M T W T F S S

DATE

[Structure Variables.]

Car → engine type
→ fuel type
→ fuel tank capacity
→ seating capacity

different type Variables.

Storing all these variables (information) in different variables for each type of car is time consuming and memory consuming.

Can we use Array?

Arrays can store different variable but from the same type.

→ our requirment is to store data of different types.

Structure is the solution

↳ is a user defined type that can be used to group elements of different types into a single type.

M T W T F S S

DATE

* Declaring Structure variable.

```
Struct {
    char *engine;
    char *FuelType;
    int *fuel_tank_cap;
    int Seating_Cap;
} Car1, Car2;
```

Structure is in Global scope it is visible for all functions including main.

Variable declared in global scope

```
int main() {
    Car1.engine = "DDiS 190 engine";
    Car2.engine = "1.2 L Kappa Dual VTVT";
}
```

[dot operator → excess member of structure.]

---

```
Struct employee {
    char *name;
    int age;
    int salary;
} Variable declared in local scope

int main(){
    Struct employee empl;

int manager(){
    Struct employee manager;
```

[Structure tag]

Specify the type of the structure and name of variable

M T W T F S S

MOOD

DATE

# [TYPEDEF]

→ typedef existing_data_type new_data_type

gives freedom to the user by allowing them to Create their own types.

```
typedef struct Car {
    char *engine [50];
    Char fuel_type[10];
    int fuel_tank_Cap;
} Car ;
int main (){
    Car C1;
}
```

Car ↑ equivelant to Struct Car.

```
Struct Car {
    Char engine [50];
    char fuel_type[10];
    int fuel_tank_cap;
    Float city_mileage;
};
intmain () {
 Struct Car C1 = { "DDis190 engine", "Diesel",
 37, 19.74 };
}
```

Initializing

This way requires arrangment of

Accessing ⇒ dot operator (.)

---

Designated intialization

allow as to intialize members in any order

```
Struct abc {
  int x, y, z;
};
int main() {
  Structabc a = { .y = 20,
  .x = 10 , .z = 30 };
}
```

dot operator is essential

# Array of structure.

```
Struct Car {
    int fuel_tank_cap;
    int seating_cap;
    float city_mileage;
}
int main() {
  Struct Car [2];
  int i;
for (i=0; i<2; i++) {
    Printf ("Enter the car %d fuel tank capacity", i+1);
    Scanf ("%d", &C[i].fuel_tank.cap);
}
```

M T W T F S S

DATE

# Pointer to Structure variable

```
Struct abc {
    int x;
    int y;
};
int main () {
    Struct abc a= {0,1};
    Struct abc *ptr = &a;
    Printf ("%d %d", ptr->x , ptr->y);
    return 0;
}
```

(*ptr).x

(*&a).x

a.x

0

1 1

M T W T F S S

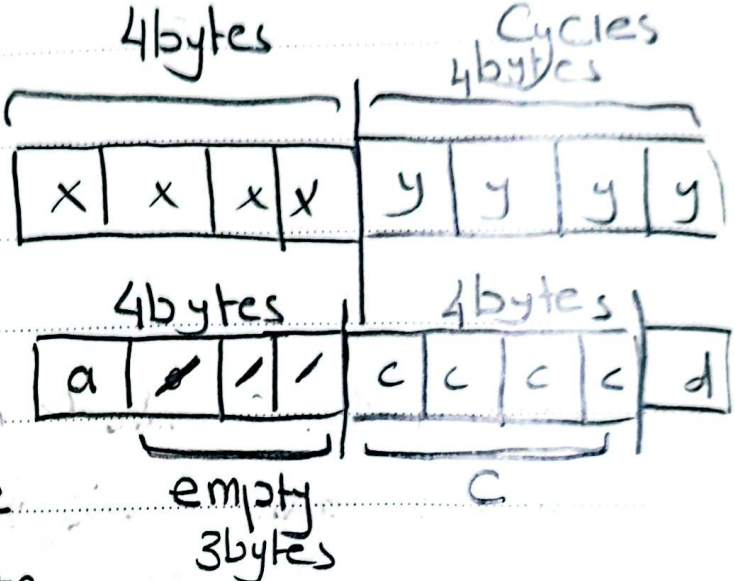
* Pragma pack(1)

# [Structure padding] & [structure packing]

Wasting memory

Wasting ~~memory~~

```
Struct abc {
    int x;  // 4bytes
    int y;  // 4bytes
    Char a;  // 1byte
    int b;  // 4byte
    Char d;  // 1byte
}
```




[Total 17bytes]

Structure pading → used by compilier to optimize memory access and aligment of data members within structure. it involves inserting extra bytes (padding) between data to ensure that they are alliged on memory boundries.

if we have 32 bit processor then it means it can access 4bytes at a time

64 bit ⟶ access 8bytes at a time.

M T W T F S S

DATE

# [passing Structure to a function]

## Call by value :

```
Struct Car {
    Char name[30];
    int price;
};
  Void Print_Car_info (Struct Car C) {
   Printf ("name : %s", C.name);
    Printf ("\n price : %d\n", C.price);
}
 int main () {
  Struct car C = {"tata", 1021};
  Print_Car_info(C);
  return 0;
}
```

↑ Passed Struct

[Call by Reference method:]

```
struct student {
        char name [50];
        int roll;
        float marks;
};
void display (struct student * student_obj)
{  Printf ("name : %s \n", student_obj -> name);
   Printf ("Roll: %d \n", student_obj -> roll );
 Printf ("Marks : %f \n", student_obj -> marks);
}
int main () {
    Struct Student st1 = {"Aman", 19, 8.5};
    display (& st1);
    return 0;
}
```

passing address of struct.

[Return structure from function.]

```
struct student {
    char name[20];
    int age;
    float marks;
};
```

return type.

```
struct student get_student_data()
{   struct student S;
    printf("Enter name : ");
    scanf("%s", S.name);
    printf("Enter age:");
    scanf("%d", &S.age);

    return S;
}

int main() {
  struct student S1 = get_student_data();
   printf("Name: %s\n", S1.name);
   printf("Age : %s\n", S1.age);
    return 0;
}
```