# *Struct in C*

# Table of content

# Introduction

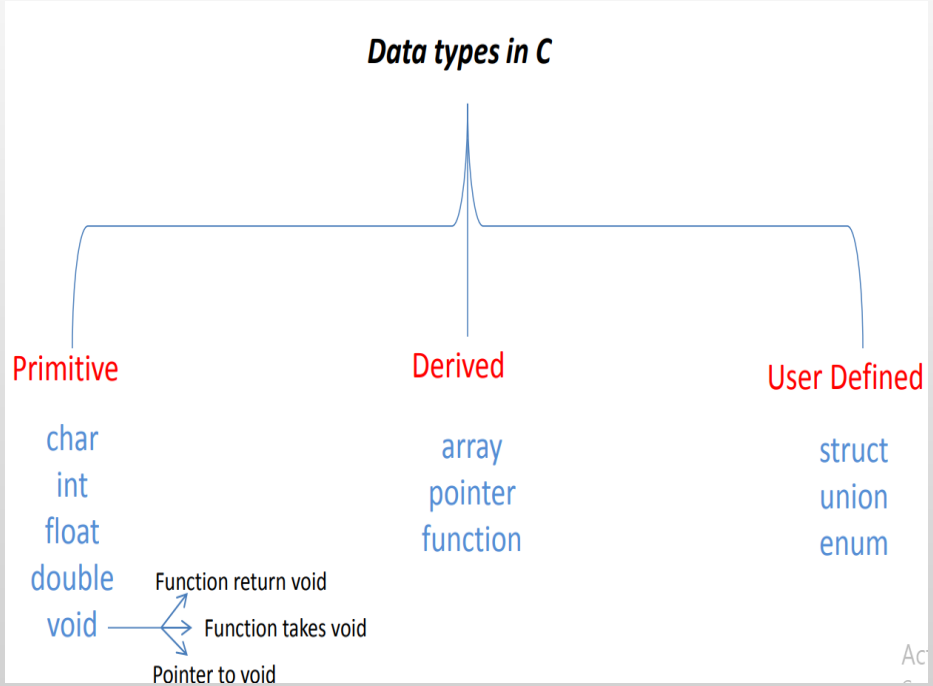
**What is Struct?**

**Struct:** is a user-defined data type that allows you to make a collection of variables of different data types under a single name .

**Why we use Struct ?**

Organizing related data: Structures allow you to group together related variables of different types under a single name. This helps in organizing data in a logical and meaningful way, making the code more readable and maintainable.

Creating complex data structures: Structures enable the creation of more complex data structures by combining multiple variables and structures. For example, you can create linked lists, trees, queues.

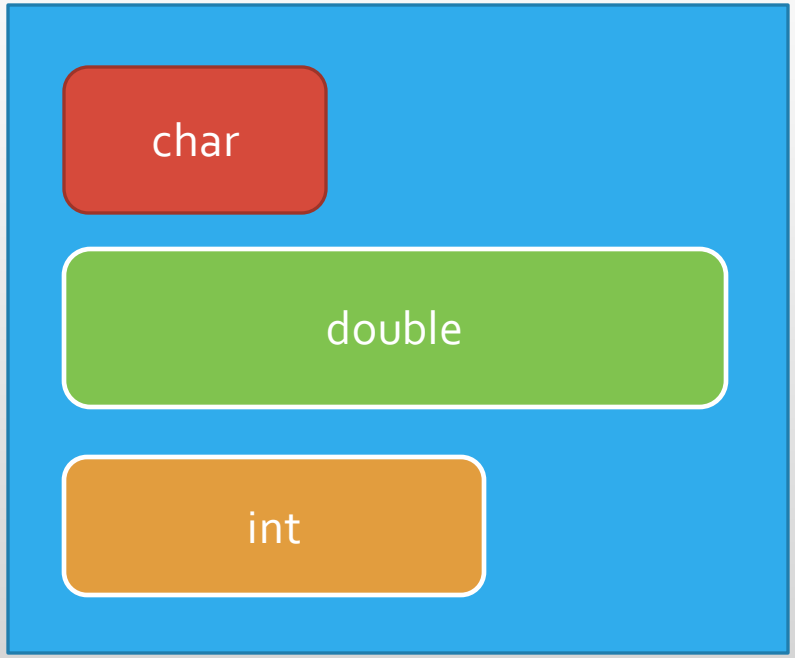


# Syntax

Syntax :

```
struct struct_name
{
        member_1_Type member_1_Name ;
        member_2_Type member_2_Name ;
        ----
        ----
};
```

