria con &

Desreferenciar con *

Acceder a miembros con `.` o `->`

Operaciones permitidas

Asignación

Tomar dirección de memoria con &

Desreferenciar con *

Acceder a miembros con `.` o `->`

Operador `sizeof`

Operaciones permitidas: Asignación

Operaciones permitidas: Asignación

```c
#include <stdio.h>
// punto-3.c

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1, p2 = {3,2};

  p1 = p2;

  printf("(%.2f, %.2f)\n", p1.x, p1.y);

  return 0;
}
```

Operaciones permitidas: Asignación

```c
#include <stdio.h>
// punto-3.c

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1, p2 = {3,2};

  p1 = p2;

  printf("(%.2f, %.2f)\n", p1.x, p1.y);

  return 0;
}
```

```
$ gcc -Wall -std=c99 -pedantic-errors punto-3.c && ./a.out
(3.00, 2.00)
$
```

Operaciones permitidas: Asignación

Operaciones permitidas: Asignación

Se puede hacer asignación si y solo si se trata del mismo tipo de estructuras...

Operaciones permitidas: Asignación

Se puede hacer asignación si y solo si se trata del mismo tipo de estructuras...

...de lo contrario hay error de compilación

Operaciones permitidas: Asignación

Se puede hacer asignación si y solo si se trata del mismo tipo de estructuras...

...de lo contrario hay error de compilación

```
struct punto_2d p1, p2 = {3,2};
struct punto_3d p3;

p3 = p2;
```

Operaciones permitidas: Asignación

Se puede hacer asignación si y solo si se trata del mismo tipo de estructuras...

...de lo contrario hay error de compilación

```
  struct punto_2d p1, p2 = {3,2};
  struct punto_3d p3;

  p3 = p2;
```

```
$ gcc -Wall -std=c99 -pedantic-errors punto-3.c && ./a.out
punto-3.c:18:8: error: incompatible types when assigning to type
                ‘struct punto_3d’ from type ‘struct punto_2d’
   18 |   p3 = p2;
      |        ^~
$
```

Operaciones no permitidas

Operaciones no permitidas

No se permite el uso de los operadores de relación (==, !=, >, <, etc)

Operaciones no permitidas

No se permite el uso de los operadores de relación (==, !=, >, <, etc)

```
struct punto_2d p1, p2 = {3,2};

p1 = p2;

if (p1 == p2)
  printf("Ok!\n");
```

Operaciones no permitidas

No se permite el uso de los operadores de relación (==, !=, >, <, etc)

```c
  struct punto_2d p1, p2 = {3,2};

  p1 = p2;

  if (p1 == p2)
    printf("Ok!\n");
```

```
$ gcc -Wall -std=c99 -pedantic-errors punto-4.c && ./a.out
punto-4.c:14:9: error: invalid operands to binary ==
                (have ‘struct punto_2d’ and ‘struct punto_2d’)
   14 |   if (p1==p2)
      |         ^~
$
```

Punteros a estructuras

Punteros a estructuras

De la misma manera que en variables de tipo escalar se usa el asterisco entre el nombre y el tipo...

Punteros a estructuras

De la misma manera que en variables de tipo escalar se usa el asterisco entre el nombre y el tipo...

...en estructuras se usa el asterisco, pero recordando que el tipo incluye la palabra reservada `struct`

Punteros a estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3,2};

  struct punto_2d *pp;

  pp = &p1;

  printf("%.2f\n", (*pp).x);

  return 0;
}
```

Punteros a estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3,2};

  struct punto_2d *pp;

  pp = &p1;

  printf("%.2f\n", (*pp).x);

  return 0;
}
```

Punteros a estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3,2};

  struct punto_2d *pp;

  pp = &p1;

  printf("%.2f\n", (*pp).x);

  return 0;
}
```

Punteros a estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3,2};

  struct punto_2d *pp;

  pp = &p1;

  printf("%.2f\n", (*pp).x);

  return 0;
}
```

Punteros a estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3,2};

  struct punto_2d *pp;

  pp = &p1;

  printf("%.2f\n", (*pp).x);

  return 0;
}
```

Punteros a estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d p1 = {3,2};

  struct punto_2d *pp;

  pp = &p1;

  printf("%.2f\n", (*pp).x);

  return 0;
}
```

Punteros a estructuras

Punteros a estructuras

Debido al orden de precedencia del operador punto deben usarse paréntesis para que la desreferencia del puntero se realice primero.

