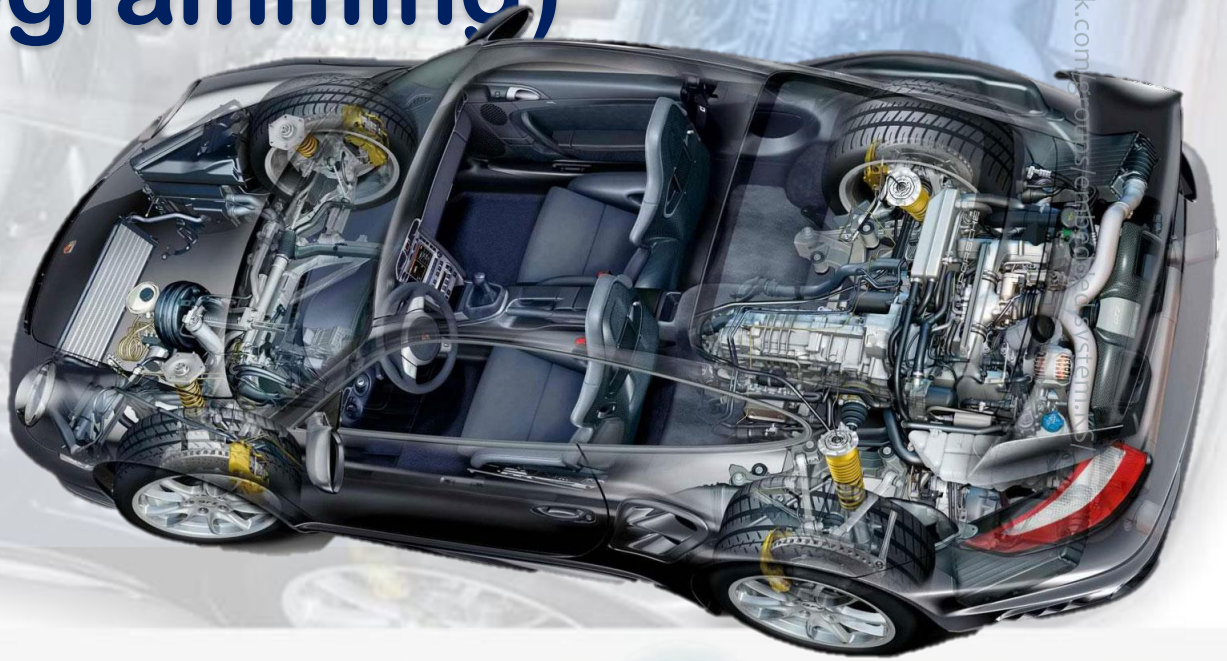
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Embedded System
unit 2 lesson 6(C Programming)

- Structures
  - Aligned and un-aligned data access on structures
  - Structure BitField
- Union
- Enum



ENG.KEROLES SHENOUDA

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Structure in C

- **Structure** is a user defined data type which hold or store heterogeneous data item or element in a singe variable. It is a Combination of primitive and derived data type.
- Structure is a data structure used to **create user-defined data types in C**. Structure allows us to combine data of different types.
- **Why Use Structure in C**
  - In C language array is also a user defined data type but array hold or store only similar type of data, If we want to store different-different type of data in then we need to defined separate variable for each type of data.
  - **Example:** Suppose we want to store Student record, then we need to store....
  - Student Name
  - Roll number
  - Class
  - Address

Why Use Structure in C Example [without Structure]

```c
#include "stdio.h"
#include "string.h"
#include "conio.h"

void main()
{
    char names[100][50], tempName[50];
    int birthDateYears[100], tempBirthDateYear;
    int birthDateMonths[100], tempBirthDateMonth;
    int birthDateDays[100], tempBirthDateDay;
    int salaries[100], tempSalary;
    int count = 0, i, j;
    char firstName[50], secondName[50];

    do
    {
        printf("Enter Employee First Name:");
        scanf("%s", firstName);

        printf("Enter Employee Second Name:");
        scanf("%s", secondName);
        strcpy(names[count], firstName);
        strcat(names[count], " ");
        strcat(names[count], secondName);

        printf("Enter Employee Birth Date (day/month/year)
                                    example(23/3/1970):");
        scanf("%d/%d/%d", &birthDateYears[count],
              &birthDateMonths[count], &birthDateDays[count]);

        printf("Enter Employee Salary:");
        scanf("%d", &salaries[count]);

        count++; if(count==100)break;

        printf("Do you want to add more,
                        press 'y' to continue?\r\n");
    }
    while(getch()=='y');

    for(i=0;i<count-1;i++)
    {
        for(j=i+1;j<count;j++)
        {
            if(  birthDateYears[i]>birthDateYears[j] ||
                 (birthDateYears[i]==birthDateYears[j] &&
                 birthDateMonths[i]>birthDateMonths[j])
                 ||
                 (birthDateYears[i]==birthDateYears[j] &&
                 birthDateMonths[i]==birthDateMonths[j]
                 && birthDateDays[i]>birthDateDays[j]))
            {
                strcpy(tempName, names[i]);
                tempBirthDateYear = birthDateYears[i];
                tempBirthDateMonth = birthDateMonths[i];
                tempBirthDateDay = birthDateDays[i];
                tempSalary = salaries[i];

                strcpy(names[i], names[j]);
                birthDateYears[i] = birthDateYears[j];
                birthDateMonths[i] = birthDateMonths[j];
                birthDateDays[i] = birthDateDays[j];
                salaries[i] = salaries[j];

                strcpy(names[j], tempName);
                birthDateYears[j] = tempBirthDateYear;
                birthDateMonths[j] = tempBirthDateMonth;
                birthDateDays[j] = tempBirthDateDay;
                salaries[j] = tempSalary;
            }
        }
    }

    for(i=0;i<count;i++)
    {
        printf("%s\t%d/%d/%d\t%d\r\n", names[i],
               birthDateDays[i], birthDateMonths[i],
               birthDateYears[i], salaries[i]);
    }
}
```

It is clear that code size is huge with respect to the problem complexity. Also the storage of the single employee information in different arrays complicates many operations like entering, sorting and printing

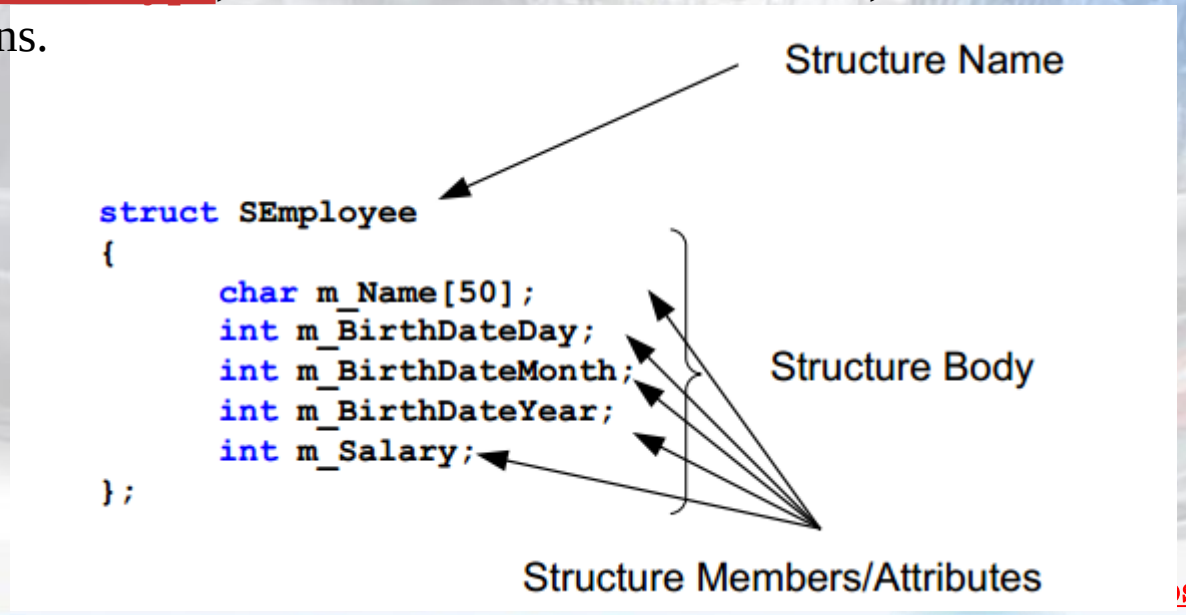
- **Structures** are used to collect related variables in one place. In this example all employee information can gathered in one place. This will simplify any further operation over the collected data, because all information items are treated as a single entity, called a **Structure**.
  Following format is used to define an employee structure, which is used to collect all employee information in one place.

```
struct SEmployee
{
        char name[50];
        int birthDateDay;
        int birthDateMonth;
        int birthDateYear;
        int Salary;
};
```

struct keyword is used to inform that all data types between the next braces {…} are members of this new data type. You can define as many data types inside the structure.

Structure

Structure is a complex data type consisting of a set of members/attributes. Each attribute can have any data type. It can be **int**, **float**, **short**…its. It can be single value or array. It can be const or variable. Also, **structures may hold another structure (nested structure).**
Programmers are free to define as many structures as the program required. Each structure must be labeled with a unique label called **Structure Name**.
Structure Name can be used as a new data type, it can be used to define variables, it can be used to define arrays and it can be passed to a functions.



Defining and Using Structure


```c
#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SEmployee
{
	char m_Name[50];
	int m_BirthDateDay;
	int m_BirthDateMonth;
	int m_BirthDateYear;
	int m_Salary;
};

void main()
{
	struct SEmployee X;

	strcpy(X.m_Name, "Ahmed Said");
	X.m_BirthDateYear = 1980;
	X.m_BirthDateMonth = 12;
	X.m_BirthDateDay = 22;
	X.m_Salary = 50;

	printf("X contains(%s, %d/%d/%d, %d)\r\n", X.m_Name,
		X.m_BirthDateDay, X.m_BirthDateMonth,
		X.m_BirthDateYear, X.m_Salary);
}
```

In above example a variable X is defined from the data type (`SEmployee`). (`X`) variable contains the following members (`m_Name`, `m_BirthDateYear`, `m_BirthDateMonth`, `m_BirthDateDay`, `m_Salary`). In this example a values is assigned to each member of (`X`) variable, then all members" values is printed.

Copying Structure Variable Contents to another Variable


```
#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SEmployee
{
	char m_Name[50];
	int m_BirthDateDay;
	int m_BirthDateMonth;
	int m_BirthDateYear;
	int m_Salary;
};

void main()
{
	struct SEmployee X, Y, Z;

	strcpy(X.m_Name, "Ahmed Said");
	X.m_BirthDateDay = 22;
	X.m_BirthDateMonth = 12;
	X.m_BirthDateYear = 1980;
	X.m_Salary = 50;

	printf("X contains(%s, %d/%d/%d, %d)\r\n", X.m_Name,
		X.m_BirthDateDay, X.m_BirthDateMonth,
		X.m_BirthDateYear, X.m_Salary);

	strcpy(Y.m_Name, X.m_Name);
	Y.m_BirthDateDay = X.m_BirthDateDay;
	Y.m_BirthDateMonth = X.m_BirthDateMonth;
	Y.m_BirthDateYear = X.m_BirthDateYear;
	Y.m_Salary = X.m_Salary;

	printf("Y contains(%s, %d/%d/%d, %d)\r\n", Y.m_Name,
		Y.m_BirthDateDay, Y.m_BirthDateMonth,
		Y.m_BirthDateYear, Y.m_Salary);

	Z = X;

	printf("Z contains(%s, %d/%d/%d, %d)\r\n", Z.m_Name,
		Z.m_BirthDateDay, Z.m_BirthDateMonth,
		Z.m_BirthDateYear, Z.m_Salary);
}
```

Employee Sort with Structures
Sort with Ages


```c
#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SEmployee
{
	char m_Name[50];
	int m_BirthDateDay;
	int m_BirthDateMonth;
	int m_BirthDateYear;
	int m_Salary;
};
```

Employee Sort with Structures Sort with Ages

```
void main()
{
	struct SEmployee employees[100], tempEmployee;
	int count = 0, i, j;
	char firstName[50], secondName[50];

	do
	{
		printf("Enter Employee First Name:");
		scanf("%s", firstName);

		printf("Enter Employee Second Name:");
		scanf("%s", secondName);
		strcpy(employees[count].m_Name, firstName);
		strcat(employees[count].m_Name, " ");
		strcat(employees[count].m_Name, secondName);

		printf("Enter Employee Birth Date (day/month/year)
			example(23/3/1970):");
		scanf("%d/%d/%d",
			&employees[count]. m_BirthDateDay,
			&employees[count].m_BirthDateMonth,
			&employees[count].m_BirthDateYear);

		printf("Enter Employee Salary:");
		scanf("%d", &employees[count].m_Salary);

		count++; if(count==100)break;

		printf("Do you want to add more, press 'y' to
			continue?\r\n");
	}
	while(getch()=='y');
```


The code (**SEmployee employees[100]**) means that defining an array of 100 variable of type (**struct SEmployee**)

Employee Sort with Structures
Sort with Ages

```
        for(i=0;i<count-1;i++)
        {
                for(j=i+1;j<count;j++)
                {
                        if(
employees[i].m_BirthDateYear>employees[i].m_BirthDateYear ||
(employees[i].m_BirthDateYear==employees[j].m_BirthDateYear &&
employees[i].m_BirthDateMonth>employees[j].m_BirthDateMonth)||
(employees[i].m_BirthDateYear==employees[j].m_BirthDateYear &&
employees[i].m_BirthDateMonth==employees[j].m_BirthDateMonth
&& employees[i].m_BirthDateDay>employees[j].m_BirthDateDay))
                        {
                                tempEmployee = employees[i];
                                employees[i] = employees[j];
                                employees[j] = tempEmployee;
                        }
```

Employee Sort with Structures
Sort with Ages


```
            }
        }

        for(i=0;i<count;i++)
        {
            printf("%s\t%d/%d/%d\t%d\r\n",
            employees[i].m_Name, employees[i].m_BirthDateDay,
            employees[i].m_BirthDateMonth,
            employees[i].m_BirthDateYear,
            employees[i].m_Salary);
        }
}
```

With Structure Vs without Structure

Employees Exchange without Structures	Employees Exchange with Structures

`tempSerial = serials[i];`
`strcpy(tempName, names[i]);`
`tempBirthDateYear = birthDateYears[i];`
`tempBirthDateMonth = birthDateMonths[i];`
`tempBirthDateDay = birthDateDays[i];`
`tempSalary = salaries[i];`
 	
`serials[i] = serials[j];`
`strcpy(names[i], names[j]);`
`birthDateYears[i] = birthDateYears[j];`
`birthDateMonths[i] = birthDateMonths[j];`
`birthDateDays[i] = birthDateDays[j];`
`salaries[i] = salaries[j];`
 	
`serials[j] = tempSerial;`
`strcpy(names[j], tempName);`
`birthDateYears[j] = tempBirthDateYear;`
`birthDateMonths[j] = tempBirthDateMonth;`
`birthDateDays[j] = tempBirthDateDay;`
`salaries[j] = tempSalary;`
	`tempEmployee = employees[i];`
	
	`employees[i] = employees[j];`
	
	`employees[j] = tempEmployee;`

Initializing Structure Variables

```
void main()
{
        struct SEmployee X = {"Ahmed Said", 22, 12, 1980, 500};
        printf("X contains(%s, %d/%d/%d, %d)\r\n", X.m_Name,
                X.m_BirthDateDay, X.m_BirthDateMonth,
                X.m_BirthDateYear, X.m_Salary);
}
```

The values `{"Ahmed Said", 22, 12, 1980, 500}` are used to initialize members respctively.

