

Structure and Union Types

Chapter 10

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To learn how to declare a **struct** data type which consists of several data fields, each with its own name and data type
- To understand how to use a **struct** to store data for a structured object or record
- To learn how to use dot notation to process individual fields of a structured object
- To learn how to use **structs** as function parameters and to return function results
- To understand the relationship between parallel arrays and arrays of structured objects

User-Defined Structure Types

- record
 - a collection of information about one data object
- structure type
 - a data type for a record composed of multiple components
- hierarchical structure
 - a structure containing components that are structures

User-Defined Structure Types

Name: Jupiter

Diameter: 142,800 km

Moons: 16

Orbit time: 11.9 years

Rotation time: 9.925 hours

```
#define STRSIZ 10

typedef struct {
    char    name[STRSIZ];
    double  diameter;           /* equatorial diameter in km    */
    int     moons;             /* number of moons              */
    double  orbit_time,        /* years to orbit sun once      */
           rotation_time;     /* hours to complete one        */
                                   revolution on axis            */
} planet_t;
```

User-Defined Structure Types

Another syntax:

```
struct Planet {  
    char name[70];  
    double diameter;  
    int moons;  
    double orbit_time,  
           rotation_time;  
};  
// in a function  
struct Planet p1, p2;
```

User-Defined Structure Types

Name: Jupiter

Diameter: 142,800 km

Moons: 16

Orbit time: 11.9 years

Rotation time: 9.925 hours

I will always use this syntax

```
#define STRSIZ 10

typedef struct {
    char    name[STRSIZ];
    double  diameter;           /* equatorial diameter in km    */
    int     moons;             /* number of moons              */
    double  orbit_time,       /* years to orbit sun once      */
           rotation_time;    /* hours to complete one
                               revolution on axis            */
} planet_t;
```

Individual Components of a Structured Data Object

- direct component selection operator
 - a period placed between a structure type variable and a component name to create a reference to the component

```
planet_t p1;  
p1.moons = 10;  
printf("p1 has %d moons\n", p1.moons);
```

```
strcpy(current_planet.name, "Jupiter");  
current_planet.diameter = 142800;  
current_planet.moons = 16;  
current_planet.orbit_time = 11.9;  
current_planet.rotation_time = 9.925;
```

Variable `current_planet`, a structure of type `planet_t`

<code>.name</code>	J u p i t e r \ 0 ? ?
<code>.diameter</code>	142800.0
<code>.moons</code>	16
<code>.orbit_time</code>	11.9
<code>.rotation_time</code>	9.925

Structure Data Type as Input and Output Parameters

- When a structured variable is passed as an input argument to a function, **all of its component values are copied** into the components of the function's corresponding formal parameter.

Structure Data Type as Input and Output Parameters

- When such a variable is used as an output argument, the address-of operator must be applied in the same way that we would pass output arguments of the standard types `char`, `int`, and `double`.

FIGURE 10.2 Function with a Structured Input Parameter

```
1. /*
2.  * Displays with labels all components of a planet_t structure
3.  */
4. void
5. print_planet(planet_t pl) /* input - one planet structure */
6. {
7.     printf("%s\n", pl.name);
8.     printf("  Equatorial diameter: %.0f km\n", pl.diameter);
9.     printf("  Number of moons: %d\n", pl.moons);
10.    printf("  Time to complete one orbit of the sun: %.2f years\n",
11.           pl.orbit_time);
12.    printf("  Time to complete one rotation on axis: %.4f hours\n",
13.           pl.rotation_time);
14. }
```

FIGURE 10.3 Function Comparing Two Structured Values for Equality

```
1. #include <string.h>
2.
3. /*
4.  * Determines whether or not the components of planet_1 and planet_2 match
5.  */
6. int
7. planet_equal(planet_t planet_1, /* input - planets to          */
8.             planet_t planet_2) /*          compare          */
9. {
```

(continued)

FIGURE 10.3 (continued)

```
10.     return (strcmp(planet_1.name, planet_2.name) == 0    &&
11.            planet_1.diameter == planet_2.diameter      &&
12.            planet_1.moons == planet_2.moons            &&
13.            planet_1.orbit_time == planet_2.orbit_time  &&
14.            planet_1.rotation_time == planet_2.rotation_time);
15. }
```

