CS210: Data Structures

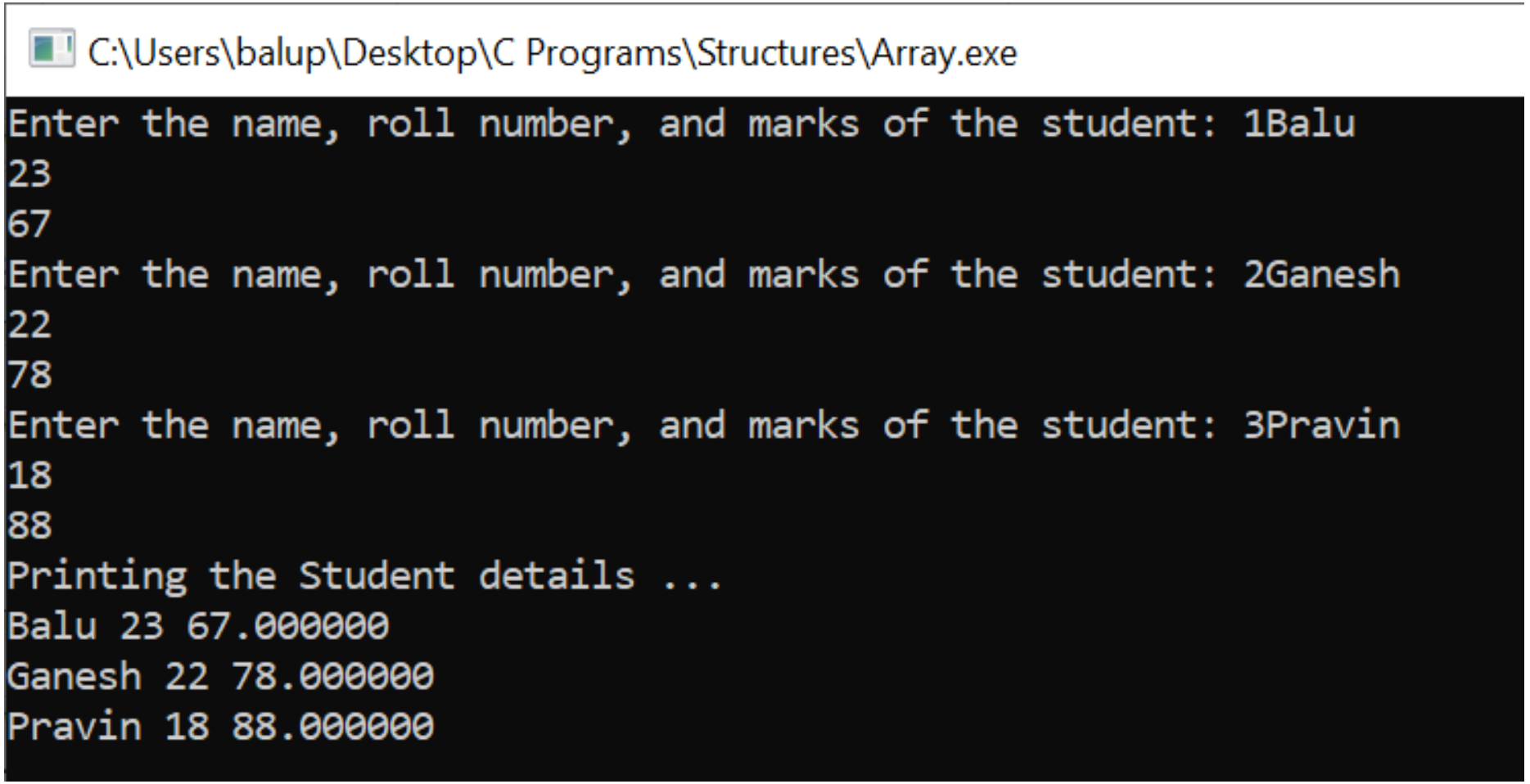
Dr. Balu L. Parne
Assistant Professor,
CSE, SVNIT Surat.

