

EXERCISE 10

Content:

1. C Structure
2. typedef in C
3. C Array of Structures
4. C Nested Structure

I. C Structure

Why use 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.

What is Structure?

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. Structures can simulate the use of classes and templates as it can store various information

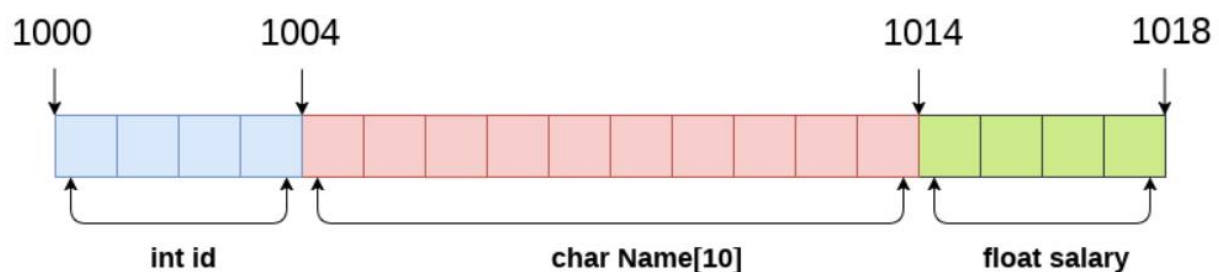
The `struct` keyword is used to define the structure. Let's see the syntax to define the structure in C.

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

Let's see the example to define a structure for an entity employee in C:

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

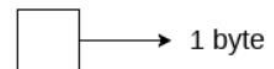
The following image shows the memory allocation of the structure employee that is defined in the above example:



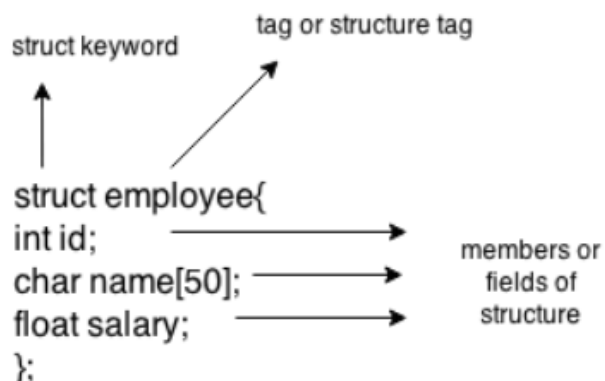
```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

$\text{sizeof (emp)} = 4 + 10 + 4 = 18 \text{ bytes}$

where;
 $\text{sizeof (int)} = 4 \text{ byte}$
 $\text{sizeof (char)} = 1 \text{ byte}$
 $\text{sizeof (float)} = 4 \text{ byte}$



Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



Declaring 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.

1st way:

Let's see 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;
};
```

Now write given code inside the main() function.

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

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

2nd way:

Let's see another way to declare variable at the time of defining the structure.

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

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.

