SNS COLLEGE OF TECHNOLOGY

Coimbatore-36.

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

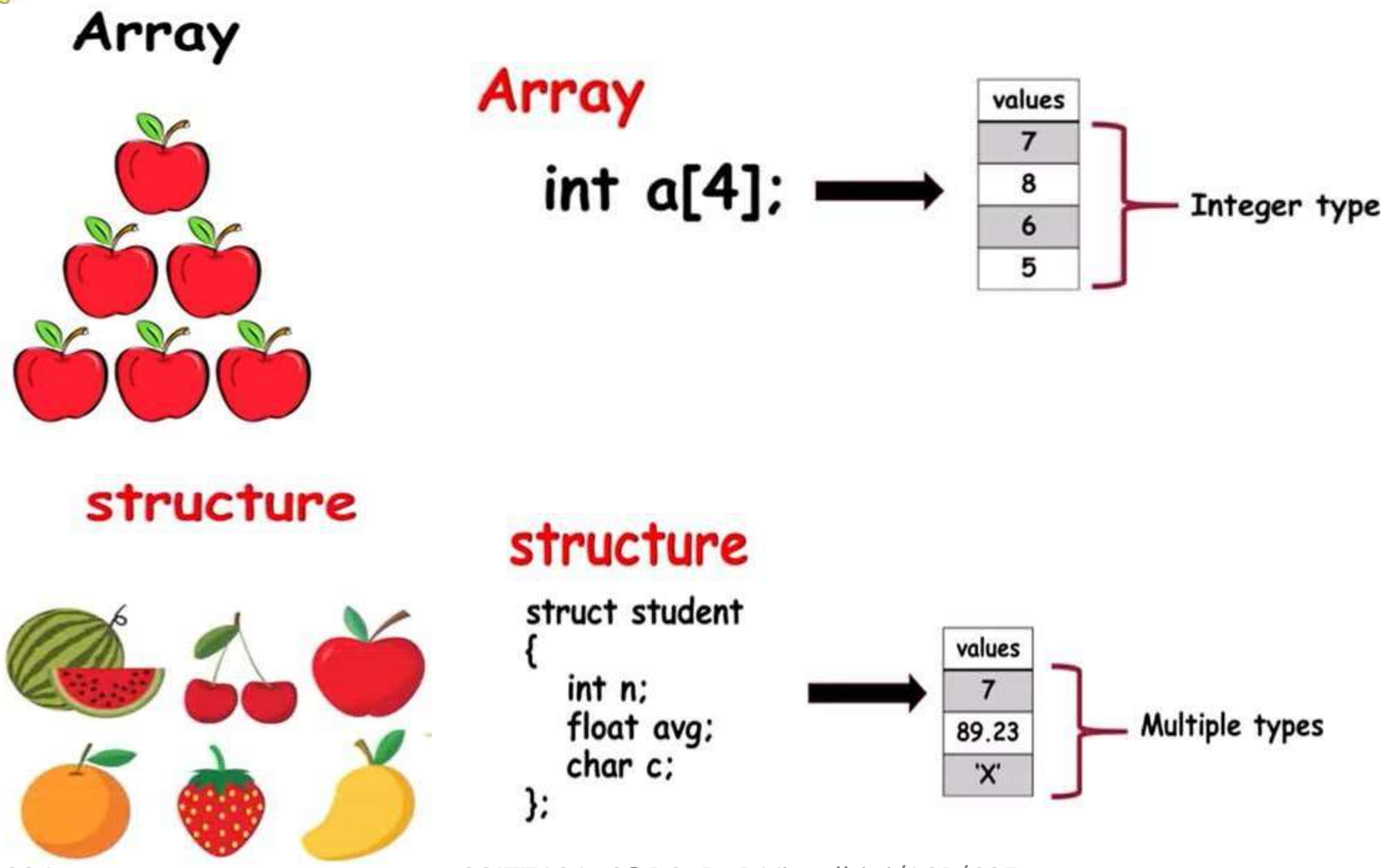
COURSE NAME : 23ITT101 PROGRAMMING IN C AND DATA STRUCTURES

I YEAR/ II SEMESTER

UNIT – V STRUCTURES UNIONS AND FILES

Dr.B.Vinodhini
Associate Professor
Department of Computer Science and Engineering

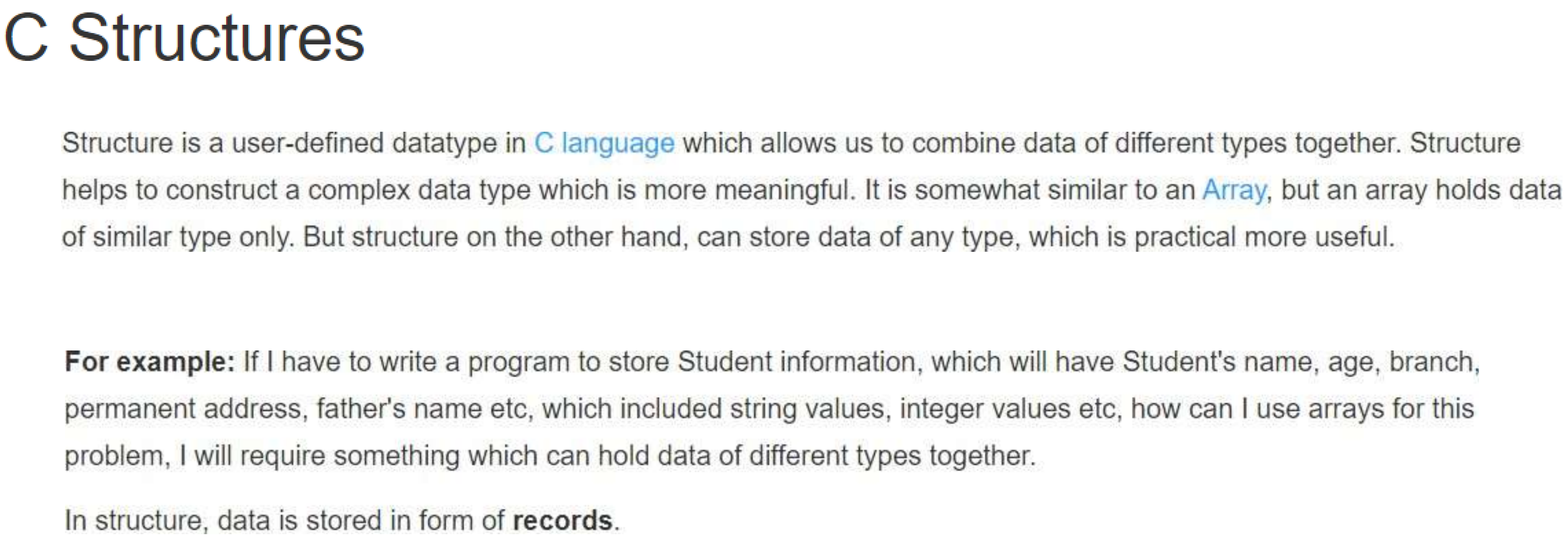
STRUCTURES



```
int a[4];
```

```
struct student
{
    int n;
    float avg;
    char c;
};
```