Introduction to Structures

Introduction to Structure

- In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types.
- For example, an entity **Student** may have its name (string), roll number (int), marks (float).
- To store such type of information regarding an entity student, we have the following approaches:
  - Construct individual arrays for storing names, roll numbers, and marks.
  - Use a special data structure to store the collection of different data types.

```c
#include<stdio.h>
int main ()
{
  char names[2][10]; // 2-dimensioanal character array names
  int roll_numbers[2],i;
  float marks[2];
  for (i=0;i<3;i++)
  {
    printf("Enter the name, roll number, and marks of the student: %d", i+1);
    scanf("%s %d %f",&names[i],&roll_numbers[i],&marks[i]);
  }
  printf("Printing the Student details ...\n");
  for (i=0;i<3;i++)
  {
    printf("%s %d %f\n",names[i],roll_numbers[i],marks[i]);
  }
  return 0;
}
```


C:\Users\balup\Desktop\C Programs\Structures\Array.exe
Enter the name, roll number, and marks of the student: 1Balu
23
67
Enter the name, roll number, and marks of the student: 2Ganesh
22
78
Enter the name, roll number, and marks of the student: 3Pravin
18
88
Printing the Student details ...
Balu 23 67.000000
Ganesh 22 78.000000
Pravin 18 88.000000

Introduction to Structure

- The above program may fulfill our requirement of storing the information of an entity student.
- However, the program is very complex, and the complexity increase with the amount of the input.
- The elements of each of the array are stored contiguously, but all the arrays may not be stored contiguously in the memory.
- C provides you with an additional and simpler approach where you can use a special data structure, i.e., **structure, in which, you can group all the information of different data type regarding an entity.**

```c
#include<stdio.h>
int main()
{
    char name[3] ;
    float price[3] ;
    int pages[3], i ;
    printf ("\nEnter names, prices and no. of pages of 3 books\n");
    for (i = 0 ; i <= 2 ; i++)
    {
        scanf("%c %f %d", &name[i], &price[i], &pages[i]);
    }
    printf ("\n And this is what you entered\n");
    for ( i = 0 ; i <= 2 ; i++)
    {
      printf ( "%c %f %d\n", name[i], price[i], pages[i]);
    }
    return 0;
}
```

Introduction to Structure

- The given approach no doubt allows you to store names, prices and number of pages. But as you must have realized, it is an unwieldy approach that obscures the fact that you are dealing with a group of characteristics related to a single entity—the book.
- The program becomes more difficult to handle as the number of items relating to the book go on increasing. For example, we would be required to use a number of arrays, if we also decide to store name of the publisher, date of purchase of book, etc. To solve this problem, C provides a special data type—the structure.

Why structs in C?

- Suppose, you want to store information about a person: his/her name, citizenship number, and salary. You can create different variables name, citNo and salary to store this information.

- What if you need to store information of more than one person? Now, you need to create different variables for each information per person**: name1, citNo1, salary1, name2, citNo2, salary2,etc**.

- A better approach would be to have a collection of all related information under a single name Person structure and use it for every person.

What is Structure?

- A structure is a user defined data type that groups logically related data items of different data types into a single unit.
- All the elements of a structure are stored at contiguous memory locations.
- A variable of structure type can store multiple data items of different data types under the one name.
  - E.g. data of employee in a company such as name, Employee ID, salary, address, phone number.

What is Structure?

- Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.
- A structure contains a number of data types grouped together. These data types may or may not be of the same type.
- Structure in c is a user-defined data type that enables us to store the collection of different data types.
- Each element of a structure is called a member.
- In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.