Accessing members of the structure

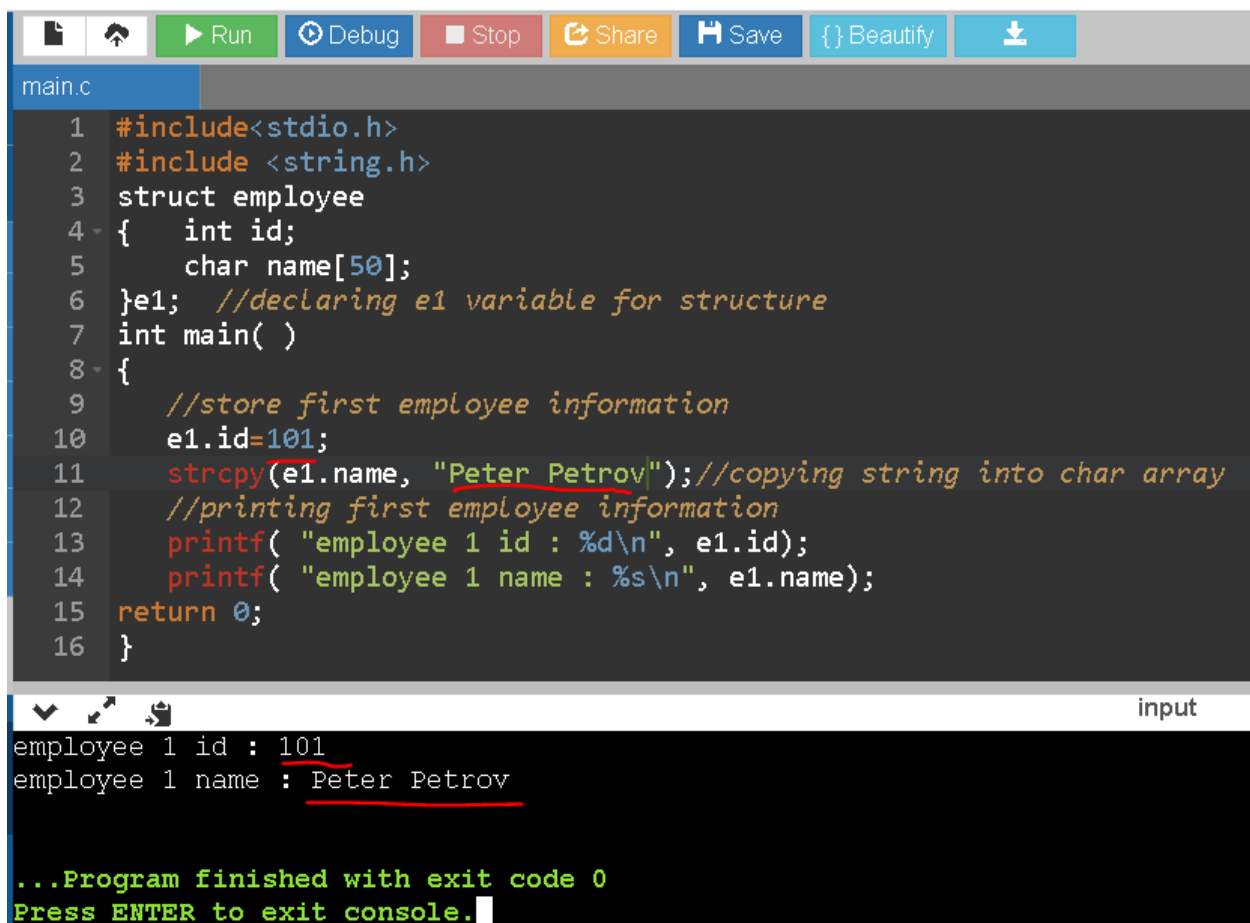
There are two ways to access structure members:

- By . (member or dot operator)
- By -> (structure pointer operator)

Let's see the code to access the id member of p1 variable by. (member) operator:

p1.id

C Structure example: Let's see a simple example of structure in C language.



