“To Iterate is Human, to Recurse, Divine”

- James O. Coplien

CSE102
Computer Programming with C

2016-2017 Fall Semester

Structures

Largely adapted from J.R. Hanly, E.B. Koffman, F.E. Sevilgen, and others…

Structure

- A structure is a collection of one or more variables possibly of different types, grouped together under a single name for convenient handling.
- Ex: Planet type
  - Name
  - Diameter
  - Number of moons
  - Number of years to complete one solar orbit
  - Number of hours to complete one rotation.

Structure definition

```
typedef struct
{
        char    name[20];
        double  diameter;
        int     moons;
        double  orbit_time,
                rotation_time;
} planet_t;

planet_t  my_planet;
```

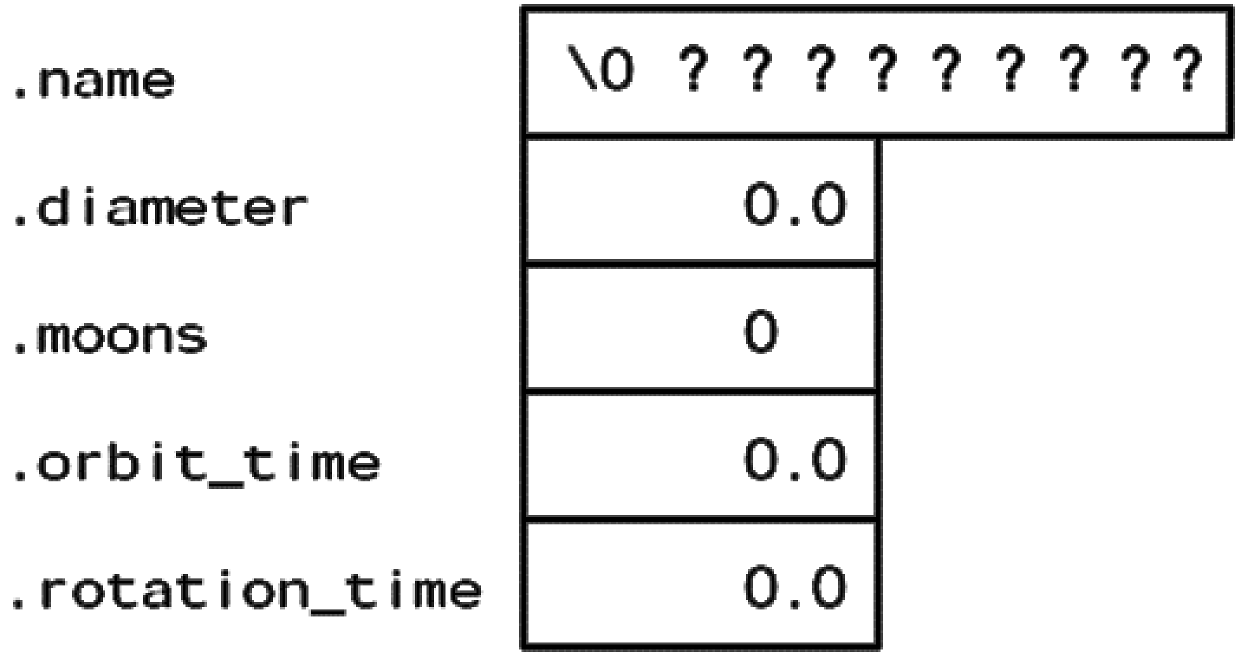
Structure definition (Cont'd)

- A name chosen for a component of one structure may be the same as the name of a component of another structure or the same as the name of a variable
- The **typedef** statement itself allocates no memory
- A variable declaration is required to allocate storage space for a structured data object

```
planet_t    current_planet,
            previous_planet,
            blank_planet = {"", 0.0, 0, 0.0, 0.0};
```

Structure definition (Cont'd)

Variable blank_planet, a structure of type planet_t



.name	\0 ? ? ? ? ? ? ? ? ?
.diameter	0.0
.moons	0
.orbit_time	0.0
.rotation_time	0.0

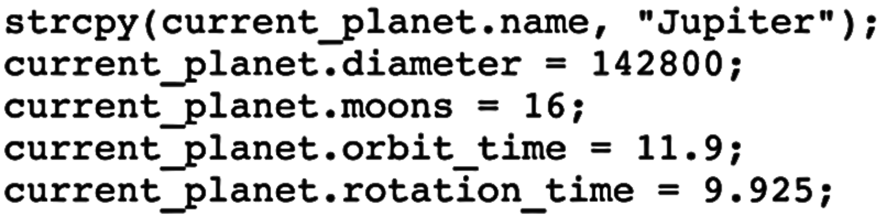
Structure definition (Cont’d)