STRUCTURES



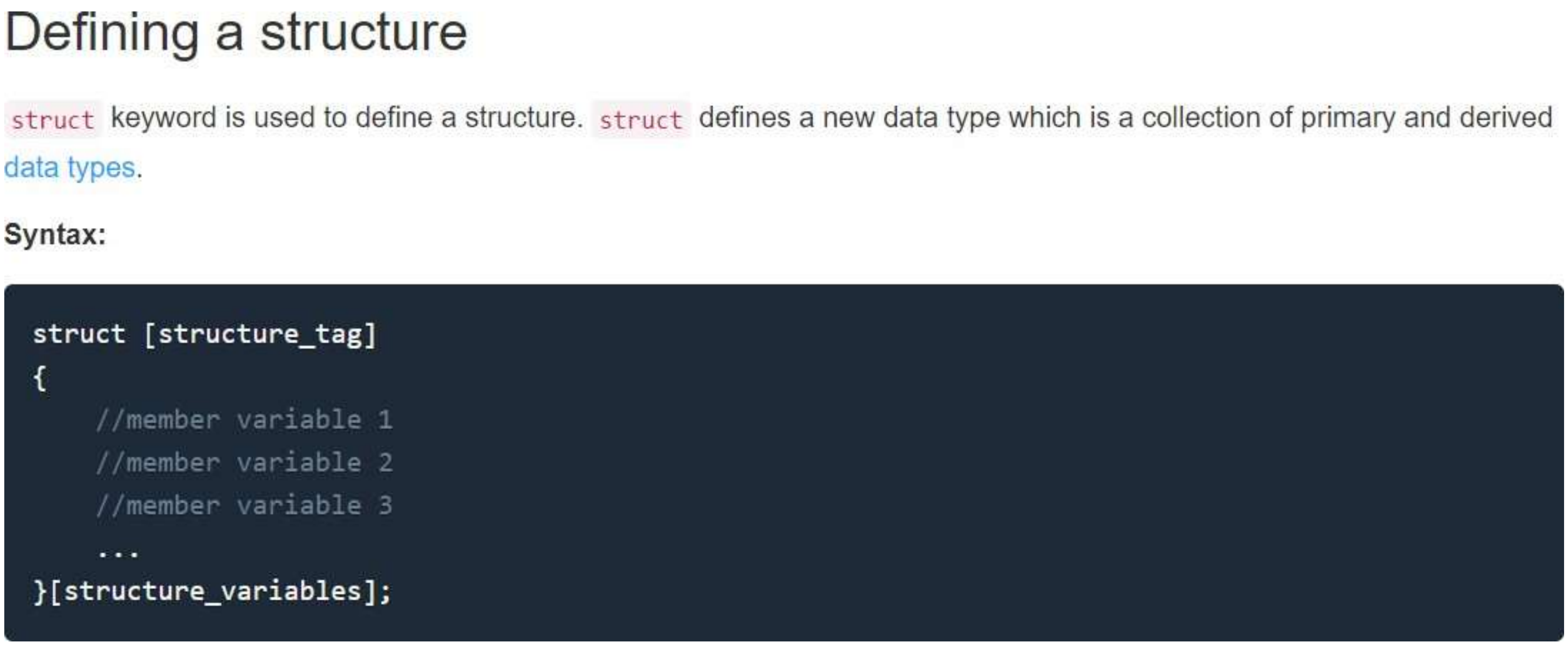
C Structures

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

For example: If I have to write a program to store Student information, which will have Student's name, age, branch, permanent address, father's name etc, which included string values, integer values etc, how can I use arrays for this problem, I will require something which can hold data of different types together.

In structure, data is stored in form of **records**.