Defining Structure

- The **struct** keyword is used to define the structure. The general form of a structure declaration statement is given below:

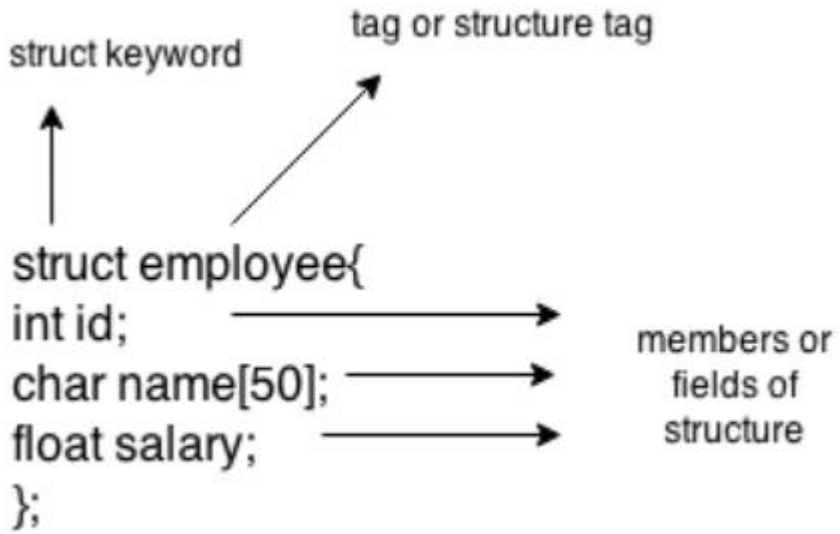
```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Defining Structure

- For Example: Here, a derived type **struct Person and struct book** is defined.

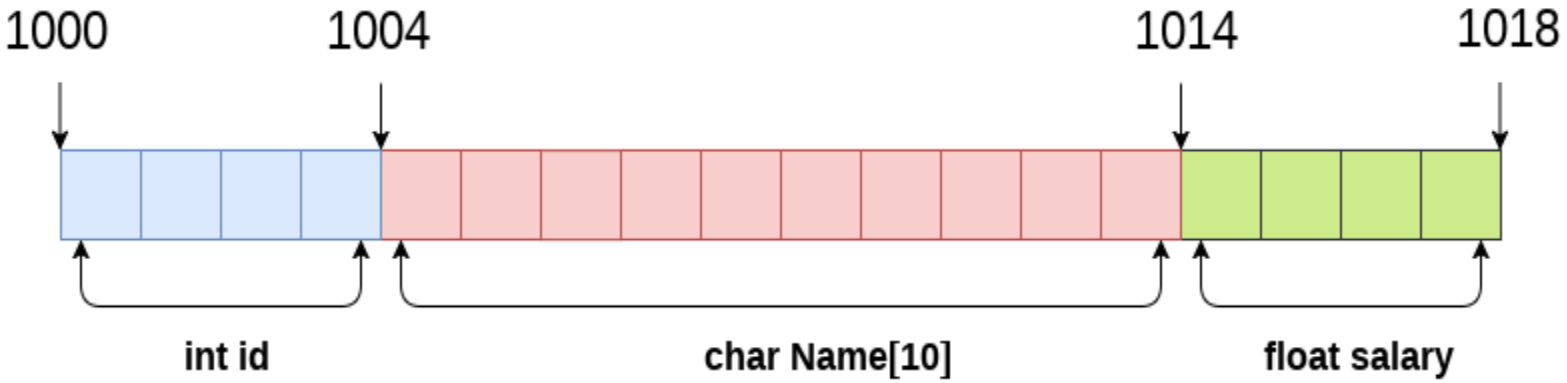
```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};
```

```
struct book
{
    char name ;
    float price ;
    int pages ;
} ;
```



Defining Structure:

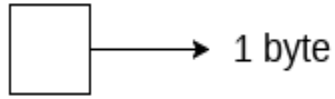
```
struct employee
{   int id;
    char name[20];
    float salary;
};
```