Nested Structure Definition

As mentioned before, it is applicable to define structure members with any data type, even other structures type.

Structure

```
struct SEmployee
{
        char m_Name[50];
        int m_BirthDateDay;
        int m_BirthDateMonth;
        int m_BirthDateYear;
        int m_GraduationDateDay;
        int m_GraduationDateMonth;
        int m_GraduationDateYear;
        int m_Salary;
};
```

Nested Structure

```
struct SDate
{
        int m_Day;
        int m_Month;
        int m_Year;
};
struct SEmployee
{
        char m_Name[50];
        struct SDate m_BirthDate;
        struct SDate m_GraduationDate;
        int m_Salary;
};
```

Nested Structure

```
SEmployee X = {"Ahmed Said", {22, 12, 1990},
                             {2, 7, 1970}, 5000};
printf("X contains(%s, %d/%d/%d, %d/%d/%d, %d)\r\n",
       X.m_Name, X.m_BirthDate.m_Day,
       X.m_BirthDate.m_Month, X.m_BirthDate.m_Year,
       X.m_GraduationDate.m_Day,
       X.m_GraduationDate.m_Month,
       X.m_GraduationDate.m_Year, X.m_Salary);
```

The code (**X.m_BirthDate.m_Day**) means reading the **m_Day** member of **m_BirthDate** of **X** variable.

Employee Sort with Nested Structures


```c
#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SDate
{
	int m_Day;
	int m_Month;
	int m_Year;
};

struct SEmployee
{
	char m_Name[50];
	struct SDate m_GraduationDate;
	struct SDate m_BirthDate;
	int m_Salary;
};


void main()
```

Employee Sort with Nested Structures

```
{
        struct SEmployee employees[100], tempEmployee;
        int count = 0, i, j;
        char firstName[50], secondName[50];

        do
        {
                printf("Enter Employee First Name:");
                scanf("%s", firstName);

                printf("Enter Employee Second Name:");
                scanf("%s", secondName);
                strcpy(employees[count].m_Name, firstName);
                strcat(employees[count].m_Name, " ");
                strcat(employees[count].m_Name, secondName);

                printf("Enter Employee Birth Date (day/month/year)
                                                example(23/3/1970):");
                scanf("%d/%d/%d",
                        &employees[count].m_BirthDate.m_Day,
                        &employees[count].m_BirthDate.m_Month,
                        &employees[count].m_BirthDate.m_Year);

                printf("Enter Employee Graduation Date
                            (day/month/year) example(23/3/1970):");
                scanf("%d/%d/%d",
                        &employees[count].m_GraduationDate.m_Day,
                        &employees[count].m_GraduationDate.m_Month,
                        &employees[count].m_GraduationDate.m_Year);

                printf("Enter Employee Salary:");
                scanf("%d", &employees[count].m_Salary);

                count++; if(count==100)break;

                printf("Do you want to add more, press 'y'
                                                to continue?\r\n");
        }
        while(getch()=='y');
```


Employee Sort with Nested Structures

```
        for(i=0;i<count-1;i++)
        {
                for(j=i+1;j<count;j++)
                {
                        if(
employees[i].m_BirthDate.m_Year>employees[i].m_BirthDate.m_Year ||
(employees[i].m_BirthDate.m_Year==employees[j].m_BirthDate.m_Year &&
employees[i].m_BirthDate.m_Month>employees[j].m_BirthDate.m_Month) ||
(employees[i].m_BirthDate.m_Year==employees[j].m_BirthDate.m_Year &&
employees[i].m_BirthDate.m_Month==employees[j].m_BirthDate.m_Month &&
employees[i].m_BirthDate.m_Day>employees[j].m_BirthDate.m_Day))
                        {
                                tempEmployee = employees[i];
                                employees[i] = employees[j];
                                employees[j] = tempEmployee;
                        }
                }
```

Employee Sort with Nested Structures


```
        }

        for(i=0;i<count;i++)
        {
                printf("%s\t%d/%d/%d\t%d/%d/%d\t%d\r\n",
                                employees[i].m_Name,
                                employees[i].m_BirthDate.m_Day,
                                employees[i].m_BirthDate.m_Month,
                                employees[i].m_BirthDate.m_Year,
                                employees[i].m_GraduationDate.m_Day,
                                employees[i].m_GraduationDate.m_Month,
                                employees[i].m_GraduationDate.m_Year,
                                employees[i].m_Salary);

        }
}
```

Using Structures with Functions

Read and Print Employee and Date Values using Functions


```c
#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SDate
{
	int m_Day;
	int m_Month;
	int m_Year;
};

struct SEmployee
{
	char m_Name[50];
	struct SDate m_GraduationDate;
	struct SDate m_BirthDate;
	int m_Salary;
};

struct SDate ReadDate(char dateName[])
{
	struct SDate date;
	printf("Enter %s (day/month/year) example(23/3/1970):",
	                                                dateName);
```

Using Structures with Functions

Read and Print Employee and Date Values using Functions

```
        scanf("%d/%d/%d", &date.m_Day, &date.m_Month,
                                        &date.m_Year);
        return date;
}

struct SEmployee ReadEmploee()
{
        struct SEmployee employee;
        char firstName[50], secondName[50];

        printf("Enter Employee First Name:");
        scanf("%s", firstName);

        printf("Enter Employee Second Name:");
        scanf("%s", secondName);

        strcpy(employee.m_Name, firstName);
        strcat(employee.m_Name, " ");
        strcat(employee.m_Name, secondName);

        employee.m_BirthDate = ReadDate("Employee Birth Date");
        employee.m_GraduationDate =
                    ReadDate("Employee Graduation Date");

        printf("Enter Employee Salary:");
        scanf("%d", &employee.m_Salary);

        return employee;
}
```


Using Structures with Functions

Read and Print Employee and Date Values using Functions


```
void PrintEmploee(struct SEmployee emploee)
{
	printf("%s\t%d/%d/%d\t%d/%d/%d\t%d\r\n",
				emploee.m_Name,
				emploee.m_BirthDate.m_Year,
				emploee.m_BirthDate.m_Month,
				emploee.m_BirthDate.m_Day,
				emploee.m_GraduationDate.m_Year,
				emploee.m_GraduationDate.m_Month,
				emploee.m_GraduationDate.m_Day,
				emploee.m_Salary);
}

void main()
{
	struct SEmployee X = ReadEmploee();
	PrintEmploee(X);
}
```

Write A program Adding Two Complex Numbers using Structure And Function

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Lab

```
struct SComplex
{
	double m_R; //Real Part
	double m_I; //Imaginary Part
};
```

```
struct SComplex ReadComplex(char name[])
```

```
struct SComplex AddComplex(struct SComplex A, struct SComplex B)
```

```
void PrintComplex(char name[], struct SComplex C)
```


```c
#include "stdio.h"

struct SComplex
{
	double m_R; //Real Part
	double m_I; //Imaginary Part
};

struct SComplex ReadComplex(char name[])
{
	struct SComplex C;
	printf("Enter %s Complex Value (Ex: 5, -3):", name);
	scanf("%lf, %lf", &C.m_R, &C.m_I);

	return C;
}

struct SComplex AddComplex(struct SComplex A, struct SComplex
B)
```

Lab Solution


```c
#include "stdio.h"

struct SComplex
{
	double m_R; //Real Part
	double m_I; //Imaginary Part
};

struct SComplex ReadComplex(char name[])
{
	struct SComplex C;
	printf("Enter %s Complex Value (Ex: 5, -3):", name);
	scanf("%lf, %lf", &C.m_R, &C.m_I);

	return C;
}

struct SComplex AddComplex(struct SComplex A, struct SComplex
B)
```

Lab Solution


```
{
        struct SComplex C;
        C.m_R = A.m_R + B.m_R;
        C.m_I = A.m_I + B.m_I;
        return C;
}

void PrintComplex(char name[], struct SComplex C)
{
        printf("%s = (%lf) + j (%lf)\r\n", name, C.m_R, C.m_I);
}

void main()
{
        struct SComplex X, Y, Z;

        X = ReadComplex("X");
        Y = ReadComplex("Y");
        Z = AddComplex(X, Y);
        PrintComplex("Z", Z);
}
```

Lab Solution

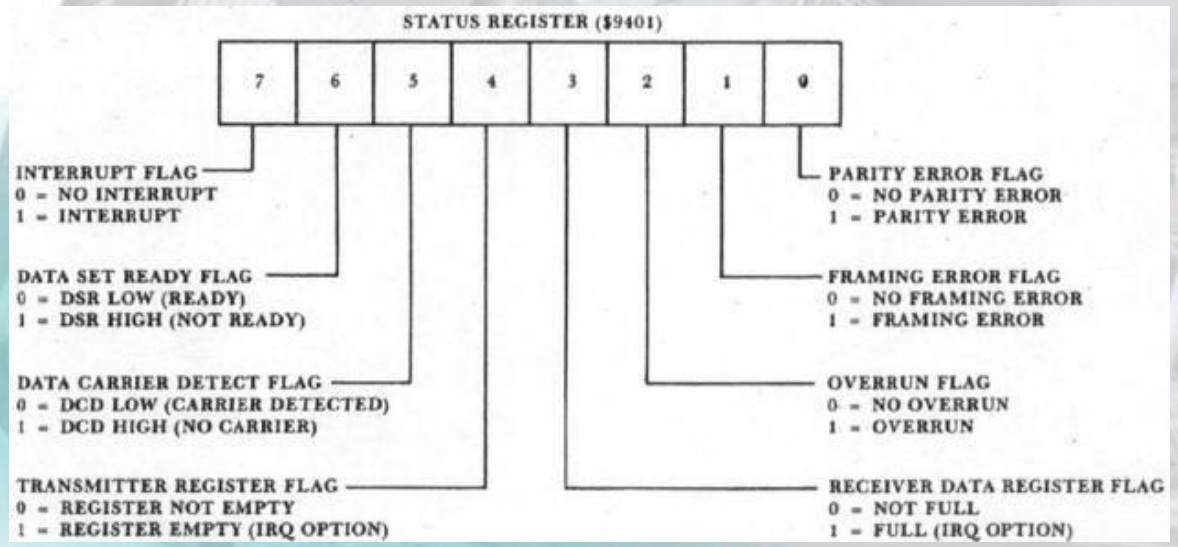
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Access Wrong Bit on the Register

one of the most famous issue for the embedded programmer

Functionality issue



https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Bit-Fields

THINK FIRST THEN ANSWER

HOW TO USE STRUCTURE ON ACCESS SPECIFIC BIT ON THE REGISTERS ?

https://www.facebook.com/groups/embedded.system.KS/

Bit-Fields

- Unlike some other computer languages, **C** has a built-in feature, called ***a bit-field***, that allows you to **access a single bit**. Bit-fields can be useful for a number of reasons, such as:
  - If storage is **limited**, you can store **several Boolean (true/false) variables** in **one byte.**
  - Certain devices transmit status information encoded into one or more bits within a byte
  - Certain encryption routines need to access the bits within a byte.

Bit-Fields

- **A bit-field** must be a **member** of a **structure or union**. It defines how long, in bits, the field is to be.
  The general form of a bit-field definition is
  type name: length;
- *type* is the type of the bit-field, and *length* is the number of bits in the field. The type of a bitfield must be **int**, **signed**, or **unsigned**. (C99 also allows a bit-field to be of type **_Bool**.)
- **Bit-fields are frequently used when analyzing input from a hardware device.**

Bit-Fields

For example, the status port of a serial communications adapter might return a status byte organized like this:

Bit	Meaning When Set
0	Change in clear-to-send line
1	Change in data-set-ready
2	Trailing edge detected
3	Change in receive line
4	Clear-to-send
5	Data-set-ready
6	Telephone ringing
7	Received signal

```
struct status_type {
    unsigned char delta_cts:1;
    unsigned char delta_dsr:1;
    unsigned char tr_edge:1;
    unsigned char delta_rec:1;
    unsigned char cts:1;
    unsigned char dsr:1;
    unsigned char ring:1;
    unsigned char rec_line:1;
} status;
```


Bit-Fields

- You might use statements like the ones shown here to enable a program to determine when it can send or receive data:

Bit	Meaning When Set
0	Change in clear-to-send line
1	Change in data-set-ready
2	Trailing edge detected
3	Change in receive line
4	Clear-to-send
5	Data-set-ready
6	Telephone ringing
7	Received signal

```
struct status_type {
  unsigned char delta_cts:1;
  unsigned char delta_dsr:1;
  unsigned char tr_edge:1;
  unsigned char delta_rec:1;
  unsigned char cts:1;
  unsigned char dsr:1;
  unsigned char ring:1;
  unsigned char rec_line:1;
} status;
```

```
status = get_port_status();
if(status.cts) printf("clear to send");
if(status.dsr) printf("data ready");
```

What is the Output ?

