

```
#include <stdio.h>
struct car
{
    int size;
    char *name;
};
int main()
{
    struct car car1;

    car1.name = "bmw";
    car1.size = 10;

    struct car car2;

    car2.name = "ferrari";
    car2.size = 20;

    printf(" car1 %d, %s\n",car1.size,car1.name);
    printf(" car2 %d, %s\n",car2.size,car2.name);
    return 0;
}
```

in this code is struct car does not loading in memory until you make a variable of type struct car car1
I mean now var store in memory have the name size or name only car1.size and car1.name

please explain what happen when I init struct and make variable of his type in memory layout

In C, when you define a `struct`, you are essentially creating a blueprint for a compound data type that encapsulates multiple other data types. This `struct` definition by itself does not cause any memory allocation. It merely informs the compiler about the new data type and how it should be structured in memory once an instance of it is created.

Here is the breakdown of what happens when you define the `struct` and then create instances of it:

- 1 `struct car` Definition: When you define `struct car`, the compiler understands that any variable of type `struct car` will have two members: an `int` named `size` and a `char*` named `name`. It does not allocate memory at this point; it only knows the size that will be required for each instance of `struct car`.
- 2 Declaration of `car1`: When you declare `struct car car1`;, the compiler allocates memory for `car1`. The amount of memory allocated is enough to store an `int` (for `size`) and a `char*` (for `name`). These will be contiguous in memory, with `car1.size` directly followed by `car1.name`.
- 3 Initialization of `car1`: When you write `car1.name = "bmw"`; and `car1.size = 10`;, you are storing the value `10` in the memory location reserved for `car1.size` and storing the address of the string literal `"bmw"` in the memory reserved for `car1.name`.
- 4 Declaration of `car2`: Similarly, when `struct car car2`; is declared, another block of memory is allocated, separate from `car1`, but of the same size and structure. Again, you have memory for an `int` and a `char*`.
- 5 Initialization of `car2`: The assignments `car2.name = "ferrari"`; and `car2.size = 20`; put the value `20` in the memory for `car2.size` and the address of the string literal `"ferrari"` in the memory for `car2.name`.

So in memory, you would have two separate areas allocated for `car1` and `car2`, each with their `size` and `name`. Here is a simplified view of how it might look in memory:

```
car1:
+-----+-----+
| size | name |
+-----+-----+
|  10  | address of "bmw" |
+-----+-----+

car2:
+-----+-----+
| size | name |
+-----+-----+
|  20  | address of "ferrari" |
+-----+-----+
```

The string literals themselves (`"bmw"` and `"ferrari"`) are typically stored in a different part of the memory that is read-only and is used for constant data. The `name` members of `car1` and `car2` are pointers that refer to these locations.

Note that the actual layout in memory may include padding or other compiler-specific optimizations or arrangements, but logically this is how you can envision it.

From ChatGPT, designed with  FancyGPT