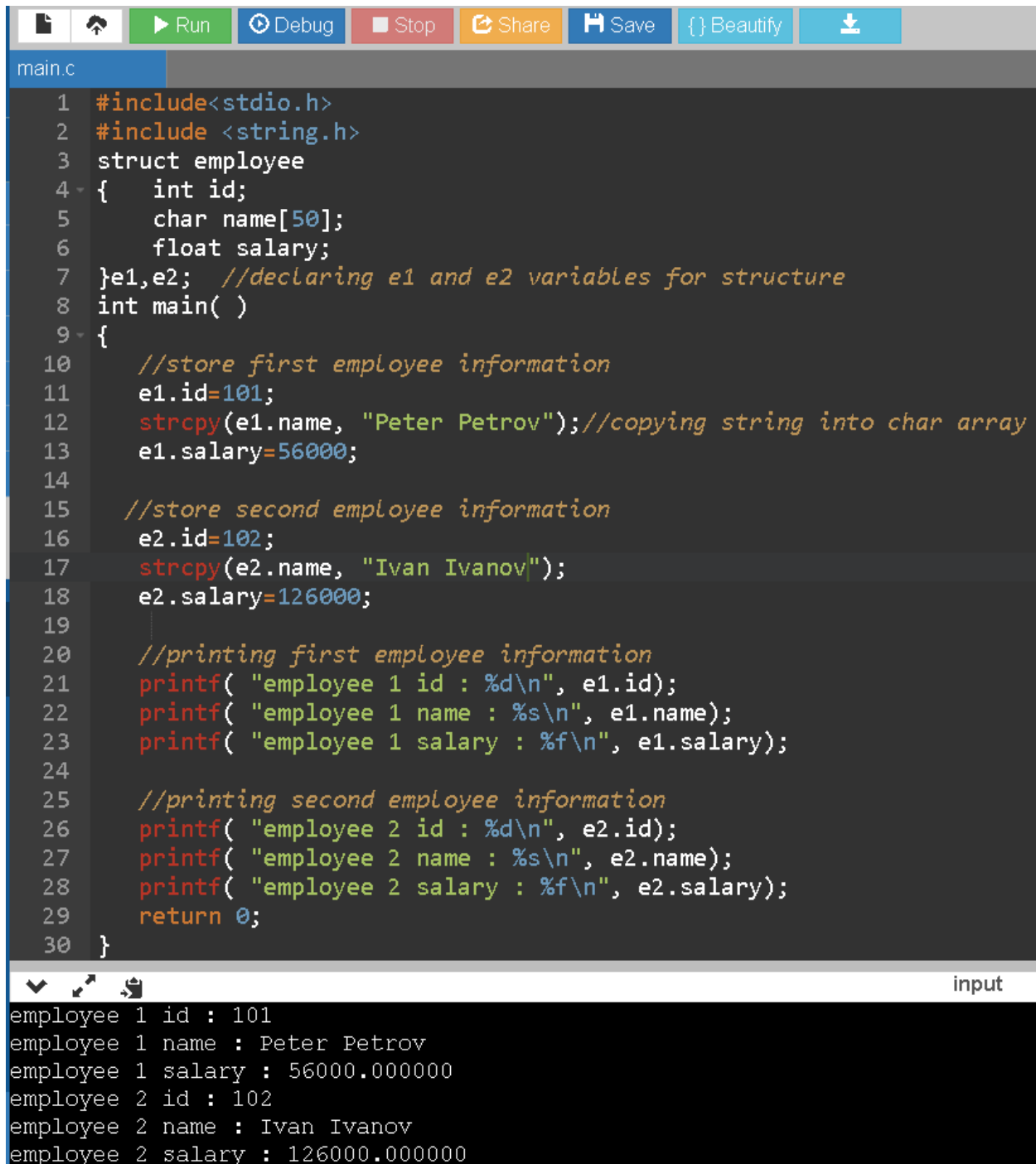
```
main.c
1  #include<stdio.h>
2  #include <string.h>
3  struct employee
4  {   int id;
5      char name[50];
6  }e1; //declaring e1 variable for structure
7  int main( )
8  {
9      //store first employee information
10     e1.id=101;
11     strcpy(e1.name, "Peter Petrov");//copying string into char array
12     //printing first employee information
13     printf( "employee 1 id : %d\n", e1.id);
14     printf( "employee 1 name : %s\n", e1.name);
15     return 0;
16 }
```

input

```
employee 1 id : 101
employee 1 name : Peter Petrov

...Program finished with exit code 0
Press ENTER to exit console.
```

Let's see another example of the structure in C language to store many employees information.



```
main.c
1  #include<stdio.h>
2  #include <string.h>
3  struct employee
4  {   int id;
5      char name[50];
6      float salary;
7  }e1,e2; //declaring e1 and e2 variables for structure
8  int main( )
9  {
10     //store first employee information
11     e1.id=101;
12     strcpy(e1.name, "Peter Petrov");//copying string into char array
13     e1.salary=56000;
14
15     //store second employee information
16     e2.id=102;
17     strcpy(e2.name, "Ivan Ivanov");
18     e2.salary=126000;
19
20     //printing first employee information
21     printf( "employee 1 id : %d\n", e1.id);
22     printf( "employee 1 name : %s\n", e1.name);
23     printf( "employee 1 salary : %f\n", e1.salary);
24
25     //printing second employee information
26     printf( "employee 2 id : %d\n", e2.id);
27     printf( "employee 2 name : %s\n", e2.name);
28     printf( "employee 2 salary : %f\n", e2.salary);
29     return 0;
30 }
```

input

```
employee 1 id : 101
employee 1 name : Peter Petrov
employee 1 salary : 56000.000000
employee 2 id : 102
employee 2 name : Ivan Ivanov
employee 2 salary : 126000.000000
```

II. typedef in C

The **typedef** is a keyword used in C programming to provide some meaningful names to the already existing variable in the C program. It behaves similarly as we define the alias for the commands. In short, we can say that this keyword is used to redefine the name of an already existing variable.

