Structures in 'C'

- Structures is a data structure used to create user-defined data types in 'c'
- Structures allows us to combine data of different types.

Creation of Structure

```
struct tag_name
{
    member_element-1;
    member_element-2;
    member-element-3;
    member-element-4;
};
```

so important

This is a syntax to create a structure in C.

```
struct CarModel
{
    unsigned int    carNumber;
    uint32_t        carPrice;
    uint16_t        carMaxSpeed;
    float           carWeight;
};
```

Structures variables

```
Struct CarModel CarBMW, CarFord, CarHonda;
```

User defined data type

(Memory will be consumed when you create structure variables.)

→ What is the data type of CarBMW?

→ The data type of CarBMW variables is "struct CarModel"

→ Structure definition doesn't consume any memory. Its just a description or a record.

```
struct carModel
{
    uint32_t carNumber;   → 4 bytes
    uint32_t carPrice;    → 4 bytes
    uint16_t carMaxSpeed; → 2 bytes
    float carWeight;      → 4 bytes
};
```

when you try to calculate the size of this struct you will find 14 bytes own by own. But the program will give you 16 bytes as a result. This causes because our programm padding to structures and it cost 2 bytes more.

```
struct carHeight
{
    uint8_t carA; → 1 bytes
};
```

* structure type cannot contain itself as a member.

But in this example, when you are trying to calculate you will see computer also will say 1 bytes. Because in this example, there is no padding.

Aligned / un-aligned data access

- For efficiency, the complier generates instructions to store variables on their natural size boundry addreses in the memory.
- This also works for structures. Member elements of a structure are located on their natural size boundry.

Natural Size Boundry

char

Address 0x....0, 0x....1, 0x....2, 0x....3 ....

short

Address 0x....0, 0x....2, 0x....4, 0x....6 ....

int

Address 0x....0, 0x....4, 0x....8 ....

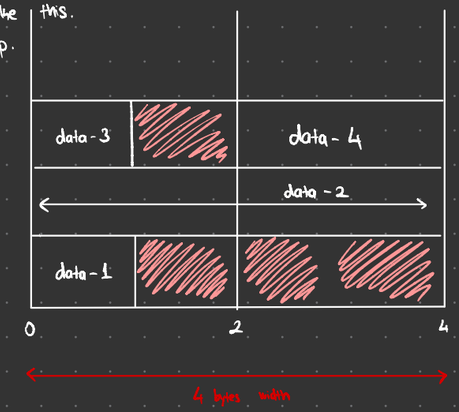
```
Memory Adress Content
=====================
0064FEB8,   11
0064FEB9,   FF
0064FEBA,   64
0064FEBB,   0
0064FEBC,   EE
0064FEBD,   EE
0064FEBE,   FF
0064FEBF,   FF
0064FEC0,   22
0064FEC1,   16
0064FEC2,   CD
0064FEC3,   AB
```

C data_allingment.c

```c
#include <stdio.h>
#include <stdint.h>

struct dataSet
{
    char data1;
    int data2;
    char data3;
    short data4;
};

int main()
{
    struct dataSet data;

    data.data1 = 0x11;
    data.data2 = 0xFFEEEEEE;
    data.data3 = 0x22;
    data.data4 = 0xABCD;

    uint8_t *ptr;

    ptr = (uint8_t*)&data;

    uint32_t totalSize = sizeof(struct dataSet);

    printf("Memory Adress Content \n");
    printf("=====================\n");

    for(uint32_t i=0 ; i <totalSize ; i++)
    {
        printf("%p,  %X\n",ptr,*ptr);
        ptr++;
    }

    getchar();
}
```

char, int, char, short (2 bytes)

when we have this code it will show like this. in memory map.

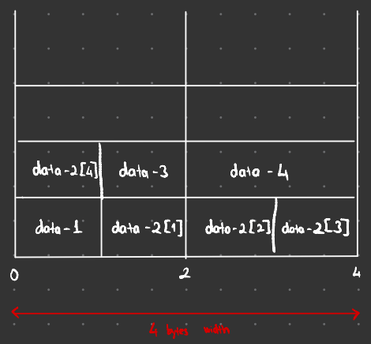


Size of (struct data) = 12 bytes.

these values was on the memory, we only used adresses which I showed with arraw.

if you do not want to packing, you need to use

```c
struct data
{
    char data1;
    int data2;
    char data3;
    short data4;
}__atribute__(packed);
```



size of it = 8 bytes.