```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte

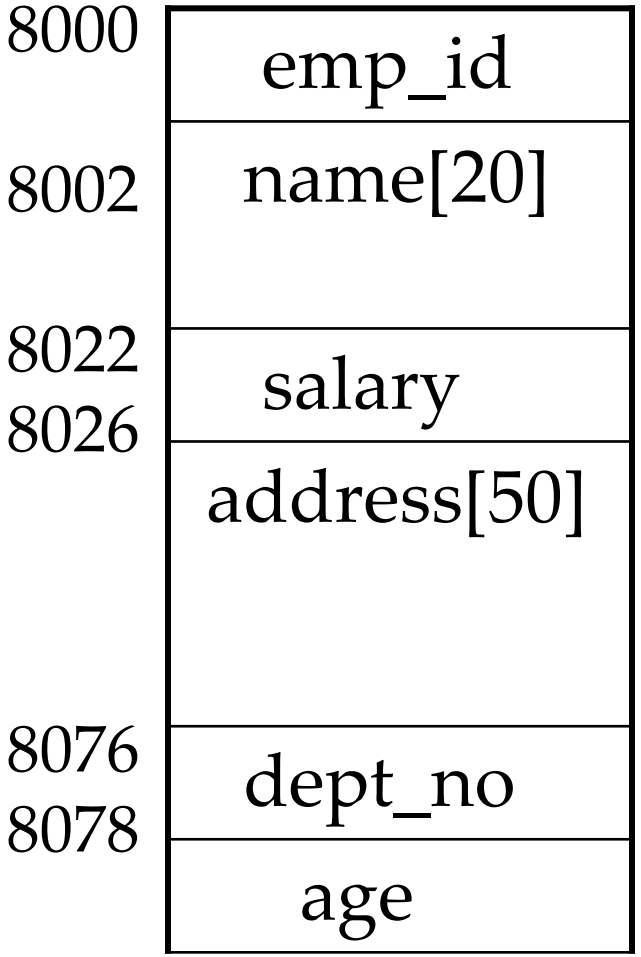


Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure.

Defining structure…

```
struct employee
{
    int  emp_id;
    char name[20];
    float salary;
    char address[50];
    int dept_no;
    int age;
};
```

Memory Allocation



Declaring a Structure Variable

- A structure has to be declared, after the body of structure is defined.
- The syntax of declaring a structure is

  struct <struct_name> <variable_name>;
- E.g. struct employee e1;
- Here e1 variable contains 6 members that are defined in structure.

Declaring a structure variable:

- We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:
  - By struct keyword within main() function
  - By declaring a variable at the time of defining the structure.

Declaring structure variable:By struct keyword within main() function

- The example to declare the structure variable by **struct** keyword. It should be declared within the main function.

```
struct employee
{   int id;
    char name[50];
    float salary;
};
```

```
struct employee e1, e2;
```

- The variables e1 and e2 can be used to access the values stored in the structure.

Declaring structure variable:By struct keyword within main() function

- The example to declare the structure variable by **struct** keyword. It should be declared within the main function.

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};

int main()
{
    struct Person person1, person2, p[20];
    return 0;
}
```

- Here, two variables person1, person2, and an array variable p having 20 elements of type **struct Person** are created.

By declaring a variable at the time of defining the structure:

- Another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;
```

- The variables e1 and e2 can be used to access the values stored in the structure.

Declaring structure variable:

- If we desire, we can combine the declaration of the structure type and the structure variables in one statement.
-

```
struct book
{
      char  name ;
      float  price ;
      int  pages ;
} ;
struct book  b1, b2, b3 ;
```

Is same as:

```
struct book
{
      char  name ;
      float  price ;
      int  pages ;
} b1, b2, b3 ;
```

Declaring structure variable:

- If we desire, we can combine the declaration of the structure type and the structure variables in one statement.

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
} person1, person2, p[20];
```

- Here, two variables person1, person2, and an array variable p having 20 elements of type **struct Person** are created.

Initializing a Structure Members

- The members of individual structure variable is initialized one by one or in a single statement.
- struct employee e1 = {1, “Hemant”,12000, “3 vikas colony new delhi”,10, 35}; OR

```
e1.emp_id=1;
strcpy(e1.name,“Hemant”);
e1.salary=12000;
strcpy(e1.address,“3  vikas colony new delhi”);
e1.dept_no=10;
e1.age=35;
```

Initializing structure variable:

- Like primary variables and arrays, structure variables can also be **initialized** where they are declared. The format used is quite similar to that used to initialize arrays.

