# STRUCTURES

# CONSIDER THE FOLLOWING SCENARIO

- Let us say we need to maintain the data about students in a college. What data would we need to store for each student?
  - Name
  - Roll number
  - Marks of each semester
  - Date of admission etc.
- Would all the above data have to be stored consecutively in the memory?
  - Yes
- Can we use arrays for the above?
  - No.
- Why can't arrays be used?
  - The data types are all different

# HOW TO SOLVE SUCH PROBLEM?

- We need a technique to:
  - Store elements of different data types
  - In consecutive memory locations
- For the above purpose, we can use *structures*.
- Structures are user-created data types,
- as opposed to built-in data types such as int, char, float etc…
- Structures allow us to store elements of different data types in consecutive memory locations.

# STEP 1: DEFINING A STRUCTURE

Syntax:

```
struct structure_name
{
datatype member1;
datatype member2;
.........
datatype membern;
};
```

Name of the USER-DEFINED data type

Elements you want to include in your structure.

# STEP 1: DEFINING A STRUCTURE

Syntax:

```
struct structure_name
{
datatype member1;
datatype member2;
………
datatype membern;
};
```

Example:

```
struct student
{
char name[10];
int roll_no;
float marks;
};
```

# STEP 1: DEFINING A STRUCTURE

- Structure definition is to specify the name of the structure and what members it contains.
- The structure definition can be local (inside a function) or global (outside all the functions).
- The keyword “struct” is required during structure definition.
- Each element of a structure is called as member or data member.
- However, defining a structure does not actually CREATE any actual space in the memory.
- We need to go to Step no.2….

# STEP 2: CREATING A VARIABLE

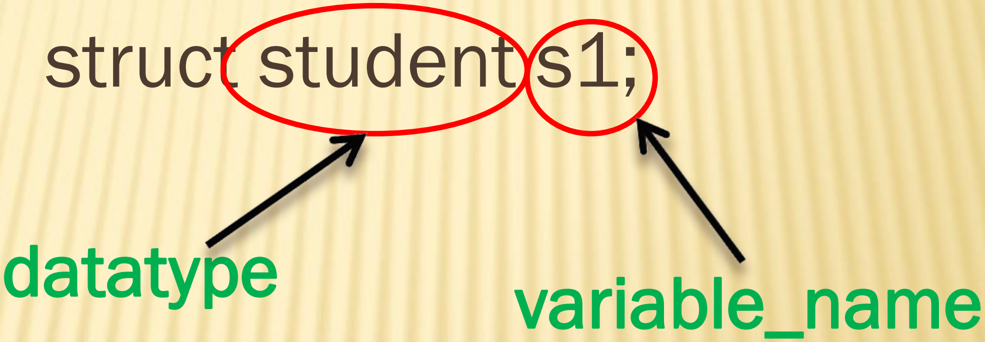
- Structure definition only gives the specifications of the user-defined data-type.
- However, no actual/physical memory will be allocated through definition.
- Only when we declare variables, memory is created for *int, float, char* etc.
- Same principle applies to structures also.
- The syntax for structure variable declaration is the same as any other variable, except keyword ”struct”.

# STEP 2: CREATING A VARIABLE

- Syntax:

  struct datatype variable_name;

- Considering the previous definition of "student" structure:



struct student s1;

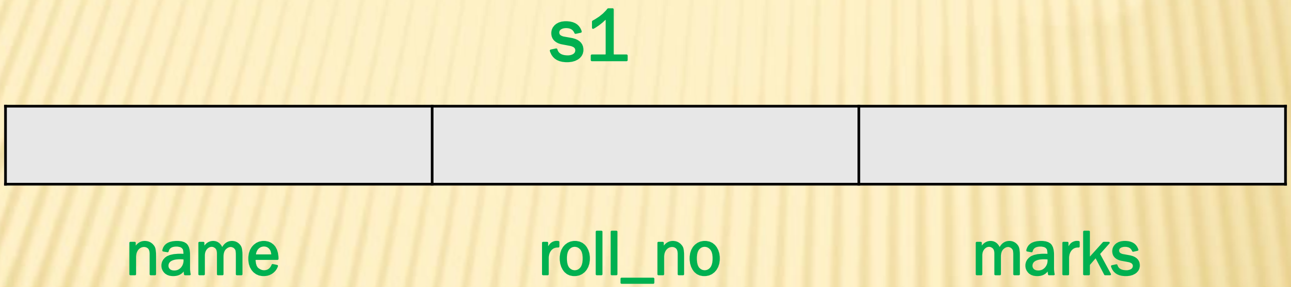
datatype

variable_name

- We can create multiple variables of the same structure in a single line using a ",".
- Each structure variable will have the same members as defined in the structure definition.