- Hierarchical structure
  - a structure containing components that are structures
- Example

```
typedef struct {
   double diameter;
   planet_t planets[9];
   char galaxy[STRSIZ];
} solar_sys_t;
```

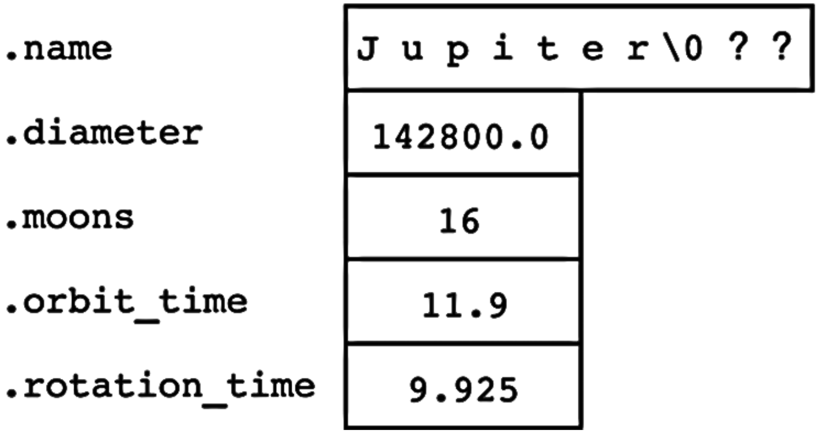
Assigning Values

- Direct component selection operator: a dot (.) placed between a structure type variable and a component name to create a reference to the component



```
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



Manipulating Structures

```
printf("%s's equatorial diameter is %.1f km.\n",
            current_planet.name, current_planet.diameter);
```

→ Jupiter's equatorial diameter is 142800.0 km.

- With no component selection operator refers to the entire structure