Note : No memory consume




```
struct struct_name  object_name ;
struct struct_name object_name = { member_1_Value , Member_2_Value, …. };
```

We use the dot operator ( . ) to access any member in the structure.

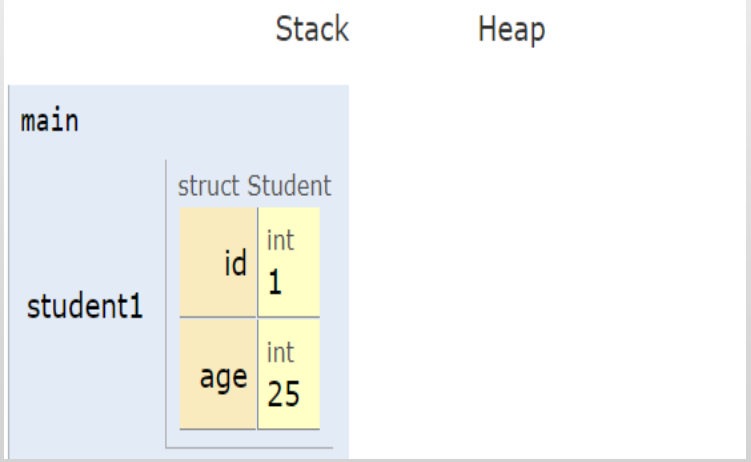
object_name.member_name

# Simple Code

```c
#include <stdio.h>

// Define the struct
struct Student
{
    int id;
    int age;
};

void main(void)
{
    // Create a struct Object
    struct Student student1;

    // Assign values to struct members
    student1.id = 1;
    student1.age = 25;

    // Access and print struct members
    printf("Id: %d\n", student1.id);
    printf("Age: %d\n", student1.age);

}
```

Print output (drag lower right corner to resize)

```
Id: 1
Age: 25
```



# Object

```c
#include <stdio.h>

// Define the struct
struct Student
{
    int id;
    int age;
};

struct Student Gstudent;

void main(void)
{
    // Create a struct Object
    struct Student Lstudent;
}
```

Note : Never make init value in struct

Global Object

Local Object

# Struct Arithmetic

No Arithmetic operation allowed on the struct

Only the assignment operator can be used with the structure, to copy a content of structure to another structure.

```c
#include <stdio.h>
// Define the struct
struct Student
{
    int id;
    int age;
};

void main(void)
{
    struct Student student1 = { 1 , 25 };
    struct Student student2 ;

    student2 = student1 ;

    student2 = student2 + 2 ;

    struct Student student = student 1 + student2;
}
```


Allowed


Not Allowed


Not Allowed

# Struct with typedef

typedef can be used with structure. It would add the value of defining an object from structure by the structure name only without mentioning the word struct.

Syntax : typedef Old_Type New_Type ;

without typedef

```c
#include <stdio.h>
// Define the struct with typedef
struct student
{
    char id;
    int age;
};

void main(void)
{
    // Create a struct Object
    struct student student1;
}
```

with typedef

```c
#include <stdio.h>
// Define the struct with typedef
typedef struct student
{
    char id;
    int age;
} Student;

void main(void)
{
    // Create a struct Object
    Student student1;
}
```