# STEP 2: CREATING A VARIABLE

- Once a structure variable is created, how will its memory representation be?
- Considering the student variable s1,



- Can you guess the size of bytes allocate to s1?
  - 10 + 2 + 4 = 16 bytes
- Size of a structure variable = total size of all its members
- NOTE: Unlike array name, structure variable name is NOT a pointer. It is simply a variable name.

# STEP 3: INITIALIZING STRUCTURE VARIABLE

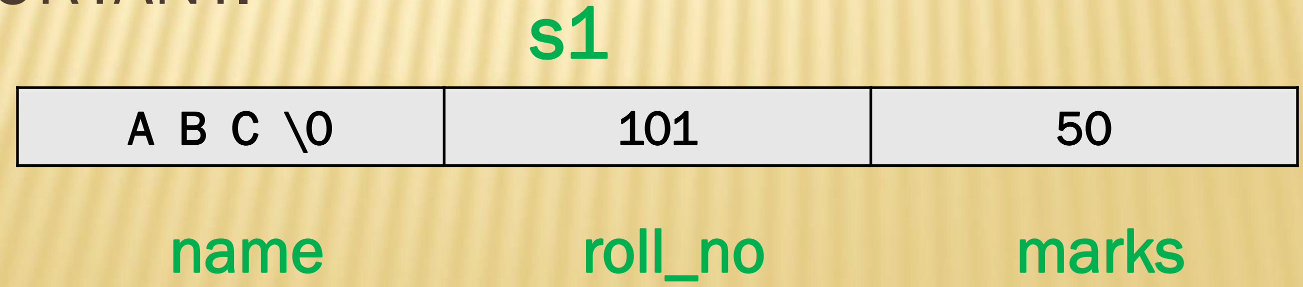
- The first method to initialize a structure variable is in the line of declaration:
- Syntax:

struct struct_name var_name = {member1_val, member2_val.... membern_val};

- Example:

struct student s1 = {"ABC", 101, 50};

- Note: In this initialization method, order of values is IMPORTANT.



s1

| | | |
|---|---|---|
| A B C \0 | 101 | 50 |
| name | roll_no | marks |

# STEP 4: ACCESSING A STRUCTURE

- Suppose we want to access any member of a structure, what information do you need?
  - Which structure variable ? (as there can be many)
  - Which member ?
- Syntax:

variable_name . member_name

dot operator

- Example, to access roll_no of student s1:

s1.roll_no

# LET US BRING IT ALL TOGETHER….

```
#include<stdio.h>
struct student
{
char name[10];
int roll_no;
float marks;
};
int main()
{
struct student s1={“ABC”,101, 50},s2={“XYZ”, 102,60};
printf(“%d”, s1.roll_no); → 101
printf(“%f, s2.marks ); → 60
printf(“%s, s2.name); → XYZ
}
```

# OPERATIONS ON A STRUCTURE VARIABLE

- Suppose we have the following declaration:

  struct student s1 = {“ABC”, 101, 50}, s2;

- No binary / unary operations such as ,

  s1 == s1, s1 + s2 , s1 > s2 etc… are allowed

- Except for the SIMPLE ASSIGNMENT operator “=“

- If we say, s2 = s1, all the members of s1 will be copied into s2, member by member.

# USER-INPUT OF A STRUCTURE VARIABLE

- Example:

```
struct student s1;
printf("Enter the details of a student");
gets(s1.name);
scanf("%d%f", s1.roll_no, s1.marks);
```

- Note: Order of input is not important in this method.

# IDENTIFY CORRECT/INCORRECT

```
1) struct student s1,s2,s3; → Valid
2) printf("%d", student. name); → INValid
3) int main()
{
struct student = { "ABC", 101, 50 }; → INValid
}
4) struct student
{                                   → Valid
char name[10], parent_name[10], address[30];
};
```

# PROGRAM IT

- Create an Employee structure with name, ID and salary. Input details of two employees and print all the details of the employee with the higher salary.
- Create a structure called Complex to store complex numbers (a + bi). Input two complex numbers, add them and display the resultant complex number.