

Intro a C Structs

With ❤ by @vichoeq & @KnowYourselfs

Clases y Herencia



?



```
class Slime(Enemy):  
    def __init__(self):  
        self.hit_points = 20  
        self.attack_dmg = 4  
  
    ...
```

Clases y Herencia



```
class Slime(Enemy):  
    def __init__(self):  
        self.hit_points = 20  
        self.attack_dmg = 4  
  
    ...
```

struct

struct - Contenedores de variables



```
struct perro
{
    char* nombre;
    char* raza;
    double edad;
    bool good_boy;
};
```



```
from collections import namedtuple

perro = namedtuple(
    'nombre',
    'raza',
    'edad',
    'good_boy'
)
```

struct - Sintaxis



```
struct nombre_del_struct
{
    type_1 campo_1;
    type_2 campo_2;
    ...
    type_n campo_n;
};
```

Un *struct* es una tupla de n **variables** de *tipos* distintos, cada una con su propio nombre.

Cada una de estas **variables** es un **campo** del *struct*.

struct - Contenedores de variables



```
struct perro
{
    char* nombre;
    char* raza;
    double edad;
    bool good_boy = true;
};
```



No se pueden definir valores default
para los **campos**

struct y el STACK

struct y el **STACK** - Inicialización



```
struct point
{
    double x;
    double y;
};
```

```
struct point p = {5.12, 2.78};
struct point q = {.x = 5.12, .y = 2.78};
struct point r = {
    .x = 5.12,
    .y = 2.78
};
```

Los **campos** de un *struct* están contiguos en memoria.

La sintaxis de inicialización es igual que en un **arreglo**.

Se puede especificar el valor a asignar a cada **campo**, por nombre.

struct y el **STACK** - Asignando un **literal**



```
struct point p;  
p = {.x = 7.18, .y = 2.78};
```

```
$ gcc main.c -o main  
error: expected expression before '{' token
```



```
struct point p;  
p = (struct point){.x = 7.18, .y = 2.78};
```

```
$ gcc main.c -o main  
$ ./main
```

struct y el STACK - acceso



```
struct point
{
    double x;
    double y;
};
```

```
struct point p = {.x = 7.18, .y = 2.52};

if(p.x < 10)
{
    p.y *= 2;
}
```

Para acceder a cada campo se usa un punto.

struct y el **STACK** - asignación



```
struct point
{
    double x;
    double y;
};
```

```
struct point p;

p.x = 7.18;
p.y = 2.52;
```

Podemos asignar el struct **campo** por **campo** usando el punto.

struct y typedef



```
struct point
{
    double x;
    double y;
};

typedef struct point Point;
```

```
Point A[2];
A[0] = (Point){.x = 1.26, .y = 25.3};
A[1] = (Point){.x = 0.54, .y = 1.55};
```

Podemos utilizar **typedef** para definir un **alias** sin **struct**, así nos ahorramos esos preciosos 7 caracteres.

Para los siguientes ejemplos usaremos **Point** aquí definido.

struct y typedef



```
typedef struct point
{
    double x;
    double y;
} Point;
```

También podemos usar **typedef** junto con la definición del **struct**

Esto es posible ya que el **struct** entero es un **tipo**

struct y typedef



```
typedef struct point
{
    double x;
    double y;
} Point;
```

También podemos usar **typedef** junto con la definición del **struct**

Esto es posible ya que el **struct** entero es un **tipo**

struct*

*struct** y el **HEAP** - Inicialización



```
Point* a = malloc(sizeof(Point));  
*a = (Point){.x = 0.25, .y = 0.56 };
```

Hay que pedir memoria para el *struct* mediante **malloc** y **sizeof**.

*struct** - acceso



```
Point* a = malloc(sizeof(Point));
*a = (Point){.x = 0.25, .y = 0.56 };

if(p->x < 10)
{
    p->y *= 2;
}
```

Para **acceder** a cada **campo** se usa la flecha "**->**".