# Struct with Function

1) Function that takes a struct as a parameter

```c
#include <stdio.h>

// Define the struct with typedef
typedef struct student
{
    char id;
    int age;
}Student;

// Function that takes a struct as a parameter
void printStudent( Student S)
{
    printf("Id: %d\n", S.id);
    printf("Age: %d\n", S.age);
}

void main(void)
{
    // Create a struct Object
    Student student1 = { 1 , 25 };
    // Call the function and pass the struct as an argument
    printStudent(student1);
}
```

# Struct with Function

2) Function that returns a struct

```c
#include <stdio.h>

// Define the struct with typedef
typedef struct student
{
    char id;
    int age;
}Student;

// Function that returns a struct
Student CreatStudent(char id , int age  )
{
  Student S;
  S.id = id ;
  S.age = age ;
  return S;
}

void main(void)
{
    // Call the function and store the returned struct
    Student student1 = CreatStudent( 1 , 25 );
    // Access and print the values of the returned struc
    printf("Id: %d\n", student1.id);
    printf("Age: %d\n", student1.age);
}
```

Note : you can return more than one value from function by struct

# Struct with Function

3) Function that takes a struct as a parameter and returns a struct

Note : not allowed to declare or define Function inside Struct

```
typedef struct student
{
    char id;
    int age;
    void print(void);
    void print(void)
    {
      printf("Error\n");
    }
}Student;
```