FOLLOW US

Press here

33

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.s main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>
struct status_type {
    unsigned char delta_cts:1;
    unsigned char delta_dsr:1;
    unsigned char tr_edge:1;
    unsigned char delta_rec:1;
    unsigned char cts:1;
    unsigned char dsr:1;
    unsigned char ring:1;
    unsigned char rec_line:1;
} status;
int main(int argt ,char**argv) {
    status.cts = 1 ;
    printf ("sizeof structure = %d",sizeof(status));
    return 0   ;
}
```

main.s main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>
struct status_type {
    unsigned int delta_cts:1;
    unsigned int delta_dsr:1;
    unsigned int tr_edge:1;
    unsigned int delta_rec:1;
    unsigned int cts:1;
    unsigned int dsr:1;
    unsigned int ring:1;
    unsigned int rec_line:1;
} status;
int main(int argt ,char**argv) {
    status.cts = 1 ;
    printf ("sizeof structure = %d",sizeof(status));
    return 0   ;
}
```

What is the Output ?

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

main.s | main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>
struct status_type {
    unsigned char delta_cts:1;
    unsigned char delta_dsr:1;
    unsigned char tr_edge:1;
    unsigned char delta_rec:1;
    unsigned char cts:1;
    unsigned char dsr:1;
    unsigned char ring:1;
    unsigned char rec_line:1;
} status;
int main(int argt ,char**argv) {
    status.cts =
    printf ("size
    return 0   ;
}
```

Problems | AVR Supported MCUs | T

<terminated> (exit value: 0) session2.exe [C/C+

atus));

```
sizeof structure = 1
```

main.s | main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>
struct status_type {
    unsigned int delta_cts:1;
    unsigned int delta_dsr:1;
    unsigned int tr_edge:1;
    unsigned int delta_rec:1;
    unsigned int cts:1;
    unsigned int dsr:1;
    unsigned int ring:1;
    unsigned int rec_line:1;
} status;
int main(int argt ,char**argv) {
    status.cts = 1 ;
    printf ("sizeof structure = %d",sizeof(status));
    return 0   ;
}
```

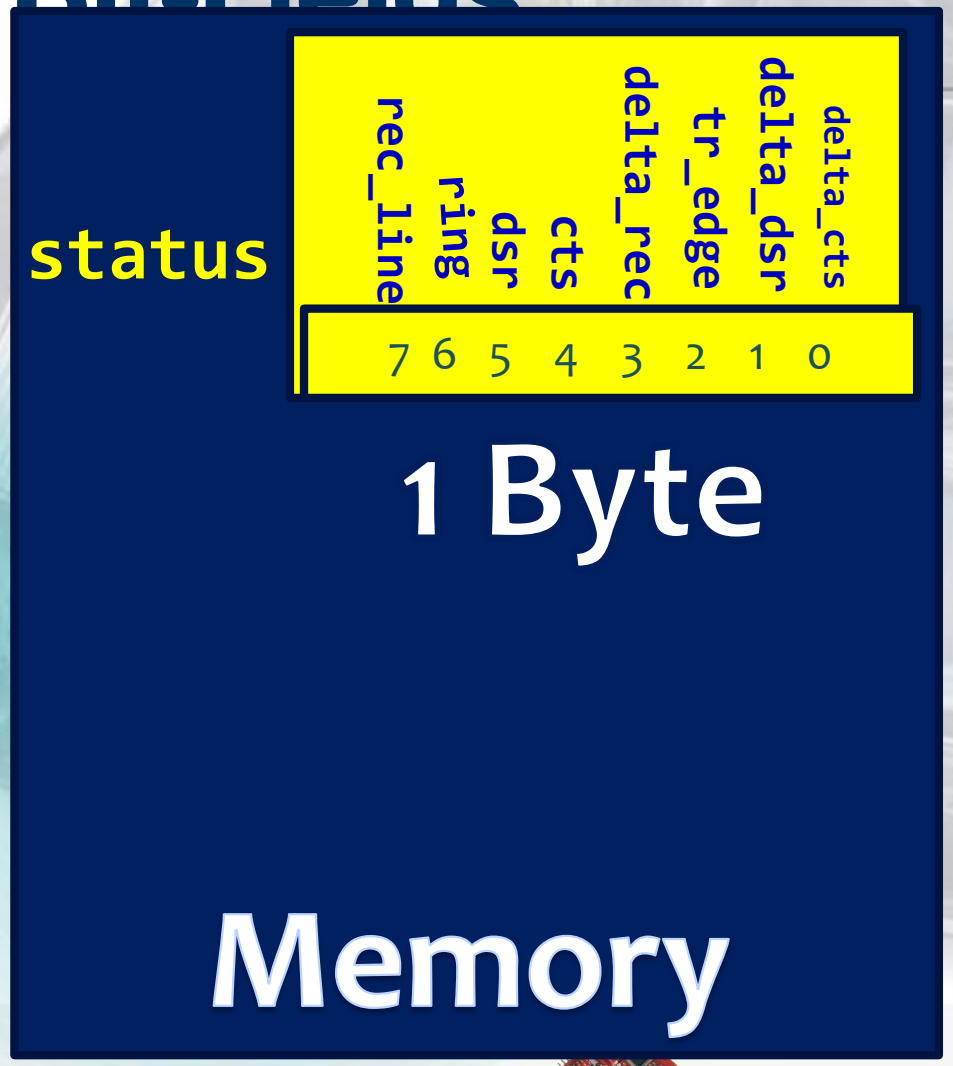
Problems | AVR Supported MCUs | Task

<terminated> (exit value: 0) session2.exe [C/C++ A

```
sizeof structure = 4
```

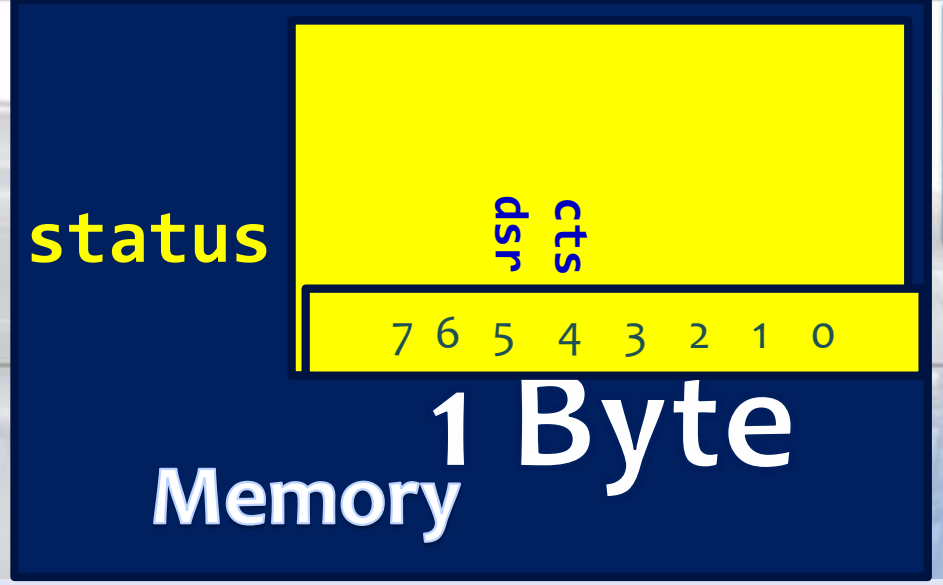
.com/groups/embedded.system.KS/

Bit-Fields




```
struct status_type {
    unsigned char delta_cts:1;
    unsigned char delta_dsr:1;
    unsigned char tr_edge:1;
    unsigned char delta_rec:1;
    unsigned char cts:1;
    unsigned char dsr:1;
    unsigned char ring:1;
    unsigned char rec_line:1;
} status;
```

Bit-Fields



- **You do not have to name each bit-field**.
  This makes it easy to reach the bit you want, bypassing unused ones.
- For example, if you only care about the **cts** and **dsr** bits, you could declare the **status_type** structure like this:
- Also, notice that the bits after **dsr** do not need to be specified if they are not used.

main.s | main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>
struct status_type {
    unsigned char :4 ;
    unsigned char cts:1;
    unsigned char dsr:1;
} status;
int main(int argt ,char**argv) {
    status.cts = 1 ;
    printf ("sizeof structure = %d",sizeof(status));
    return 0   ;
}
```

Bit-Fields

What is the Output ?

- It is valid to mix normal structure members with bit-fields. For

main.s | main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>
struct emp {
  float pay;
  unsigned lay_off:     1; /* lay off or active */
  unsigned hourly:      1; /* hourly pay or wage */
  unsigned deductions: 3; /* IRS deductions */
}test;
int main(int argt ,char**argv) {
    printf ("sizeof structure = %d",sizeof(test));
    return 0   ;
}
```

What is the Output ?

- It is valid to mix normal structure members with bit-fields. For

main.s | main.c

```
1  //Prepared by Eng.Keroles
2  #include <stdio.h>
3  struct
4  {
5    float
6    unsig
7    unsig
8    unsig
9  }test;
10 int mai
11 {
12     printf ("sizeof structure = %d",sizeof(test));
13     return 0   ;
14 }
```

Problems | AVR Supported MCUs | Tasks | C

<terminated> (exit value: 0) session2.exe [C/C++ Applicatio

```
sizeof structure = 8
```

https://w

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Bit-Fields

Bit-fields have certain restrictions

- you cannot take the address of a bit-field.
- **Bit-fields cannot be arrayed.**
- You cannot know, from machine to machine, whether the fields will run from right to left or from left to right; have some machine dependencies

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Aligned and un-aligned data access on structures

LEARN-IN-DEPTH

What is the output ?

```c
#include "stdio.h"

struct Sperson {
    unsigned char weight ;
};

void main ()
{
struct Sperson man = {100 } ;
printf ("size of man =%d ", sizeof(struct Sperson));
}
```


#LEARN_IN DEPTH

#Be_professional_in embedded_system

What is the output ?

```c
#include "stdio.h"

struct Sperson {
    unsigned char weight ;
};

void main ()
{
struct Sperson man = {100 } ;
printf ("size of man =%d ", sizeof(struct Sperson));
}

```



```
Select Windows PowerShell
PS C:\MinGW\bin> .\gcc.exe .\main.c -o .\main.exe
PS C:\MinGW\bin> .\main.exe
size of man =1
PS C:\MinGW\bin>
```

What is the output ?

```c
#include "stdio.h"

struct Sperson {
    unsigned char weight ;
    unsigned int age ;
};

void main ()
{
struct Sperson man = {100 , 50 } ;
printf ("size of man =%d ", sizeof(struct Sperson));
}

```


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

What is the output ? Why ?!

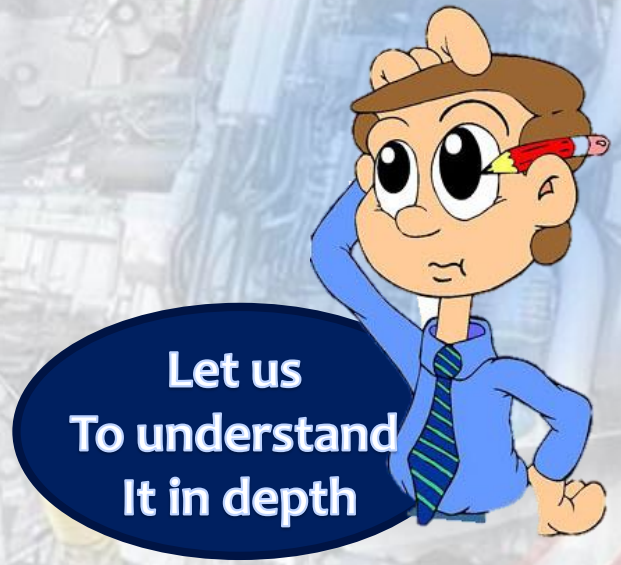
```c
#include "stdio.h"

struct Sperson {
    unsigned char weight ;
    unsigned int age ;
};

void main ()
{
struct Sperson man = {100 , 50 } ;
printf ("size of man =%d ", sizeof(struct Sperson));
}

