Char array in a struct - incompatible assignment? [duplicate]

Asked 10 years, 2 months ago Active 11 months ago Viewed 110k times

32

11

This question already has an answer here:
Structure Problem in C *12 answers*

I tried to find out what a struct really 'is' and hit a problem, so I have really 2 questions:

1) What is saved in 'sara'? Is it a pointer to the first element of the struct?

2) The more interesting question: Why doesn't it compile? GCC says "test.c:10: error: incompatible types in assignment" and I can't figure out why... (This part has been solved by your answers already, great!)

```
#include <stdio.h>

struct name {
    char first[20];
    char last[20];
};

int main() {
    struct name sara;
    sara.first = "Sara";
    sara.last = "Black";
    printf("struct direct: %x\n",sara);

    printf("struct deref: %x\t%s\n", *sara, *sara);


}
```

Thanks for your help!

c struct char

edited Aug 18 '09 at 8:45

asked Aug 18 '09 at 8:34

Patrick
3,250 4 32 63

marked as duplicate by harper, Jean-François Fabre ♦ c Nov 27 '18 at 20:22

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

8 Answers

63

This has nothing to do with structs - arrays in C are not assignable:

```
char a[20];
a = "foo";   // error
```

you need to use strcpy:

```
strcpy( a, "foo" );
```

or in your code:

```
strcpy( sara.first, "Sara" );
```

answered Aug 18 '09 at 8:37
anon

4 +1 For `strcpy` over `strncpy` ... :) – GManNickG Aug 18 '09 at 8:49

I would have used char* first; sara.first = "Sara", since I used fixed size arrays only when it's really required; Is it a good or bad practice (in C, not C++)? – gramm Aug 18 '09 at 9:06

1 @gramm If you do that, you are committing yourself to dynamic memory management, which may not be necessary or desirable. – anon Aug 18 '09 at 9:55

@gramm There are plenty of cases where an array is much more well-behaved. With `char *`, `sara.first="Sara"; sara.first[0]='s';` would die horrible because the string `"Sara"` was in read-only memory. – Dave Dec 14 '11 at 5:02

2 @GManNickG Could you elaborate on why `strncpy` is bad? – Nic Hartley May 12 '16 at 22:50

10

Or you could just use dynamic allocation, e.g.:

```
struct name {
  char *first;
  char *last;
};

struct name sara;
sara.first = "Sara";
sara.last = "Black";
printf("first: %s, last: %s\n", sara.first, sara.last);
```

answered Sep 14 '14 at 12:12

Scott Gardner
7,015 1 31 31

use `strncpy` to make sure you have no buffer overflow.

4

```
char name[]= "whatever_you_want";
strncpy( sara.first, name, sizeof(sara.first)-1 );
sara.first[sizeof(sara.first)-1] = 0;
```

edited Aug 18 '09 at 8:51

answered Aug 18 '09 at 8:40

TimW
7,534 1 24 29

1 This assumes that silently truncating the data doesn't cause your program to fail somewhere else down the line. It's usually better to explicitly handle the case where the string doesn't fit in the buffer, than either to invoke undefined behaviour with strcpy, or put your program in an unexpected state with strncpy. But I generalise - if the rest of your program accounts for silent truncation (e.g. never expects that sara.first will match "whatever_you_want") then the program state is not unexpected. – Steve Jessop Aug 18 '09 at 11:17

1 @onebyone this behavior must be documented in the contract. – TimW Aug 18 '09 at 11:31

If that's sufficient, then why not just document that the input string must be no longer than the struct can cope with? The problem is one of designing APIs which help clients avoid errors. IMO, silently ignoring what is probably an error does not do this. But as I say, I'm generalising, and it might not be an error. If the function is called "StoreFirst20CharsOf", then we can reasonably assume that anyone glancing at the calling code will understand what it does. So there are a few uses for strncpy. – Steve Jessop Aug 18 '09 at 11:35

StoreFirst20CharsOf is definitely the way to do it ... – TimW Aug 18 '09 at 11:47

@SteveJessop: Simply documenting that the input string must not exceed a particular length leaves the possibility for a buffer overflow. IMHO, it's better to document that the API will truncate names longer than 20 characters, and not introduce a buffer overflow vulnerability. And the truncation need not be silent; the API could signal the caller that truncation occurred, if the caller cares. While truncation may *sometimes* result in a bug, a buffer overflow is *always* a bug (and one of the worst kind). – Dan Moulding Feb 28 '13 at 14:07

You can also initialise it like this:

4

```
struct name sara = { "Sara", "Black" };
```

Since (as a special case) you're allowed to *initialise* char arrays from string constants.

Now, as for what a struct actually is - it's a compound type composed of other values. What `sara` actually looks like in memory is a block of 20 consecutive char values (which can be referred to using `sara.first`, followed by 0 or more padding bytes, followed by another block of 20 consecutive char values (which can be referred to using `sara.last`). All other instances of the `struct name` type are laid out in the same way.

In this case, it is very unlikely that there is any padding, so a `struct name` is just a block of 40 characters, for which you have a name for the first 20 and the last 20.

You can find out how big a block of memory a `struct name` takes using `sizeof(struct name)`, and you can find out where within that block of memory each member of the structure is placed at


200k 32 264 403

2

`sara` is the struct itself, not a pointer (i.e. the variable representing location on the stack where *actual struct data* is stored). Therefore, `*sara` is meaningless and won't compile.

answered Aug 18 '09 at 8:51


Pavel Minaev
87.9k 23 199 271

ah, ok... Thanks – Patrick Aug 18 '09 at 9:13

0

The Sara structure is a memory block containing the variables inside. There is nearly no difference between a classic declarations :

```
char first[20];
int age;
```

and a structure :

```
struct Person{
char first[20];
int age;
};
```

In both case, you are just allocating some memory to store variables, and in both case there will be 20+4 bytes reserved. In your case, Sara is just a memory block of 2x20 bytes.

The only difference is that with a structure, the memory is allocated as a single block, so if you take the starting address of Sara and jump 20 bytes, you'll find the "last" variable. This can be useful sometimes.

check http://publications.gbdirect.co.uk/c_book/chapter6/structures.html for more :) .

answered Aug 18 '09 at 9:18


gramm
9,274 6 22 24

Sara is a memory block of *at least* 2 * 20 bytes. It might be more, if there's padding. – caf Aug 18 '09 at 9:23

And `offsetof(struct name, last)` is not necessarily 20, either (although it almost certainly is). – caf Aug 18 '09 at 9:24

Didn't know that. Is there a possibility of padding when specifying variable size ? For example, unsigned char speed : 7 ; unsigned char flags : 3; etc ... I'm using this at work and never had a problem with padding and data alignement. – gramm Aug 18 '09 at 9:32

If the bitfield follows a non-bitfield member, then there can be padding *before* it. There isn't allowed to be padding *between* two adjacent bitfields, unless they don't fit into the same "basic storage unit" that's being

later, it's laid out like this" – Dave Dec 14 '11 at 5:07

0

You can use strcpy to populate it. You can also initialize it from another struct.

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

struct name {
    char first[20];
    char last[20];
};

int main() {
    struct name sara;
    struct name other;

    strcpy(sara.first,"Sara");
    strcpy(sara.last, "Black");

    other = sara;

    printf("struct: %s\t%s\n", sara.first, sara.last);
    printf("other struct: %s\t%s\n", other.first, other.last);

}
```

answered Dec 25 '16 at 7:06

Xofo

534 1 9 24

0

You can try in this way. I had applied this in my case.

```
#include<stdio.h>

struct name
{

  char first[20];

  char last[30];

};

//globally

// struct name sara={"Sara","Black"};

int main()

 {

//locally

struct name sara={"Sara","Black"};
```

```
}
```

edited Nov 24 '18 at 5:42

answered Nov 23 '18 at 19:33

MD Rakibuz Sultan

149 1 6