```
struct book
{
        char  name[10] ;
        float  price ;
        int  pages ;
} ;
struct book  b1 = { "Basic", 130.00, 550 } ;
struct book  b2 = { "Physics", 150.80, 800 } ;
```

```c
#include<stdio.h>
int main( )
{
    struct employee
    {
        int id;
        char name[50];
        float salary;
    };
    struct employee e1 = {101, "Sonu Jaiswal",  56000}; //declaring e1 and e2 variables for structure
    struct employee e2 = {102, "James Bond", 126000};  //store employee information
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
   printf( "employee 1 salary : %f\n", e1.salary);
   //printing second employee information
   printf( "employee 2 id : %d\n", e2.id);
   printf( "employee 2 name : %s\n", e2.name);
   printf( "employee 2 salary : %f\n", e2.salary);
   return 0;
}
```

Points to be consider while Declaring a Structure Type:

- The closing brace in the structure type declaration must be followed by a semicolon.
- It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the 'form' of the structure.
- Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined. In very large programs they are usually put in a separate header file, and the file is included (using the preprocessor directive #include) in whichever program we want to use this structure type.
- If a structure variable is initiated to a value {0}, then all its elements are set to a value 0, as in e3 in the given program. This is a handy way of initializing structure variables. In absence of this, we would have been required to initialize each individual element to a value 0.

Which approach is good:

- If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.

- If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in main() function.

Access members of a structure:

- There are two types of operators used for accessing members of a structure:
  - By . (member or dot operator)
  - By -> (structure pointer operator)
- In arrays we can access individual elements of an array using a subscript. Structures use a different scheme. They use a dot (.) operator.
- Suppose, you want to access the salary of person2. Here's how you can do it:

```
person2.salary
```

- **Note that before the dot there must always be a structure variable and after the dot there must always be a structure element.**

Accessing a Structure Members

- The structure members cannot be directly accessed in the expression.
- They are accessed by using the name of structure variable followed by a dot and then the name of member variable.
- E.g.
  - e1.emp_id, e1.name, e1.salary, e1.address, e1.dept_no, e1.age.
  - The data with in the structure is stored and printed using dot operator in scanf and printf statement in c program.

How structure Elements are Stored:

- Whatever be the elements of a structure, they are always stored in contiguous memory locations.

```c
#include<stdio.h>
int main()
{
    struct book
    {
        char name[2];
        float price;
        int pages;
    };
    struct book b1 = {"CP", 780.50, 50};
    printf("\n Address of name = %u", &b1.name);
    printf("\n Address of price = %u", &b1.price);
    printf("\n Address of pages = %u", &b1.pages);
}
```