```

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
size of man =8
PS C:\MinGW\bin>
```


#LEARN_IN DEPTH

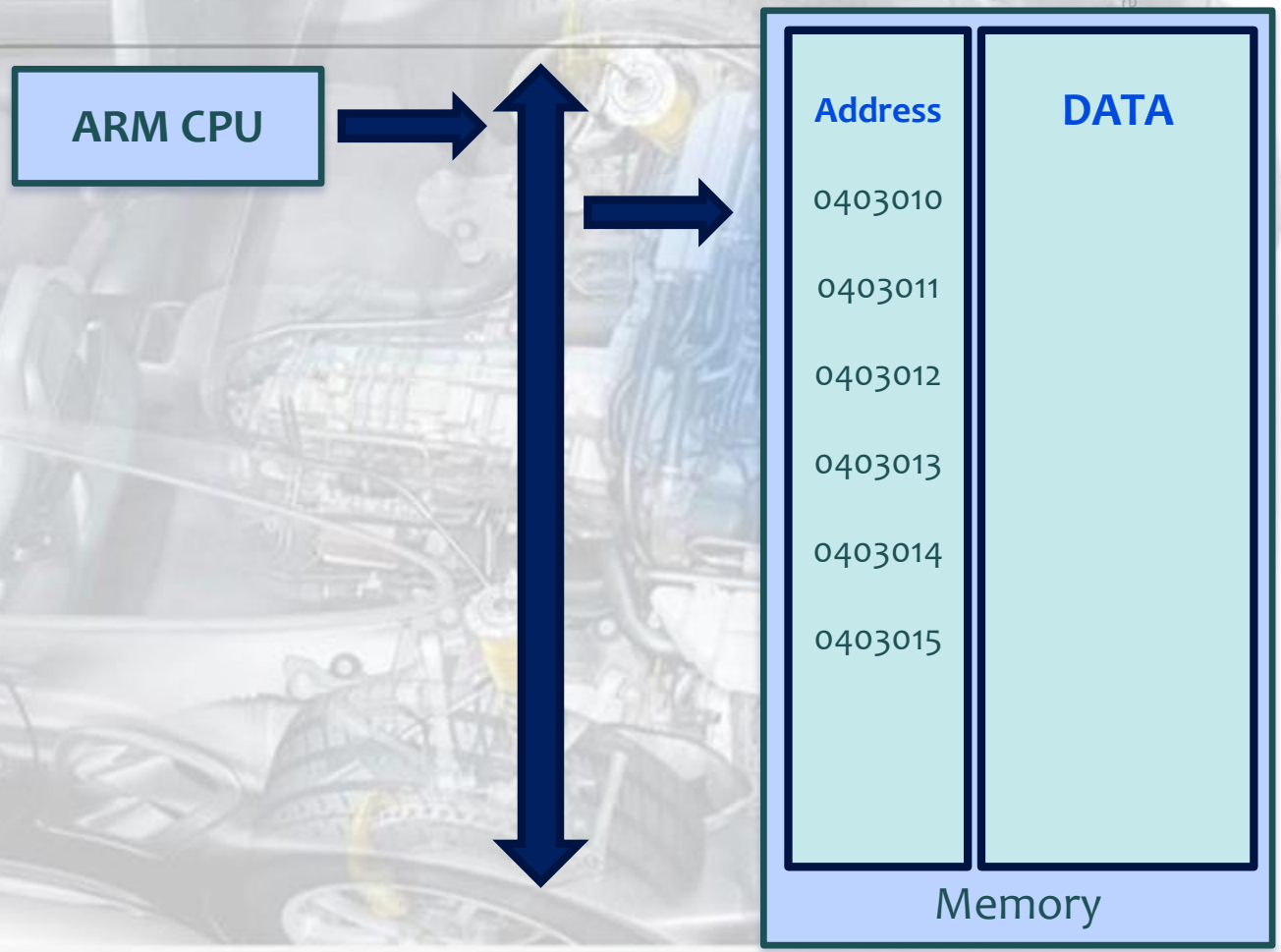
#Be_professional_in embedded_system

Aligned and un-aligned data access on structures



Aligned/un-aligned data acess

- For **efficiency**, the compiler generates instructions to store variables on their **natural size boundary** addresses in the memory.
- So Member elements of a structure are located on their natural size boundary.

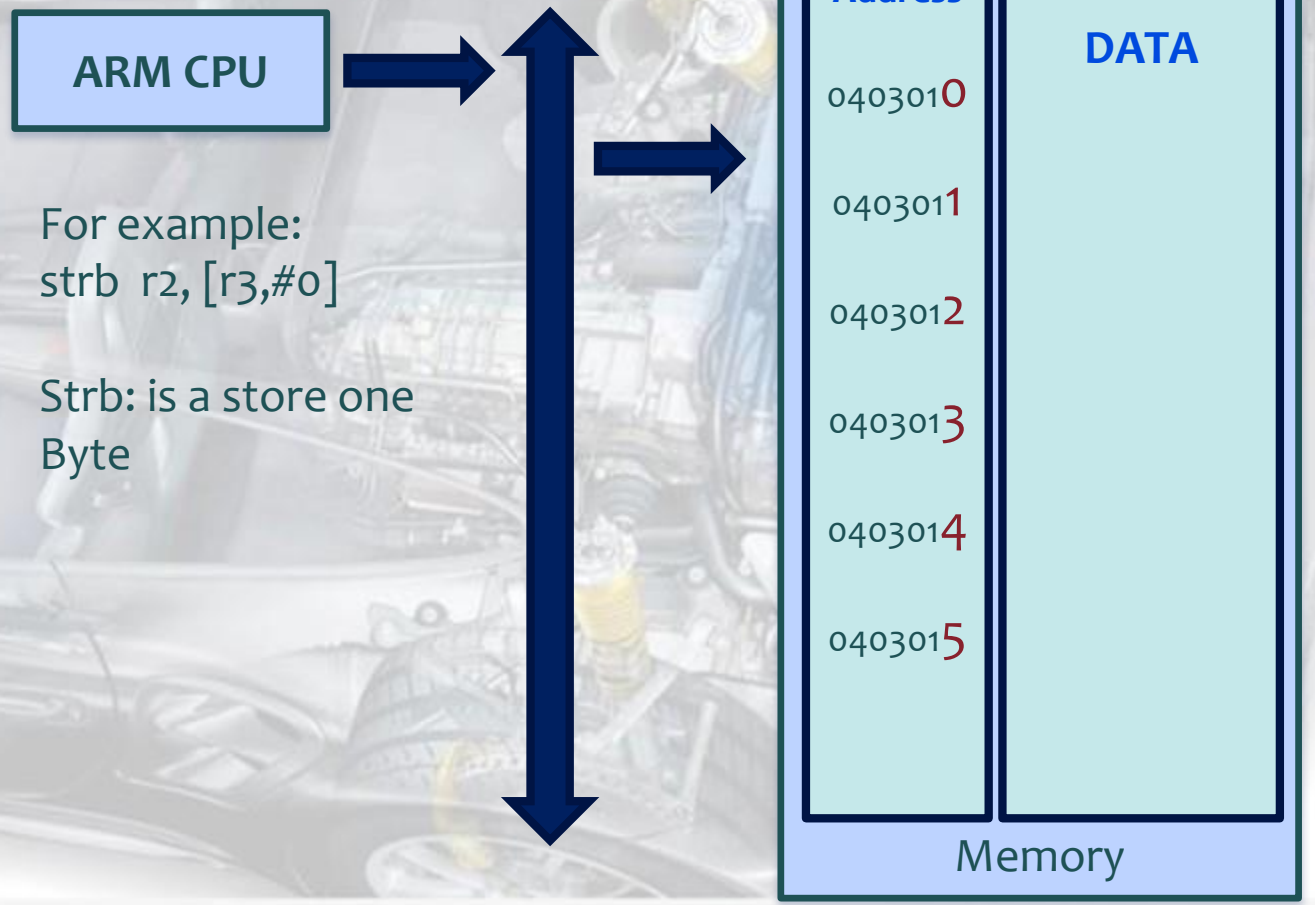


Aligned/un-aligned data acess

- For **efficiency**, the compiler generates instructions to store variables on their **natural size boundary** addresses in the memory.
- So Member elements of a structure are located on their natural size boundary.

char x=1 ;

- **The natural size of a char data type** is **1 byte**, this data-type variable can be placed then at any data address

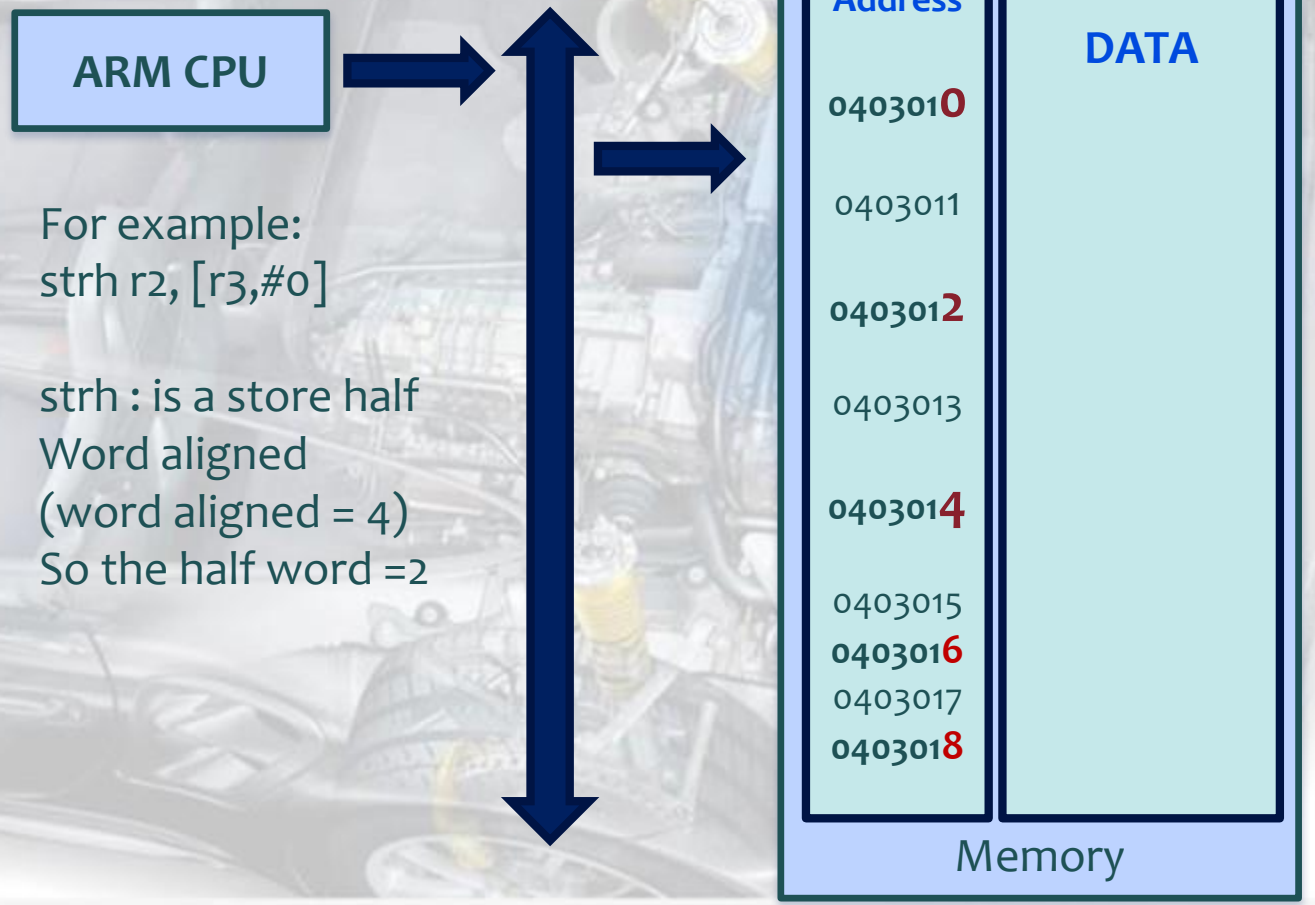


Aligned/un-aligned data acess

- For **efficiency**, the compiler generates instructions to store variables on their **natural size boundary addresses** in the memory.
- So Member elements of a structure are located on their natural size boundary.

short x=1 ;

- **The short** type variable size is 2 bytes, in this case the address number of two consecutive variables cannot be consecutive, it has to be placed after two bytes to make space to variable in memory



Aligned/un-aligned data acess

- For **efficiency**, the compiler generates instructions to store variables on their **natural size boundary** addresses in the memory.
- So Member elements of a structure are located on their natural size boundary.

Int x=1 ;

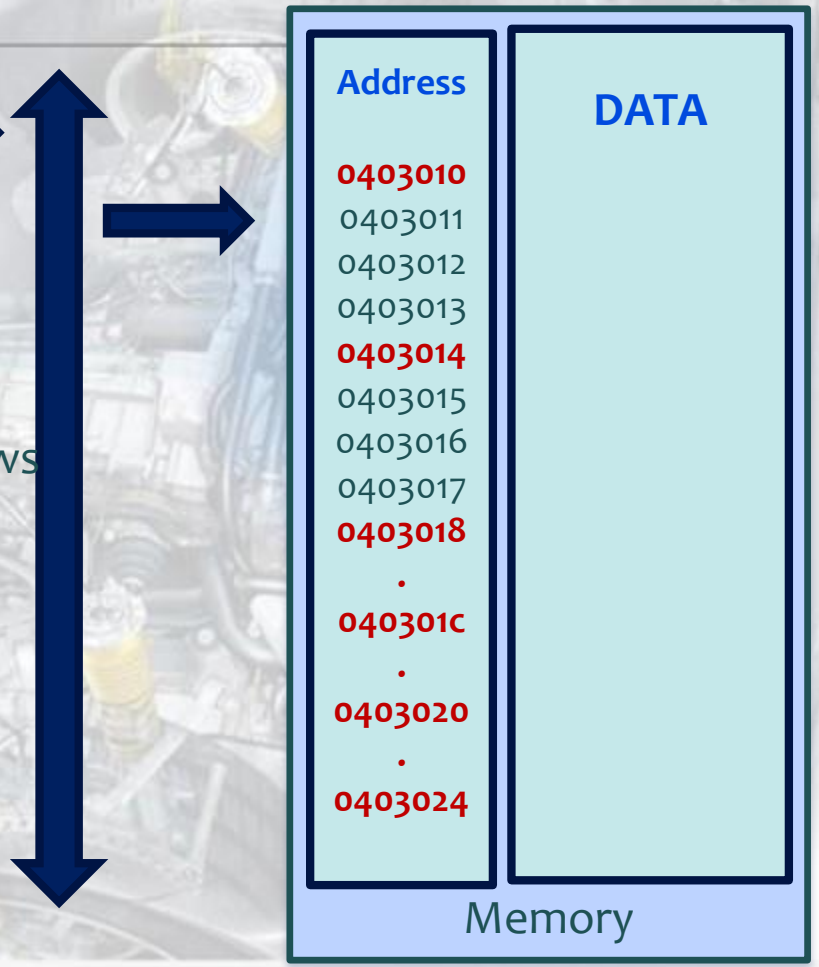
- The same can be said for int datatype which **natural size** is equal to 4 bytes.

This type of storage which always respect the **natural size of variables** is called **aligned**.

ARM CPU

For example:
str r2, [r3,#0]

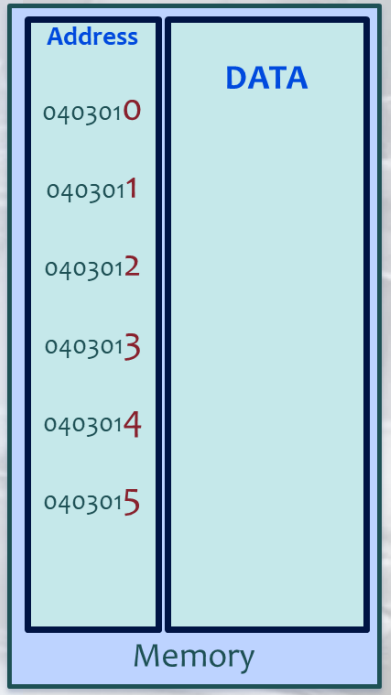
The str command allows data storage just if the data is word aligned (each 4 bytes)



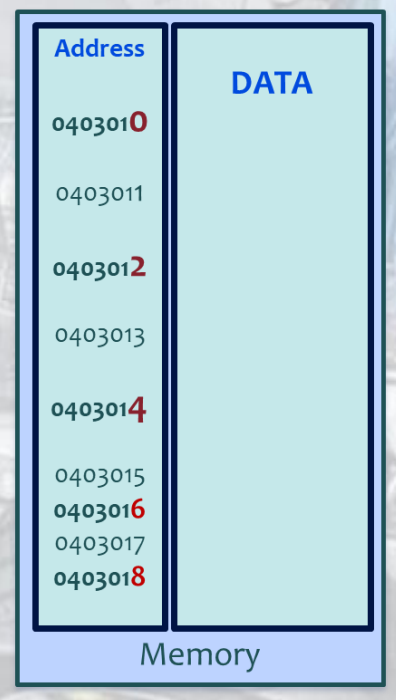
- When compile the code the compiler generate instructions to store the variables in memory according to the variables **naturel size boundary** that is called

aligned data storage

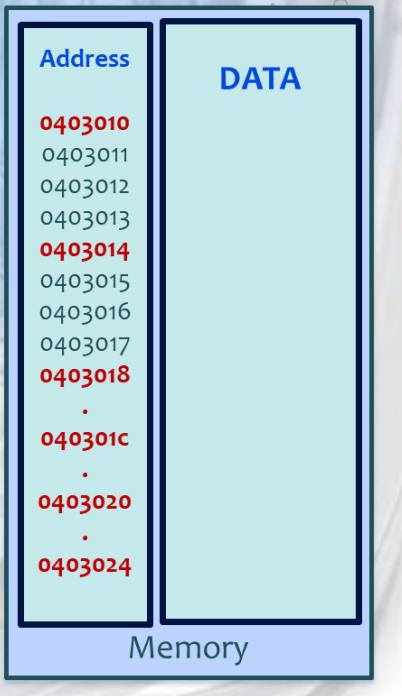
Char x;



Short x;



Int x;



Let us deep dive on this example