```
previous_planet = current_planet;
```

- Direct component operator (.) has the highest precedence.

Structures as Arguments

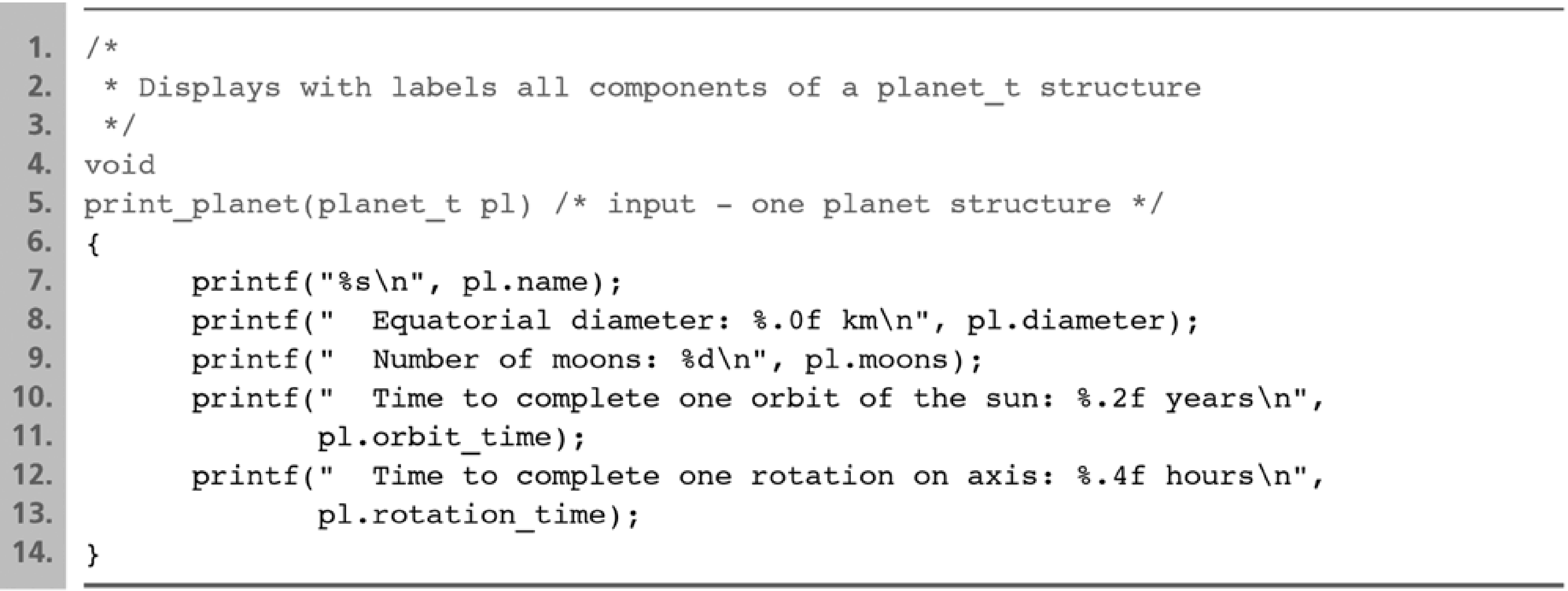
- When a structured variable is passed as an input argument to a function, all of its component *values* are copied into the components of the function’s corresponding formal parameter.

- When such a variable is used as an output argument, the address-of operator must be applied.

- The equality and inequality operators cannot be applied to a structured type as a unit.

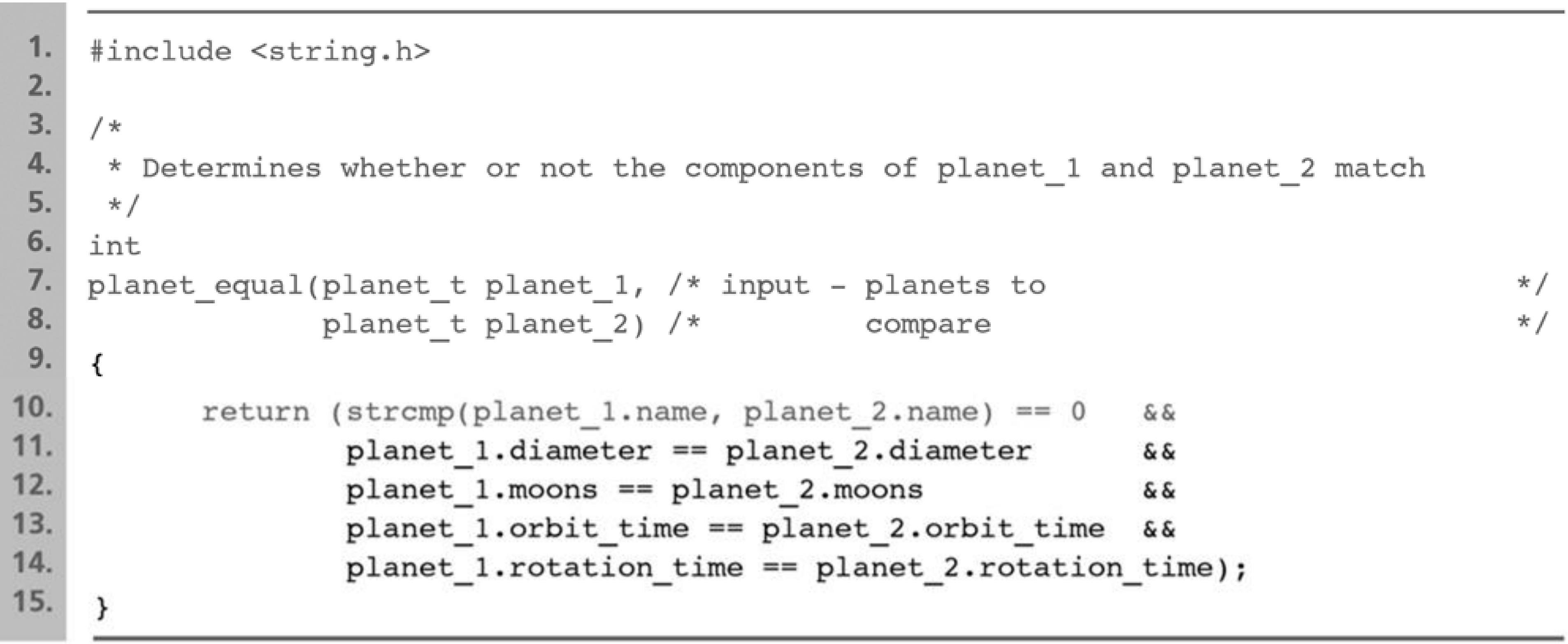
Structured Input Parameter

print_planet(current_planet);



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