```
 Address of name = 6487568
 Address of price = 6487572
 Address of pages = 6487576
--------------------------------
Process exited after 0.3755 seconds with return value 28
Press any key to continue . . .
```

Array of Structure:

- C allows to create an array of variables of type structure.
- The array of structure is used to store the large number of similar records.
- For example,
  - Store the records of 100 employees
- The method to define and access the array element of array of structure is similar to other array. The syntax to define the array of structure is
    - **struct <struct_name> <array_name> [<size>];**
- For Example:-
- **struct employee emp[100];**

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

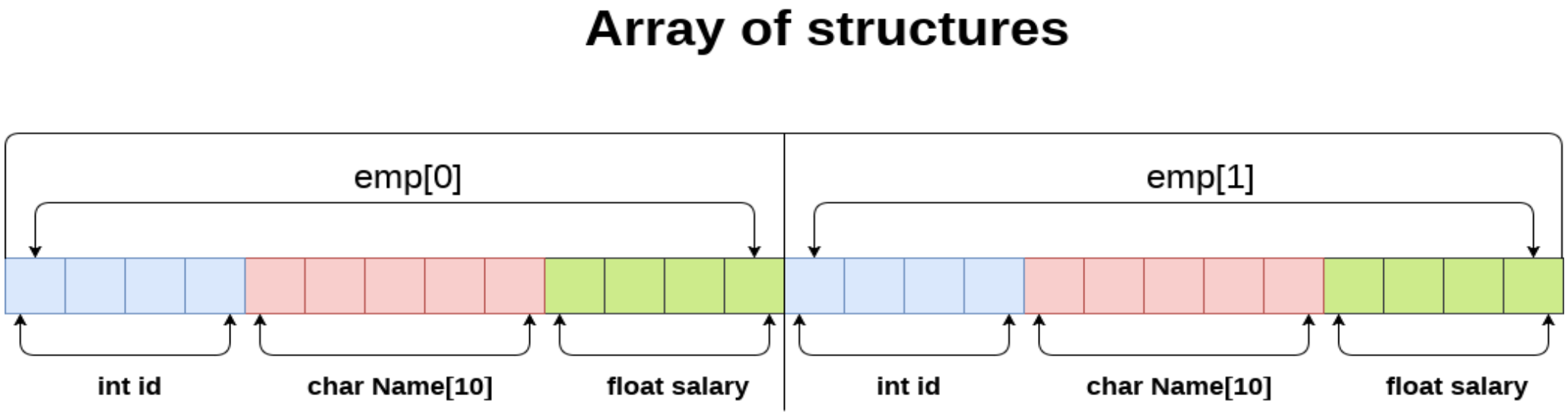
Why Array of Structure?

- In the above program, we have stored data of 3 students in the structure.
- However, the complexity of the program will be increased if there are 20 students. In that case, we will have to declare 20 different structure variables and store them one by one. This will always be tough since we will have to declare a variable every time we add a student.
- Remembering the name of all the variables is also a very tricky task.
- However, c enables us to declare an array of structures by using which, we can avoid declaring the different structure variables; instead we can make a collection containing all the structures that store the information of different entities.

Array of Structure:

- An array of structures 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 Structure:




```
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>
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;
}
```

Additional Features of Structures:

- The values of a structure variable can be assigned to another structure variable of the same type using the assignment operator.
- Ability to copy the contents of all structure elements of one variable into the corresponding elements of another structure variable is rather surprising, since C does not allow assigning the contents of one array to another just by equating the two. As we saw earlier, for copying arrays we have to copy the contents of the array element by element.
- This copying of all structure elements at one go has been possible only because the structure elements are stored in contiguous memory locations. Had this not been so, we would have been required to copy structure variables element by element.

Assignment Operator:

- The value of one structure variable is assigned to another variable of same type using assignment operator.
- If the e1 and e2 are structure variables of type employee then the statement

  e1 = e2;

- assigns value of structure variable e2 to e1.
- The value of each member of e2 is assigned to corresponding members of e1.

Assignment….

- However,

  e1 = = e2; OR e1 != e2; statements are not permitted !!!
- In case you want to compare two structures, you have to compare them member by member.

Nested Structure:

- C provides us the feature of nesting one structure within another structure by using which, complex data types are created.
- For example, we may need to store the address of an entity employee in a structure.
- The attribute address may also have the subparts as street number, city, state, and pin code.
- Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee. Consider the following program.

```c
#include<stdio.h>
struct address
{
    char city[20];
    int pin;
    char phone[14];
};
struct employee
{
    char name[20];
    struct address add;
};
void main ()
{
    struct employee emp;
    printf("Enter employee information : empname, empaddresscity, employee pin, employeephone ?\n");
    scanf("%s %s %d %s",&emp.name,&emp.add.city, &emp.add.pin, &emp.add.phone);
    printf("Printing the employee information....\n");
    printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
}
```

Structures within Structures

- C allows to define a variable of structure type as a member of other structure type.
- The syntax to define the structure within structure is.