- When you thinks about performace, spliting the data is bad, But for memory is gives you 4 more bytes.

```c
struct dataSet
{
    char data1;
    int data2;
    char data3;
    short data4;
};

int main(void)
{
    struct dataSet data;
    // DO NOT FORGET TO DEFINE STRUCT.

    data.data1 = 0xAA;
    data.data2 = 0xFFFFEEEE;
    data.data3 = 0x55;
    data.data4 = 0xA5A5;

    printf("data.data1 = %d\n",data.data1);
    printf("data.data2 = %d\n",data.data2);
    printf("data.data3 = %d\n",data.data3);
    printf("data.data4 = %d\n",data.data4);

}
```

```
08000280:   push    {r7, lr}
08000282:   sub     sp, #16
08000284:   add     r7, sp, #0
08000286:   movs    r3, #170        ; 0xaa      } → 0xAA
store byte → strb   r3, [r7, #4]
0800028a:   ldr     r3, [pc, #64]   ; (0x80002cc <main+76>)   } → 0xFFFFEEEE
store →     str     r3, [r7, #8]
0800028e:   movs    r3, #85 ; 0x55      } → 0x55
store byte → strb   r3, [r7, #12]
08000292:   movw    r3, #42405      ; 0xa5a5    } → 0xA5A5
store half → strh   r3, [r7, #14]
08000298:   ldrb    r3, [r7, #4]
0800029a:   mov     r1, r3
0800029c:   ldr     r0, [pc, #48]   ; (0x80002d0 <main+80>)
0800029e:   bl      0x8000600 <printf>
080002a2:   ldr     r3, [r7, #8]
080002a4:   mov     r1, r3
080002a6:   ldr     r0, [pc, #44]   ; (0x80002d4 <main+84>)
080002a8:   bl      0x8000600 <printf>
080002ac:   ldrb    r3, [r7, #12]
080002ae:   mov     r1, r3
080002b0:   ldr     r0, [pc, #36]   ; (0x80002d8 <main+88>)
080002b2:   bl      0x8000600 <printf>
```

→ this is how does it look like when you debug with aligned unpacked structure.

Text 5112

for this little code the difference in assembly is 32 lines.

```c
#include <stdint.h>
#include <stdio.h>

struct dataSet
{
    char data1;
    int data2;
    char data3;
    short data4;
} __attribute__((packed));

int main(void)
{
    struct dataSet data;
    // DO NOT FORGET TO DEFINE STRUCT.

    data.data1 = 0xAA;
    data.data2 = 0xFFFFEEEE;
    data.data3 = 0x55;
    data.data4 = 0xA5A5;

    printf("data.data1 = %d\n",data.data1);
    printf("data.data2 = %d\n",data.data2);
    printf("data.data3 = %d\n",data.data3);
```

with help of this code block, we told "we need to unalign our memory map."

```
08000284:   add     r7, sp, #0
17              data.data1 = 0xAA;
08000286:   movs    r3, #170        ; 0xaa
store byte  strb    r3, [r7, #0]
18              data.data2 = 0xFFFFEEEE;
0800028a:   movs    r3, #0
0800028c:   orn     r3, r3, #17
store byte  strb    r3, [r7, #1]
08000292:   movs    r3, #0
08000294:   orn     r3, r3, #17
store byte  strb    r3, [r7, #2]
0800029a:   mov.w   r3, #4294967295
store byte  strb    r3, [r7, #3]
080002a0:   mov.w   r3, #4294967295
store byte  strb    r3, [r7, #4]
19              data.data3 = 0x55;
080002a6:   movs    r3, #85 ; 0x55
080002a8:   strb    r3, [r7, #5]
20              data.data4 = 0xA5A5;
080002aa:   movw    r3, #42405      ; 0xa5a5
080002ae:   strh    r3, [r7, #6]
22              printf("data.data1 = %d\n",data.data1);
080002b0:   ldrb    r3, [r7, #0]
```

for 4 bytes variable we need a lot of code blocks. Because of that, it will affect our performance.

→ "This is what, when we code with packed"

Text 5144

typedef is used to give an alias name to primitive and user defined data types

1. a false name, especially one used by criminals
2. a name that can be used to find computer files, command, etc.

not advanced or complicated in structure.

Typedef with Structure

tag name of struct

```
struct CarModel
{
    unsigned int    carNumber;
    uint32_t        carPrice;
    uint16_t        carMaxSpeed;
    float           carWeight;
};
```

`struct CarModel car BMW, carFord;` → for defining

```
typedef struct "there is no tag name in here"
{
    unsigned int    carNumber;
    uint32_t        carPrice;
    uint16_t        carMaxSpeed;
    float           carWeight;
} CarModel_t;
```

alias name or typedef name for this structure. When you use this name, you can create the variables.