Syntax of typedef

typedef <existing_name> <alias_name>

In the above syntax, 'existing_name' is the name of an already existing variable while 'alias name' is another name given to the existing variable.

For example, suppose we want to create a variable of type unsigned int, then it becomes a tedious task if we want to declare multiple variables of this type. To overcome the problem, we use a typedef keyword.

typedef unsigned int unit;

In the above statements, we have declared the unit variable of type unsigned int by using a typedef keyword.

Now, we can create the variables of type unsigned int by writing the following statement:

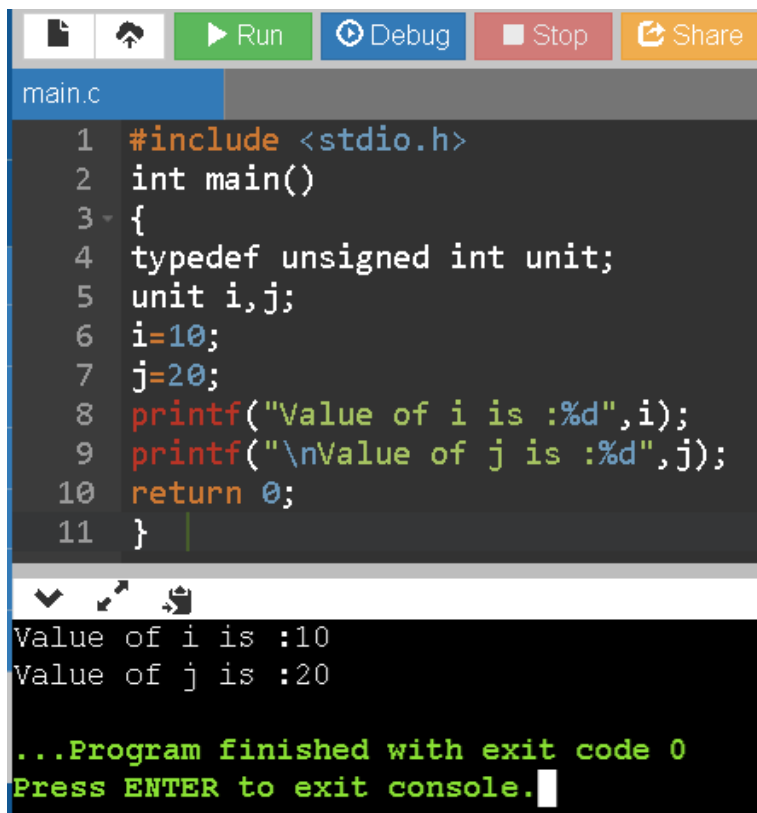
unit a, b;

instead of writing:

unsigned int a, b;

Till now, we have observed that the typedef keyword provides a nice shortcut by providing an alternative name for an already existing variable. This keyword is useful when we are dealing with the long data type especially, structure declarations.

Let's understand through a simple example:

A screenshot of a code editor window showing a C program named 'main.c'. The code defines a typedef for 'unsigned int' as 'unit', declares two variables 'i' and 'j' of type 'unit', assigns 'i' the value 10 and 'j' the value 20, and prints their values. The program is executed, and the output is displayed in a console window below the code editor. The console shows the printed values and a message indicating the program finished successfully.

```
1 #include <stdio.h>
2 int main()
3 {
4     typedef unsigned int unit;
5     unit i,j;
6     i=10;
7     j=20;
8     printf("Value of i is :%d",i);
9     printf("\nValue of j is :%d",j);
10    return 0;
11 }
```

Value of i is :10
Value of j is :20

...Program finished with exit code 0
Press ENTER to exit console.