Punteros a estructuras

Debido al orden de precedencia del operador punto deben usarse paréntesis para que la desreferencia del puntero se realice primero.

De lo contrario el compilador da un error, ya que el operador punto espera una estructura y un miembro al que acceder, no un puntero.

Punteros a estructuras

Punteros a estructuras

Para simplificar la notación y disminuir la posibilidad de errores se usa el operador *flecha* (->)

Punteros a estructuras

Para simplificar la notación y disminuir la posibilidad de errores se usa el operador *flecha* (->)

En lugar de

```
printf("%.2f\n", (*pp).x);
```

Punteros a estructuras

Para simplificar la notación y disminuir la posibilidad de errores se usa el operador *flecha* (->)

En lugar de

```
printf("%.2f\n", (*pp).x);
```

se puede usar

```
printf("%.2f\n", pp->x);
```

Punteros a estructuras

Para simplificar la notación y disminuir la posibilidad de errores se usa el operador *flecha* (->)

En lugar de

```
printf("%.2f\n", (*pp).x);
```

se puede usar

```
printf("%.2f\n", pp->x);
```

El operador flecha espera un miembro a la derecha y un puntero a una estructura a la izquierda

Arreglos de estructuras

Arreglos de estructuras

Para definir un arreglo de estructuras simplemente se usa el corchete en el nombre de la variable

Arreglos de estructuras

Para definir un arreglo de estructuras simplemente se usa el corchete en el nombre de la variable

```
struct punto_2d puntos[10];
```

Arreglos de estructuras

Para definir un arreglo de estructuras simplemente se usa el corchete en el nombre de la variable

```
struct punto_2d puntos[10];
```

En el ejemplo se crea un arreglo de 10 elementos, cada uno de tipo `struct punto_2d`

Arreglos de estructuras

Para definir un arreglo de estructuras simplemente se usa el corchete en el nombre de la variable

```
struct punto_2d puntos[10];
```

En el ejemplo se crea un arreglo de 10 elementos, cada uno de tipo `struct punto_2d`

Se accede a los miembros de cada estructura, primero a través del uso del índice que corresponde en el arreglo

Arreglos de estructuras

Arreglos de estructuras

```c
#include <stdio.h>

struct punto_2d {
  float x;
  float y;
};

int main (void) {
  struct punto_2d puntos[5] = {{0,0},{6,7}};
  int i;

  puntos[0].x = 3; puntos[0].y = 3;
  puntos[3].x = 4; puntos[3].y = 5;

  for(i=0; i<5 ; i++)
    printf("(%.2f, %.2f)\n", puntos[i].x, puntos[i].y);

  return 0;
}
```

Arreglos de estructuras

Arreglos de estructuras

En el caso de

```
puntos[0].x = 3;
```

se opera primero el corchete por estar más a la izq.

Arreglos de estructuras

En el caso de

```
puntos[0].x = 3;
```

se opera primero el corchete por estar más a la izq.

La variable `puntos` antes que nada es un arreglo por lo que el elemento se accesa con los corchetes.

Arreglos de estructuras

En el caso de

```
puntos[0].x = 3;
```

se opera primero el corchete por estar más a la izq.

La variable `puntos` antes que nada es un arreglo por lo que el elemento se accesa con los corchetes.

Luego, el elemento accesado con los corchetes es de tipo `struct puntos_2d`, por lo que entonces se puede accesar al miembro de la estructura con el operador punto.

Arreglos asignados dinámicamente

Arreglos asignados dinámicamente

```c
#include<stdio.h>
#include<stdlib.h>

struct personal {
  int dni;
  char nombre[80];
  int legajo;
};

int main (void)
{
  struct personal *p;
  int i, n=10;

  p = malloc (n*sizeof (struct personal));

  for(i = 0; i < n; i++) {
    printf("Ingrese Nombre: "); scanf(" %80[^\n]s", (p+i)->nombre);
    printf("Ingrese DNI: "); scanf("%d", &(p+i)->dni);
    printf("Ingrese Legajo: "); scanf("%d", &(p+i)->legajo);
  }

  // continua
```

Arreglos asignados dinámicamente

```
  for(i = 0; i < n; i++) {
    printf("Nombre: %s\n", (p+i)->nombre);
    printf("DNI: %d\n", (p+i)->dni);
    printf("Legajo: %d\n", (p+i)->legajo);
  }

  free(p);
  return 0;
}
```