```c
#include "stdio.h"

struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};

struct Sdata gdata ;

void main ()
{
    printf ("Online_Diploma, LEARn-In-Depth \n");
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;
    int total_size= sizeof(struct Sdata);
    printf ("size of struct Sdata (non packing) =%d \n", sizeof(struct Sdata));
    dump_memory (&gdata ,total_size );
}

dump_memory (char* ptr ,  int size )
{
    int i ;
    for (i=0 ;i<size ; i++)
    {
        printf ("%p     %X \n",ptr,(unsigned char)*ptr);
        ptr++ ;
    }
}
```

Let us deep dive on this example

```c
#include "stdio.h"

struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};

struct Sdata gdata ;

void main ()
{
    printf ("Online_Diploma, LEARn-In-Depth \n");
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;
    int total_size= sizeof(struct Sdata);
    printf ("size of struct Sdata (non packing) =%d \n", sizeof(struct Sdata));
    dump_memory (&gdata ,total_size );
}

dump_memory (char* ptr ,  int size )
{
    int i ;
    for (i=0 ;i<size ; i++)
    {
        printf ("%p     %X \n",ptr,(unsigned char)*ptr);
        ptr++ ;
    }
}
```

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
Online_Diploma, LEARn-In-Depth
size of struct Sdata (non packing) =12
00407070        AA
00407071        0
00407072        0
00407073        0
00407074        FF
00407075        FF
00407076        FF
00407077        FF
00407078        55
00407079        0
0040707A        A5
0040707B        A5
PS C:\MinGW\bin>
```

naturel size boundary
For char is 1 `
SO no problem here

Let us deep dive on this example

```c
#include "stdio.h"

struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};

struct Sdata gdata ;

void main ()
{
    printf ("Online_Diploma, LEARn-In-Depth \n");
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;
    int total_size= sizeof(struct Sdata);
    printf ("size of struct Sdata (non packing) =%d \n", sizeof(struct Sdata));
    dump_memory (&gdata ,total_size );
}

dump_memory (char* ptr ,  int size )
{
    int i ;
    for (i=0 ;i<size ; i++)
    {
        printf ("%p     %X \n",ptr,(unsigned char)*ptr);
        ptr++ ;
    }
}
```

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
Online_Diploma, LEARn-In-Depth
size of struct Sdata (non packing) =12
00407070        AA
00407071        0
00407072        0
00407073        0
00407074        FF
00407075        FF
00407076        FF
00407077        FF
00407078        55
00407079        0
0040707A        A5
0040707B        A5
PS C:\MinGW\bin>
```

naturel size boundary
For int is 4 `
SO the next natural
size boundary is 0x..7
And will padding zero
For ..71,..72,... 73

#LEARN_IN DEPTH

#Be_professional_in embedded_system

Let us deep dive on this example

```c
#include "stdio.h"

struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};

struct Sdata gdata ;

void main ()
{
    printf ("Online_Diploma, LEARn-In-Depth \n");
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;
    int total_size= sizeof(struct Sdata);
    printf ("size of struct Sdata (non packing) =%d \n", sizeof(struct Sdata));
    dump_memory (&gdata ,total_size );
}

dump_memory (char* ptr ,  int size )
{
    int i ;
    for (i=0 ;i<size ; i++)
    {
        printf ("%p     %X \n",ptr,(unsigned char)*ptr);
        ptr++ ;
    }
}
```

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
Online_Diploma, LEARn-In-Depth
size of struct Sdata (non packing) =12
00407070        AA
00407071        0
00407072        0
00407073        0
00407074        FF
00407075        FF
00407076        FF
00407077        FF
00407078        55
00407079        0
0040707A        A5
0040707B        A5
PS C:\MinGW\bin>
```

naturel size boundary
For char is 1 `
SO no problem here

Let us deep dive on this example

```c
#include "stdio.h"

struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};

struct Sdata gdata ;

void main ()
{
    printf ("Online_Diploma, LEARn-In-Depth \n");
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;
    int total_size= sizeof(struct Sdata);
    printf ("size of struct Sdata (non packing) =%d \n", sizeof(struct Sdata));
    dump_memory (&gdata ,total_size );
}

dump_memory (char* ptr ,  int size )
{
    int i ;
    for (i=0 ;i<size ; i++)
    {
        printf ("%p     %X \n",ptr,(unsigned char)*ptr);
        ptr++ ;
    }
}
```