Using typedef with structures

Consider the below structure declaration:

```
struct student
{
    char name[20];
    int age;
};
struct student s1;
```

In the above structure declaration, we have created the variable of student type by writing the following statement:

```
struct student s1;
```

The above statement shows the creation of a variable, i.e., s1, but the statement is quite big. To avoid such a big statement, we use the typedef keyword to create the variable of type student.

```
struct student
{
    char name[20];
    int age;
};
typedef struct student stud;
stud s1, s2;
```

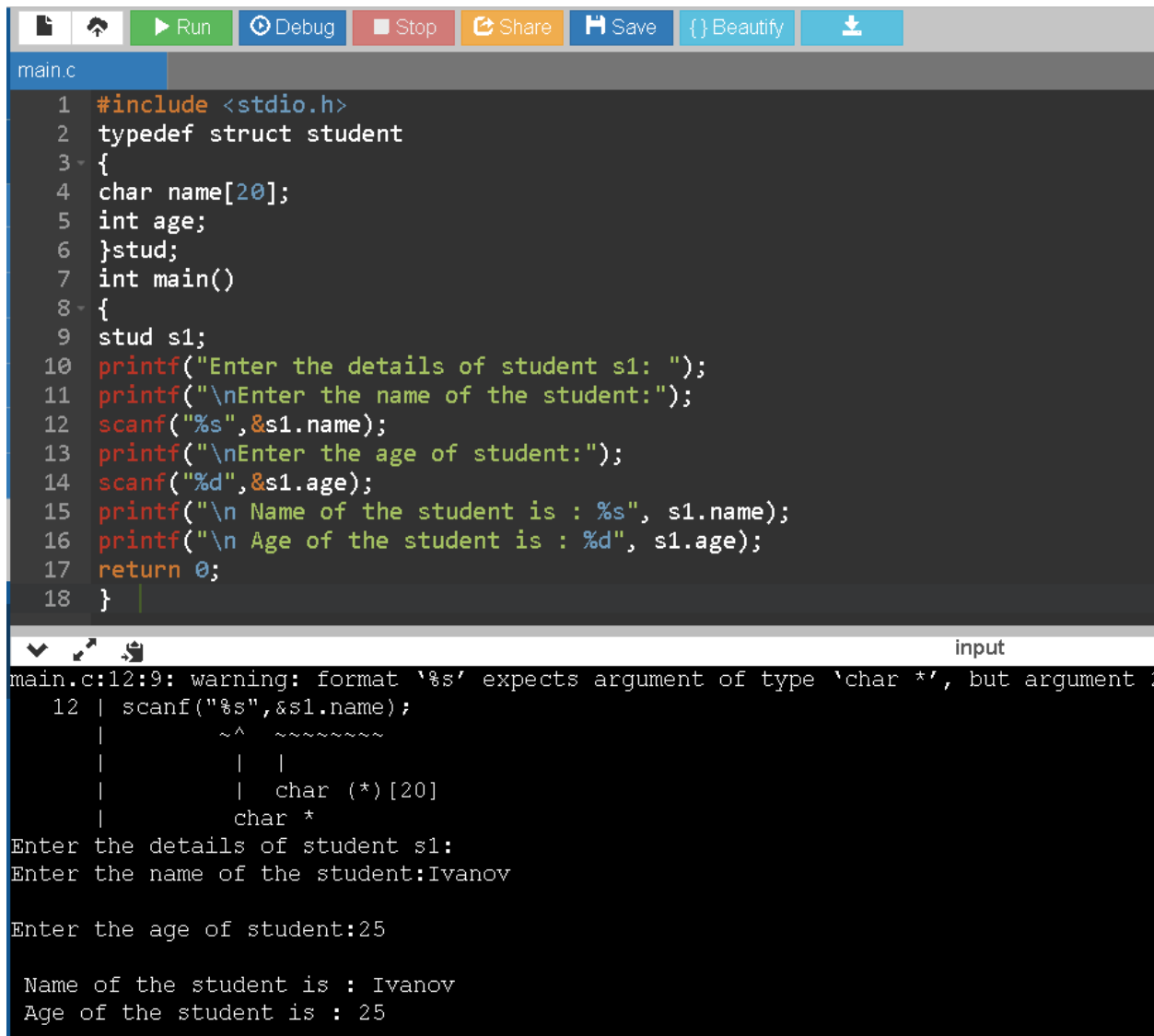
In the above statement, we have declared the variable stud of type struct student. Now, we can use the stud variable in a program to create the variables of type struct student.

The above typedef can be written as:

```
typedef struct student
{
    char name[20];
    int age;
} stud;
stud s1,s2;
```

From the above declarations, we conclude that typedef keyword reduces the length of the code and complexity of data types. It also helps in understanding the program.

