**>>>>>>>>> STRUCTURE <<<<<<<<<<**

1) Stucture/Union can't have a static member in C.

2) Structure can't have array of bit field.

3) IT is implementation dependent to assign an out of range value to a bit field member.

but gcc output is as following.....

i.e

struct tag

{

int i:1;

int j:2;

int k:3;

int l:4;

};

struct st s;

s.i =1;

s.j =2;

s.k =5;

s.l =10;

+ \*\*OUTPUT:\*\*

i = -1

j = -2

k = -3

l = -6

Because MSB will be used as sign bit.

-----

i = | 1 | = -1

-----

--------

j = | 1 | 0| = -2 // Compiler store negative data in 2's compliment in memory.

--------

-------------

k = | 1 | 0 | 1 | = -3 // 2's compliment

-------------

-----------------

l = | 1 | 0 | 1 | 0 | = -6 // 2's compilement

-----------------

4) we can't have pointer to bit field members they may not start at a byte boundry.

5) A special unnammmed bit field of size 0 is used to force alignment on next boundry.

6) Float and double is not allowed for bit field.

7)

# headerfiles

struct tag t; // it is posssible.

stuct tag

{

char ch;

int arr[4];

};

main()

{

t.ch = 'A';

t.Arr[0] = 12;

PF("%c %d\n",t.ch,t.arr[0]);

}

OUTPUT : NO Error

A,12

8) Two structure can't be compare.Although they are same type.But assignment two structure of same type

is possible.

i.e

struct tag p1 ={1};

struct tag p2 ={1};

struct tag p3 = p1; // NO ERROR,all value of p1 will accomodate into p3.

if(p1 == p2) PF("Equal"); // Compiler Error

else PF("NOT Equal");

9) Structure can't be initilize with declaration. Because structure datatype declare no memory

allocate. Memory allocate when variable are created.

10) Designated initilization of structure allows random initilize of a structure members.

11) >>>>>>> Bit FIELD <<<<<<<<<<

case 1:

struct tag{

unsigned int a; // First 4 bytes

unsigned int b; // second 4 bytes

unsigned int c; // thrid 4 bytes

};

>>>>>>> OUTPUT: 12 bytes

case 2:

struct tag{

unsigned int a:1; // fisrt 4 bytes

unsigned int b:31; // will get accomodated in the 1st 4 Bytes

unsigned int c:1; // Second 4 Bytes are allocate

};

>>>>>>> OUTPUT: 8 Bytes

case 3:

struct tag{

unsigned int a:1; // fisrt 4 bytes

unsigned int b:32; // will not get accomodated in the 1st 4 Bytes so 2nd Byte will allocate

unsigned int c:1; // will not accomodate in Second 4 Bytes So 3rd Byte will allocate

};

>>>>>>> OUTPUT: 12 Bytes

case 4:

struct tag{

unsigned int a:1; // fisrt 4 bytes

unsigned int b:1; // will get accomodated in the 1st 4 Bytes

unsigned int c:1; // will get accomodated int 1st 4 Bytes

};

>>>>>>> OUTPUT: 4 Bytes

case 5:

struct tag{

unsigned int a:32; // fisrt 4 bytes

unsigned int b:1; // Second 4 Bytes are allocate.

unsigned int c:1; // will get accomodate in the second 4 bytes.

};

>>>>>>> OUTPUT: 8 Bytes

case 6: // Special case

struct tag{

unsigned int a:5; // fisrt 4 bytes

unsigned int :0; // will get accomodated in the 1st Bytes

unsigned int c:8; // Second 4 Bytes are allocate

};

>>>>>>> OUTPUT: 8 Bytes

12 ) Find structure size :

Trick : structre Padding

(\*) structure size always 4 bytes memory boundry align.

so sizeof( struct \_node )%4 == 0

(\*) Structre padding happen according to largest structre member.

(#) struct \_node

{

int a;

int b;

char c[10];

};

(\*) Note : we calculate size of this structre is 18 but 18 is not devisible by 4. So padding

will happen according to largest member of Structre which is int and sizeof int is 4 bytes.

>>>> Sizeof(struct \_node) ==> 20 Bytes

(#) struct \_node

{

char a; // 0-3

int b; // 4-7

int d; // 8-11

char c; //12-15

};

(\*) int always take address which will start from

>>>> Sizeof(struct \_node) ==> 16 Bytes

(#) struct \_node

{

char a;

int b;

double d;

char c;

}node;

>>>> Sizeof(struct \_node) ==> 24 Bytes

(\*) Note : Structre padding happen according to largest structre member.

In this case padding happening 8 bytes memory allign.

struct \_node

{

char a;

double d;

char c;

}node;

>>>> Sizeof(struct \_node) ==> 24 Bytes