STRUCTURES



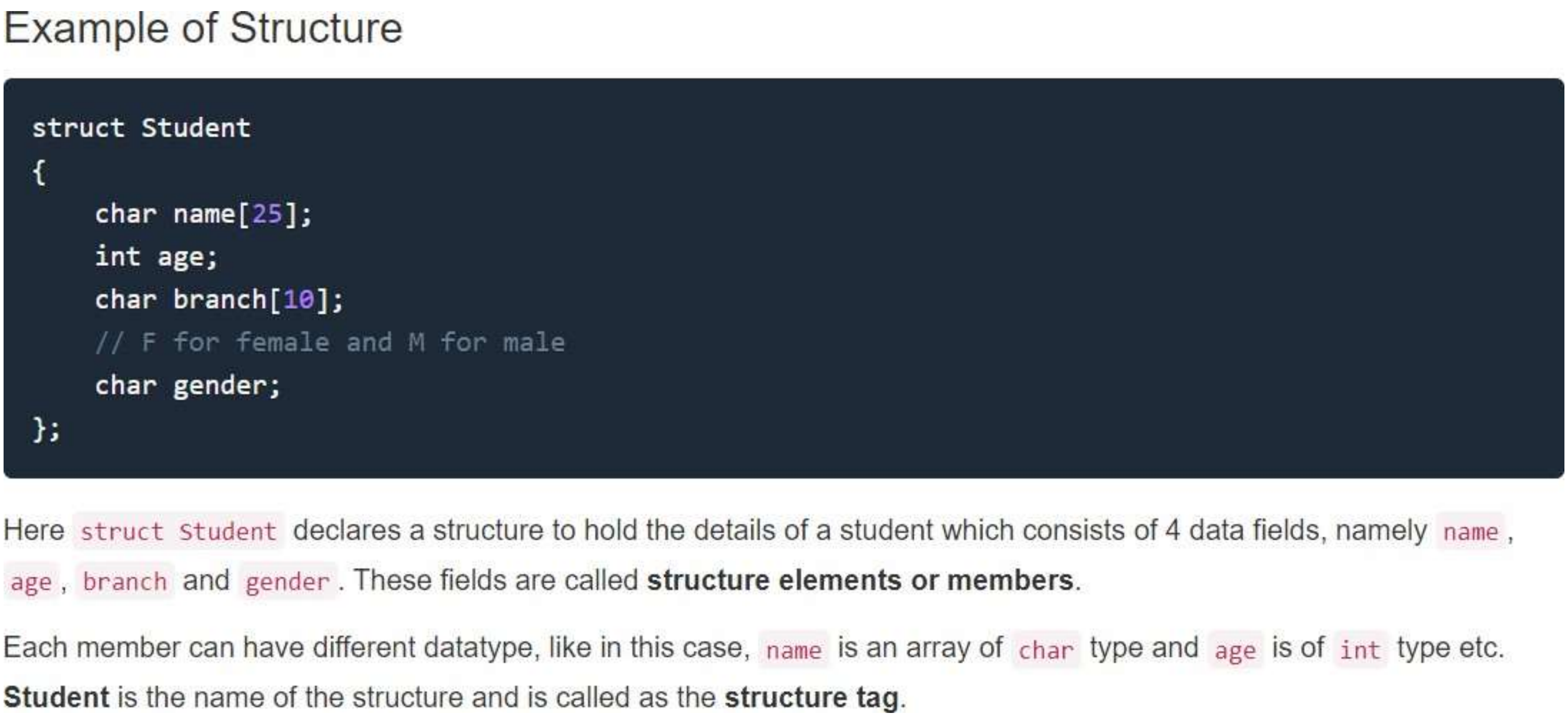
Defining a structure

struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived data types.

Syntax:

```
struct [structure_tag]
{
    //member variable 1
    //member variable 2
    //member variable 3
    ...
}[structure_variables];
```

