# **Struct vs Union**

Struct và Union là 2 kiểu dữ liệu User-defined (Người dùng tự định nghĩa) trong C, là kiểu dữ liệu chứa các thành phần dữ liệu khác bên trong.

  Hai kiểu dữ liệu này được sử dụng khá phổ biến trong các thư viện cho vi điều khiển (ví dụ thư viện STD STM32), và quản lý sự kiện trong các ứng dụng nhúng, IOT.

1. ***Về đặc điểm thì khác nhau rõ ràng nhất là việc chiếm dụng bộ nhớ***

A blackboard with white text and symbols

Description automatically generated

1. Dữ liệu của các thành viên của **Struct** được lưu trữ ở những vùng nhớ khác nhau, có thể tuân theo nguyên tắc padding.

**=> Memory sử dụng lớn hơn hoặc bằng tổng kích thước các thành phần.**

*Để hiểu rõ về việc này, chúng ta hãy cùng tìm hiểu sơ qua Data Structure Alignment & Padding.*

Data structure alignment là gì? Nó hoạt động như thế nào? Lâu lâu chúng ta vẫn hay nghe software này không tương thích với hệ điều hành 64bit hay software này chỉ chạy trên HĐH 32bit gì đó, vậy liên quan gì tới data structure alignment không?

Bên cạnh đó chúng ta cần một số kiến thức nền về hệ thống để có thể hiểu sâu hơn. Mình sẽ cung cấp cho các bạn trong bài viết này (nhưng chỉ là tổng quan thôi, vì nó khá là chuyên sâu, nếu có thời gian mình sẽ viết thêm những bài về nó) và mình sẽ cung cấp keyword để các bạn có thể tra google thêm.

* 1. **Data Structure Alignment là gì?**

Theo Wikipedia: Data Structure is the way data is arranged and accessed in computer memory. Có nghĩa là khi data load lên memory sẽ được CPU sắp xếp lại để tiện cho việc truy xuất tối ưu nhất có thể.  
Chúng ta đều biết rằng các CPU hiện đại của chúng ta luôn thao tác trên memory theo từng khối ở địa chỉ là một số chẵn, không thể thao tác trên địa chỉ là số lẻ được. Như vậy với mstruct của chúng ta, ví dụ biến “char c” nằm trên memory có địa chỉ chẵn, nếu biến “int i” nằm kế tiếp thì sẽ có địa chỉ lẻ rất phức tạp để CPU thao tác với biến “int i” này. Vì vậy chúng ta có thêm 2 khái niệm sau:

* Data alignment: sắp xếp data sao cho địa chỉ của các biến luôn là số chẵn và phù hợp với hệ thống
* Data padding: để làm được việc alignment như ở trên chúng ta cần phải “padding” thêm một số byte vào sau biến “char c” để khi đó biến “int i” có thể ở địa chỉ chẵn
  1. **Sau đây mình sẽ giới thiệu thêm về hệ thống.**
* Trên các hệ thống khác nhau, chúng ta sẽ có sự khác nhau giữa size của các data type như sau:
* Lúc nãy ở trên chúng ta có biết được rằng CPU chỉ thao tác trên memory theo từng khối ở địa chỉ chẵn. Hay nói chính xác hơn khối ở đây là 1 WORD size. Tùy thuộc vào kiến trúc của CPU, nhưng thông dụng nhất có các size như sau:
  + Trên system 16bit: 1 WORD = 16bit = 2 bytes
  + Trên system 32bit: 1 WORD = 32bit = 4 bytes
  + Trên system 64bit: 1 WORD = 64bit = 8 bytes
* N-byte alignment:  
  chúng ta có các loại n-byte alignment: 2-byte align, 4-byte align, 8-byte align.
  1. **Cách tính kích thước bộ nhớ**
* Theo mình nghĩ, số byte mỗi lần đọc nó sẽ lấy kích thước của kiểu dữ liệu lớn nhất trong Struct (đã kiểm tra nhưng mâu thuẫn so với ở phía trên, vì tùy hệ thống mỗi lần đọc là 1 WORD và kích thước của 1 WORD khác nhau tùy vào system 16 bit, 32 bit hay 64 bit
* Kiểu dữ liệu chỉ có thể đặt ở ô nhớ có vị trí chia hết cho chính nó
* Ví dụ kiểu int (4 byte) thì đặt tại địa chỉ chia hết cho 4 là: 4, 8, 12,...
* Còn kiểu char (1 byte) thì đặt ở bất kì ô nhớ nào, thường sẽ đặt liền kề với địa chỉ của biến trước đó

1. Dữ liệu của các thành viên trong **Union** được lưu trữ trong cùng 1 vùng nhớ.

    => **Memory sử dụng = Kích thước thành viên lớn nhất.**

1. **Về mặt ứng dụng thì rõ ràng nhiều trường hợp có thể dùng Struct thay cho Union vì nó bao quát và tường minh hơn.**
2. **Nhưng trong các ứng dụng lớn, có nhiều sự kiện, thì việc dùng Union sẽ giúp tiết kiệm bộ nhớ hơn rất nhiều (Đặc biệt với ứng dụng sử dụng Vi điều khiển có bộ nhớ hạn chế).**
3. **Một ứng dụng khác mình dùng Union là quản lý các thanh ghi trong vi điều khiển (STM32).**

<https://www.laptrinhdientu.com/2021/08/Union.html>

https://www.laptrinhdientu.com/2021/08/UnionEx2.html