Programación de Computadores

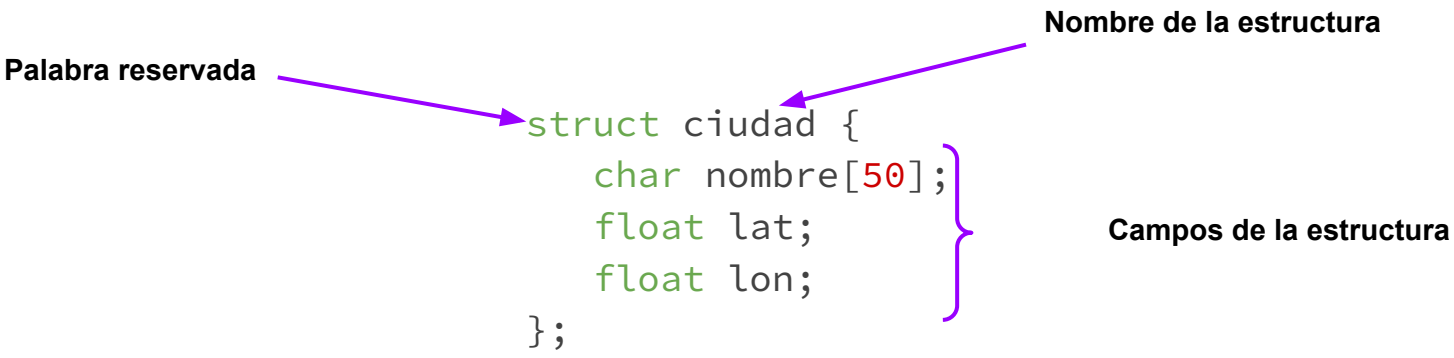
Tema 13: Estructuras y uniones

Carrera Ingeniería Civil en Informática
y Ciencias de la Computación
Universidad de Concepción

José Fuentes - jfuentess@inf.udec.cl

Estructuras (struct)

Una colección de variables relacionadas, agrupadas bajo un mismo nombre



Estructuras: en contexto

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

struct ciudad {
   char nombre[50];
   float lat;
   float lon;
};

void print_city(struct ciudad c) {
   printf("%s (%f, %f)\n", c.nombre, c.lat, c.lon);
}

int main() {
   struct ciudad c1;
   strcpy(c1.nombre, "Concepcion");
   c1.lat = -36.8271;
   c1.lon = -73.0503;

   print_city(c1);
   return 0;
}
```

Acceso a los campos de la estructura

Declaración de una variable de tipo struct ciudad

Ver: struct_ciudad.c

Estructuras: en contexto

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

struct fecha {
   int d; // Día
   int m; // Mes
   int a; // Año
};

void print_fecha(struct fecha f) {
   printf("%d/%d/%d\n", f.d, f.m, f.a);
}

struct fecha siguiente_dia1(struct fecha f) {
   f.d = 1 + (f.d % 30); // Asumiendo meses de 30 días
   if(f.d == 1) {
      f.m = 1 + (f.m % 12);
      if(f.m == 1)
         f.a++;
   }
   return f;
}
```

Ver: struct_fecha.c

```c
void siguiente_dia2(struct fecha *f) {
   // Nota: f->d es equivalente a (*f).d
   f->d = 1 + (f->d % 30);
   if(f->d == 1) {
      f->m = 1 + (f->m % 12);
      if(f->m == 1)
         f->a++;
   }
}

int main() {
   struct fecha hoy = {29, 11, 2021};
   print_fecha(hoy);
   struct fecha sig = siguiente_dia1(hoy);
   print_fecha(sig);
   siguiente_dia2(&sig);
   print_fecha(sig);

   return 0;
}
```

Salida

```
> 29/11/2021
> 30/11/2021
> 1/12/2021
```

Estructuras: mezclando estructuras

```c
struct fecha {
    int d; // Día
    int m; // Mes
    int a; // Año
};

struct ciudad {
    char nombre[50];
    float lat;
    float lon;
    // Fundación
    struct fecha f;
};
```

Ver: struct_ciudad_fecha.c

```c
void print_city(struct ciudad c) {
    printf("%s (%f, %f). Fundada el %d/%d/%d.\n",
                            c.nombre, c.lat,
                            c.lon, c.f.d,
                            c.f.m, c.f.a);
}

int main() {
    struct ciudad c1;
    strcpy(c1.nombre, "Concepcion");
    c1.lat = -36.8271;
    c1.lon = -73.0503;
    c1.f.d = 27;
    c1.f.m = 9;
    c1.f.a = 1544;

    print_city(c1);
    return 0;
}
```

Estructuras: arreglos

```c
struct fecha {
   int d; // Día
   int m; // Mes
   int a; // Año
};

struct fecha fechas[5];
fechas[0].d = 29;
fechas[0].m = 11;
fechas[0].a = 2021;

fechas[1].d = 30;
fechas[1].m = 11;
fechas[1].a = 2021;
...
```

vs

```c
int d[5]; // Día
int m[5]; // Mes
int a[5]; // Año

d[0] = 29;
m[0] = 11;
a[0] = 2021;

d[1] = 30;
m[1] = 11;
a[1] = 2021;
...
```

Typedef

```c
struct fecha {
   int d; // Día
   int m; // Mes
   int a; // Año
};

typedef struct fecha fecha_t;

fecha_t f1 = {11, 11, 2000};
struct fecha f2 = {11, 11, 2000};
```

```c
typedef unsigned char byte;
typedef int int32_t;
typedef long int64_t;
typedef unsigned int uint32_t;
typedef unsigned long uint64_t;


byte c = 64; // Equivalente a @ en ASCII
int32_t x = -10;
uint32_t y = 55;
uint64_t z = 34359738368;
```

Ver: struct_typedef.c

Padding y packing

```
struct X {
   char c; // 1 byte
   /* Padding: 3 bytes */
   int d;  // 4 bytes
   char e; // 1 byte
   /* Padding: 3 bytes */
};
```

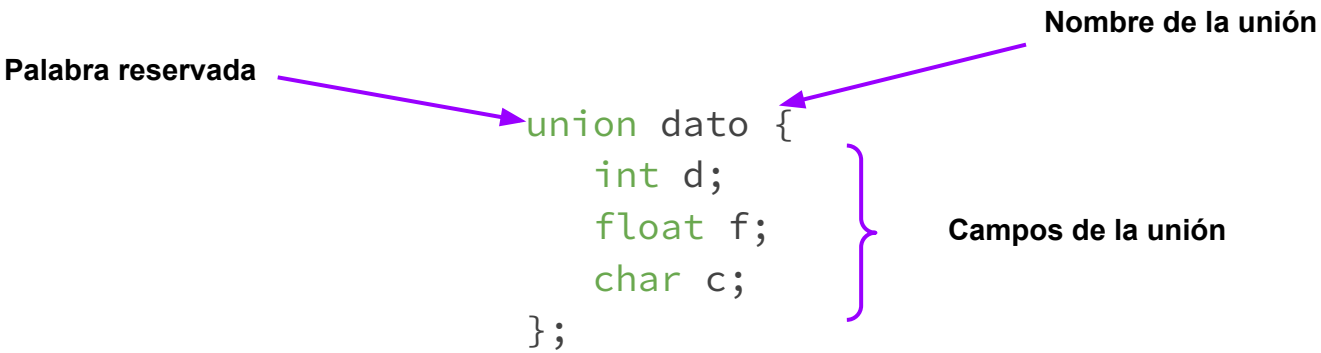
```
struct Y {
   char c; // 1 byte
   char e; // 1 byte
   /* Padding: 2 bytes */
   int d;  // 4 bytes
};
```

```
struct __attribute__((__packed__)) Z {
   char c; // 1 byte
   char e; // 1 byte
   int d;  // 4 bytes
};
```

Ver: struct_padding.c

Unión (union)

Una colección de variables relacionadas, agrupadas bajo un mismo nombre. Similar a *struct*, pero en una unión sólo un campo puede tener un valor válido a la vez.



Uniones proveen una manera de manipular diferentes tipos de datos en una misma área de memoria.

