The struct data type is used when we want to put together data of different types in one capsule. Obviously, we cannot use arrays because all members of the array have to be of the same type. A struct variable is therefore a collection of members (data) of different types. Before you declare a struct variable, you must first define the struct type of that variable:

For example:

```c
// Definition of struct employee
struct employee
{
    string id;
    string name;
    char gender;
    int numDepend;
    money rate;
    money totWages;
};
```

In this case, any variable of type employee can have data members: id, name, gender, numDepend, rate, and totWages.

After that, you can declare the struct variables engineer and assistant to be of that type:

```c
employee engineer, assistant;
```

To access the members of the struct variable, we put a “period” between the variable name and the member name, e.g.,

```c
engineer.name engineer.id assistant.gender etc.
```

**Question:** Can an array be a member of a struct?

**Answer:** Yes. Here is an example:
// Arrays can be members of structs, e.g.
struct studentRecord {
    string name;
    string id;
    float grade[10];
    float GPA;
};

studentRecord student;

student.grade[3] = 3.6; cin >> student.name;
for( i = 0; i < 10; i++) cin >> student.grade[i];

---

Question: Can a member of a struct be itself a smaller struct?
Answer: Yes. Example:

// A member of a struct may itself be a struct, e.g.
struct nameType {
    string first;
    string middle;
    string last;
};

struct personInfo {
    nameType name;
    string address;
    string phone;
};

personInfo person;
cin >> person.address;
cin >> person.phone;
cin >> person.name.first;  // Here we are using two dots
cin >> person.name.last;   // Here we are using two dots

---

Question: Can we build arrays of structs?
Answer: Yes. In this case each element of the array is itself a struct, e.g.,

// A struct may be an element of an array of structs.
personInfo staff[100];
cout << staff[i].address;
cout << staff[i].name.first;
for( i = 0; i < N; i++) {
    cout << staff[i].phone;
    cout << staff[i].name.family;
}
Question: Can we copy (assign) a struct variable to another struct variable?
Answer: Yes, but only when both are of the same struct type, e.g.,

```c
personInfo person;
person = staff[6];
```

Struct variables enjoy the same properties of other variables. So, they may be passed as function parameters by value or by reference. Even a typed function may return a struct value.

**Example (1):**
Declare a struct type `item` for an item with weight and total price, both are of type `float`.
Write a function `maxval (x,y,z)` to receive two items (x) and (y) and return the more valuable item (z), i.e. the item with the higher price per weight ratio.

**Solution:**

```c
struct item
{float weight, price; }

void maxval (item x, item y, item & z)
{
    if ((x.price / x.weight) >= (y.price / y.weight)) z = x else z = y;
}
```

Or, we can return the struct directly:

```c
item maxval (item x, item y)
{
    if ((x.price / x.weight) >= (y.price / y.weight)) return x else return y;
}
```

**Example (2):**
A color image is stored as a 2-D array `A[][]` of pixels, with N rows and M columns. Each pixel `A[i][j]` is represented as a `struct` with three integer values representing its color components (Red, Green, Blue). To convert the pixel color to grey-level, we take the average of the three color components (rounded to the nearest integer) as the “brightness”. Write a function to receive such array and return another array `B[][]` containing the “brightness” of each pixel in the image. Consider N and M to be global constants.
Solution

const int N = ...; const int M = ...;
struct pixel {int red, green, blue; };
void Brightness (pixel A[][M], int B[][M])
{
    int i, j, b;   pixel p;
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
        {
            p = A[i][j];
            b = (p.red + p.green + p.blue)/3.0 + 0.5;
            // or b = floor((p.red + p.green + p.blue)/3.0 + 0.5);
            B[i][j] = b;
        }
}