STRUCTURES



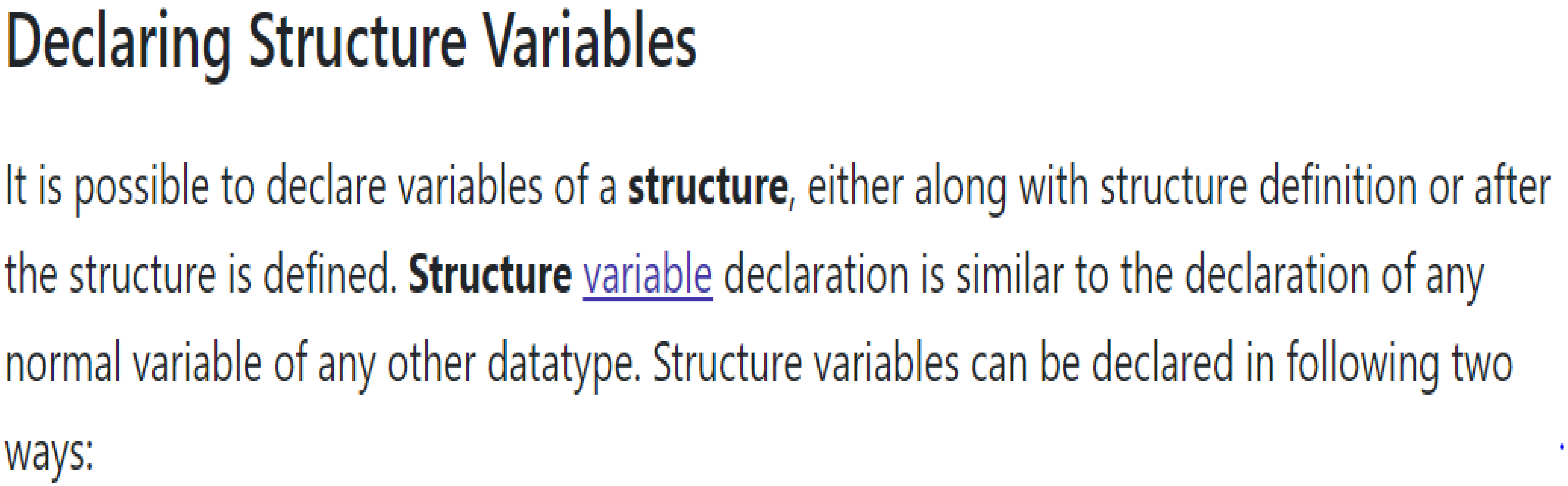
Example of Structure

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    // F for female and M for male
    char gender;
};
```

Here `struct Student` declares a structure to hold the details of a student which consists of 4 data fields, namely `name`, `age`, `branch` and `gender`. These fields are called **structure elements or members**.

Each member can have different datatype, like in this case, `name` is an array of `char` type and `age` is of `int` type etc. **Student** is the name of the structure and is called as the **structure tag**.

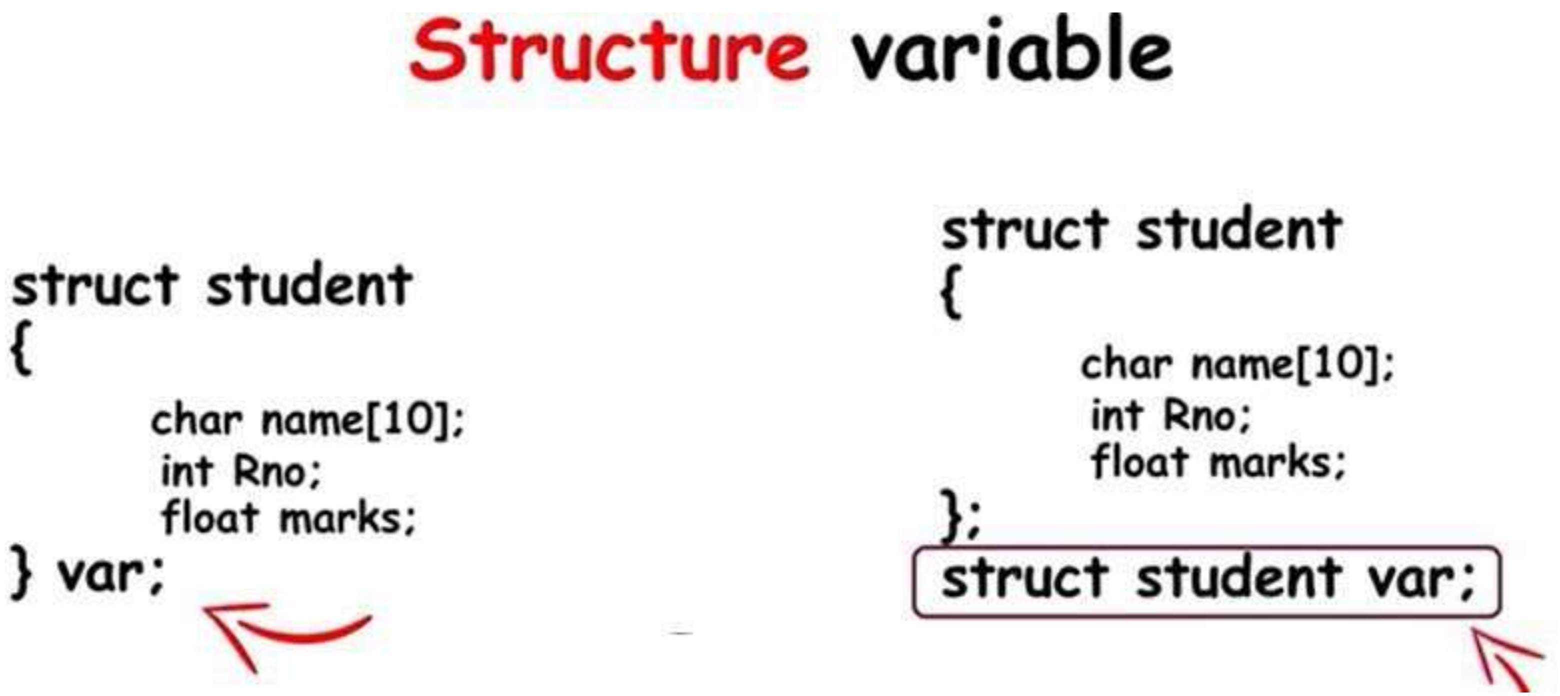
STRUCTURES



Declaring Structure Variables

It is possible to declare variables of a **structure**, either along with structure definition or after the structure is defined. **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype. Structure variables can be declared in following two ways:

STRUCTURES

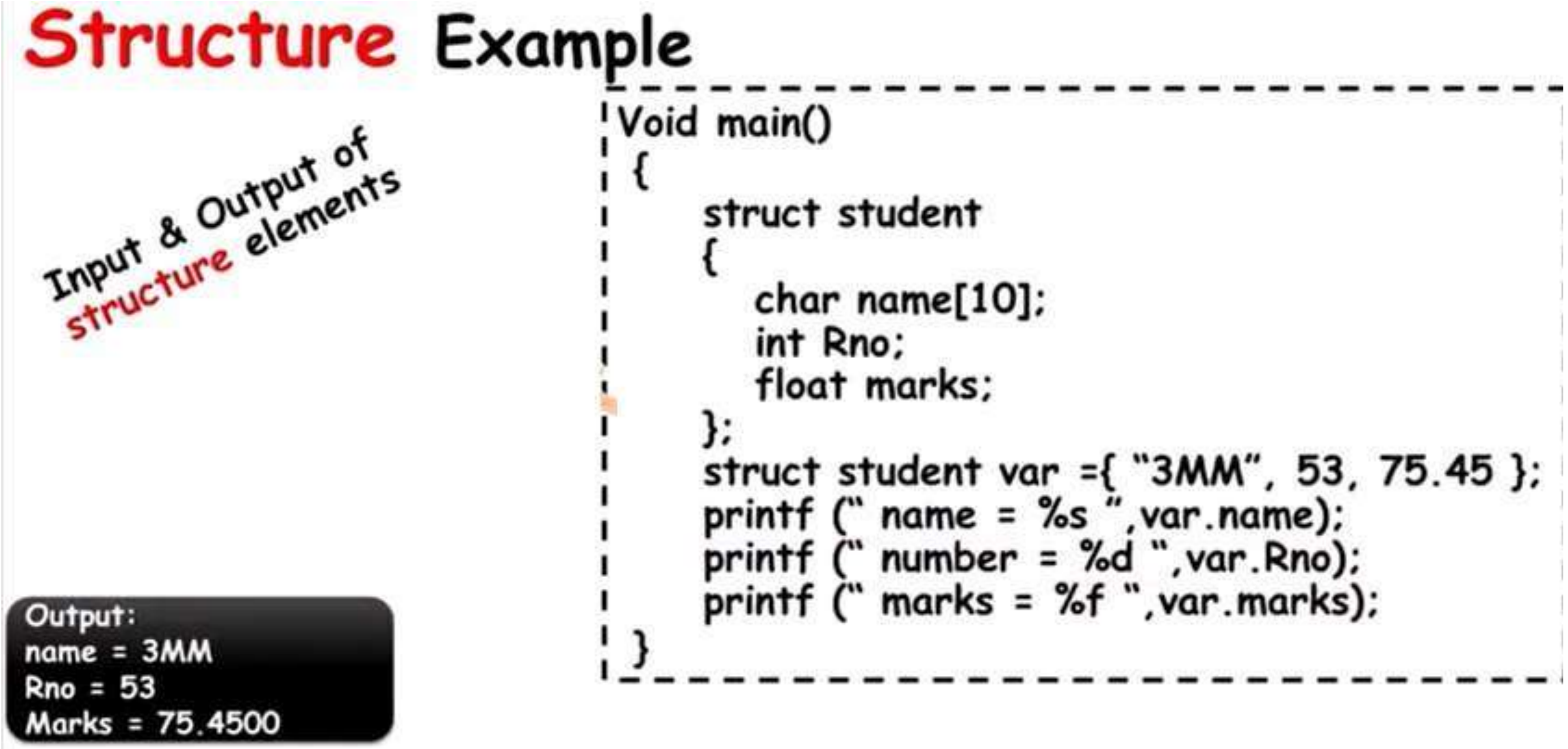


Structure variable

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

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

STRUCTURES



Structure Example

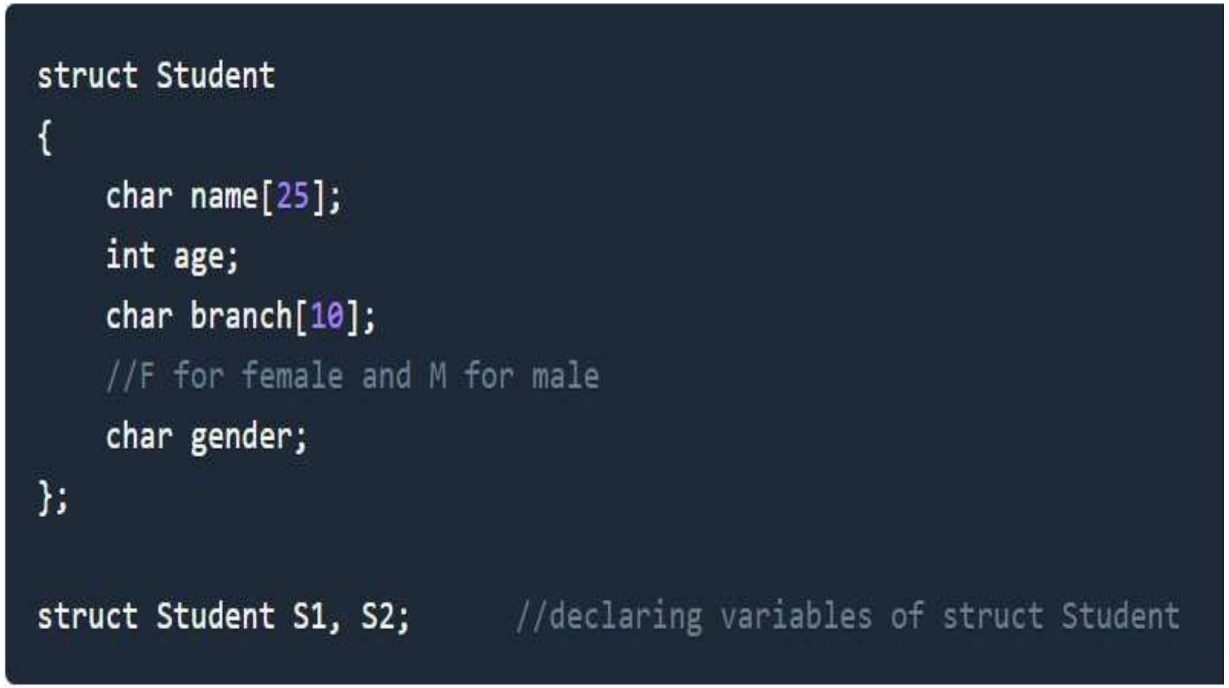
Input & Output of structure elements

```
Void main()
 {
      struct student
      {
         char name[10];
         int Rno;
         float marks;
      };
      struct student var ={ “3MM”, 53, 75.45 };
      printf (“ name = %s ”,var.name);
      printf (“ number = %d ”,var.Rno);
      printf (“ marks = %f ”,var.marks);
 }
```

```
Output:
name = 3MM
Rno = 53
Marks = 75.4500
```

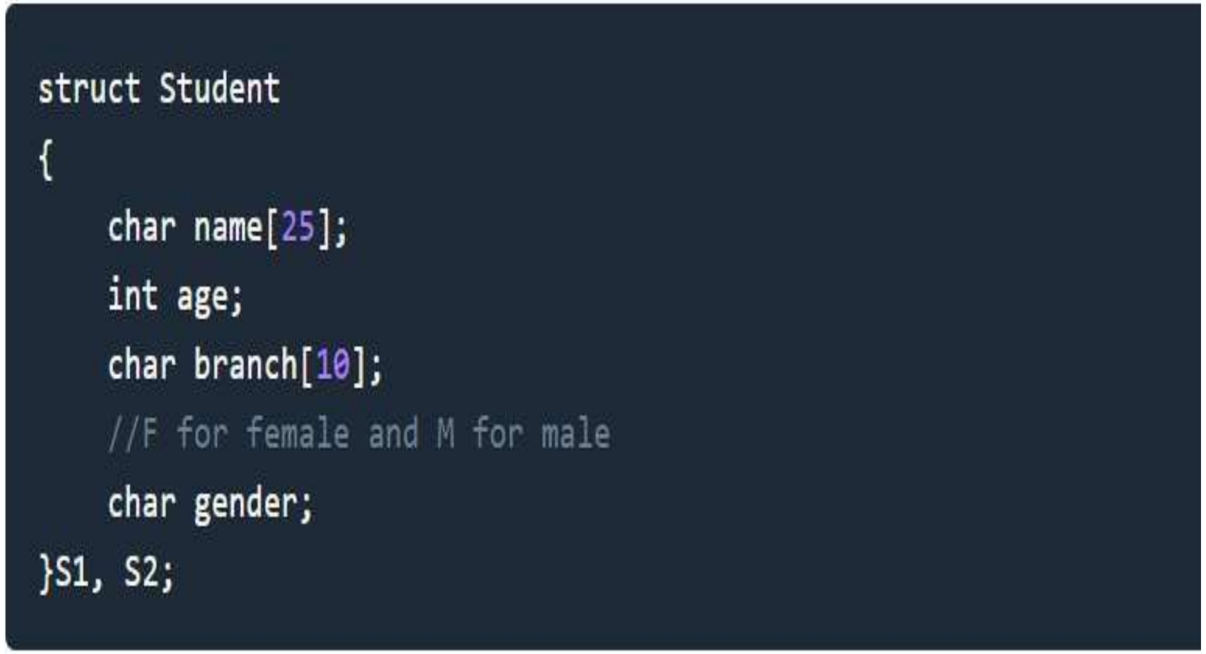
STRUCTURES

1) Declaring Structure variables separately



```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};

struct Student S1, S2;      //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

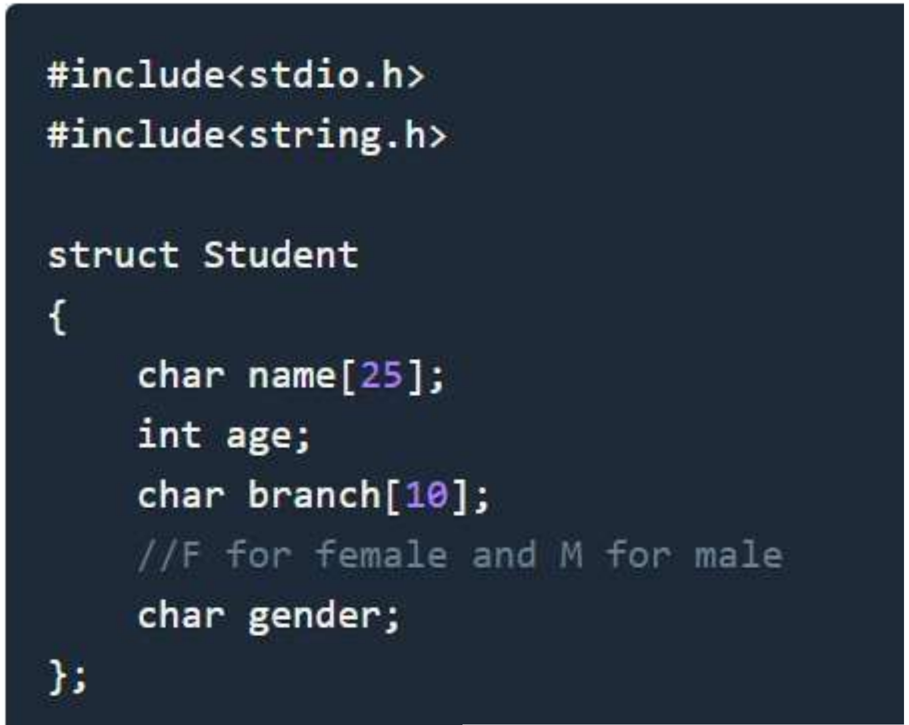


```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
}S1, S2;
```

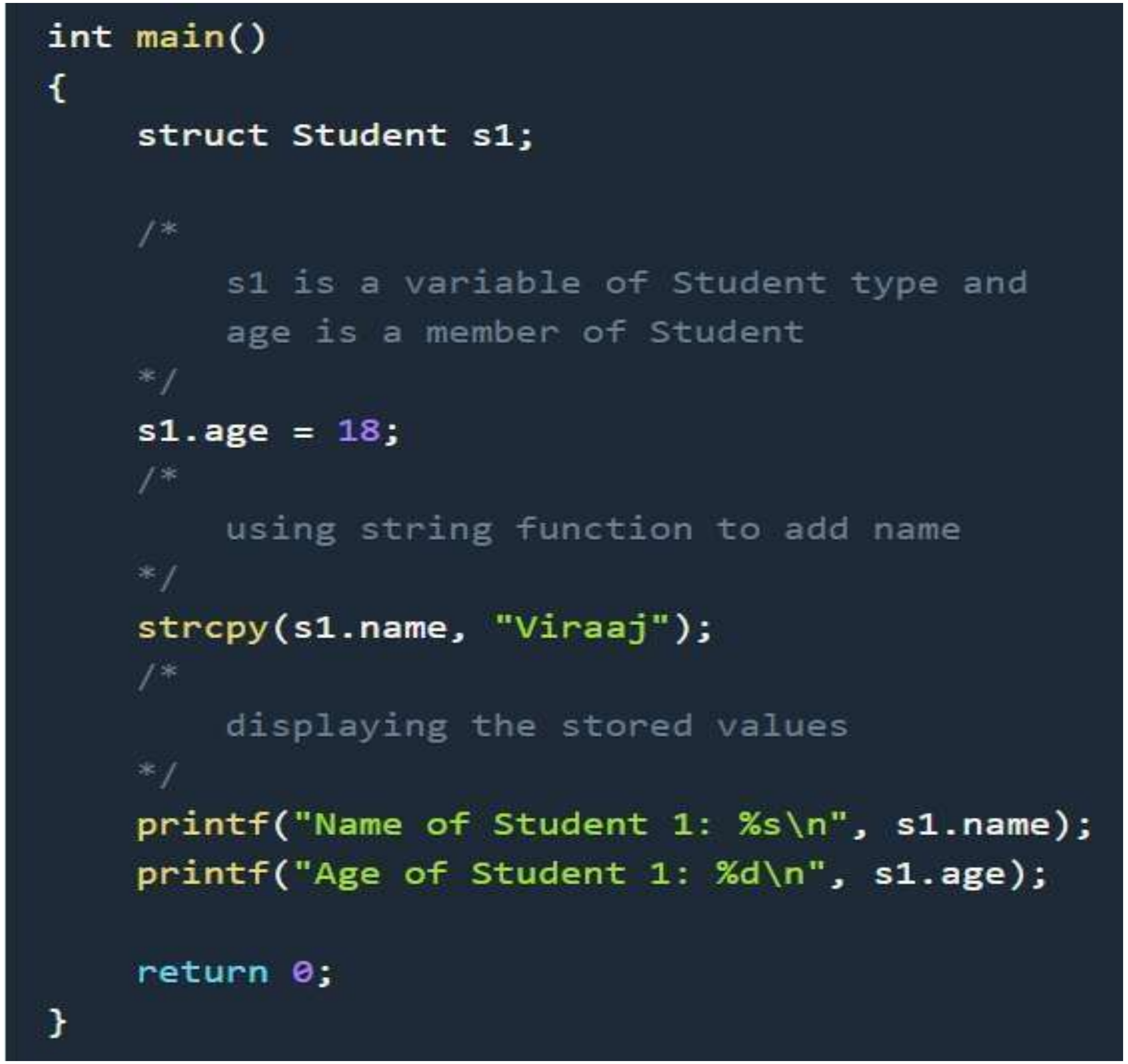
Here `S1` and `S2` are variables of structure `Student`. However this approach is not much recommended.

STRUCTURES

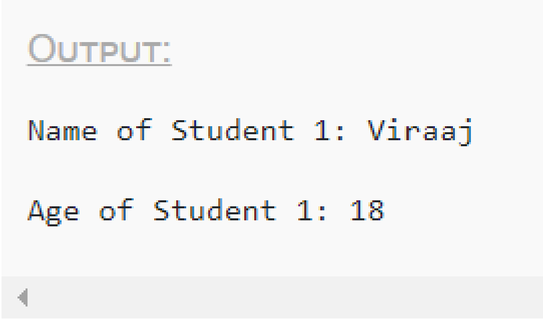
For example:



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

struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};
```



```c
int main()
{
    struct Student s1;

