```
int x;
myint y;
typedef myint smallint;
smallint z;
Printf("enter two values");
Scanf("%d%d", &x, &y);
z=x+y;
Printf("Sum value is: %d", z);
getch();
return 0;
}
```

Output:

enter two values: 4, 2

Sum value is: 6

Structure padding:

* Structure padding is a concept in c that adds the one more empty bytes between the memory addresses to align the data in memory
* Suppose we create a user defined structure. When we create an object of this structure, then the contiguous memory will be allocated to the structure member.

```
Struct student
{
  Char a;
  Char b;
  Int c;
} stu;
```

* In this,
  Student - structure name
  stu1 - object of the structure
* After the creation of an object, a contiguous block of memory is allocated to its structure members.
* Memory will be allocated to 'a', then 'b' after that 'c'
  Now we calculate the size of the struct student

```
struct student
{
    char a;   // 1 byte
    char b;   // 1 byte
    int c;    // 4 byte
};
```

In the above case, we calculate the size of the 'student', size come to 6 bytes. But this answer is wrong. Now, we will understand the concept of structure padding.

Structure padding:

* Generally the processor does ~~not~~ not read 1 byte at a time. It reads 1 word at a time.
* If we have 32 bit processor, then the processor reads 4 byte at a time, which means that 1 word equal to 4 bytes.

  ie 1 word = 4 byte

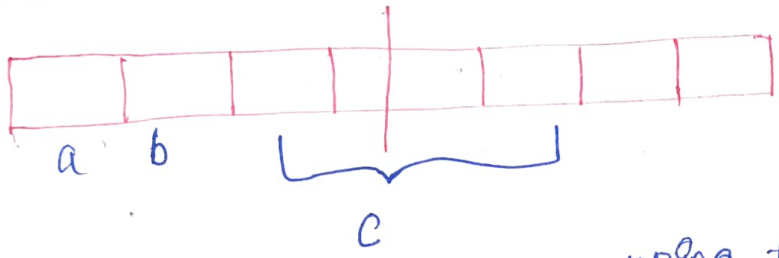
  ∴ 1 word = 8 byte [in 64 bit processor]

  So size of the word depends architecture.

Why structure padding:

```
struct student
{
  char a;    // 1 byte
  char b;    // 1 byte
  Char c;    // 4 byte
}
```

* If we have a 32 bit processor (4 byte at a time) then the pictorial representation of the memory for the above structure would be.

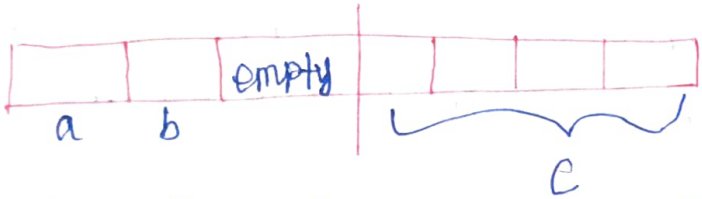


* As we know that structure occupies the contiguous block of memory as shown in the above diagram. ie 1 byte for char 'a', 1 byte for char 'b', and 4 byte for char 'c'
* The 4 bytes can be accessed at a time as we are considering the 32 bit architecture. The problem is that so in one CPU cycle,

  1 byte of char a, 1 byte of char b, and 2 bytes of int c can be accessed.

* we will not face any problem during accessing of a and b variables as both the variables can be accessed in one CPU cycle, but int c variable as 2 CPU cycles are required to access the value of the 'c' variable.

* Suppose we do not want to access 'a' and 'b' variables, we only want to access the variable 'c' which requires two cycles. This is an unnecessary wastage of CPU cycles.
* Due to this reason, the structure padding concept was introduced to save the number of CPU cycle.
* The structure padding is done by the compiler automatically



* In order to achieve the structure padding, an empty row is created on the left [ref picture] and two bytes which are occupied by the 'c' variable on the left are shifted to the right. So all the four bytes of 'c' variable are on the right.
* Now the 'c' variable can be accessed in a single CPU cycle. After structure padding, the total memory occupied by the structure is 8 bytes

a - 1 byte
b - 1 byte
empty - 2 bytes
c - 4 bytes

which is greater than previous one. Although the memory is wasted in this case, the variable can be accessed in within a single CPU cycle