*struct** - asignación



```
struct point
{
    double x;
    double y;
};
```

```
struct point p = malloc(sizeof(Point));

p->x = 7.18;
p->y = 2.52;
```

Podemos asignar el *struct* campo por campo usando la flecha.

struct.struct

struct.struct



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct kid
{
    char* name;
    struct dog dog;
};
```

```
struct kid matias = {
    .name = "matías",
    .dog = {
        .name = "bruno",
        .age = 3,
        .good_boy = true
    }
};
```

C nos permite definir *struct* que contienen *struct*.

La inicialización de un *struct* dentro de un *struct* es igual.

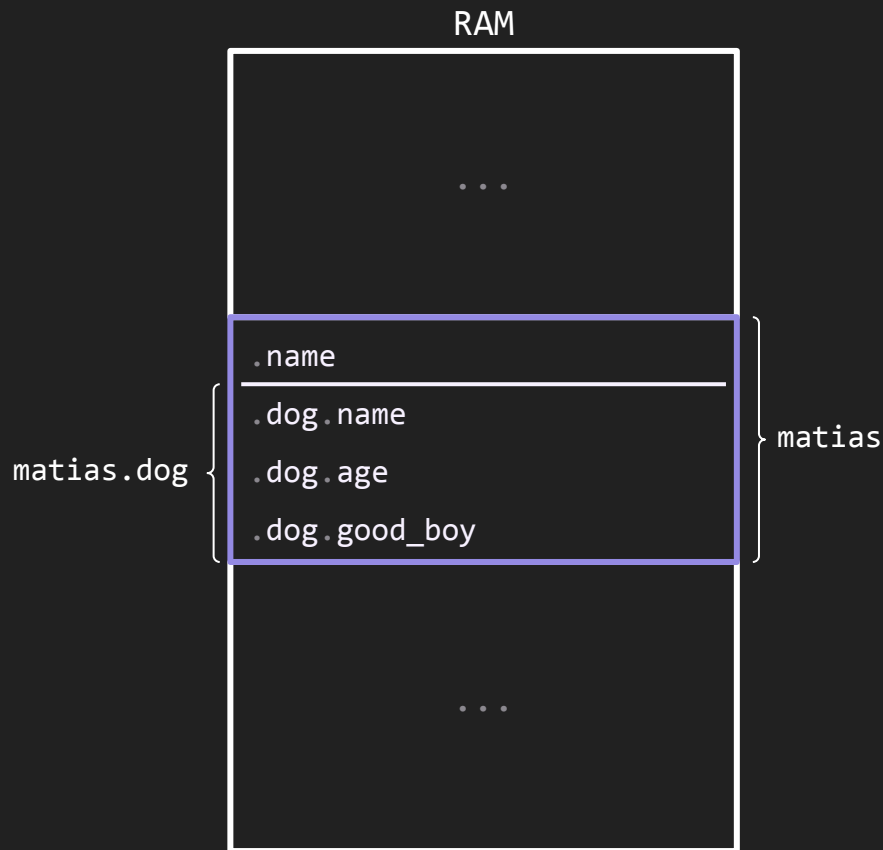
struct.struct



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct kid
{
    char* name;
    struct dog dog;
};
```

```
struct kid matias = {
    .name = "matías",
    .dog = {
        .name = "bruno",
        .age = 3,
        .good_boy = true
    }
};
```



struct.struct



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct kid
{
    char* name;
    struct dog dog;
};
```

```
struct kid matias = {
    .name = "matías",
    .dog = {
        .name = "bruno",
        .age = 3,
        .good_boy = true
    }
};
```



struct.struct



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct kid
{
    char* name;
    struct dog dog;
};
```

```
struct kid matias = {
    .name = "matías",
    .dog = {
        .name = "bruno",
        .age = 3,
        .good_boy = true
    }
};
```



struct.struct



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct kid
{
    char* name;
    struct dog dog;
};
```

```
struct kid matias = {
    .name = "matías",
    .dog = {
        .name = "bruno",
        .age = 3,
        .good_boy = true
    }
};
```



struct.struct



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct kid
{
    char* name;
    struct dog dog;
};
```

```
struct kid matias = {
    .name = "matías",
    .dog = {
        .name = "bruno",
        .age = 3,
        .good_boy = true
    }
};
```



*struct.struct**

struct.struct*



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```

C nos permite definir *struct* que contienen *struct**.

La inicialización de un *struct** dentro de un *struct* requiere el uso de *malloc/calloc*.

Se podría en el **STACK**, pero no es recomendado.

