03.09.2024

Laboratório 5

Theo Canuto - 2311293

Professor Raúl Renteria

INF1018 - 3WA

2024.2

1) O array `b` ocupa 8 bytes na memória, pois é composto por inteiros, cada um com 4 bytes, e o array possui 2 elementos (4 bytes * 2 = 8 bytes). Já o array `c` pode ser visto como um array de arrays, onde cada `a[i]` contém um array com 3 elementos. Assim, em `a[2][3]`, temos 6 elementos, e como cada elemento é do tipo `short` (2 bytes), o total de memória ocupada é 12 bytes (6 elementos * 2 bytes). Em ambos os casos, a alocação de memória é contínua. O endereço do array aponta para o seu primeiro elemento, e o próximo elemento está no endereço anterior somado ao tamanho do tipo do array.

Ao testar a função `dump` na `main`, confirmei que os tamanhos dos arrays são os indicados, e o endereço do primeiro elemento é o mesmo que o do array. À medida que os endereços são incrementados de acordo com `sizeof(tipo)`, chegamos a um novo elemento do array, que é exibido conforme a ordenação little-endian.

2) Chamando a função `dump` para essa estrutura, percebemos que os campos de cada elemento da `struct` são alocados sequencialmente na memória (com cada um ordenado segundo a ordenação little-endian). Como esperado, as variáveis inteiras `a` e `c` ocupam 4 bytes cada na memória, mas a variável `short` `b` também ocupa 4 bytes devido ao padding de 2 bytes. Isso demonstra que a alocação de memória na `struct` é sequencial, mas não necessariamente contígua, pois os valores devem ser alinhados em endereços múltiplos de seus tamanhos, garantindo o alinhamento da `struct`.

3) Em todos os casos de `struct`, seus campos são alocados sequencialmente na memória, mas nem sempre de forma contígua, variando de acordo com a estrutura. O padding é representado por `x`.

a) Tamanho: 12 bytes
Organização: |c1|x|x|x|i|i|i|i|c2|x|x|x|

b) Tamanho: 16 bytes
Organização: |l|l|l|l|l|l|l|l|c|x|x|x|x|x|x|x|

c) Tamanho: 8 bytes
Organização: |i|i|i|i|c1|c2|x|x|

d) Tamanho: 24 bytes
Organização: |l|l|l|l|l|l|l|l|c1|x|x|x|x|x|x|x|c2|x|x|x|x|x|x|x|

e) Tamanho: 3 bytes
Organização: |c1|c2|c3|

f) Tamanho: 16 bytes
Organização: |s1|s1|x|x|i|i|i|i|c1|c2|c3|x|s2|s2|x|x|

Para o caso das `unions`, diferentes tipos de dados compartilham o mesmo local de memória, e apenas um elemento pode armazenar um valor em um determinado momento. Dessa forma, menos memória é utilizada, e o tamanho da `union` corresponde ao tamanho do maior elemento, pois os endereços devem ser múltiplos do tamanho do maior elemento armazenado. `?` representa um valor que estava previamente armazenado no endereço antes da atribuição de um novo valor.

g) Tamanho: 8 bytes
Organização:
- inteiro -> |i|i|i|i|?|x|x|x|
- char -> |c|c|c|c|c|x|x|x|

h) Tamanho: 6 bytes
Organização:
- short -> |s|s|?|?|?|x|
- char -> |c|c|c|c|c|x|