Synatax Error

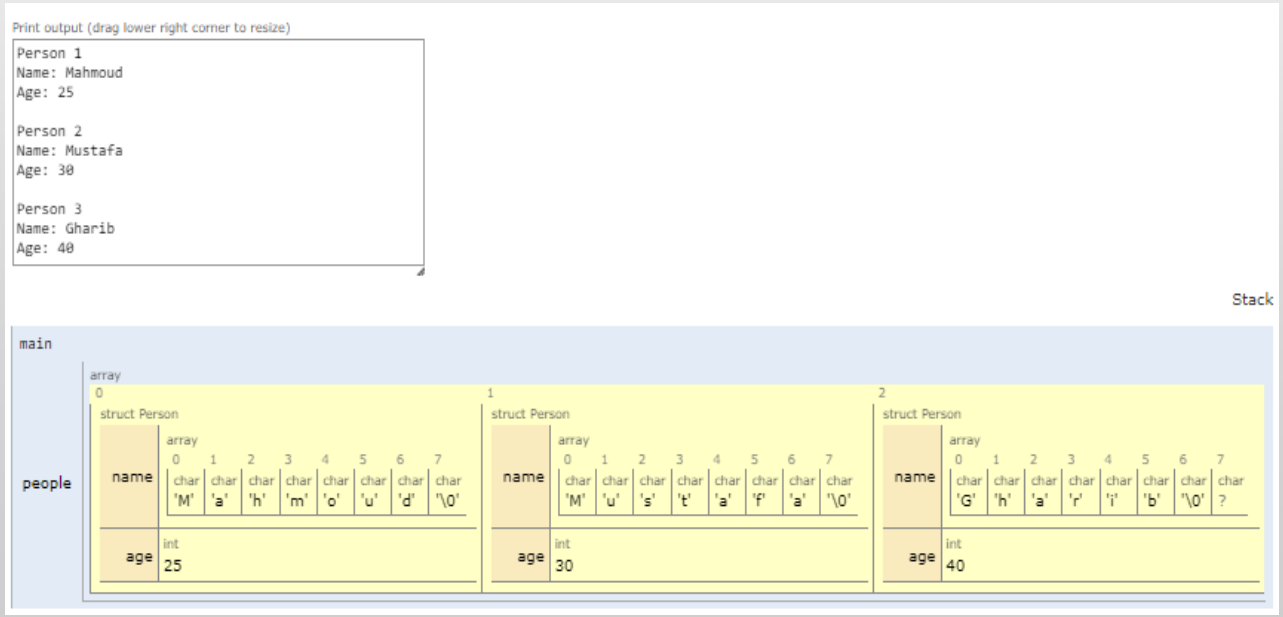
# Struct with Array

1) Create member inside struct as an Array
2) Create an array of structs

```c
#include <stdio.h>
// Define the struct
struct Person
{
    char name[8];
    int age;
};

void main(void)
{
    // Create an array of structs
    struct Person people[3];

  // Assign values to struct members for each element in the array
    strcpy(people[0].name, "Mahmoud");
    people[0].age = 25;

    strcpy(people[1].name, "Mustafa");
    people[1].age = 30;

    strcpy(people[2].name, "Gharib");
    people[2].age = 40;

// Access and print the values of the struct members
    for (int i = 0; i < 3; i++)
    {
        printf("Person %d\n", i + 1);
        printf("Name: %s\n", people[i].name);
        printf("Age: %d\n", people[i].age);
        printf("\n");
    }
}
```

Print output (drag lower right corner to resize)

```
Person 1
Name: Mahmoud
Age: 25

Person 2
Name: Mustafa
Age: 30

Person 3
Name: Gharib
Age: 40
```



# Struct with Pointer

1) Create member inside struct as a Pointer

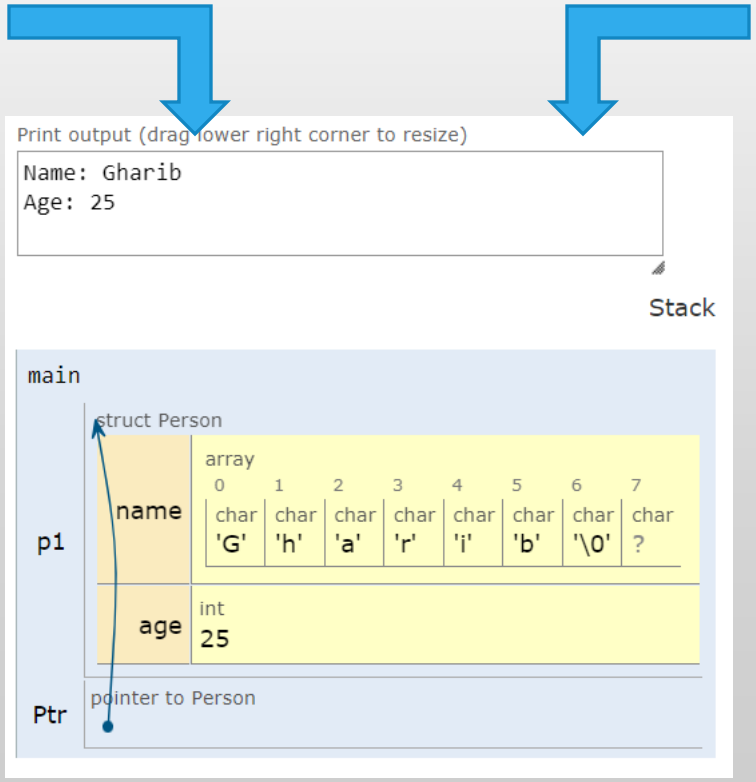
```c
#include <stdio.h>
// Define the struct
struct Person
{
    char *name;
    int age;
};

void main(void)
{

  struct Person p1;

  strcpy(p1.name, "Gharib");
  p1.age = 25;

  printf("Name: %s\n", p1.name);
  printf("Age: %d\n", p1.age);
  printf("\n");

}
```

# Struct with Pointer

2) Create a Pinter that point to struct

Note : The arrow operator -> is used with pointer to struct as replacement to the dereference operator * and the dot operator .

```c
#include <stdio.h>
// Define the struct
struct Person
{
    char name[8];
    int age;
};

void main(void)
{

  struct Person p1;
  struct Person *Ptr = &p1;

  strcpy((*Ptr).name, "Gharib");
  (*Ptr).age = 25;

  printf("Name: %s\n", p1.name);
  printf("Age: %d\n", p1.age);
  printf("\n");

}
```




```c
#include <stdio.h>
// Define the struct
struct Person
{
    char name[8];
    int age;
};

void main(void)
{

  struct Person p1;
  struct Person *Ptr = &p1;

  strcpy(Ptr -> name, "Gharib");
  Ptr -> age = 25;

  printf("Name: %s\n", p1.name);
  printf("Age: %d\n", p1.age);
  printf("\n");

}
```