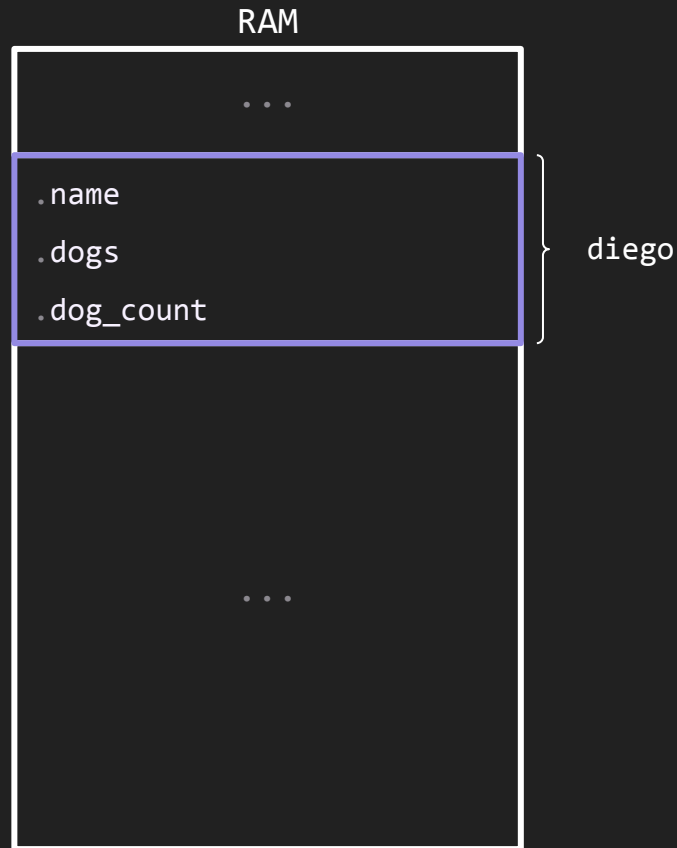
struct.struct*



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```



struct.struct*



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```

RAM

...

```
.name = "diego"
.dogs
.dog_count
```

} diego

...

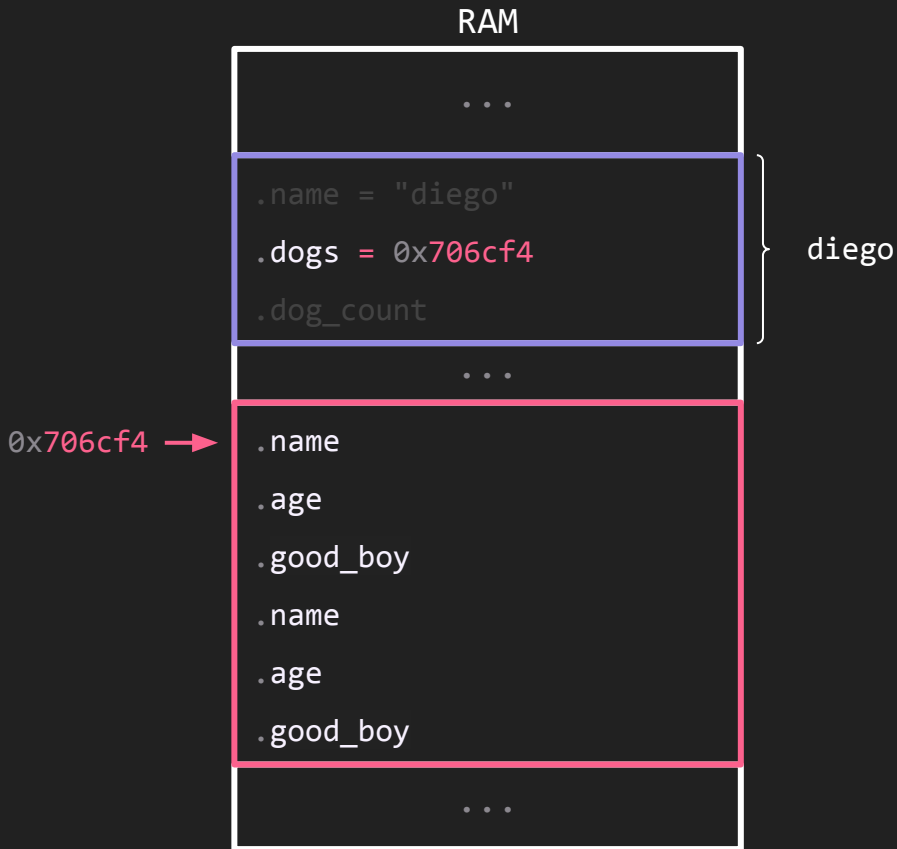
struct.struct*



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};

struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```



struct.struct*

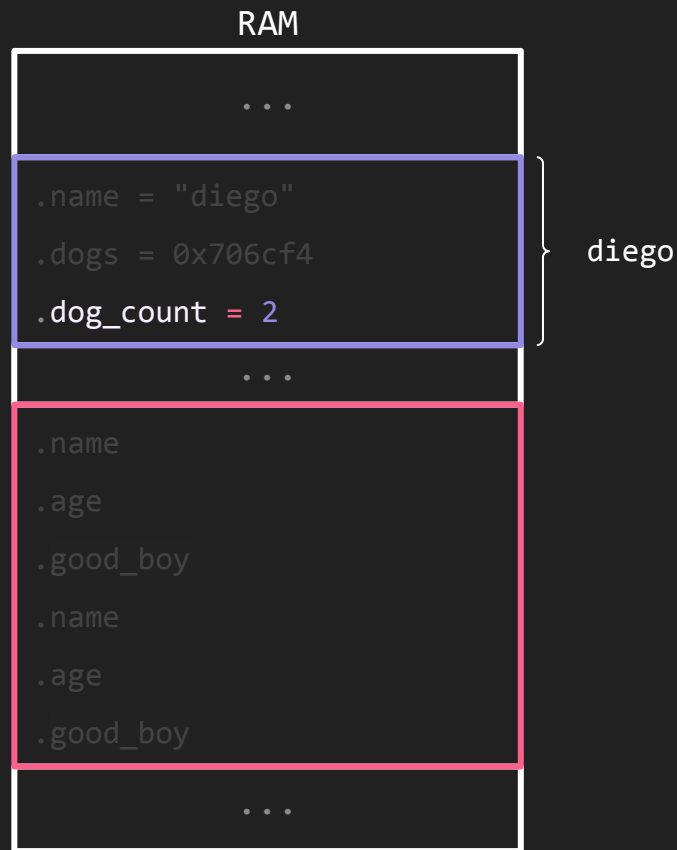


```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};

struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```

0x706cf4 →



struct.struct*

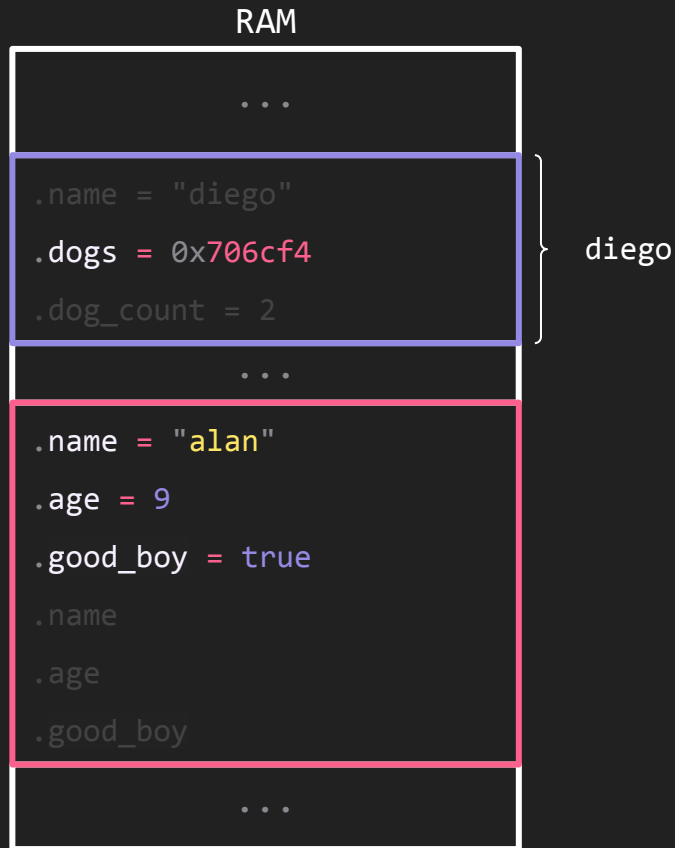


```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```

0x706cf4 →



struct.struct*



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
};
```

```
struct adult
{
    char* name;
    struct dog* dogs;
    int dog_count;
};
```

```
struct adult diego = {
    .name = "diego",
    .dogs = calloc(2, sizeof(struct dog)),
    .dog_count = 2
};
diego.dogs[0] = (struct dog) {"alan", 9, true};
diego.dogs[1] = (struct dog) {"casi", 5, true};
```

RAM

...

```
.name = "diego"
.dogs = 0x706cf4
.dog_count = 2
```

} diego

0x706cf4 →

...

```
.name = "alan"
.age = 9
.good_boy = true
.name = "casi"
.age = 5
.good_boy = true
```

...

struct recursivo

struct recursivo



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog friend;
};
```

RAM

?

struct recursivo



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog friend;
};
```



RAM

...