Let's see another example where we typedef the structure declaration.



```
1 #include <stdio.h>
2 typedef struct student
3 {
4     char name[20];
5     int age;
6 }stud;
7 int main()
8 {
9     stud s1;
10    printf("Enter the details of student s1: ");
11    printf("\nEnter the name of the student:");
12    scanf("%s",&s1.name);
13    printf("\nEnter the age of student:");
14    scanf("%d",&s1.age);
15    printf("\n Name of the student is : %s", s1.name);
16    printf("\n Age of the student is : %d", s1.age);
17    return 0;
18 }
```

main.c:12:9: warning: format '%s' expects argument of type 'char *', but argument 2

12		scanf("%s",&s1.name);
		~^ ~~~~~
		char (*)[20]
		char *

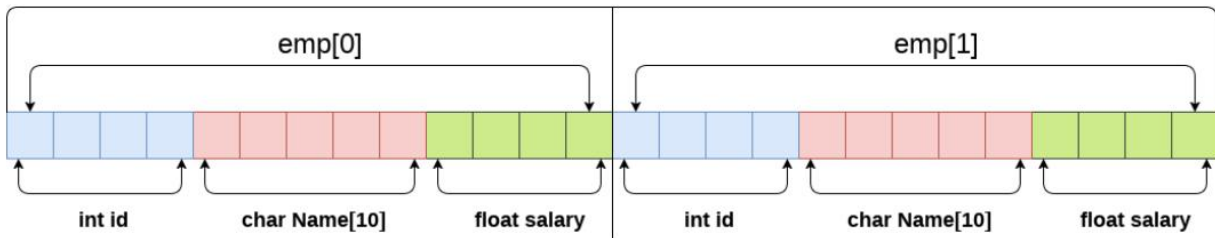
Enter the details of student s1:
Enter the name of the student:Ivanov

Enter the age of student:25

Name of the student is : Ivanov
Age of the student is : 25

III. C Array of Structures

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.



```

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`

LAB: Create a program that create an array of structures that stores information of 5 students and prints it.

```

main.c
1  #include<stdio.h>
2  #include <string.h>
3  struct student{
4  int rollno;
5  char name[10];
6  };
7  int main(){
8  int i;
9  struct student st[5];
10 printf("Enter Records of 5 students");
11 for(i=0;i<5;i++){
12 printf("\nEnter Rollno:");
13 scanf("%d",&st[i].rollno);
14 printf("\nEnter Name:");
15 scanf("%s",st[i].name);
16 }
17 printf("\nStudent Information List:");
18 for(i=0;i<5;i++){
19 printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
20 }
21 return 0;
22 }

```

Result/Output:

```
Enter Records of 5 students
Enter Rollno:1

Enter Name:Peter

Enter Rollno:2

Enter Name:Ivan

Enter Rollno:3

Enter Name:Zeki

Enter Rollno:4

Enter Name:Muhhamed

Enter Rollno:5

Enter Name:Alper

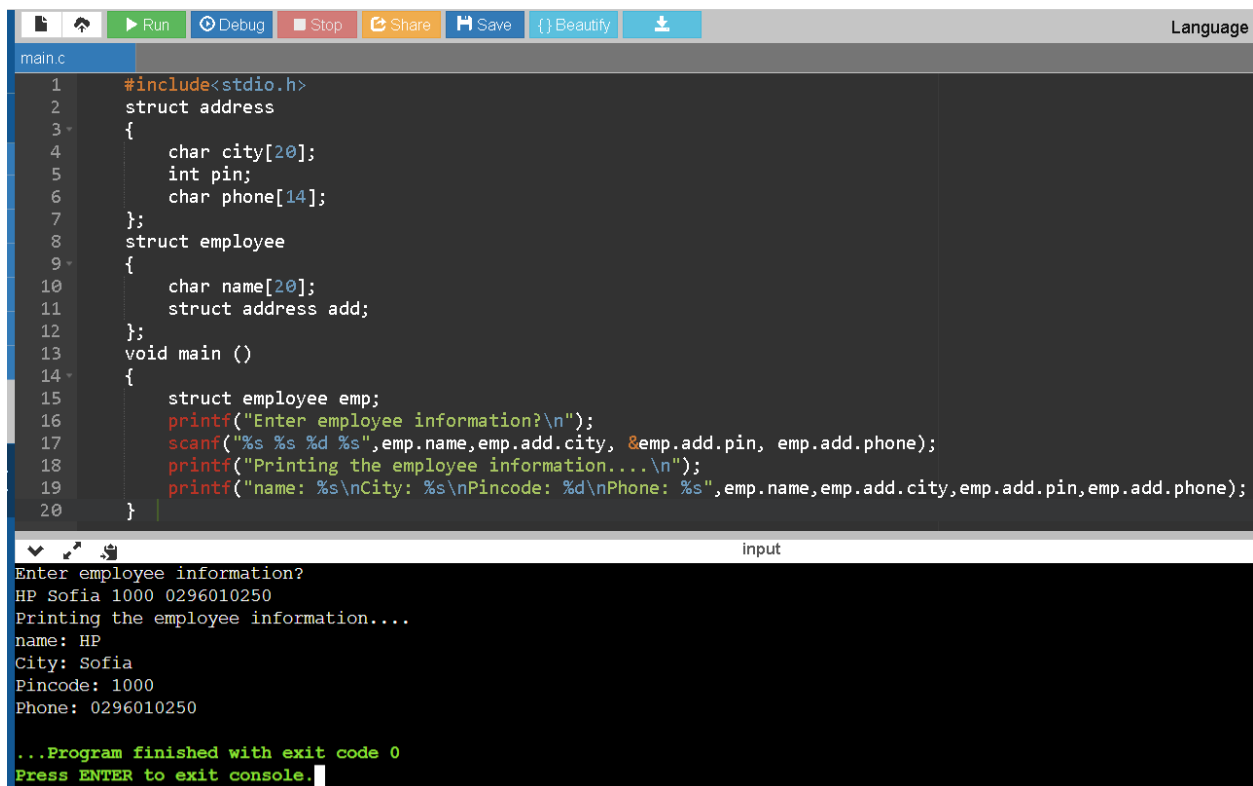
Student Information List:
Rollno:1, Name:Peter
Rollno:2, Name:Ivan
Rollno:3, Name:Zeki
Rollno:4, Name:Muhhamed
Rollno:5, Name:Alper

...Program finished with exit code 0
Press ENTER to exit console.
```

IV. C 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.

Example:



```
1 #include<stdio.h>
2 struct address
3 {
4     char city[20];
5     int pin;
6     char phone[14];
7 };
8 struct employee
9 {
10     char name[20];
11     struct address add;
12 };
13 void main ()
14 {
15     struct employee emp;
16     printf("Enter employee information?\n");
17     scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
18     printf("Printing the employee information...\n");
19     printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
20 }
```

input

```
Enter employee information?
HP Sofia 1000 0296010250
Printing the employee information....
name: HP
City: Sofia
Pincode: 1000
Phone: 0296010250
...Program finished with exit code 0
Press ENTER to exit console.
```

The structure can be nested in the following ways.

- By separate structure
- By Embedded structure

Separate structure

Here, we create two structures, but the dependent structure should be used inside the main structure as a member. Consider the following example.

```
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

The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but it can not be used in multiple data structures. Consider the following example.

```

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

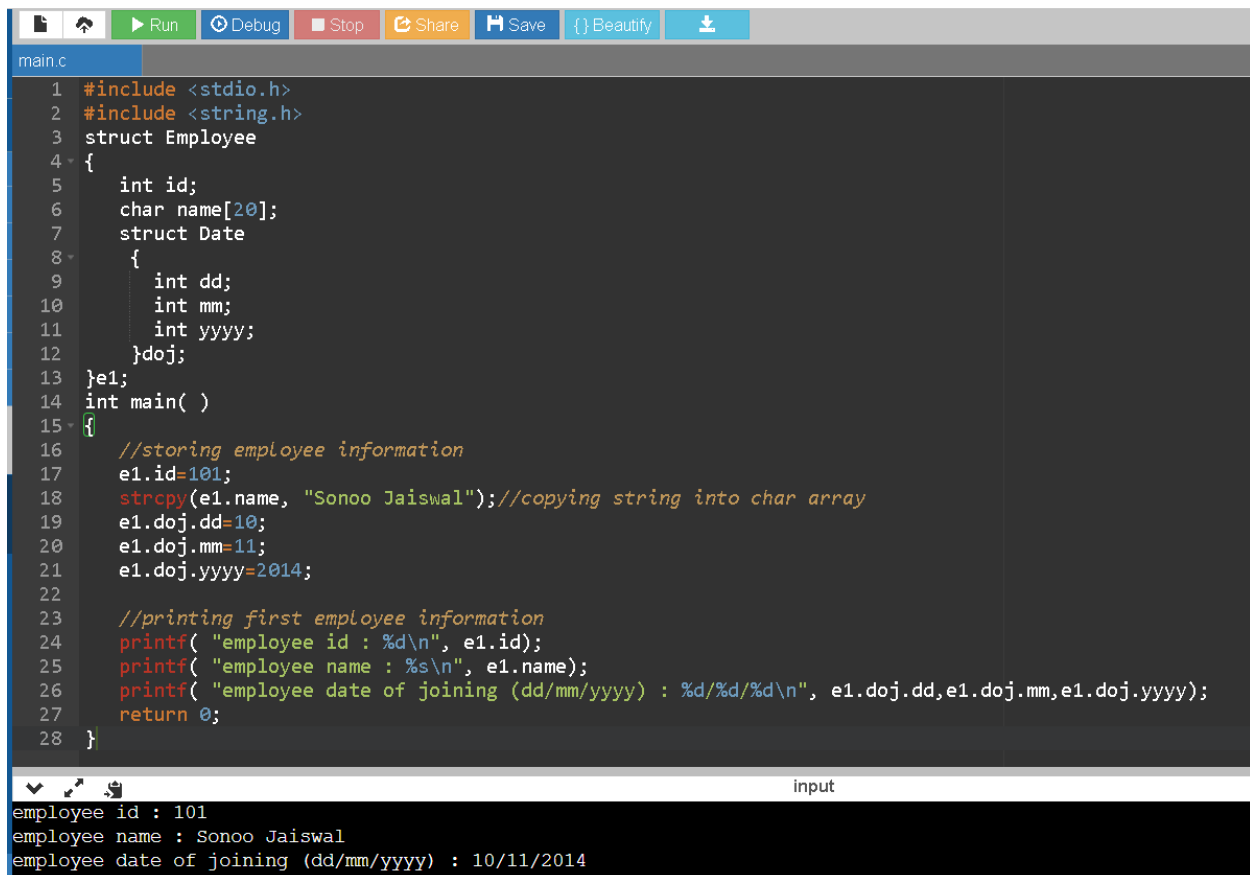
```

Accessing Nested Structure

We can access the member of the nested structure by Outer_Structure.Nested_Structure.member as given below:

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

LAB: Nested Structure example:



```
main.c
1 #include <stdio.h>
2 #include <string.h>
3 struct Employee
4 {
5     int id;
6     char name[20];
7     struct Date
8     {
9         int dd;
10        int mm;
11        int yyyy;
12    }doj;
13 }e1;
14 int main( )
15 {
16     //storing employee information
17     e1.id=101;
18     strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
19     e1.doj.dd=10;
20     e1.doj.mm=11;
21     e1.doj.yyyy=2014;
22
23     //printing first employee information
24     printf( "employee id : %d\n", e1.id);
25     printf( "employee name : %s\n", e1.name);
26     printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.mm,e1.doj.yyyy);
27     return 0;
28 }
```

input

```
employee id : 101
employee name : Sonoo Jaiswal
employee date of joining (dd/mm/yyyy) : 10/11/2014
```

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. Consider the following example to pass the structure variable employee to a function display() which is used to display the details of an employee.

```
1  #include<stdio.h>
2  struct address
3  {
4      char city[20];
5      int pin;
6      char phone[14];
7  };
8  struct employee
9  {
10     char name[20];
11     struct address add;
12 };
13 void display(struct employee);
14 void main ()
15 {
16     struct employee emp;
17     printf("Enter employee information?\n");
18     scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
19     display(emp);
20 }
21 void display(struct employee emp)
22 {
23     printf("Printing the details....\n");
24     printf("%s %s %d %s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
25 }
```

input

Enter employee information?
Petrov Varna 5000 0888123123
Printing the details....
Petrov Varna 5000 0888123123

...Program finished with exit code 0
Press ENTER to exit console.