Union: en contexto

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

union dato {
   int d;
   float f;
   char c;
   char s[8];
};

void print_dato(union dato v) {
   printf("%d, %f, %c, %s\n", v.d, v.f, v.c, v.s);
}

int main() {
   union dato var;
   var.d = 103;
   print_dato(v);
   printf("Tamaño: %ld bytes\n", sizeof(union dato)); // 8 bytes
   return 0;
}
```

Acceso a los campos de la unión

Declaración de una variable de tipo *union dato*

Ver: union.c

Enumeraciones: enum

Palabra reservada

Nombre del enum

Campos de enum (Todos constantes)

Declaración de variables enum

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

enum estado {ACTIVO, INACTIVO};
enum rating {EXCELENTE=5, BUENA=4, NORMAL=3, BAJA=2, MALA=1};
enum semana {LU, MA, MI, JU=10, VI, SA, DO};

int main() {
   enum estado e_actual;
   enum rating calidad;
   enum semana dia;

   e_actual = INACTIVO;
   calidad = NORMAL;
   dia = LU;

   printf("%d %d\n", ACTIVO, INACTIVO);
   printf("%d %d %d %d %d\n", EXCELENTE, BUENA, NORMAL, BAJA, MALA);
   printf("%d %d %d %d %d %d %d\n", LU, MA, MI, JU, VI, SA, DO);

   return 0;
}
```

Ver: enum.c

Union vs Struct

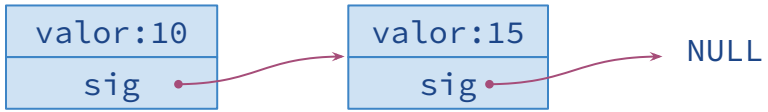
```c
typedef union u_dato t_u;

union u_dato {
   int d;
   float f;
   char c;
   char s[8];
};

printf("Tamaño: %ld bytes\n",
     sizeof(t_u)); // 8 bytes
```

```c
typedef struct s_dato t_s;

struct s_dato {
   int d;
   float f;
   char c;
   char s[8];
};

printf("Tamaño: %ld bytes\n",
     sizeof(t_s)); // 20 bytes
```

Ver: cmp_union_struct.c

Struct: listas enlazadas

```c
struct nodo {
   int valor;
   struct nodo sig;
};
```

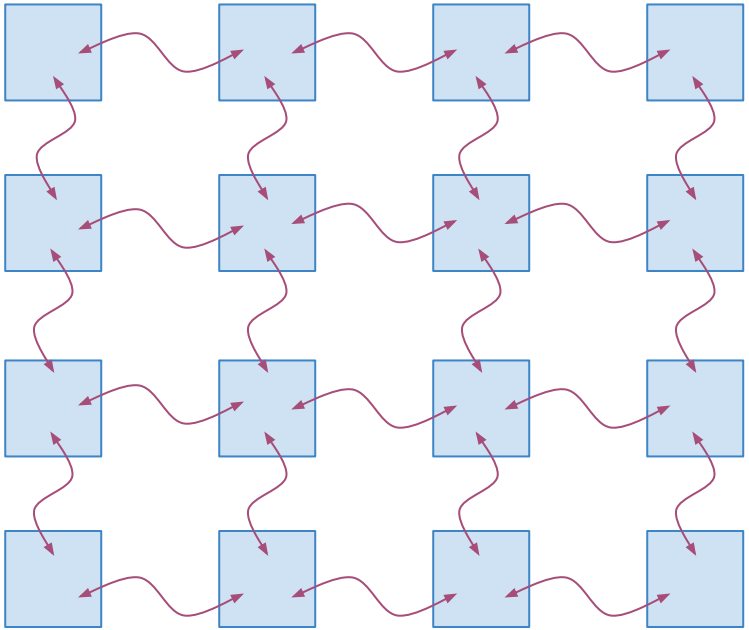



```c
int main() {
   struct nodo nodo1;
   struct nodo nodo2;

   nodo1.valor=10;
   nodo1.sig = &nodo2;

   nodo2.valor=15;
   nodo2.sig = NULL;

   return 0;
}
```

Ver: lista_simple.c

¿Cómo implementar este patrón?

Opción 1: `arreglos 2D`
Ventajas vs desventajas



Opción 2: `struct + punteros`
Ventajas vs desventajas

¡A practicar!

Ejemplo 1:
lista_completa.c

Ejemplo 2:
tienda_juegos.c

Ejemplo 3:
lista_doble.c

Ejemplo 4:
votacion.c

Ejemplo 5:
tienda_juegos.c

Ejemplo 6:
poligonos.c