Structure

```
Struct rect
{   int length;
    int breadth;
};
int main()
{   Struct rect r;          ⟶ declaration
    Struct rect r2 = {10, 5}  → declaration + initialization

    r.length = 15;
    r.breadth = 10;
    printf("Area of Rectangle is %d",
                r.length * r.breadth);
}   return 0;
```

Eg →

```
struct abc
{   char a;   1 byte
    char b;   1 byte
    int c;    4 bytes
} var;
```

Total 6 bytes ✗ wrong

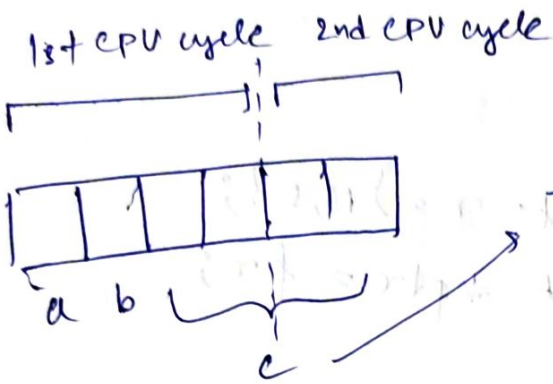
(#) Structure padding

→ processor doesn't read 1 byte at a time from memory

→ it reads 1 word at a time.

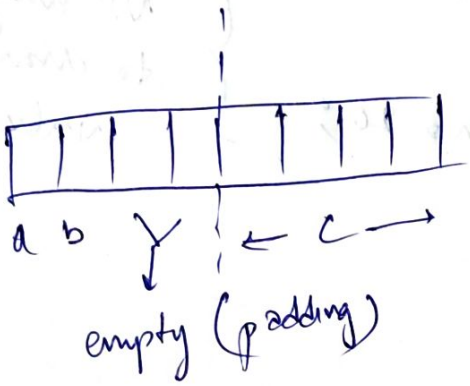
If we have a 32 bit processor → it can access 4 bytes at a time

If we have a 64-bit processor → it can access 8 bytes at a time



To access a single variable we have to access two CPU cycle.

Now,



→ So total size

= 1+1+2(empty) +4 = 8 byte

Structure packing

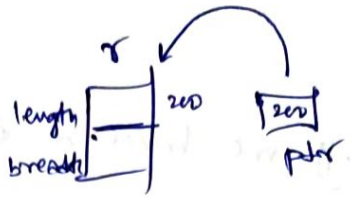
↳ If we want to avoid the padding and save the storage then we use packing

```
#pragma pack(1)
struct abc
{   char a;
    char b;
    int c;
} var;
```

Total 6 bytes ✓ (but 2 CPU cycles)

Pointer to a structure

```
Struct rect
{  int length;
   int breadth;
};
int main()
{  struct rect r = {10,5};
   struct rect *ptr = &r;
```




```
   (*p).length = 20;
        or
   p -> length = 20;
```

} Use any of these to change the value.

Arrow operator

```
}
```

(#) in Heap

```
struct rect
{  int length;
   int breadth;
};
int main()
{
   struct rect *ptr;
   ptr = (struct rect *) malloc (sizeof( struct rect));
   ptr -> length = 20;
```