    /*
        s1 is a variable of Student type and
        age is a member of Student
    */
    s1.age = 18;
    /*
        using string function to add name
    */
    strcpy(s1.name, "Viraaj");
    /*
        displaying the stored values
    */
    printf("Name of Student 1: %s\n", s1.name);
    printf("Age of Student 1: %d\n", s1.age);

    return 0;
}
```



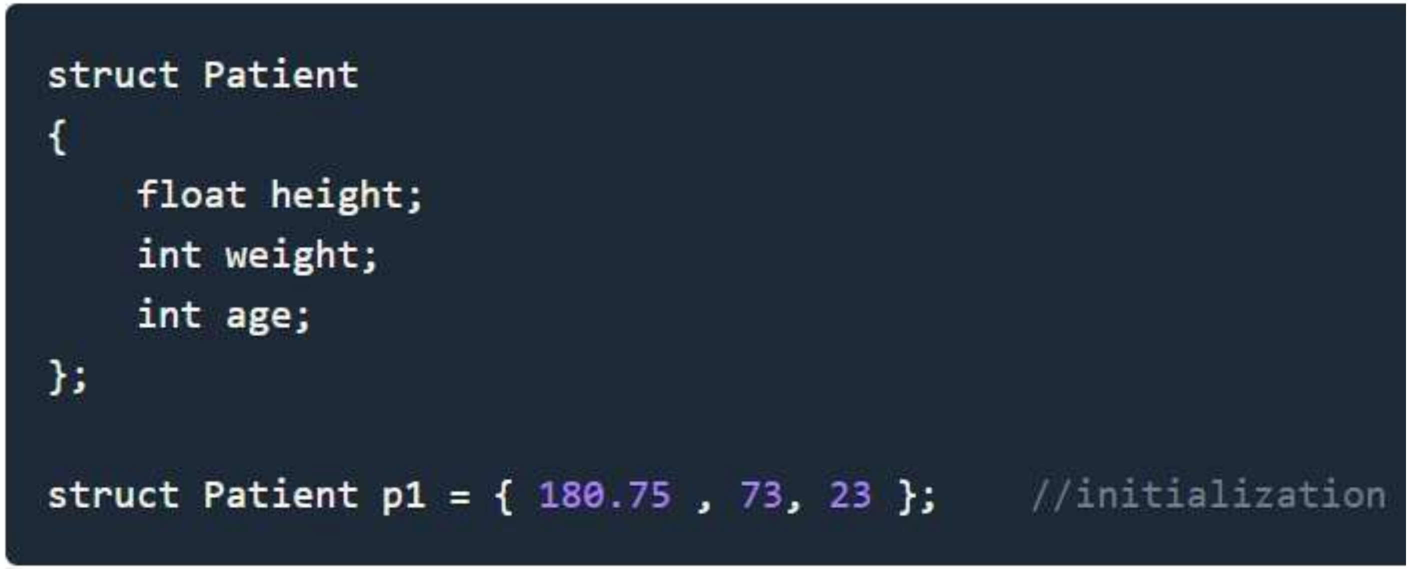
OUTPUT:

```
Name of Student 1: Viraaj

Age of Student 1: 18
```

STRUCTURES

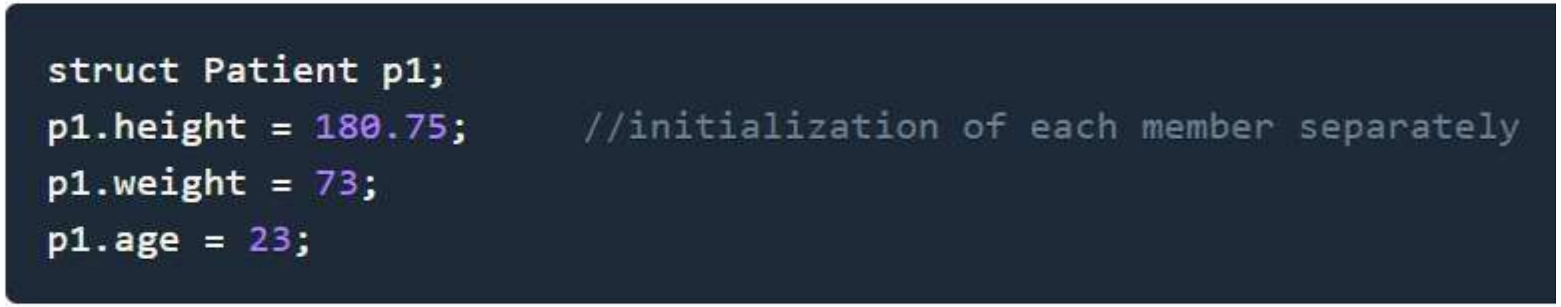
Structure Initialization

Like a variable of any other datatype, structure variable can also be initialized at compile time.



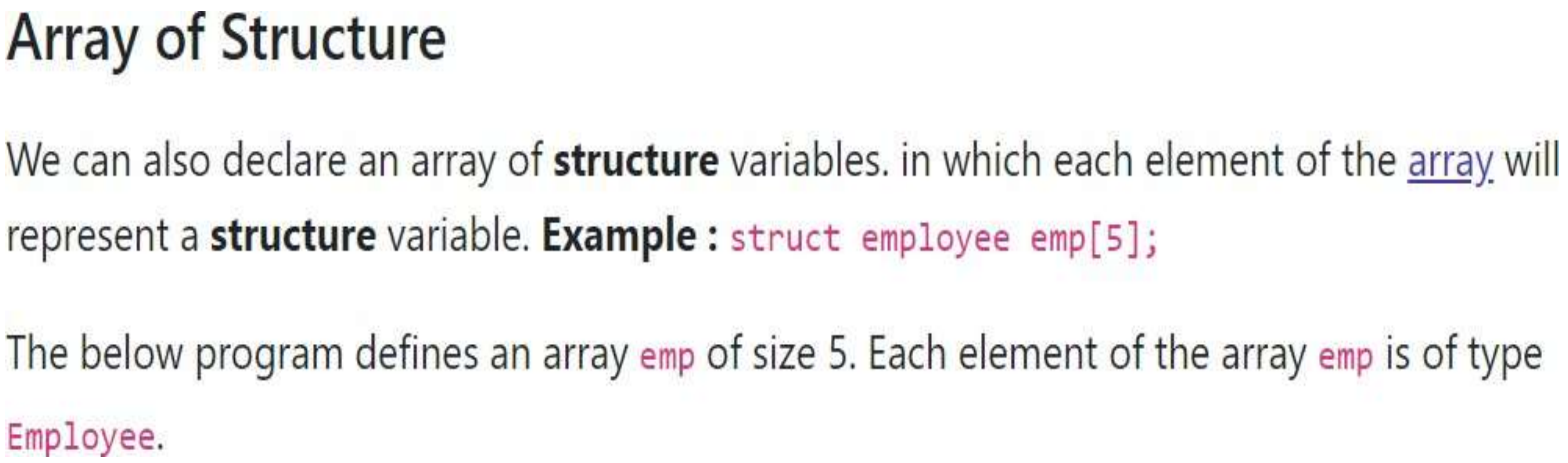
```
struct Patient
{
    float height;
    int weight;
    int age;
};