```
struct <struct_name>{
 <data_type> <variable_name>;
                struct <struct_name>
                    { <data_type> <variable_name>;
                   ……..}<struct_variable>;
 <data_type> <variable_name>;
 };
```

Nested Structure:

- The structure can be nested in the following ways.
  - **By separate structure :** we create two structures, but the dependent structure should be used inside the main structure as a member.
  - **By Embedded structure:** The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but **it cannot be used in multiple data structures.**

Separate Structure:

```
struct Date
{
   int dd;
   int mm;
   int yyyy;
};
struct Employee
{
   int id;
   char name[20];
   struct Date doj;
}emp1;
```

As you can see, doj (date of joining) is the variable of type Date. Here doj is used as a member in Employee structure. In this way, we can use Date structure in many structures.

Embedded Structure:

```
struct Employee
{
   int id;
   char name[20];
   struct Date
    {
      int dd;
      int mm;
      int yyyy;
    }doj;
}emp1;
```

Structures within Structures…

- Accessing Structures within Structures
  - The data member of structure within structure is accessed by using two period (.) symbol.
  - We can access the member of the nested structure by Outer_Structure.Nested_Structure.

    struct _var. nested_struct_var. struct_member;

    For Example:-

    e1.doj.dd
    e1.doj.mm
    e1.doj.yyyy

Passing structure to function:

- Just like other variables, a structure can also be passed to a function.
- We may pass the structure members into the function or pass the structure variable at once.
- To pass individual elements would become more tedious as the number of structure elements goes on increasing. A better way would be to pass the entire structure variable at a time.
- Similar to variables of built-in types, you can also pass structure variables to a function.
- You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

Passing Structure to Function

- The individual elements of a structure variable can be passed to a function as a parameter.

```
struct employee
{
   int  emp_id;
   char name[20];
   float salary;
};
```

Passing Structure to Function…

```
void printdata(int, char *, float);
void main ( )
  {
    struct employee e1;
    printf (“Enter the employee id of employee”);
    scanf(“%d”,&e1.emp_id);
    printf (“Enter the name of employee”);
    scanf(“%s”,e1.name);
    printf (“Enter the salary of employee”);
    scanf(“%f”,&e1.salary);
    printdata (e1.emp_id,e1.name,e1.salary);
    getch();
  }
```

Passing Structure to Function…

```
void printdata(int id, char *name, float sal)
 {
   printf (“\nThe employee id of employee is : %d”, id);
   printf (“\nThe name of employee is : %s”, name);
   printf (“\nThe salary of employee is : %f”, sal);
 }
```

Here, we have passed the base address of the arrays name and the value stored in emp_id as well as salary. Thus, this is a mixed call- a call by reference as well as call by value.

Passing Structure to Function

- The structure variable can be passed to a function as a parameter.

```
struct employee
{
   int  emp_id;
   char name[20];
   float salary;
};
```

Passing Structure to Function…

```
void printdata(struct employee emp);
void main ( )
  {
    struct employee e1;
    printf (“Enter the employee id of employee”);
    scanf(“%d”,&e1.emp_id);
    printf (“Enter the name of employee”);
    scanf(“%s”,e1.name);
    printf (“Enter the salary of employee”);
    scanf(“%f”,&e1.salary);
    printdata (e1);
    getch();
  }
```

Passing Structure to Function…

```
void printdata(struct employee emp)
 {
    printf (“\nThe employee id of employee is : %d”, emp.emp_id);
    printf (“\nThe name of employee is : %s”, emp.name);
    printf (“\nThe salary of employee is : %f”, emp.salary);
 }
```

Function Returning Structure

- The function can return a variable of structure type like a integer and float variable.

```
struct employee
{
   int  emp_id;
   char name[20];
   float salary;
};
```

Function Returning Structure…

```
struct employee getdata();
void main ( )
  {
    struct employee emp;
    emp=getdata();
    printf (“\nThe employee id of employee is :%d”, emp.emp_id);
    printf (“\nThe name of employee is : %s”, emp.name);
    printf (“\nThe salary of employee is : %f”, emp.salary);
    getch();
  }
```

Function Returning Structure…

```
struct employee getdata( )
 {
    struct employee e1;
    printf (“Enter the employee id of employee”);
    scanf(“%d”,&e1.emp_id);
    printf (“Enter the name of employee”);
    scanf(“%s”,e1.name);
    printf (“Enter the salary of employee”);
    scanf(“%f”,&e1.salary);
    return(e1);
 }
```

Thank You…!!! ☺