Structures in C

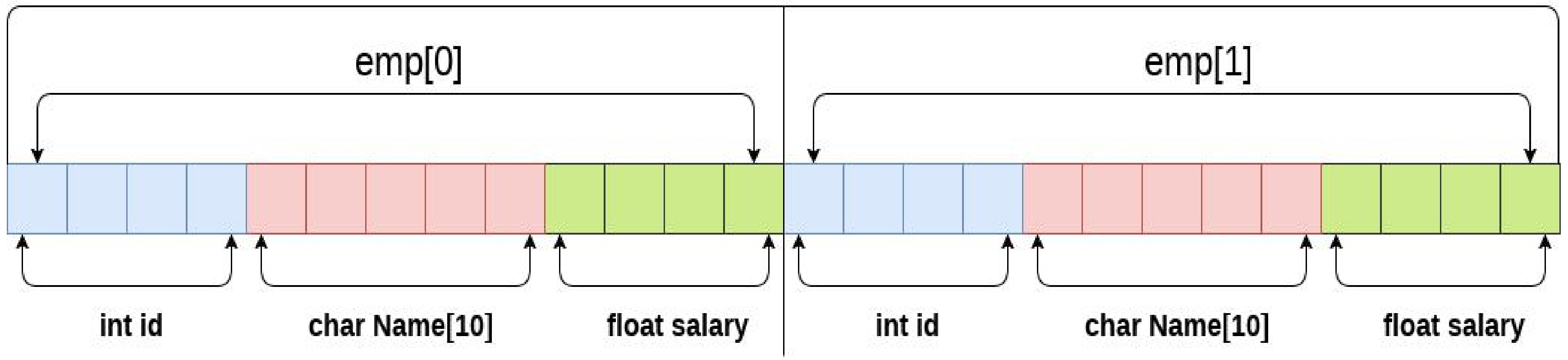
PROGRAM TO STORE DATA OF 3 STUDENTS

```c
#include<stdio.h>
struct student
{
    char name[20];
    int id;
    float marks;
};
void main()
{
    struct student s1,s2,s3;
    int dummy;
    printf("Enter the name, id, and marks of student 1 ");
    scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
    scanf("%c",&dummy);
    printf("Enter the name, id, and marks of student 2 ");
    scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
    scanf("%c",&dummy);
    printf("Enter the name, id, and marks of student 3 ");
    scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
    scanf("%c",&dummy);
    printf("Printing the details....\n");
    printf("%s %d %f\n",s1.name,s1.id,s1.marks);
    printf("%s %d %f\n",s2.name,s2.id,s2.marks);
    printf("%s %d %f\n",s3.name,s3.id,s3.marks);
}
```

Array of Structures in C

- An array of structres in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Array of structures



```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

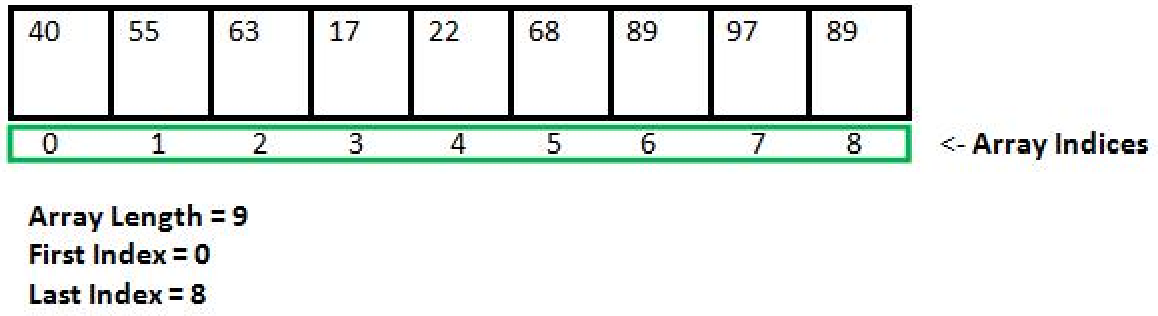
```c
#include<stdio.h>
#include <string.h>
struct student{
int rollno;
char name[10];
};