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
Online_Diploma, LEARn-In-Depth
size of struct Sdata (non packing) =12
00407070        AA
00407071        0
00407072        0
00407073        0
00407074        FF
00407075        FF
00407076        FF
00407077        FF
00407078        55
00407079        0
0040707A        A5
0040707B        A5
PS C:\MinGW\bin>
```

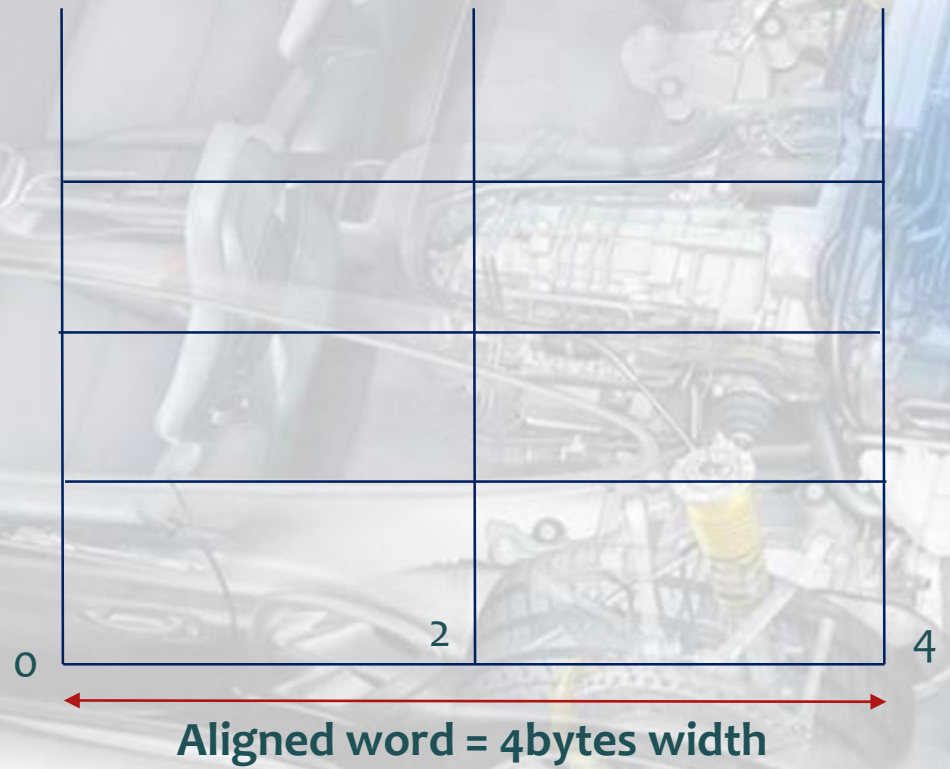
naturel size boundary For short is 2` So the next aligned nature Size boundary will be 0x.7A And padding the intermediate byte by 0

https://www.facebook.com/groups/embedded.system.KS/

Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

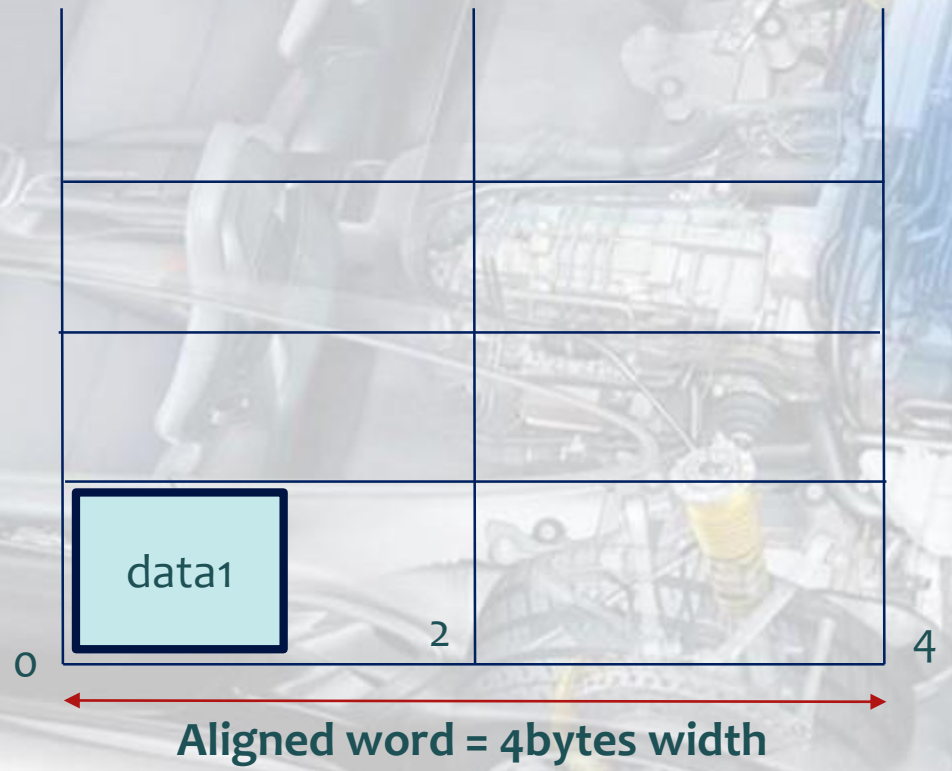
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

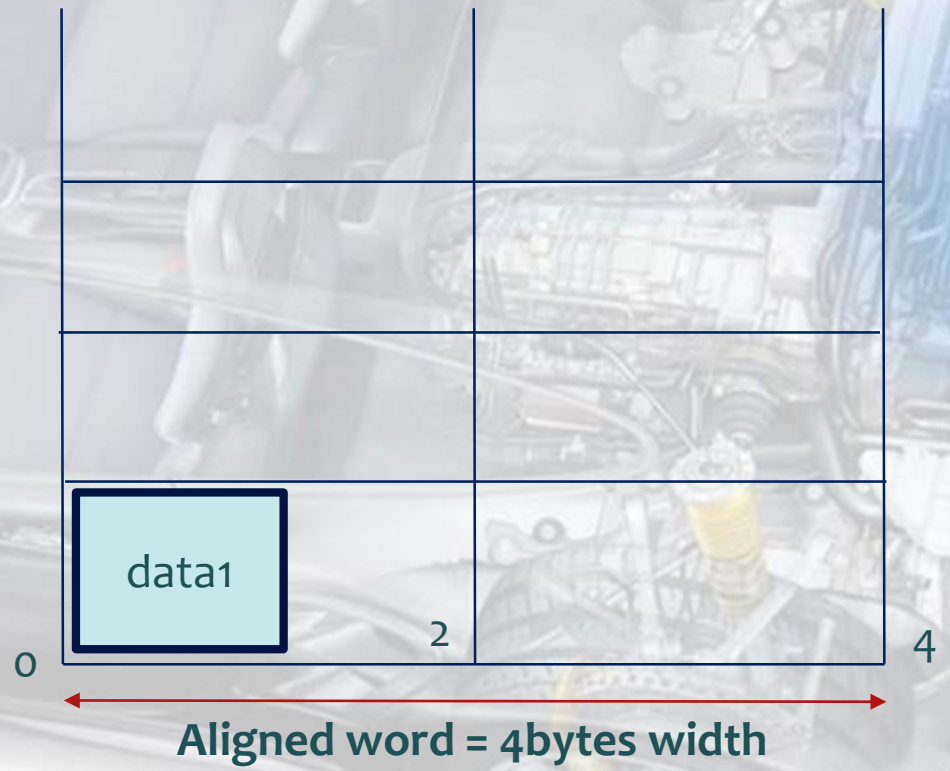
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

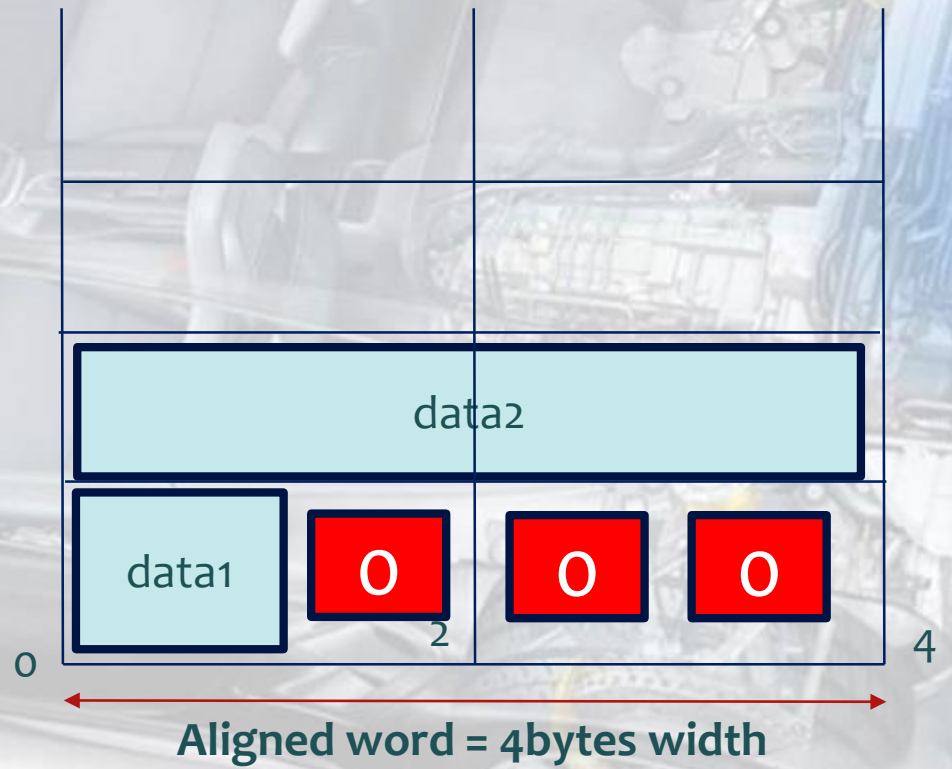
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

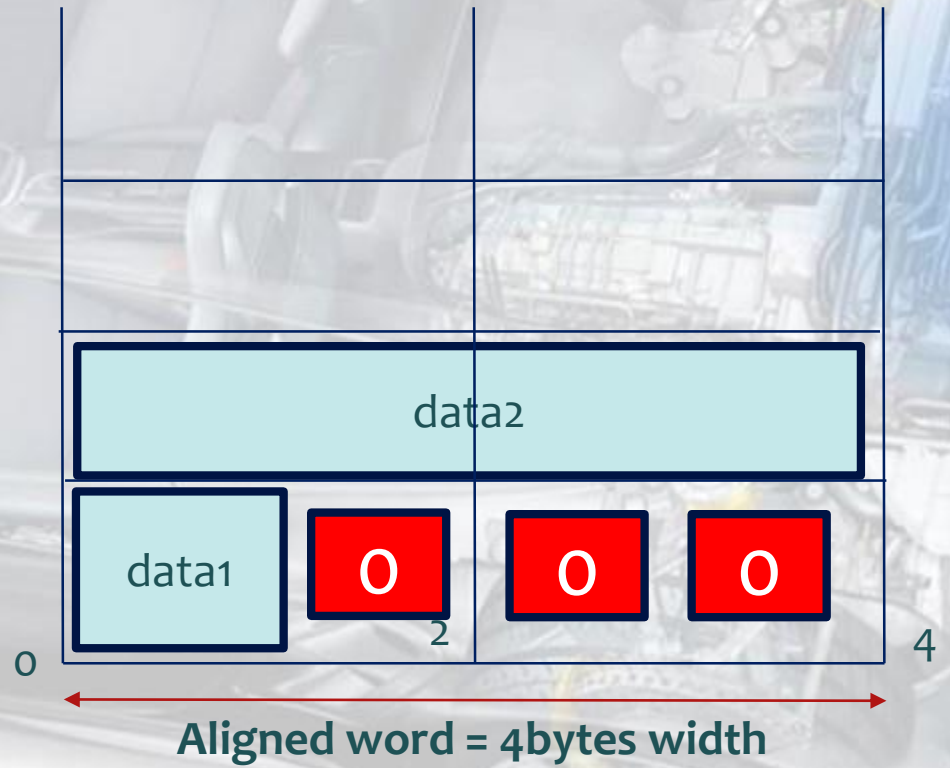
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

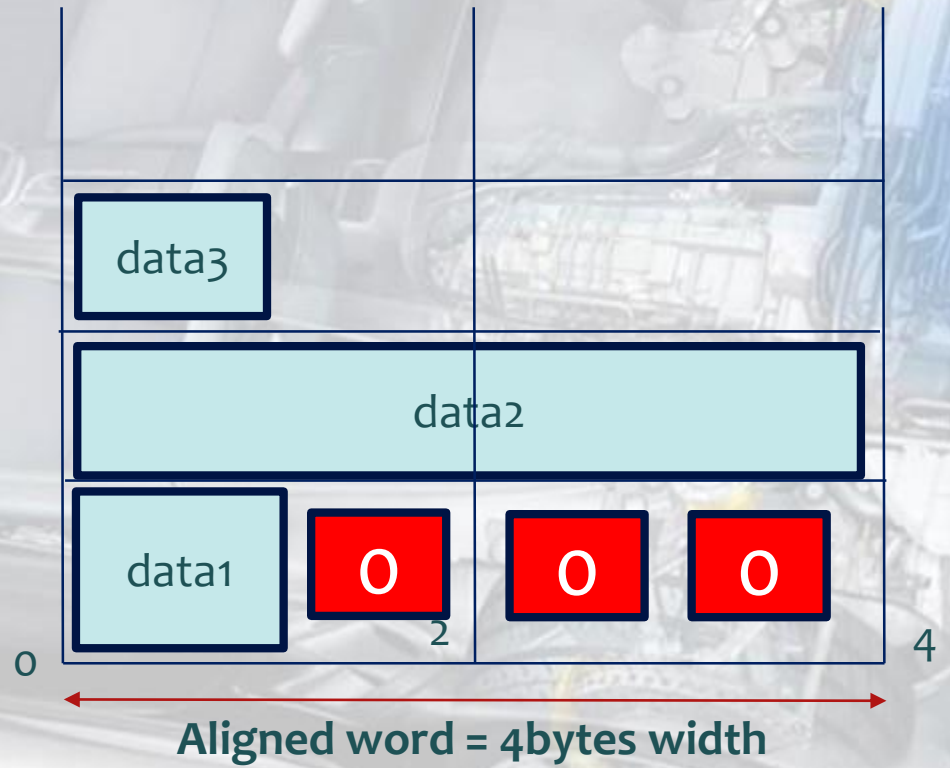
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

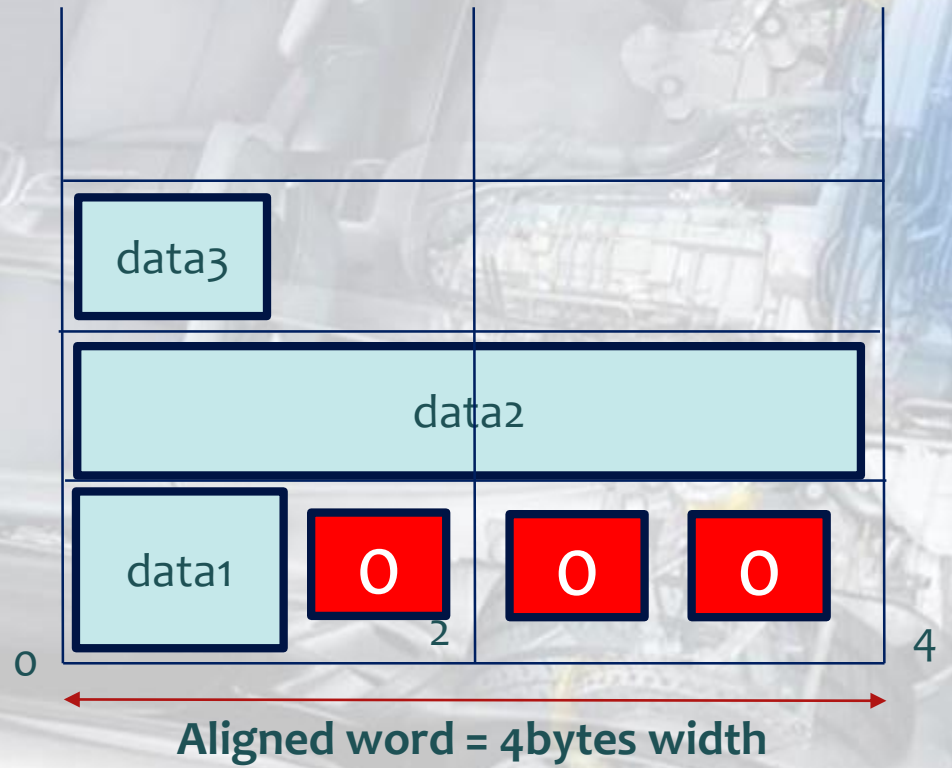
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

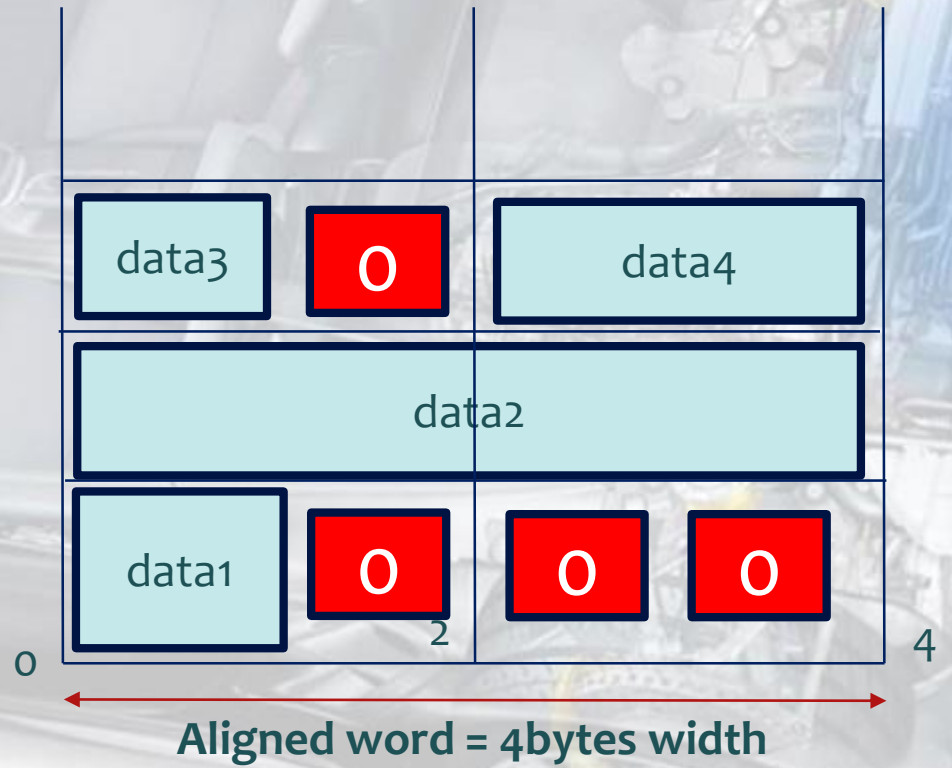
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

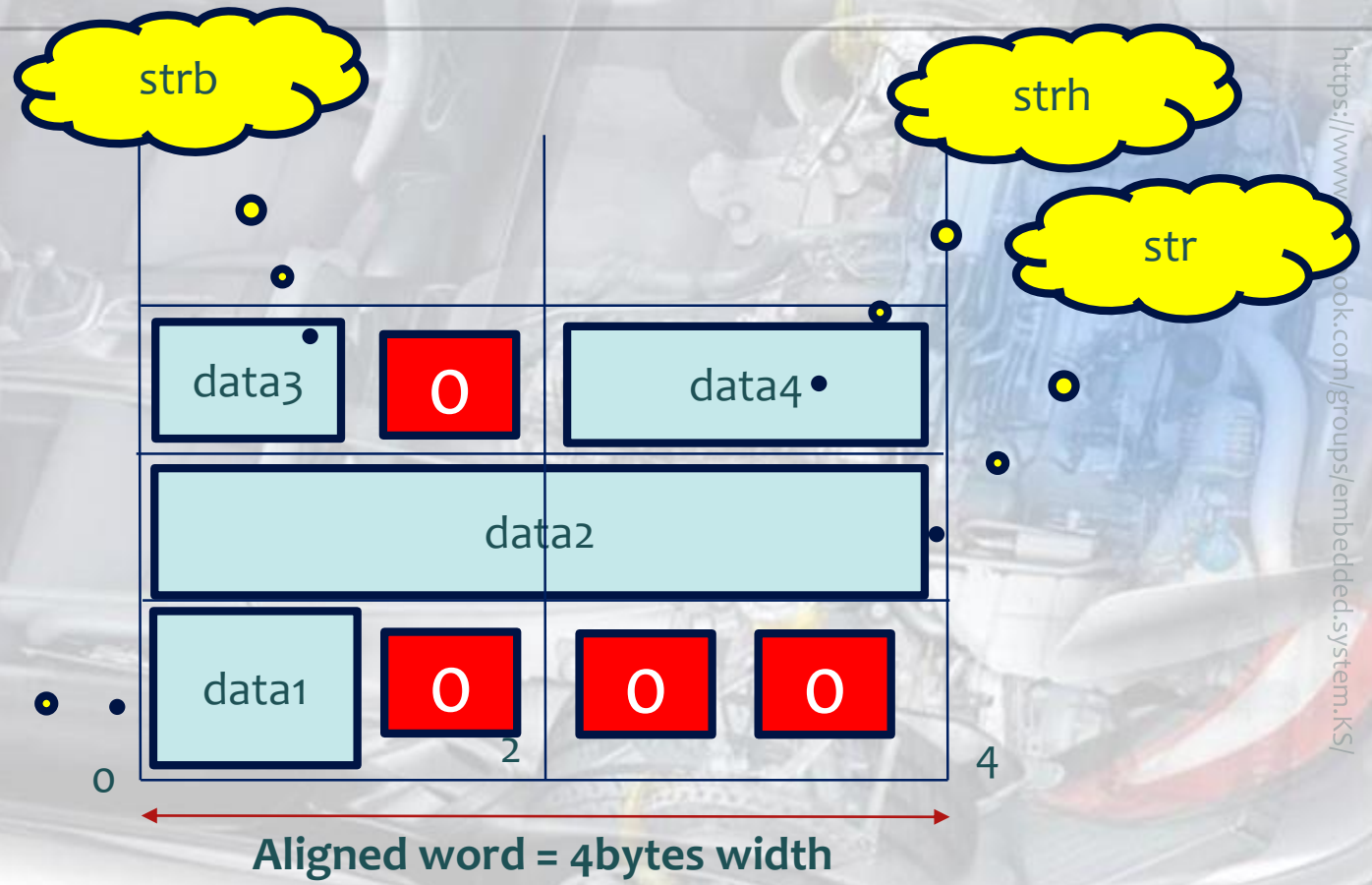
size of struct Sdata (non packing) =12



Structure padding

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

size of struct Sdata (non packing) =12



Structure packing

- To enable packing:

```
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
}__attribute__ ((packed));
```

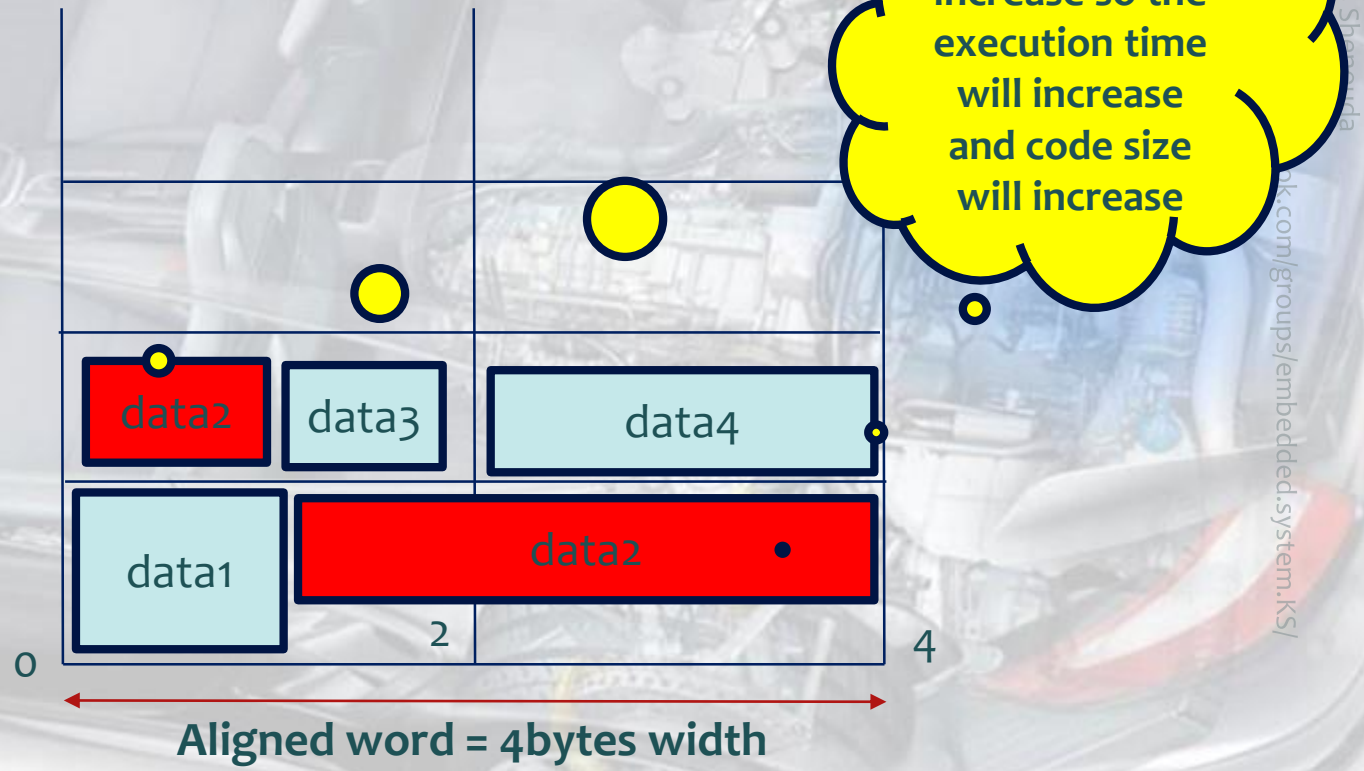
```
#pragma pack(1)
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
```

```
#pragma pack(push, 1)
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};
#pragma pack(pop)
```


Structure padding

```
3 #pragma pack(1)
4 struct Sdata {
5     unsigned char data1 ;
6     unsigned int data2 ;
7     unsigned char data3 ;
8     unsigned short data4 ;
9 };
```

size of struct Sdata (non packing) =8



Structure packing

```c
#include "stdio.h"

#pragma pack(1)
struct Sdata {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
};

struct Sdata gdata ;

dump_memory (char*ptr , int size);
void main ()
{
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;
    printf ("size of man =%d \n", sizeof(struct Sdata));
    int total_size = sizeof(struct Sdata) ;
    dump_memory (&gdata , total_size);
}
dump_memory (char*ptr , int size)
{
    int i ;
    for (i=0 ; i<size ; i++)
    {
        printf ("%p      %X \n",ptr , (unsigned char)*ptr);
        ptr++;
    }
}
```

```
Windows PowerShell
PS C:\MinGW\bin> .\main.exe
size of man =8
00407070       AA
00407071       FF
00407072       FF
00407073       FF
00407074       FF
00407075       55
00407076       A5
00407077       A5
PS C:\MinGW\bin>
```


Prove that the Code size will increase (instructions will increase)

Compile for ARM

```c
volatile unsigned int * const UART0DR = (unsigned int *)0x101f1000;

//#pragma pack(1)
struct Sperson {
    unsigned char data1 ;
    unsigned int data2 ;
    unsigned char data3 ;
    unsigned short data4 ;
} gdata;
//__attribute__ ((packed))

//struct Sperson female ;

void print_uart0(const char *s) {
    while(*s != '\0') { /* Loop until end of string */
    *UART0DR = (unsigned int)(*s); /* Transmit char */
    s++; /* Next char */
    }
}

void c_entry() {
    gdata.data1 = 0XAA ;
    gdata.data2= 0xFFFFFFFF ;
    gdata.data3= 0x55 ;
    gdata.data4= 0xA5A5 ;

print_uart0("############   VersatilePB physical Board  ############\n");

print_uart0("Structure lab!\n");

}
```


Arm Code analysis

packing is enabled
So uninitialized data =8 bytes

Code size is greater that packing version

Windows PowerShell

```
PS D:\courses\new_diploma\C\diploma4\ARM\bin> .\build.sh
PS D:\courses\new_diploma\C\diploma4\ARM\bin> .\arm-none-eabi-size.exe .\test.elf
   text    data     bss     dec     hex filename
    288       0       8     296     128 .\test.elf
PS D:\courses\new_diploma\C\diploma4\ARM\bin> .\build.sh
PS D:\courses\new_diploma\C\diploma4\ARM\bin> .\arm-none-eabi-size.exe .\test.elf
   text    data     bss     dec     hex filename
    264       0      12     276     114 .\test.elf
PS D:\courses\new_diploma\C\diploma4\ARM\bin>
```

Non-packing
So uninitialized data =12 bytes

Prove that in depth ☺

Using gdb or any binary utilities prove that

binary utilities objdump -lSd

packing

```
70 void c_entry() {
71    10060: e92d4800  push  {fp, lr}
72    10064: e28db004  add fp, sp, #4
73 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:22
74    gdata.data1 = 0XAA ;
75    10068: e59f3054  ldr r3, [pc, #84] ; 100c4 <c_entry+0x64>
76    1006c: e3e02055  mvn r2, #85 ; 0x55
77    10070: e5c32000  strb  r2, [r3]
78 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:23
79    gdata.data2= 0xFFFFFFFF ;
80    10074: e59f3048  ldr r3, [pc, #72] ; 100c4 <c_entry+0x64>
81    10078: e3e02000  mvn r2, #0
82    1007c: e5c32001  strb  r2, [r3, #1]
83    10080: e3e02000  mvn r2, #0
84    10084: e5c32002  strb  r2, [r3, #2]
85    10088: e3e02000  mvn r2, #0
86    1008c: e5c32003  strb  r2, [r3, #3]
87    10090: e3e02000  mvn r2, #0
88    10094: e5c32004  strb  r2, [r3, #4]
89 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:24
90    gdata.data3= 0x55 ;
91    10098: e59f3024  ldr r3, [pc, #36] ; 100c4 <c_entry+0x64>
92    1009c: e3a02055  mov r2, #85 ; 0x55
93    100a0: e5c32005  strb  r2, [r3, #5]
94 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:25
95    gdata.data4= 0xA5A5 ;
96    100a4: e59f3018  ldr r3, [pc, #24] ; 100c4 <c_entry+0x64>
97    100a8: e59f2018  ldr r2, [pc, #24] ; 100c8 <c_entry+0x68>
98    100ac: e1c320b6  strh  r2, [r3, #6]
99 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:27
100
101 print_uart0("############ VersatilePB physical Board  ###########\n");
102    100b0: e59f0014  ldr r0, [pc, #20] ; 100cc <c_entry+0x6c>
103    100b4: ebffffd5  bl  10010 <print_uart0>
104 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:29
105
```

Non-packing

```
70 void c_entry() {
71    10060: e92d4800  push  {fp, lr}
72    10064: e28db004  add fp, sp, #4
73 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:22
74    gdata.data1 = 0XAA ;
75    10068: e59f303c  ldr r3, [pc, #60] ; 100ac <c_entry+0x4c>
76    1006c: e3e02055  mvn r2, #85 ; 0x55
77    10070: e5c32000  strb  r2, [r3]
78 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:23
79    gdata.data2= 0xFFFFFFFF ;
80    10074: e59f3030  ldr r3, [pc, #48] ; 100ac <c_entry+0x4c>
81    10078: e3e02000  mvn r2, #0
82    1007c: e5832004  str r2, [r3, #4]
83 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:24
84    gdata.data3= 0x55 ;
85    10080: e59f3024  ldr r3, [pc, #36] ; 100ac <c_entry+0x4c>
86    10084: e3a02055  mov r2, #85 ; 0x55
87    10088: e5c32008  strb  r2, [r3, #8]
88 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:25
89    gdata.data4= 0xA5A5 ;
90    1008c: e59f3018  ldr r3, [pc, #24] ; 100ac <c_entry+0x4c>
91    10090: e59f2018  ldr r2, [pc, #24] ; 100b0 <c_entry+0x50>
92    10094: e1c320ba  strh  r2, [r3, #10]
93 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:27
94
95 print_uart0("############ VersatilePB physical Board  ###########\n");
96    10098: e59f0014  ldr r0, [pc, #20] ; 100b4 <c_entry+0x54>
97    1009c: ebffffdb  bl  10010 <print_uart0>
98 D:\courses\new_diploma\C\diploma4\ARM\bin/test.c:29
```

Using gdb non-packing structure on arm board

```
Breakpoint 1, c_entry () at test.c:22
22              gdata.data1 = 0XAA ;
(gdb) display/7i $pc
1: x/7i $pc
=> 0x10068 <c_entry+8>: ldr     r3, [pc, #60]   ; 0x100ac <c_entry+76>
   0x1006c <c_entry+12>:        mvn     r2, #85 ; 0x55
   0x10070 <c_entry+16>:        strb    r2, [r3]
   0x10074 <c_entry+20>:        ldr     r3, [pc, #48]   ; 0x100ac <c_entry+76>
   0x10078 <c_entry+24>:        mvn     r2, #0
   0x1007c <c_entry+28>:        str     r2, [r3, #4]
   0x10080 <c_entry+32>:        ldr     r3, [pc, #36]   ; 0x100ac <c_entry+76>
(gdb)
```

```
23              gdata.data2= 0xFFFFFFFF ;
1: x/7i $pc
=> 0x10074 <c_entry+20>:        ldr     r3, [pc, #48]   ; 0x100ac <c_entry+76>
   0x10078 <c_entry+24>:        mvn     r2, #0
   0x1007c <c_entry+28>:        str     r2, [r3, #4]
   0x10080 <c_entry+32>:        ldr     r3, [pc, #36]   ; 0x100ac <c_entry+76>
```

```
24              gdata.data3= 0x55 ;
1: x/7i $pc
=> 0x10080 <c_entry+32>:        ldr     r3, [pc, #36]   ; 0x100ac <c_entry+76>
   0x10084 <c_entry+36>:        mov     r2, #85 ; 0x55
   0x10088 <c_entry+40>:        strb    r2, [r3, #8]
   0x1008c <c_entry+44>:        ldr     r3, [pc, #24]   ; 0x100ac <c_entry+76>
```

```
25              gdata.data4= 0xA5A5 ;
1: x/7i $pc
=> 0x1008c <c_entry+44>:        ldr     r3, [pc, #24]   ; 0x100ac <c_entry+76>
   0x10090 <c_entry+48>:        ldr     r2, [pc, #24]   ; 0x100b0 <c_entry+80>
   0x10094 <c_entry+52>:        strh    r2, [r3, #10]
```


Using gdb packing structure on arm board

```
Breakpoint 1, c_entry () at test.c:22
22              gdata.data1 = 0XAA ;
1: x/7i $pc
=> 0x10068 <c_entry+8>: ldr     r3, [pc, #84]   ; 0x100c4 <c_entry+100>
   0x1006c <c_entry+12>:        mvn     r2, #85 ; 0x55
   0x10070 <c_entry+16>:        strb    r2, [r3]
```

```
(gdb) s
23              gdata.data2= 0xFFFFFFFF ;
1: x/7i $pc
=> 0x10074 <c_entry+20>:        ldr     r3, [pc, #72]   ; 0x100c4 <c_entry+100>
   0x10078 <c_entry+24>:        mvn     r2, #0
   0x1007c <c_entry+28>:        strb    r2, [r3, #1]
   0x10080 <c_entry+32>:        mvn     r2, #0
   0x10084 <c_entry+36>:        strb    r2, [r3, #2]
   0x10088 <c_entry+40>:        mvn     r2, #0
   0x1008c <c_entry+44>:        strb    r2, [r3, #3]
```

```
24              gdata.data3= 0x55 ;
1: x/7i $pc
=> 0x10098 <c_entry+56>:        ldr     r3, [pc, #36]   ; 0x100c4 <c_entry+100>
   0x1009c <c_entry+60>:        mov     r2, #85 ; 0x55
   0x100a0 <c_entry+64>:        strb    r2, [r3, #5]
   0x100a4 <c_entry+68>:        ldr     r3, [pc, #24]   ; 0x100c4 <c_entry+100>
   0x100a8 <c_entry+72>:        ldr     r2, [pc, #24]   ; 0x100c8 <c_entry+104>
   0x100ac <c_entry+76>:        strh    r2, [r3, #6]
```

```
25              gdata.data4= 0xA5A5 ;
1: x/7i $pc
=> 0x100a4 <c_entry+68>:        ldr     r3, [pc, #24]   ; 0x100c4 <c_entry+100>
   0x100a8 <c_entry+72>:        ldr     r2, [pc, #24]   ; 0x100c8 <c_entry+104>
   0x100ac <c_entry+76>:        strh    r2, [r3, #6]
   0x100b0 <c_entry+80>:        ldr     r0, [pc, #20]   ; 0x100cc <c_entry+108>
   0x100b4 <c_entry+84>:        bl      0x10010 <print_uart0>
   0x100b8 <c_entry+88>:        ldr     r0, [pc, #16]   ; 0x100d0 <c_entry+112>
   0x100bc <c_entry+92>:        bl      0x10010 <print_uart0>
(gdb)
```

enum

- An enumeration is a user-defined data type that consists of integral constants.
  To define an enumeration, keyword enum is used.