# Struct with Bit field

Bit field : is a member of a struct that has size of certain number of bits
used to 1) optimize memory usage
2) to store and modify data at the bit level

Syntax : type name : number_of_bits ;

```c
#include <stdio.h>

// Define a structure with bitfields
struct Flags
{
    unsigned char f1 : 1;
    unsigned char f2 : 3;
    unsigned char f3 : 5;
};

void main(void)
{
    // Declare a structure variable
    struct Flags f;

    // Assign values to the bitfields
    f.f1 = 1;
    f.f2 = 7;
    f.f3 = 14;

    // Print the values of the bitfields
    printf("flag1: %d\n", f.f1);
    printf("flag2: %d\n", f.f2);
    printf("flag3: %d\n", f.f3);
    printf("size of struct : %d\n", sizeof(f));
}
```

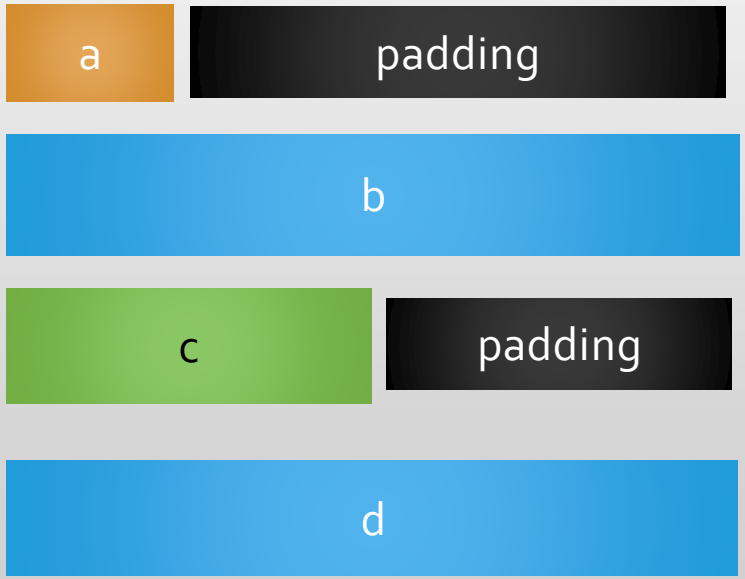
size of struct : 2 Byte

# Size of Struct

( Size of Struct  > =  sum of its member sizes )

```
struct S
{
   char          a ;
   int           b ;
   short int     c ;
   int           d ;
};
```

Assuming 32 bit memory width

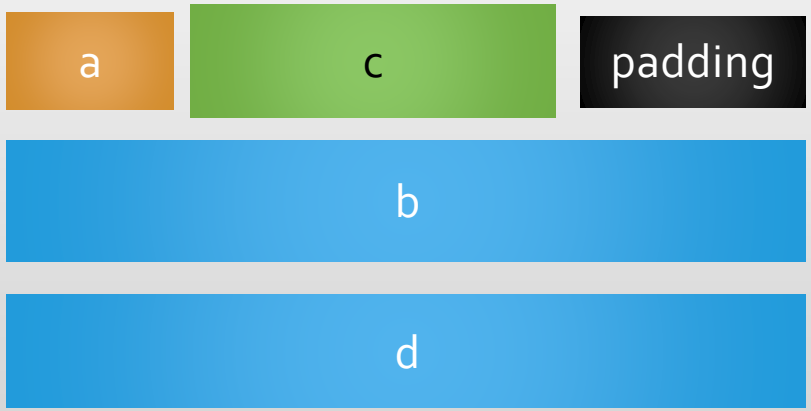


# Size of Struct

The general rule states that always arrange the members in ascending order or descending order to get the lowest size.

```
struct S
{
  char          a ;
  short int     c ;
  int           b ;
  int           d ;
};
```

Assuming 32 bit memory width




ANY
QUESTIONS?