int main(){
int i;
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```

Difference between Structure and Array in C

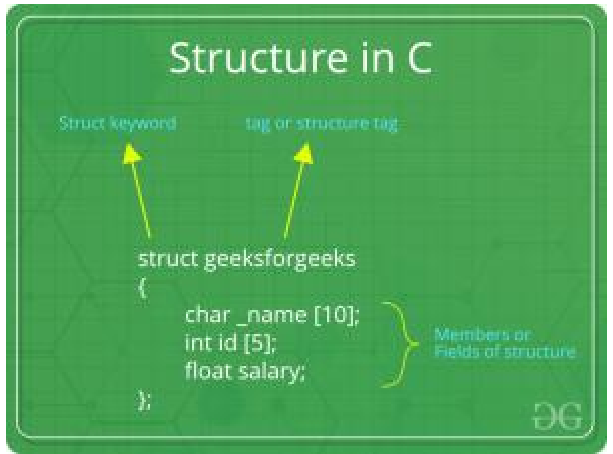
Array in C

An array is collection of items stored at continuous memory locations.



Structure in C

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.



```
struct geeksforgeeks
{
        char _name [10];
        int id [5];
        float salary;
};
```

ARRAY	STRUCTURE
Array refers to a collection consisting of elements of homogenous data type.	Structure refers to a collection consisting of elements of heterogenous data type.
Array uses subscripts or “[ ]” (square bracket) for element access	Structure uses “.” (Dot operator) for element access
Array is pointer as it points to the first element of the collection.	Structure is not a pointer
Instantiation of Array objects is not possible.	Instantiation of Structure objects is possible.
Array size is fixed and is basically the number of elements multiplied by the size of an element.	Structure size is not fixed as each element of Structure can be of different type and size.
Bit filed is not possible in an Array.	Bit filed is possible in an Structure.

NESTED STRUCTURES

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.

- The structure variables can be a normal structure variable or a pointer variable to access the data.
- i.e to say:

1. Structure within structure in C using normal variable
2. Structure within structure in C using pointer variable

```
#include <stdio.h>
#include <string.h>
struct student_college_detail
{
    int college_id;
    char college_name[50];
};
struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}stu_data;

int main()
{
    struct student_detail stu_data = {1, "Raju", 90.5, 71145, "Anna University"};
    printf(" Id is: %d \n", stu_data.id);
    printf(" Name is: %s \n", stu_data.name);
    printf(" Percentage is: %f \n\n", stu_data.percentage);
    printf(" College Id is: %d \n",
                stu_data.clg_data.college_id);
    printf(" College Name is: %s \n",
                stu_data.clg_data.college_name);
    return 0;
}
```

STRUCTURE WITHIN STRUCTURE IN C USING POINTER VARIABLE:

- This program explains how to use structure within structure in C using pointer variable. “student_college_detail’ structure is declared inside “student_detail” structure in this program. one normal structure variable and one pointer structure variable is used in this program.
- Please note that combination of .(dot) and ->(arrow) operators are used to access the structure member which is declared inside the structure.

```
#include <stdio.h>
#include <string.h>
struct student_college_detail
{     int college_id;
    char college_name[50];
};
struct student_detail
{ int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}stu_data, *stu_data_ptr;

int main()
{  struct student_detail stu_data = {1, "Raju", 90.5, 71145, "Anna
University"};
    stu_data_ptr = &stu_data;
    printf(" Id is: %d \n", stu_data_ptr->id);
    printf(" Name is: %s \n", stu_data_ptr->name);
    printf(" Percentage is: %f \n\n", stu_data_ptr->percentage);
    printf(" College Id is: %d \n", stu_data_ptr->clg_data.college_id);
    printf(" College Name is: %s \n", stu_data_ptr->clg_data.college_name);
    return 0;
}
```

Pointer to a Structure in C

pointer is a variable which points to the address of another variable of any data type like int, char, float etc. Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable. Here is how we can declare a pointer to a structure variable.

```
struct dog
{
    char name[10];
    char breed[10];
    int age;
    char color[10];
};

struct dog spike;

// declaring a pointer to a structure of type
struct dog
struct dog *ptr_dog
```