```
enum flag { const1, const2, ..., constN };
```

Here, name of the enumeration is flag.
And, const1, const2,...., constN are values of type flag.
By default, const1 is 0, const2 is 1 and so on. You can change default values of enum elements during declaration (if necessary).

```
// Changing default values of enum
enum suit {
      club = 0,
      diamonds = 10,
      hearts = 20,
      spades = 3,
};
```

Enumerated Type Declaration

- When you create an enumerated type, only blueprint for the variable is created. Here's how you can create variables of enum type.

```
enum boolean { false, true };
enum boolean check;
```

- Here, a variable check of type enum boolean is created.
- Here is another way to declare same check variable using different syntax.

```
enum boolean
{
    false, true
} check;
```

Example: Enumeration Type

```c
#include <stdio.h>

enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };

int main()
{
    enum week today;
    today = wednesday;
    printf("Day %d",today+1);
    return 0;
}
```

output

```
Day 4
```

Why enums are used in C programming?

- 1. Enum variable takes only one value out of many possible values. Example to demonstrate it,

```c
#include <stdio.h>

enum suit {
     club = 0,
     diamonds = 10,
     hearts = 20,
     spades = 3
} card;

int main()
{
          card = club;
        printf("Size of enum variable = %d bytes", sizeof(card));
        return 0;
}
```

It's because the size of an integer is 4 bytes.
This makes enum a good choice to work with flags.
You can accomplish the same task using structures.
However, working with enums gives you efficiency along with flexibility.

Output

```
Size of enum variable = 4 bytes
```

Why enums are used in C programming?

- **More readable** Here, we have added italics to our design. Note, only code for italics is written inside if statement.

```c
#include <stdio.h>

enum designFlags {
        BOLD = 1,
        ITALICS = 2,
        UNDERLINE = 4
};

int main() {
        int myDesign = BOLD | UNDERLINE;

           //      00000001
           //    | 00000100
           //      ----------
           //      00000101

        printf("%d", myDesign);

        return 0;
}
```

Output

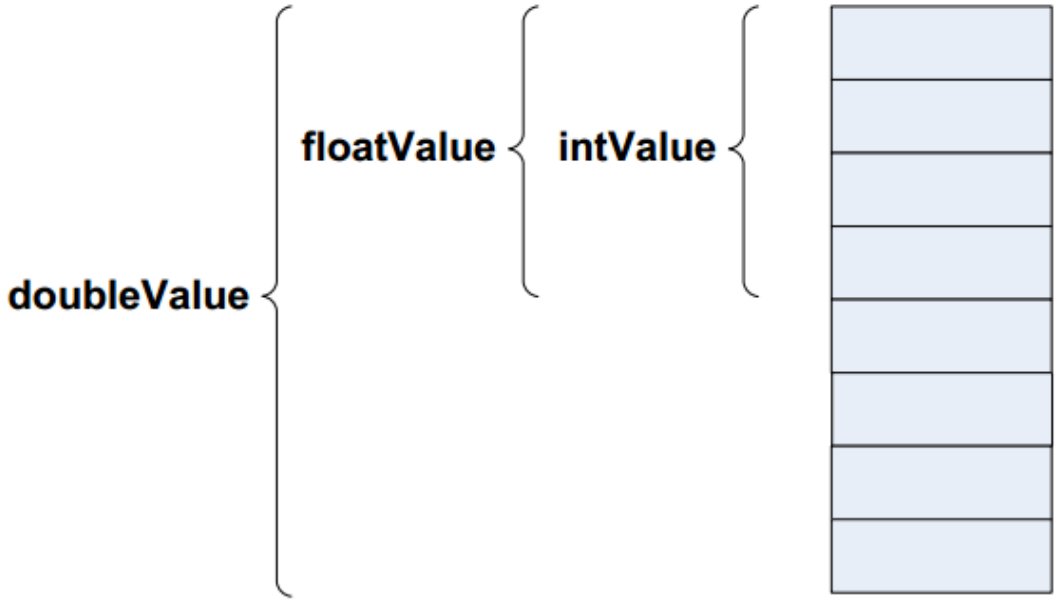
```
5
```

You can accomplish almost anything in C programming without using enumerations. However, they can be pretty handy in certain situations. That's what differentiates good programmers from great programmers.

Example: Personal Data using Enum


```c
#include "stdio.h"

enum Gender{MALE, FEMALE};

struct SPerson
{
	char m_Name[100];
	enum Gender m_Gender;
};

void main()
{
	int i;
	struct SPerson persons[] = {  {"Ahmed Ali", MALE},
	                              {"Mona Mohamed", FEMALE}};

	for(i=0;i<sizeof(persons)/sizeof(struct SPerson);i++)
	{
		printf("%s - %s\n", persons[i].m_Name,
		(persons[i].m_Gender==MALE)?"Male":"Female");
	}
}
```

Union is a special data type that supports different overlapped data types. For example following union can hold integer or float or double values, the developer is responsible on using one of those types.




```
union UNUMValue
{
        int intValue;
        float floatValue;
        double doubleValue;
};
```

Union size equals to the size of the biggest data type. For **UNUMValue** union the size equals to the double value size (8 bytes).

Using Union

- Following example illustrates how to combine the three data types (**int, float, double**) in one overlapped data type. Programmer is responsible on supplying and retrieving the data correctly. If integer data is supplied, integer data must be retrieved.

main.s | main.c

```c
//Prepared by Eng.Keroles
#include <stdio.h>

enum NUMType{INT, FLOAT, DOUBLE};

union UNUMValue
{
    int u_intValue;
    float u_floatValue;
    double u_doubleValue;
};
union UNUMValue Add(union UNUMValue value1, union UNUMValue
        value2, enum NUMType type)
{
    union UNUMValue result;
    switch(type)
    {
    case INT: result.u_intValue = value1.u_intValue +
            value1.u_intValue; break;
    case FLOAT: result.u_floatValue = value1.u_floatValue +
            value1.u_floatValue; break;
    case DOUBLE: result.u_doubleValue = value1.u_doubleValue
            + value1.u_doubleValue; break;
    }
    return result;
}
```


```c
int main(int argt ,char**argv) {


    union UNUMValue V1, V2, R;
    printf("the size of UNUMValue is %d bytes\n\n",
            sizeof(union UNUMValue));
    V1.u_intValue = 9898;
    V2.u_intValue = 8776;
    R = Add(V1, V2, INT);
    printf("int: %d + %d = %d\n",
            V1.u_intValue, V2.u_intValue, R.u_intValue);
    V1.u_floatValue = 86.82;
    V2.u_floatValue = 83.11;
    R = Add(V1, V2, FLOAT);
    printf("int: %f + %f = %f\n",
            V1.u_floatValue, V2.u_floatValue, R.u_floatValue);
    V1.u_doubleValue = 821.8;
    V2.u_doubleValue = 988.2;
    R = Add(V1, V2, DOUBLE);
    printf("int: %lf + %lf = %lf\n",V1.u_doubleValue, V2.u_doubleValue, R.u_doubleValue);

return 0   ;
}
```

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Problems | AVR Supported MCUs | Tasks | Console | Properties | AVR Device Explorer

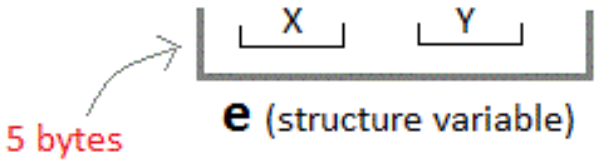
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (4/27/17, 11:16 PM)

```
the size of UNUMValue is 8 bytes

int: 9898 + 8776 = 19796
int: 86.820000 + 83.110001 = 173.639999
int: 821.800000 + 988.200000 = 1643.600000
```

Difference between union and structure

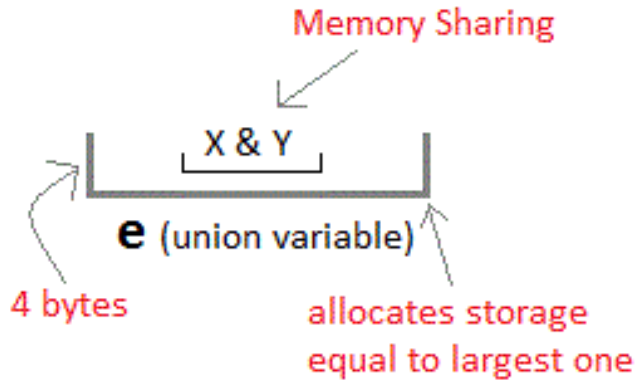
Structure

```
struct Emp
{
 char X ;     // size 1 byte
 float Y ;    // size 4 byte
} e ;
```



Unions

```
union Emp
{
 char X ;
 float Y ;
} e ;
```



Difference between union and structure

The primary difference can be demonstrated by this example:

```c
#include <stdio.h>
union unionJob
{
    //defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;

struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;

int main()
{
    printf("size of union = %d", sizeof(uJob));
    printf("\nsize of structure = %d", sizeof(sJob));
    return 0;
}
```

Output

```
size of union = 32
size of structure = 40
```


Difference between union and structure

The primary difference can be demonstrated by this example:

- **1. More memory is allocated to structures than union**
  - The amount of memory required to store a structure variable is the sum of memory size of all members.

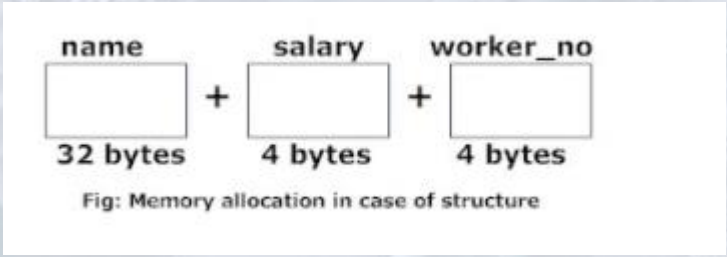


Fig: Memory allocation in case of structure

  - But, the memory required to store a union variable is the memory required for the largest element of an union.

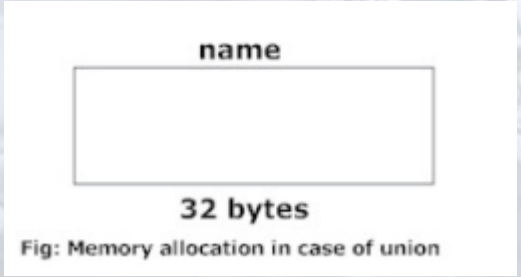


Fig: Memory allocation in case of union

- **2. Only one union member can be accessed at a time**

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

In Embedded C we will take the following topics (don't worry)

- Usage of bit-fields in embedded code
- Applicability of unions in Embedded system code

References

- The Case for Learning C as Your First Programming Language
- A Tutorial on Data Representation
- std::printf, std::fprintf, std::sprintf, std::snprintf…..
- C Programming **for Engineers, Dr. Mohamed Sobh**
- **C programming expert.**
- fresh2refresh.com/c-programming
- **C programming Interview questions and answers**
- **C – Preprocessor directives**

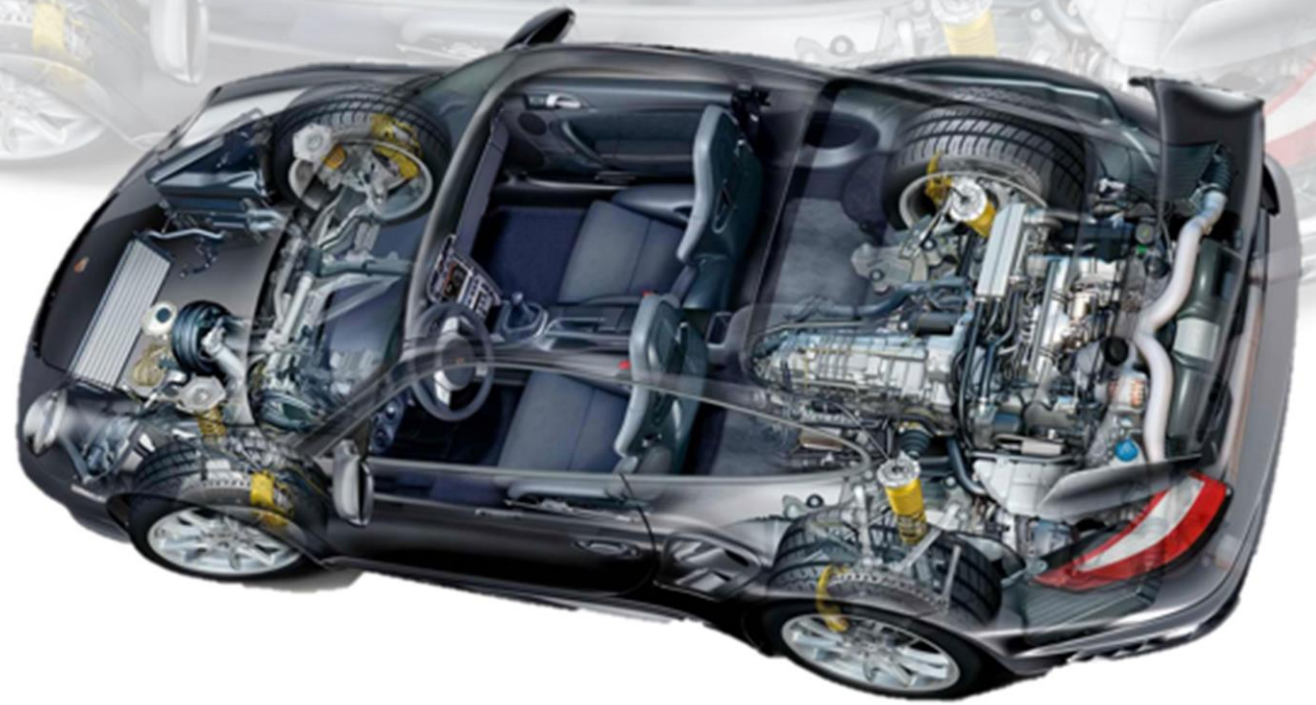
References

- The Case for Learning C as Your First Programming Language
- A Tutorial on Data Representation
- std::printf, std::fprintf, std::sprintf, std::snprintf…..
- C Programming **for Engineers, Dr. Mohamed Sobh**
- **C programming expert.**
- fresh2refresh.com/c-programming
- **C programming Interview questions and answers**
- **C – Preprocessor directives**

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Thanks and Good Luck



ENG. Keroles Shenouda
https://www.facebook.com/groups/embedded.system.KS/

https://www.facebook.com/groups/embedded.system.KS/