```
.name
.age
.good_boy
.friend.name
.friend.age
.friend.good_boy
.friend.friend.name
.friend.friend.age
.friend.friend.good_boy
.friend.friend.friend.name
.friend.friend.friend.age
```

struct recursivo



```
struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
};
```

```
struct dog A = {"bruno", 3, true, NULL};
struct dog B = {"catalina", 4, true, NULL};
```

```
A.friend = &B;
B.friend = &A;
```

RAM

0x62df70 →

```
A.name = "bruno"
A.age = 3
A.good_boy = true
A.friend = 0x7d5ad4
B.name = "catalina"
B.age = 4
B.good_boy = true
B.friend = 0x62df70
```

0x7d5ad4 →

struct recursivo - *typedef*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    Dog* friend;
} Dog;
```



```
struct dog;
typedef struct dog Dog;

Dog
{
    char* name;
    double age;
    bool good_boy;
    Dog* friend;
};
```

Programar con *struct*

Modelación orientada a objetos

Modelación orientada a *objetos*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
} Dog;

Dog* dog_init(char* name){ ... }

void dog_bark(Dog* dog){ ... }

void dog_meet(Dog* dog, Dog* friend){ ... }

void dog_destroy(Dog* dog){ ... }
```

Es lo más cercano a *clases* que podemos hacer en **C**.

Modelación orientada a *objetos*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
} Dog;
```

```
Dog* dog_init(char* name){ ... }
```

```
void dog_bark(Dog* dog){ ... }
```

```
void dog_meet(Dog* dog, Dog* friend){ ... }
```

```
void dog_destroy(Dog* dog){ ... }
```

Definimos el *struct*.

Modelación orientada a *objetos*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
} Dog;

Dog* dog_init(char* name){ ... }

void dog_bark(Dog* dog){ ... }

void dog_meet(Dog* dog, Dog* friend){ ... }

void dog_destroy(Dog* dog){ ... }
```

Definimos la función que inicializa un nuevo *struct* en el **HEAP** con *malloc/calloc*.

Modelación orientada a *objetos*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
} Dog;

Dog* dog_init(char* name){ ... }

void dog_bark(Dog* dog){ ... }

void dog_meet(Dog* dog, Dog* friend){ ... }

void dog_destroy(Dog* dog){ ... }
```

Definimos las **funciones** que usan o procesan el *struct*.

Modelación orientada a *objetos*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
} Dog;

Dog* dog_init(char* name){ ... }

void dog_bark(Dog* dog){ ... }

void dog_meet(Dog* dog, Dog* friend){ ... }

void dog_destroy(Dog* dog){ ... }
```

Las **funciones** definen el **comportamiento** del **objeto**.

Modelación orientada a *objetos*



```
typedef struct dog
{
    char* name;
    double age;
    bool good_boy;
    struct dog* friend;
} Dog;

Dog* dog_init(char* name){ ... }

void dog_bark(Dog* dog){ ... }

void dog_meet(Dog* dog, Dog* friend){ ... }

void dog_destroy(Dog* dog){ ... }
```

Definimos la función que hace **free** de la **memoria** usada por una instancia del *struct*.

Estructuras de datos

Estructuras de datos



```
struct list
{
    int value;
    struct list* next;
};

typedef struct list List;

List* list_init(){ ... }

List* list_append(List* list){ ... }

List* list_at_index(List* list, int index){ ... }

void list_destroy(List* list){ ... }
```

Sirven para organizar datos en la memoria del programa.

Estructuras de datos



```
struct list
{
    int value;
    struct list* next;
};

typedef struct list List;

List* list_init(){ ... }

List* list_append(List* list){ ... }

List* list_at_index(List* list, int index){ ... }

void list_destroy(List* list){ ... }
```

Las **funciones** definen las **operaciones** de la estructura.

tipos compuestos

tipos compuestos



```
typedef struct vector3d
{
    double x;
    double y;
    double z;
} Vec3D;

Vec3D vec3d_create(double x, double y, double z)
{ ... }

double vec3d_dot_prod(Vec3D v1, Vec3D v1)
{ ... }

Vec3D vec3d_cross_prod(Vec3D v1, Vec3D v1)
{ ... }
```

Estos tipos son *tipos* en los que no nos interesa usar *referencias*.

Actúan literalmente como contenedores, sin ninguna propiedad ni significado.

tipos compuestos



```
typedef struct vector3d
{
    double x;
    double y;
    double z;
} Vec3D;

Vec3D vec3d_create(double x, double y, double z;)
{ ... }

double vec3d_dot_prod(Vec3D v1, Vec3D v1)
{ ... }

Vec3D vec3d_cross_prod(Vec3D v1, Vec3D v1)
{ ... }
```

Las *funciones* son *operadores* del *struct*

¡Muchas Gracias!

instantiating a class or
something idk i use C



With ♥ by @vichoeq & @KnowYourselfs