`CarModel_t carBMW, carFord` → for defining

for typedef, we need add that name at the end of alias name. (just for readibility purpose) `_t`
↳ low dash low line

typedef allow us to define our own type.

Structures and Pointers

data_allingment.c

```
#include <stdio.h>
#include <stdint.h>

struct dataSet
{
    char data1;
    int data2;
    char data3;
    short data4;
}; This doesn't consume any memory!

int main()
{
    struct dataSet data; "When you write this you are starting to use memory"

    data.data1 = 0x11;
    data.data2 = 0xFFFFEEEE;
    data.data3 = 0x22;
    data.data4 = 0xABCD;

```

→ Base address of the structure in memory also happens to be the adress of the first member element.

```
Memory Adress Content
=====================
0064FEB8,  11
0064FEB9,  FF
0064FEBA,  64
0064FEBB,  0
0064FEBC,  EE
0064FEBD,  EE
0064FEBE,  FF
0064FEBF,  FF
0064FEC0,  22
0064FEC1,  16
0064FEC2,  CD
0064FEC3,  AB
```

it is also base address of structure

Let‘s say you have been given with the base address of a structure variable and asked to change the member element values what would you do?

```c
#include <stdio.h>
#include <unistd.h>

struct dataSet
{
    char data1;
    int data2;
    char data3;
    short data4;
};

void displayMemberElements(struct dataSet *pData);

int main(){

    struct dataSet data;

    data.data1 = 0x11;
    data.data2 = 0xFFFFEEEE;
    data.data3 = 0x22;
    data.data4 = 0xABCD;

    /*
    printf("Before: data.data1 = %d\n",data.data1);

    struct dataSet *pData;

    pData = &data;
    pData->data1 = 0x55;
    printf("After: data.data1 = %d\n",data.data1);
    */

    return(0);
}
```

As I said like below, we can use for transfering the structure.

→ When we are trying to change variables in struct via function or something else. We need to use pointer struct. For example, in here we made it basically in the main block.

↳ When we use pointer struct, we should use arrow.

C structPointer.c

```c
#include <stdio.h>
#include <unistd.h>

struct dataSet
{
    char data1;
    int data2;
    char data3;
    short data4;
};

void displayMemberElements(struct dataSet *pData);

int main(){

    struct dataSet data;

    data.data1 = 0x11;
    data.data2 = 0xFFFFEEEE;
    data.data3 = 0x22;
    data.data4 = 0xABCD;

    displayMemberElements(&data);
    return(0);
}

void displayMemberElements(struct dataSet *pData)
{
    printf("data1 = %X\n",pData->data1);
    printf("data1 = %X\n",pData->data2);
    printf("data1 = %X\n",pData->data3);
    printf("data1 = %X\n",pData->data4);
}
```

we can also send the structure itself! But, this is rarely used. (Lesson 161)
(struct dataSet data)

```
struct Packet{

    uint8_t crc;   → 1 byte
    uint8_t status; → 1 byte
    uint16_t payload; → 2 byte
    uint8_t bat;   → 1 byte
    uint8_t sensor; → 1 byte
    uint8_t longAddr; → 1 byte
    uint8_t shortAddr; → 1 byte
    uint8_t addrMode; → 1 byte
};
```

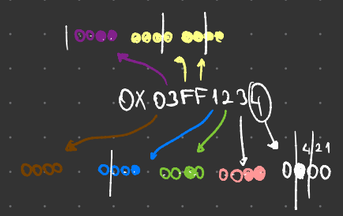
\+

9byte

```
Enter the 32 bit packed value:0xF1234ABC
decode = 0
decode = 0x1
decode = 0x957
decode = 0x6
decode = 0
decode = 0x89
decode = 0x3
decode = 0x1
Size of the struct is 10
PS C:\Users\islam\Desktop\Embedded-C\My_workspace\target\000CFolder>
```

because of padding its memory 10 byte.

And normally when you think this value was 4 byte. When you compare the values. As you can see, we lost 6 byte. We can minimize this memory consumption with using structure bit field.



Structure and Bit Fields

For preventing this memory consumption, we need to define one variable and extract various fields into variables.

```
uint32_t crc :2;
uint32_t status : 1;
uint32_t payload : 12;
```

this talks with memory and says "use only 2 bits for it".

and at the end we will only use 32 bits for this struct.