struct Patient p1 = { 180.75 , 73, 23 };    //initialization
```

or



```
struct Patient p1;
p1.height = 180.75;     //initialization of each member separately
p1.weight = 73;
p1.age = 23;
```

STRUCTURES

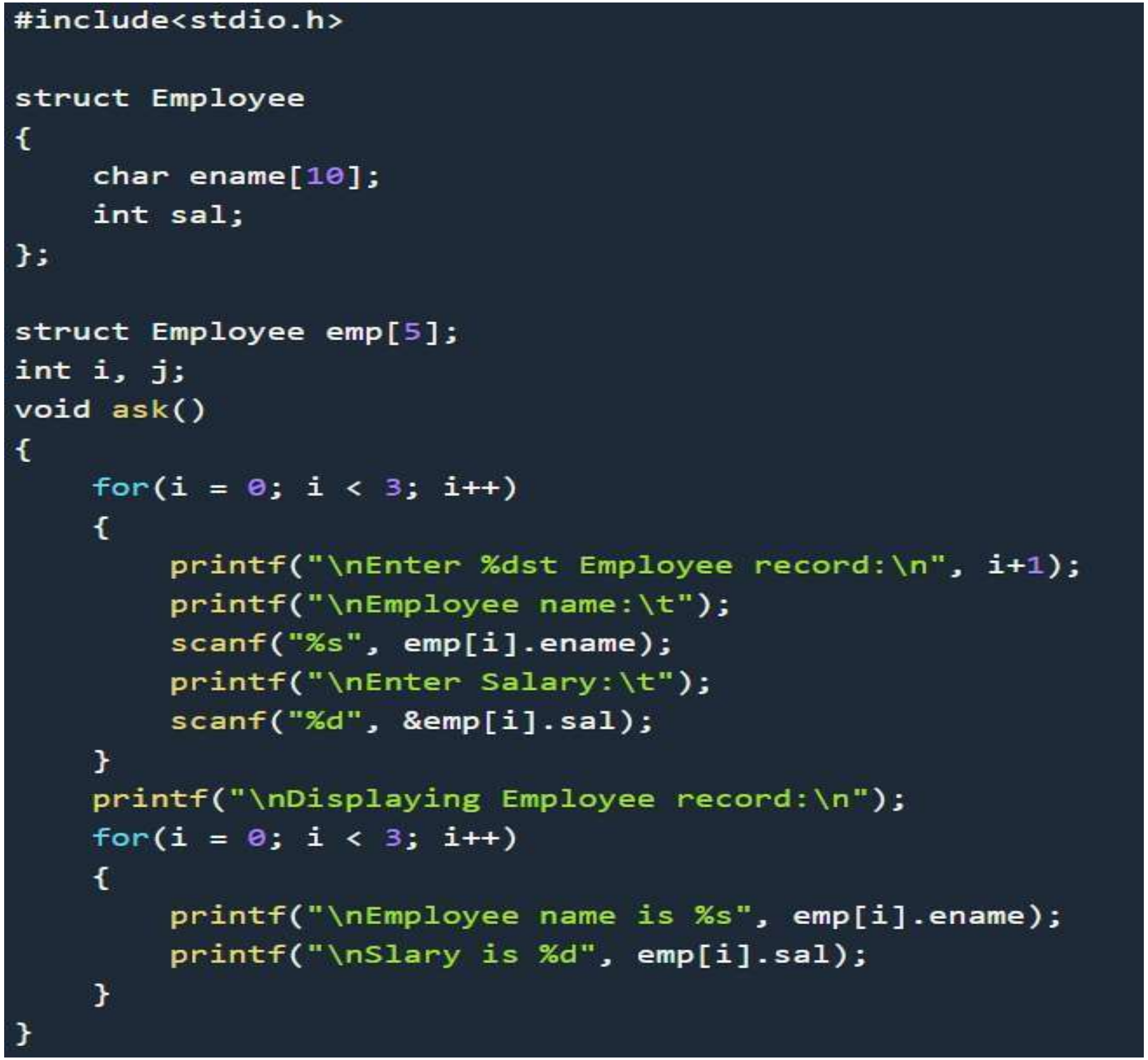


Array of Structure

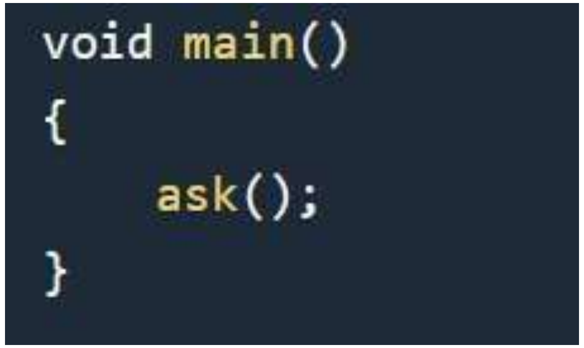
We can also declare an array of **structure** variables. in which each element of the array will represent a **structure** variable. **Example :** `struct employee emp[5];`

The below program defines an array `emp` of size 5. Each element of the array `emp` is of type `Employee`.

STRUCTURES



```c
#include<stdio.h>

struct Employee
{
    char ename[10];
    int sal;
};

struct Employee emp[5];
int i, j;
void ask()
{
    for(i = 0; i < 3; i++)
    {
        printf("\nEnter %dst Employee record:\n", i+1);
        printf("\nEmployee name:\t");
        scanf("%s", emp[i].ename);
        printf("\nEnter Salary:\t");
        scanf("%d", &emp[i].sal);
    }
    printf("\nDisplaying Employee record:\n");
    for(i = 0; i < 3; i++)
    {
        printf("\nEmployee name is %s", emp[i].ename);
        printf("\nSlary is %d", emp[i].sal);
    }
}
```



```c
void main()
{
    ask();
}
```

Thank
you