STRUCTURES

POINTER TO A STRUCTURE

- If we want to store address of a structure variable, what will be the datatype of the pointer?
  - Same as the structure
  - struct student *p;
- A structure member can be accessed either by a variable name or by using a pointer variable.
- If we access a member through a variable the syntax as we have discussed will be:

  variable_name.member_name
- If we access a structure member through a pointer variable, the syntax is:

  pointer_name ->member_name
- We use the dot operator(.) for accessing through variables.
- We use the arrow (->) to access using a pointer variable.

QUICK EXERCISE:

- Consider a student structure has been defined with name, roll and marks:

```
int main()
{
struct Student s={"ABC", 102, 50}, *p;
p=&s;
printf("%s\n", s.name); → ABC
printf("%u\n",p); → Address of variable “s “
printf("%d\n",p->roll); → 102
printf("%d\n",(*p).roll); → 102
printf("%f\n",(&s)->marks); → 50.00000
p->marks=65;
printf("%f",s.marks); → 65.00000
return 0;
}
```

CONSIDER THE FOLLOWING SCENARIO…

- You have to store the data about students in a college. The data contains, name, roll_no, date of joining, address, parents’ name etc….
- The above information has to be stored for 1000 students in the college.
- What C constructs will you use?
- Will you need structures?
  - Yes
- What else do you need apart from structures?
  - Arrays

ARRAY OF STRUCTURES

- Structure is a user-defined data type to store data of dissimilar types.
- If we want to store only roll number of 1000 students we will need:
  - An int array
- If we want to store only the marks of 1000 students we will need:
  - A float array
- But we want to store the data of 1000 students (with varying types of data). Therefore we need:
  - An array of students
  - We may use an array now as all elements are of the SAME type i.e “student”

ARRAY OF STRUCTURES

- Declaration:
- Syntax:

```
struct structure_name arrayname[SIZE];
```

- Example:

```
struct student s[3];
```

- Initialization (in the line of declaration):
- Syntax:

```
struct structure_name arrayname[SIZE]={
                        {Members of first element},
                        {Members of second element}…
                        {Members of nth element}
                        };
```