This declares a pointer ptr_dog that can store the address of the variable of type struct dog. We can now assign the address of variable spike to ptr_dog using & operator.

```
ptr_dog = &spike;
```

Accessing members using Pointer

There are two ways of accessing members of structure using pointer:

1. Using indirection (*) operator and dot (.) operator.
2. Using arrow (->) operator or membership operator.

Using Indirection (*) Operator and Dot (.) Operator

At this point ptr_dog points to the structure variable spike, so by dereferencing it we will get the contents of the spike. This means spike and *ptr_dog are functionally equivalent. To access a member of structure write *ptr_dog followed by a dot(.) operator, followed by the name of the member. For example:

(*ptr_dog).name – refers to the name of dog
(*ptr_dog).breed – refers to the breed of dog
and so on.

Parentheses around *ptr_dog are necessary because the precedence of dot(.) operator is greater than that of indirection (*) operator.

Using arrow operator (->)

The above method of accessing members of the structure using pointers is slightly confusing and less readable, that's why C provides another way to access members using the arrow (->) operator. To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member.

```
ptr_dog->name - refers to the name of dog
ptr_dog->breed - refers to the breed of dog
```

and so on.

```
#include<stdio.h>
 struct dog
{    char name[10];
    char breed[10];
    int age;
    char color[10];
};
 int main()
{    struct dog my_dog = {"tyke", "Bulldog", 5, "white"};
    struct dog *ptr_dog;
    ptr_dog = &my_dog;
     printf("Dog's name: %s\n", ptr_dog->name);
    printf("Dog's breed: %s\n", ptr_dog->breed);
    printf("Dog's age: %d\n", ptr_dog->age);
    printf("Dog's color: %s\n", ptr_dog->color);

    // changing the name of dog from tyke to jack
    strcpy(ptr_dog->name, "jack");
     // increasing age of dog by 1 year
    ptr_dog->age++;
     printf("Dog's new name is: %s\n", ptr_dog->name);
    printf("Dog's age is: %d\n", ptr_dog->age);
     // signal to operating system program ran fine
    return 0;}
```

Topics to be discussed in next class:

*Structure variables as Functions arguments
*Returning structure variables

Passing struct to function

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won’t be available to other functions unless it is passed to those functions by value or by address(reference).
- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

- **PASSING STRUCTURE TO FUNCTION IN C:**

  It can be done in below 3 ways:
- Passing structure to a function by value
- Passing structure to a function by address(reference)
- No need to pass a structure – Declare structure variable as global

1. EXAMPLE PROGRAM – PASSING STRUCTURE TO FUNCTION IN C BY VALUE:

- In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

```
#include <stdio.h>
#include <string.h>
struct student
{            int id;
            char name[20];
            float percentage;
};
 void func(struct student record);
 int main()
{
            struct student record;
             record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;
             func(record);
            return 0;
}

void func(struct student record)
{
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
}
```

2. EXAMPLE PROGRAM – PASSING STRUCTURE TO FUNCTION IN C BY ADDRESS:

- In this program, the whole structure is passed to another function by address. It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

```
• #include <stdio.h>
• #include <string.h>
• struct student
• {          int id;
•            char name[20];
•            float percentage;
• };
•  void func(struct student *record);
•  int main()
• {         struct student record;
•            record.id=1;
•           strcpy(record.name, "Raju");
•           record.percentage = 86.5;
•           func(&record);
•           return 0;
• }
•  void func(struct student *record)
• {         printf(" Id is: %d \n", record->id);
•           printf(" Name is: %s \n", record->name);
•           printf(" Percentage is: %f \n", record->percentage);
• }
```

3. EXAMPLE PROGRAM TO DECLARE A STRUCTURE VARIABLE AS GLOBAL IN C:

- Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don’t need to pass the structure to any function separately.

```
include <stdio.h>
#include <string.h>

struct student
{
            int id;
            char name[20];
            float percentage;
};
struct student record; // Global declaration of structure
void structure_demo();
int main()
{            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;
            structure_demo();
            return 0;
}
void structure_demo()
{
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
}
```

UNION

- C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.
- Union and structure in C are same in concepts, except allocating memory for their members.
- Structure allocates storage space for all its members separately.
- Whereas, Union allocates one common storage space for all its members
- We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Wher**e** as Structure allocates storage space for all its members separately.
- Many union variables can be created in a program and memory will be allocated for each union variable separately.

