一、前言

昨天在编译代码的时候，之前一直OK的一个地方，却突然出现了好几个 Warning!

本着强迫症要消灭一切警告的做法，最终定位到：是结构体内部， 指向结构体类型的指针成员变量导致的问题。

这个问题，也许永远不会碰到，之所以被我赶上了，应该是因为某个时候手贱， 误碰了键盘导致。

下面一一道来。

PS: 我的测试环境是 Ubuntu16.04-64，编译器使用系统自带的 gcc-5.4.0。

二、问题描述

1. 正常的代码

比较简单：结构体 `struct _Data2_` 的第 2 个成员变量是一个指针，指向的数据类型是结构体 `struct _Data1_`。

```
typedef struct _Data1_
{
    int a;
}Data1;

typedef struct _Data2_
{
    int b;
    struct _Data1_ *next;
}Data2;

int main()
{
    Data1 d1 = {1};
    Data2 d2 = {2, &d1};
```

```
    printf("d1 = %p \n", &d1);
    printf("d2 = %p \n", &d2);

}
```

编译、执行，都没有问题：

```
$ gcc main.c -m32  -o main
$ ./main
d1 = 0xffdc72f0
d2 = 0xffdc72f4
```

2. 错误的代码

现在我们来模拟误碰键盘操作，把 `struct _Data2_` 中 `next` 成员指向的数据类型，改为一个 不存在的结构体：

```
typedef struct _Data2_
{
    int b;
    struct _Data3_ *next;
}Data2;
```

在测试代码中，`struct _Data3_` 肯定是不存在的。

好了，现在执行编译指令 `gcc main.c -m32  -o main`，将会得到什么结果？

可以停下来稍微 思考一下。

我之前的预期是：gcc 会 报错，找不到 `struct _Data3_` 这个类型。

实际情况是：

```
$ gcc main.c -m32  -o main -I./
main.c: In function ‘main’:
main.c:18:20: warning: initialization from incompatible pointer type [-
Wincompatible-pointer-types]
     Data2 d2 = {2, &d1};
                    ^
main.c:18:20: note: (near initialization for ‘d2.next’)
$ ./main
d1 = 0xffd8ee70
d2 = 0xffd8ee74
```

好神奇吧， gcc 居然不报错！那么我们就按照 gcc 的方式来理解一下。

我们知道，编译器在遇到一个结构体类型的时候，最重要的就是需要知道结构体类型 所占据的内存空间的大小。

gcc 在遇到 `struct _Data2_` 这个字符串时，判断出它是一个用户自定义的数据类型：结构体 `_Data2`。

gcc 继续读取结构体内部的每一个字符，在读取到 `*next` 时，知道它是一个 指针。

此时它并并没确认该指针所指向的数据类型是否存在，它只是为 `next` 保留了 4 个字节的内存空间(32位系统)。

然后 gcc 在解析 `Data2 d2 = {2, &d1};` 这一行时，就发现 类型不匹配了：data2 的 next 需要的是 `struct _Data3_` 类型的指针，但是赋值的 d1 是 `struct _Data1_` 类型，于是给出警告信息。

我们用其他的编译器试一下：

(1) clang

```
$ clang main.c -m32  -o main -I./
main.c:18:20: warning: incompatible pointer types initializing 'struct
_Data3_ *' with an expression of type 'Data1 *'
      (aka 'struct _Data1_ *') [-Wincompatible-pointer-types]
    Data2 d2 = {2, &d1};
                   ^~~
1 warning generated.
$ ./main
d1 = 0xffb1b3a0
d2 = 0xffb1b398
```

(2) g++

```
$ g++ main.c -m32  -o main -I./
main.c: In function ‘int main()’:
main.c:18:23: error: cannot convert ‘Data1* {aka _Data1_*}’ to ‘_Data3_*’ in
initialization
     Data2 d2 = {2, &d1};
```

看起来，只有 g++ 进一步确认了 `_Data3_` 这个结构体类型不存在！

三、把类型改为 void 指针类型

把 `struct _Data2_` 中的 `next` 成员，改为 指向 void 型的指针，然后在 `main` 函数中操作它。

```
typedef struct _Data1_
{
    int a;
}Data1;

typedef struct _Data2_
{
    int b;
    void *next;
```

```
}Data2;

int main()
{
    Data1 d1 = {1};
    Data2 d2 = {2, &d1};

    Data1 *dn = d2.next;
    printf("dn->a = %d \n", dn->a);
}
```

编译、执行：

```
$ gcc main.c -m32  -o main -I./
$ ./main
dn->a = 1
```

可以看到： `Data1 *dn = d2.next;` 这一行把指向 void 型的 `d2.next` 赋值给指向`Data1`型的指针变量 `dn`，然后在 `printf` 语句中可以正确地打印出`dn`中的成员变量`a`。

这又回到了指针的本质： 指针就是一个地址，至于如何来解释这个地址中的内容，这是由定义这个指针时所指定的数据类型来决定的

结合代码来看：虽然`d2.next`是一个`void` 型指针，但是它的确存储了一个 地址（变量 d1 的地址）。然后把这个地址赋值给`dn` 指针，那么通过`dn`指针来操作该地址内的成员时，就取决于在定义`dn`时所指定的数据类型(`Data1`)，因此 dn->a 就可以正确的从这个地址中取出前 4 个字节，然后作为一个`int`型的数据打印出来。

以上代码，如果使用`clang`来编译，结果也是正确的。

用`g++`编译，继续报错：

```
$ g++ main.c -m32  -o main -I./
main.c: In function ‘int main()’:
main.c:23:20: error: invalid conversion from ‘void*’ to ‘Data1* {aka
_Data1_*}’ [-fpermissive]
     Data1 *dn = d2.next;
```

如果想让这个错误消除掉，在指针赋值时， 强制转换一下即可(把`void`型指针强转成`Data1`型指针，然后再赋值)：

```
Data1 *dn = (Data1 *)d2.next;
```

四、总结

这里描述的错误，几乎很少遇到，除非是像我一样误碰了键盘。

不过，从中我们也看到了一个现象：gcc编译器在面对结构体时，主要关心的是结构体在内存空间中所占用的空间大小，对其内部指向结构体类型的指针，并没有严格的检查是否存在，g++ 在这一点就做的严谨一些了。

---------- End ----------

让知识流动起来，越分享，越幸运！

星标公众号，能更快找到我！

Hi~你好，我是道哥，一枚嵌入式开发老兵。

推荐阅读

【1】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
【2】C指针的这些使用技巧，掌握后立刻提升一个Level
【3】提高代码逼格的利器：宏定义-从入门到放弃
【4】原来gdb的底层调试原理这么简单
【5】一步步分析-如何用C实现面向对象编程
【6】我最喜欢的进程之间通信方式-消息总线
【7】如何利用Google的protobuf，来思考、设计、实现自己的RPC框架
【8】都说软件架构要分层、分模块，具体应该怎么做(一)
【9】都说软件架构要分层、分模块，具体应该怎么做(二)
【10】内联汇编很可怕吗？看完这篇文章，终结它！