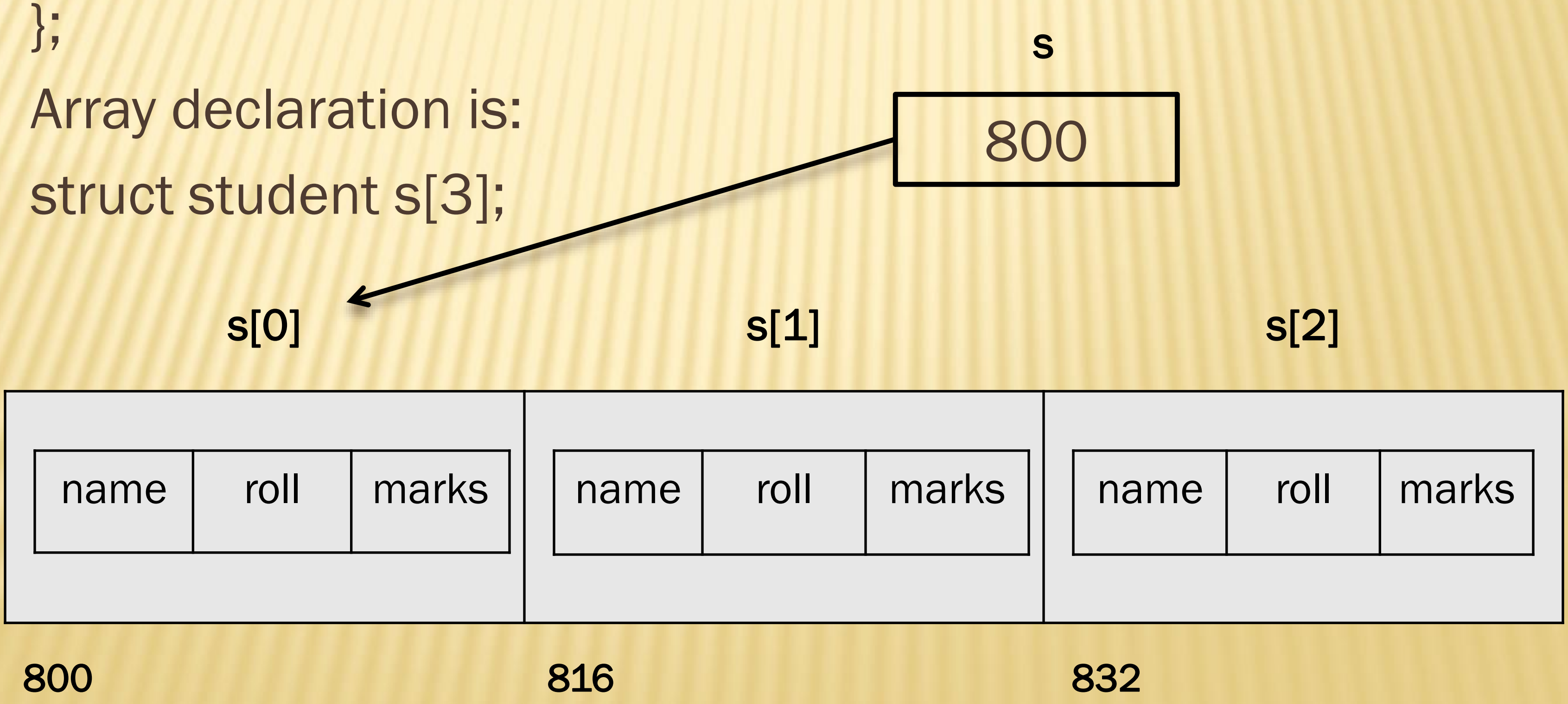
MEMORY REPRESENTATION

- Consider the following definition:

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

- Array declaration is:

```
struct student s[3];
```



s

800

s[0]

s[1]

s[2]

name	roll	marks	name	roll	marks	name	roll	marks

800

816

832

ARRAY OF STRUCTURES

- Accessing array of structure:
  - Syntax:

```
arrayname[index].member_name
```

  - Example: Print the name of the 3rd student.

```
printf("%s",s[2].name);
```

- Accessing all elements of an array of structure:

```
for(i=0;i<n;i++)
{
//Access as arrayname[i].member_name
}
```

PROGRAM IT

Create a structure IPLTeamPlayer with Player_name, runs_scored, number of matches and country. Create 10 players, input their data and calculate and find the average runs scored by all the players in the team.

UNIONS

- A union is also a user-defined data type similar to structures.
- Union also stores the different types of elements i.e heterogeneous elements.
- The main point to remember about unions is that:
  - The size of a union variable = size of the largest data member.

DIFFERENCE BETWEEN STRUCTURES AND UNIONS

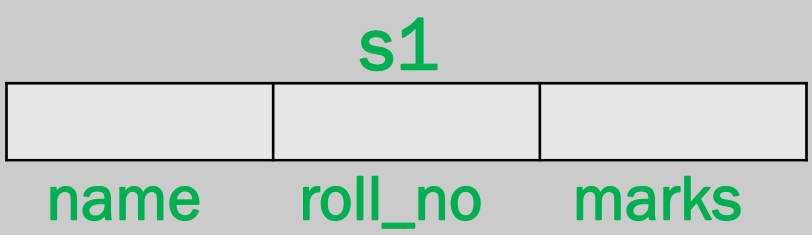
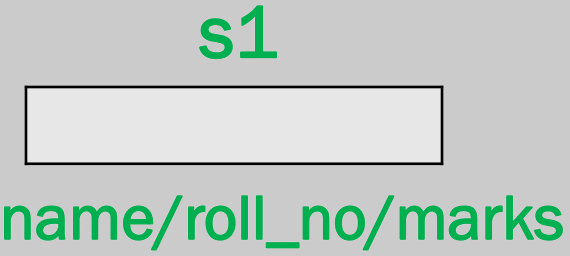
1. Definition:

Structures	Unions

struct Student
{
char name[10];
int roll;
float marks;
};
	union Student
	{
	char name[10];
	int roll;
	float marks;
	};

DIFFERENCE BETWEEN STRUCTURES AND UNIONS

2. Memory allocation:

Structure	Union
Size of a structure variable = total size of all its members	Size of a union variable = size of its largest data member.
Each structure member gets its own individual memory space	All members of a union share a common space in the memory
 s1 name roll_no marks	 s1 name/roll_no/marks
The total size of "s1" considering the previous definition of structure = 10+2+4=16 bytes	The total size of "s1" considering the previous definition of union = 10 bytes (Size given to the member "name")

DIFFERENCE BETWEEN STRUCTURES AND UNIONS

3. Accessing members and applications:

Structure	Union
All members can be initialized simultaneously in the line of declaration.	Members can be initialized only one at a time.
Any member can be accessed at any point in the program.	Only the last initialized member can be accessed. Remaining elements will give a random garbage value
Structures can be used for user-level software applications.	Unions are used in hardware level programming application such as Embedded Systems where memory is scarce.

PROGRAM EXAMPLE FOR UNIONS

```
#include <stdio.h>
#include<string.h>
union Student
{
char name[10];
int roll;
float marks;
};
int main()
{
union Student s;
strcpy(s.name,"ABC");
printf("%s\n", s.name);
printf("%d\n", s.roll);
printf("%f\n", s.marks);
s.roll=103;
printf("%s\n", s.name);
printf("%d\n", s.roll);
printf("%f\n", s.marks);
}
```

Output:

```
ABC
4407873
0.000000
g
103
0.000000
```

NESTED STRUCTURES

- Sometimes, we may want to use a variable of one user created data type as a member of another user-defined data type.
- Example: Date.
- If we create a user-defined data type called ”Date” in the format of dd,mm and yy…
- We may use it in any other structure to store date-related information, such as:
  - Employee: Date of joining, Date of leaving the company
  - Aadhar Card: Date of birth
  - Any food products: Date of manufacturing, Date of expiry

DEFINING NESTED STRUCTURES

- We can define a structure within another structure or we can define it outside and only create a variable inside another structure.

```
struct Date
{
int dd,mm,yy;
} ;
struct Student
{
char name[10];
int roll;
float marks;
struct Date dob,doj;
};
```

```
struct Student
{
struct Date
{
int dd,mm,yy;
} dob,doj;
char name[10];
int roll;
float marks;
};
```

NESTED STRUCTURES

- Initializing members of a nested structure:

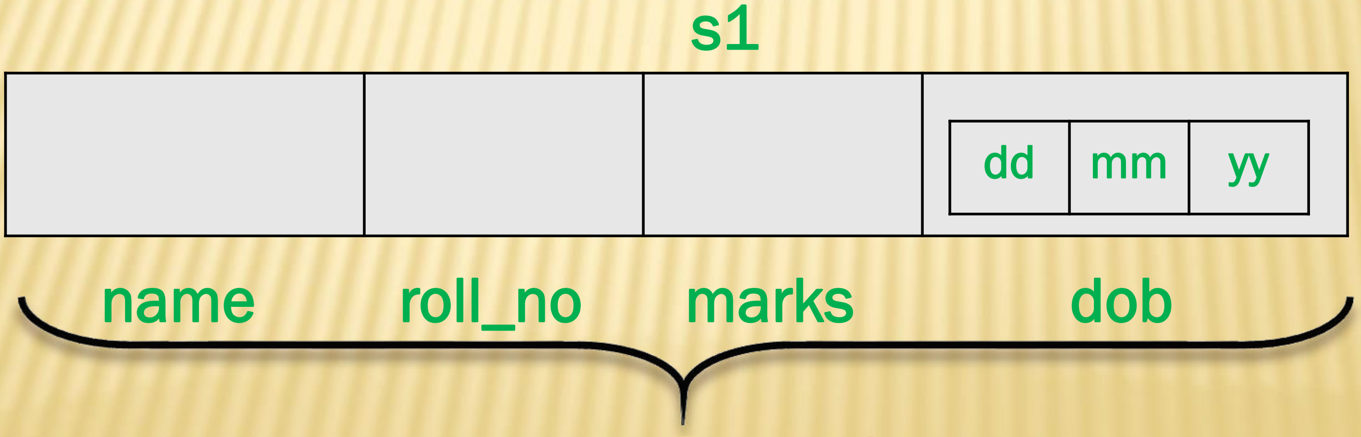
  struct Student s1 = {“ABC”,102, 50, {12,7,2010}};

- Accessing members of a nested structure:

outer_structure_variable.inner_structure_variable.member_name

- Example: Print the month of birth of student s1

  printf(“%d”, s1.dob.mm); → 7



s1

dd mm yy

name roll_no marks dob

Total size of s1 = 10 + 2 + 4 + 6 = 22 bytes

PROGRAM IT

- Create a structure Car with Car_Name, Brand, Price and Manufacturing Date using nested structures. Create 10 Cars, Input the data and sort them as per the year of manufacturing.
- Homework:

  Sort the cars as per the Date of manufacturing (Consider month and day also)