Structure Data Type as Input and Output Parameters

- indirect component selection operator
 - the character sequence `->` placed between a pointer variable and a component name creates a reference that follows the pointer to a structure and selects the component

FIGURE 10.4 Function with a Structured Output Argument

```
1. /*
2.  * Fills a type planet_t structure with input data. Integer returned as
3.  * function result is success/failure/EOF indicator.
4.  *     1 => successful input of one planet
5.  *     0 => error encountered
6.  *     EOF => insufficient data before end of file
7.  * In case of error or EOF, value of type planet_t output argument is
8.  * undefined.
9.  */
10. int
11. scan_planet(planet_t *plnp) /* output - address of planet_t structure
12.                             to fill                                     */
13. {
14.     int result;
15.
16.     result = scanf("%s%lf%d%lf%lf", (*plnp).name,
17.                   &(*plnp).diameter,
18.                   &(*plnp).moons,
19.                   &(*plnp).orbit_time,
20.                   &(*plnp).rotation_time);
21.
22.     if (result == 5)
23.         result = 1;
24.     else if (result != EOF)
25.         result = 0;
26.
27.     return (result);
28. }
```

FIGURE 10.5

Data Areas of main and scan_planet
During Execution
Of `status = scan_planet(¤t_planet);`

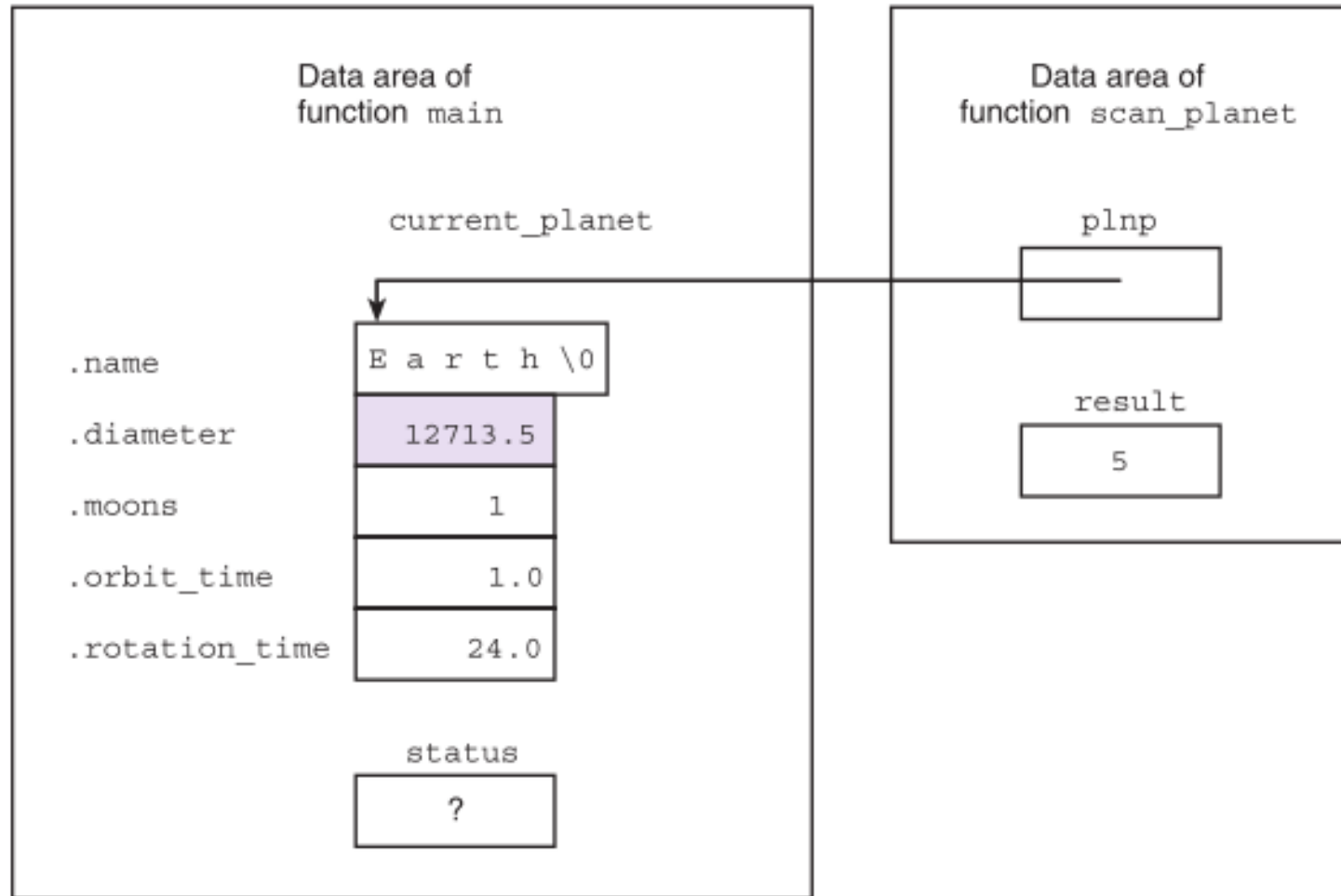


TABLE 10.2 Step-by-Step Analysis of Reference `&>(*plnp).diameter`

Reference	Type	Value
<code>plnp</code>	<code>planet_t *</code>	address of structure that <code>main</code> refers to as <code>current_planet</code>
<code>*plnp</code>	<code>planet_t</code>	structure that <code>main</code> refers to as <code>current_planet</code>
<code>(*plnp).diameter</code>	<code>double</code>	<code>12713.5</code>
<code>&>(*plnp).diameter</code>	<code>double *</code>	address of colored component of structure that <code>main</code> refers to as <code>current_planet</code>

Functions Whose Result Values are Structured

- A function that computes a structured result can be modeled on a function computing a simple result.
- A local variable of the structure type can be allocated, fill with the desired data, and returned as the function result.

Functions Whose Result Values are Structured

- The function does not return the *address* of the structure as it would with an array result.
- Rather, it returns the *values* of all components.

FIGURE 10.6 Function `get_planet` Returning a Structured Result Type

```
1.  /*
2.   * Gets and returns a planet_t structure
3.   */
4.  planet_t
5.  get_planet(void)
6.  {
7.      planet_t planet;
8.
9.      scanf("%s%lf%d%lf%lf", planet.name,
10.          &planet.diameter,
11.          &planet.moons,
12.          &planet.orbit_time,
13.          &planet.rotation_time);
14.      return (planet);
15. }
```

FIGURE 10.7 Function to Compute an Updated Time Value

```
1. /*
2.  * Computes a new time represented as a time_t structure
3.  * and based on time of day and elapsed seconds.
4.  */
5. time_t
6. new_time(time_t time_of_day,    /* input - time to be
7.                                     updated                               */
8.          int   elapsed_secs)  /* input - seconds since last update   */
9. {
10.     int new_hr, new_min, new_sec;
11.
12.     new_sec = time_of_day.second + elapsed_secs;
13.     time_of_day.second = new_sec % 60;
14.     new_min = time_of_day.minute + new_sec / 60;
15.     time_of_day.minute = new_min % 60;
16.     new_hr  = time_of_day.hour + new_min / 60;
17.     time_of_day.hour = new_hr % 24;
18.
19.     return (time_of_day);
20. }
```

Problem Solving with Structure Types

- abstract data type (ADT)
 - a data type combined with a set of basic operations

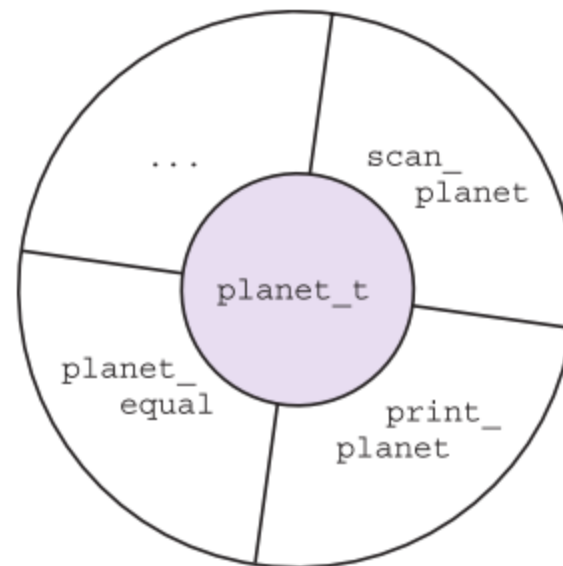


FIGURE 10.9

Data Type
planet_t and Basic
Operations

Header files

`#include<stdio.h>`

versus

`#include"class.h"`

Angle brackets versus quotes tells compiler where to look for the file