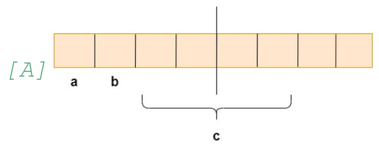
Structure Padding:

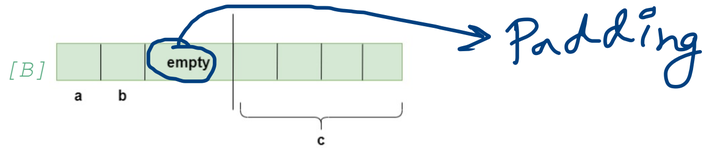
Processor reads 1 word at a time. For 32-bit processor it reads 4 bytes at a time.

1 word = 4 bytes. >> 32-bit processor
1 word = 8 bytes. >> 64-bit processor

This is expected data allignment.



This is actual data allignment done by processor.



[Case A:] In this case, 32-bit processor will process 4 bytes at a time. In case [A], there will not be any problem in processing variables "a" and "b", but to access "c" it will need two cycles. If we only want to access variable "c" in that case also processor will take 2 cycles to complete.
Total Memory occupied in [Case A] will be 6 bits.

[Case B:] Here, padding bits [2:3] or holes are introduced so that as processor accesses "c", it will take only one CPU cycle to access it.
Total Memory occupied in [Case B] will be 8 bits >> 1 bytes].

In order to check the padding bits or holes pahole in Ubuntu can be used.

```
struct temp {
        char a;
        char b;
        int c;
};
```

```
struct temp {
        char                    a;              /*     0     1 */
        char                    b;              /*     1     1 */

        /* XXX 2 bytes hole, try to pack */

        int                     c;              /*     4     4 */

        /* size: 8, cachelines: 1, members: 3 */
        /* sum members: 6, holes: 1, sum holes: 2 */
        /* last cacheline: 8 bytes */
};
```

How to avoid structure padding:
1. #pragma
2. attribute

```
#include<stdio.h>
#pragma pack(1)
```

or

```
struct base {
      char a;
      double b;
      int c;
}__attribute__((packed));
```

With __attribute__((packed));

```
struct temp {
        char                    a;              /*     0     1 */
        double                  b;              /*     1     8 */
        int                     c;              /*     9     4 */

        /* size: 13, cachelines: 1, members: 3 */
        /* last cacheline: 13 bytes */
} __attribute__((__packed__));
```

Without packed:

```
struct temp {
        char                    a;              /*     0     1 */

        /* XXX 7 bytes hole, try to pack */

        double                  b;              /*     8     8 */
        int                     c;              /*    16     4 */

        /* size: 24, cachelines: 1, members: 3 */
        /* sum members: 13, holes: 1, sum holes: 7 */
        /* padding: 4 */
        /* last cacheline: 24 bytes */
};
```