Using normal variable	Using pointer variable

Syntax:
union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
	Syntax:
	union tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};

Example:
union student
{
int mark;
char name[10];
float average;
};
	Example:
	union student
	{
	int mark;
	char name[10];
	float average;
	};

Declaring union using normal variable: union student report;	**Declaring union using pointer variable:** union student *report, rep;

Initializing union using normal variable:
union student report = {100, “Mani”, 99.5};
	Initializing union using pointer variable:
	union student rep = {100, “Mani”, 99.5};
	report = &rep;

Accessing union members using normal variable:
report.mark;
report.name;
report.average;
	Accessing union members using pointer variable:
	report -> mark;
	report -> name;
	report -> average;

```
#include <stdio.h>
#include <string.h>
 union student
{             char name[20];
             char subject[20];
             float percentage;};
 int main()
{ union student record1;
    union student record2;
     // assigning values to record1 union variable
       strcpy(record1.name, "Raju");
       strcpy(record1.subject, "Maths");
       record1.percentage = 86.50;
        printf("Union record1 values example\n");
       printf(" Name       : %s \n", record1.name);
       printf(" Subject    : %s \n", record1.subject);
       printf(" Percentage : %f \n\n", record1.percentage);
     // assigning values to record2 union variable
       printf("Union record2 values example\n");
       strcpy(record2.name, "Mani");
       printf(" Name       : %s \n", record2.name);
        strcpy(record2.subject, "Physics");
       printf(" Subject    : %s \n", record2.subject);
        record2.percentage = 99.50;
       printf(" Percentage : %f \n", record2.percentage);
       return 0;}
```

EXPLANATION FOR C UNION PROGRAM:

There are 2 union variables declared in this program to understand the difference in accessing values of union members.

Record1 union variable:

- “Raju” is assigned to union member “record1.name” . The memory location name is “record1.name” and the value stored in this location is “Raju”.
- Then, “Maths” is assigned to union member “record1.subject”. Now, memory location name is changed to “record1.subject” with the value “Maths” (Union can hold only one member at a time).
- Then, “86.50” is assigned to union member “record1.percentage”. Now, memory location name is changed to “record1.percentage” with value “86.50”.
- Like this, name and value of union member is replaced every time on the common storage space.
- So, we can always access only one union member for which value is assigned at last. We can’t access other member values.
- So, only “record1.percentage” value is displayed in output. “record1.name” and “record1.percentage” are empty.

Record2 union variable:

- If we want to access all member values using union, we have to access the member before assigning values to other members as shown in record2 union variable in this program.
- Each union members are accessed in record2 example immediately after assigning values to them.
- If we don’t access them before assigning values to other member, member name and value will be over written by other member as all members are using same memory.
- We can’t access all members in union at same time but structure can do that.

EXAMPLE PROGRAM – ANOTHER WAY OF DECLARING C UNION:

In this program, union variable “record” is declared while declaring union itself as shown in the below program.

```
#include <stdio.h>
#include <string.h>
 union student
{            char name[20];
            char subject[20];
            float percentage;
}record;
 int main()
{
             strcpy(record.name, "Raju");
            strcpy(record.subject, "Maths");
            record.percentage = 86.50;

            printf(" Name       : %s \n", record.name);
            printf(" Subject    : %s \n", record.subject);
            printf(" Percentage : %f \n", record.percentage);
            return 0;
}
```

C Structure	C Union
Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
Structure occupies higher memory space.	Union occupies lower memory space over structure.
We can access all members of structure at a time.	We can access only one member of union at a time.

Structure example:
struct student
{
int mark;
char name[6];
double average;
};
	Union example:
	union student
	{
	int mark;
	char name[6];
	double average;
	};

For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes