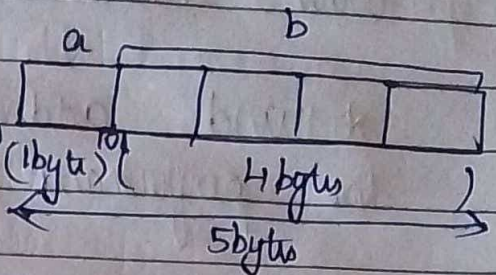
C_11b ⇒ Structure Padding in C

```
struct abc
{
    char a; → 1byte
    int b; → 4byte
};
```

Memory allocation



* We think for char 1 byte and for int 4 bytes so totally 5 bytes of memory is allocated; but it wrong.

* Because memory is not byte addressable; rather it is word addressable.

Word addressable :-

* That is, for one CPU cycle, the processor can address 4 byte for 32 bit machine and can address 8 byte for 64 bit machine.

Byte addressable :-

* For one CPU cycle, the processor can address only one byte at a time.

Problem

* If the processor is byte addressable; and it wants to access the integer variable of 4 bytes of memory; then it takes 4 CPU cycles to read it or access it. and this is the problem.

* We want to read or access the integer variable of 4 bytes of memory at a single CPU cycle; then comes word addressable memory.

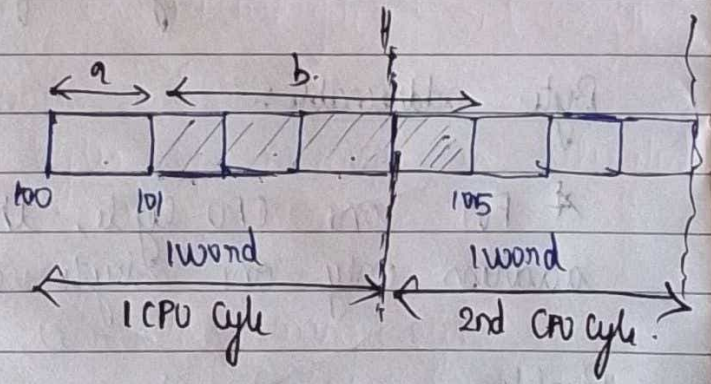
* Word addressable memory increases the performance and speed

* So nowadays memory is word addressable (ie) at one time we can access 4 bytes of memory for 32 bit machine and can access 8 bytes of memory for 64 bit machine.

Scenario:

* Now we assume that we have 32 bit machine and our word size is 4 byte; word size is nothing but number of bytes it can access memory.

```
Struct abc
{
  char a;
  int b;
};
```

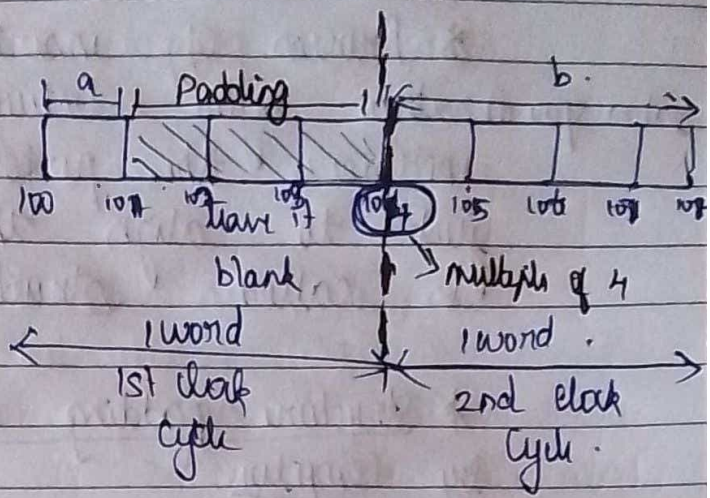


* So to access integer value; it requires 2 CPU cycle → In first cycle it can read only 3 bytes and in 2nd cycle it will read the left 1 byte

* This leads to wastage of CPU cycle and performance decreases.

* To overcome; this we go for another method.

```
struct abc
{
  char a;
  int b;
};
```



* In first clock cycle; we store or fetch for one character value and leave the remaining 3 bytes of memory free and at the next clock cycle we access or store the integer value of 4 byte at a single clock cycle.

* So in this case address of integer variable starts with multiple of 4.

* The extra bits or free space is called padding bits or memory holes or memory alignment or data alignment, to increase the speed or performance of CPU we waste the memory.

Note:-

* Padding is automatically done by the Compiler; but if we want to avoid padding then we want to include special line is program to do packing.

Date
Page

Questions

(1) Structure Padding :- (with data allignment).

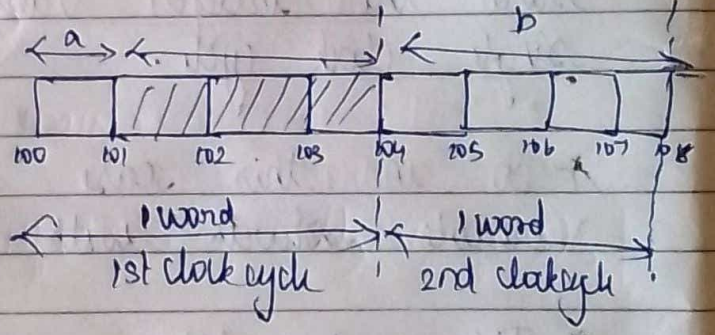
* Process of inserting extra bits or extra spaces or extra bytes between variables to increase the performance and speed just to align the data then it is called structure padding.

* Structure padding is done automatically by Compiler.

* Word size → At a single CPU cycle it can access 4 bytes of memory in 32 bit ; 8 bytes of memory in 64 bit

(1) struct abc
{
char a;
int b;
};

8 bytes of memory

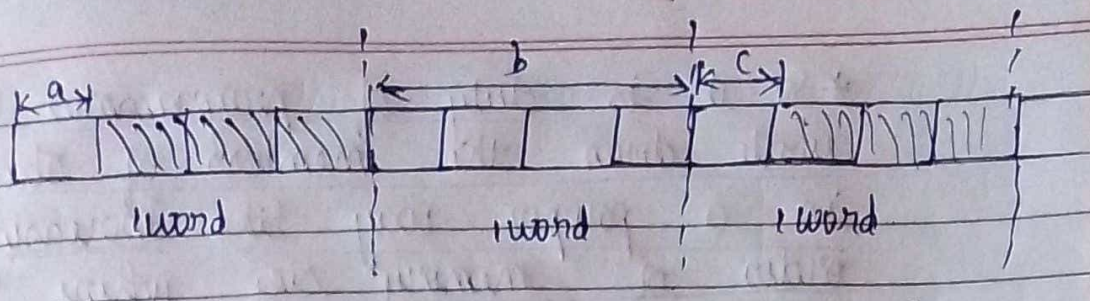


* So here 8 bytes of memory is allocated in memory.

(2) struct abc
{
char a;
int b;
char c;
};

12 bytes of memory

→ How many bytes of memory allocated with structure padding or with data allignment ?

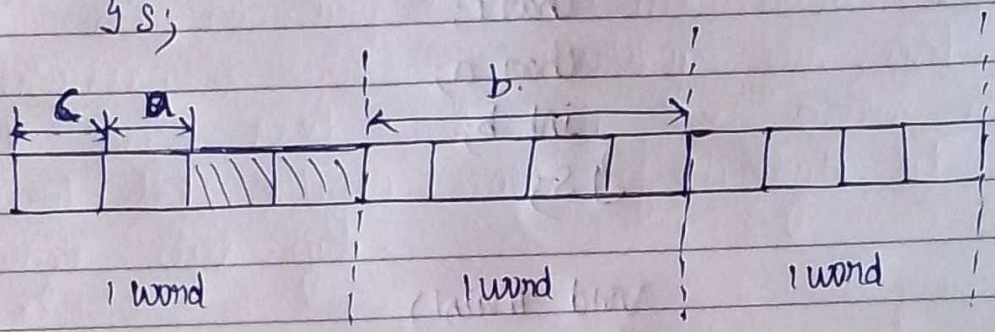


* So here 4x3 => 12 bytes of memory is allocated by Compiler.

③ struct abc → How many bytes of memory allocated with structure padding or with data alignment?

{
char c;
char a;
int b;
} s;

8 bytes of memory.



* So here 4x2 => 8 bytes of memory is allocated by Compiler.

* In ② example it takes 12 bytes and ③ example it takes only 4 bytes of memory.

* So it now clear that; to avoid wastage of memory; it the duty of programmer is such a way that memory is not wastage.

* If we go for ② example; if we declare object for 60 students; and for every student we waste 4 bytes of memory

* So it is duty of programmer to write down the members of structure in a proper way like increasing order of memory size where the memory have less wastage.

* We can check the size of memory allocated for structure using sizeof()

Ex:
```
struct ab
{
    char c;
    char a;
    int b;
} s;

void main()
{
    printf("size of s:%d", sizeof(s));
}
```

To check size of object allocated for structure

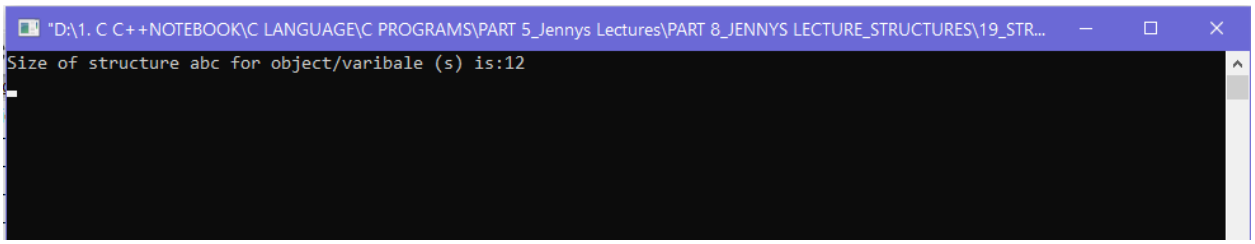
```c
#include <stdio.h>
#include <stdlib.h>
/** 18-STRUCTURE PADDING IN C (1) **/

typedef struct student
{
 int rollno;
 char  name[20];
 float marks;
}stu;

int main()
{
    stu s;
    printf("Size of student structure for object/varibale (s) is:%d\n",sizeof(s));
    getch();
}
```

```
"D:\1. C C++NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 8_JENNYS LECTURE_STRUCTURES\18_STR...
Size of student structure for object/varibale (s) is:28
```

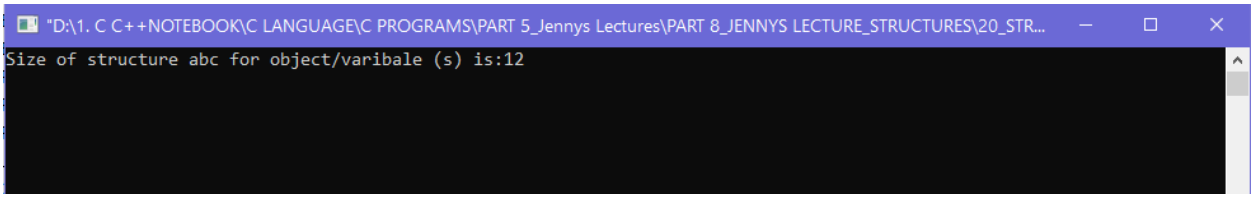
```c
#include <stdio.h>
#include <stdlib.h>
/** 18-STRUCTURE PADDING IN C (2) **/

typedef struct abc
{
 int a;
 char b;
 int c;
}abc;

int main()
{
    abc s;
    printf("Size of structure abc for object/varibale (s) is:%d\n",sizeof(s));
    getch();
}
```

```
"D:\1. C C++NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 8_JENNYS LECTURE_STRUCTURES\18_STR...
Size of structure abc for object/varibale (s) is:12
```

```c
#include <stdio.h>
#include <stdlib.h>
/** 19-STRUCTURE PADDING IN C (3) **/

typedef struct abc
{
 int a;
 int b;
 char c;
}abc;

int main()
{
    abc s;
    printf("Size of structure abc for object/varibale (s) is:%d\n",sizeof(s));
    getch();
}
```




```c
#include <stdio.h>
#include <stdlib.h>
/** 20-STRUCTURE PADDING IN C (4) **/

typedef struct abc
{
 char a;
 int b;
 char c;
}abc;

int main()
{
    abc s;
    printf("Size of structure abc for object/varibale (s) is:%d\n",sizeof(s));
    getch();
}
```



```c
#include <stdio.h>
#include <stdlib.h>
/** 21-STRUCTURE PADDING IN C (5) **/

typedef struct abc
{
 char a;
 char b;
 int c;
}abc;

int main()
{
    abc s;
    printf("Size of structure abc for object/varibale (s) is:%d\n",sizeof(s));
    getch();
}
```