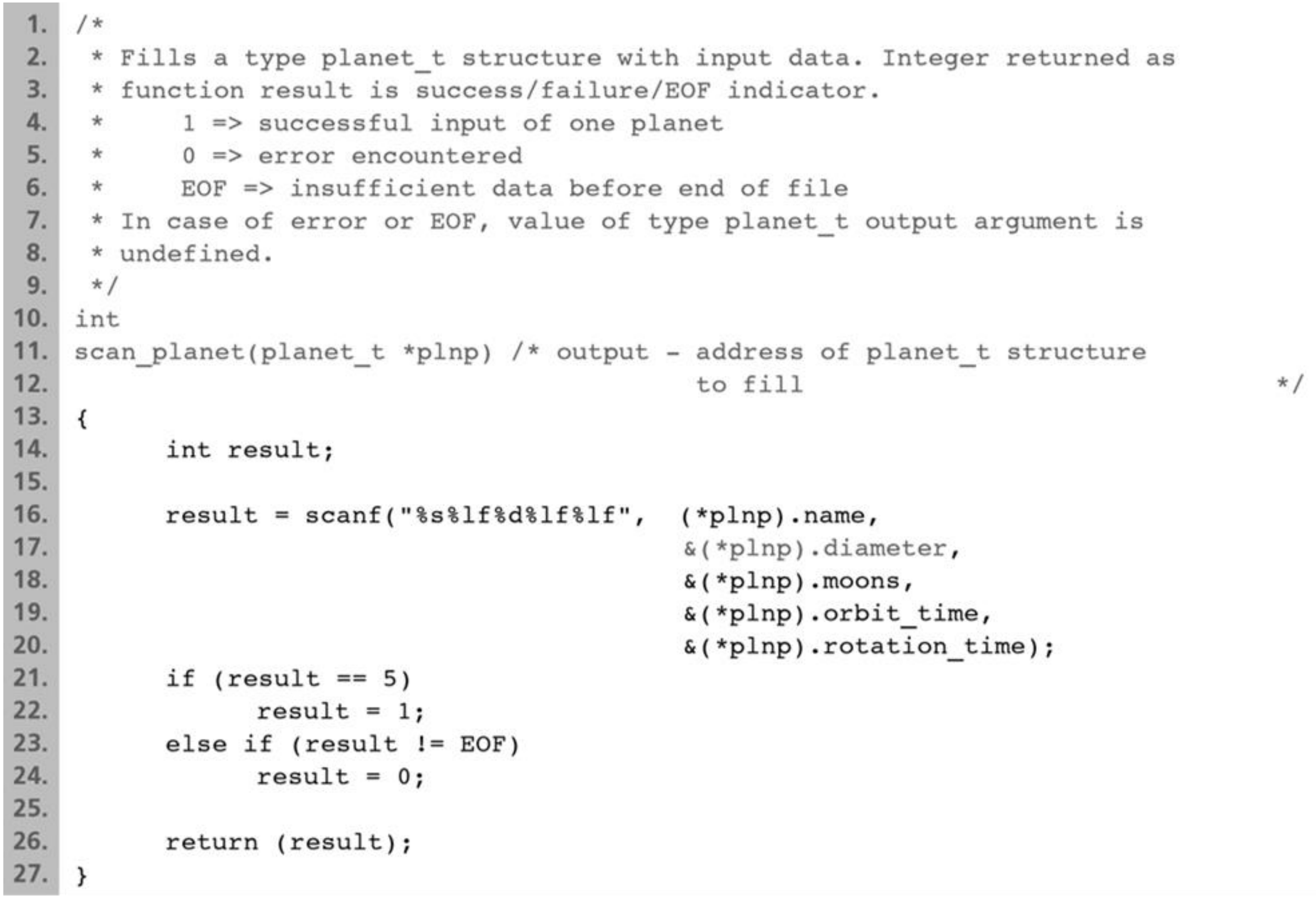
Comparing Two Structured Values



```c
#include <string.h>

/*
 * Determines whether or not the components of planet_1 and planet_2 match
 */
int
planet_equal(planet_t planet_1, /* input - planets to                          */
             planet_t planet_2) /*         compare                             */
{
      return (strcmp(planet_1.name, planet_2.name) == 0   &&
              planet_1.diameter == planet_2.diameter      &&
              planet_1.moons == planet_2.moons            &&
              planet_1.orbit_time == planet_2.orbit_time  &&
              planet_1.rotation_time == planet_2.rotation_time);
}
```

Structured Output Argument

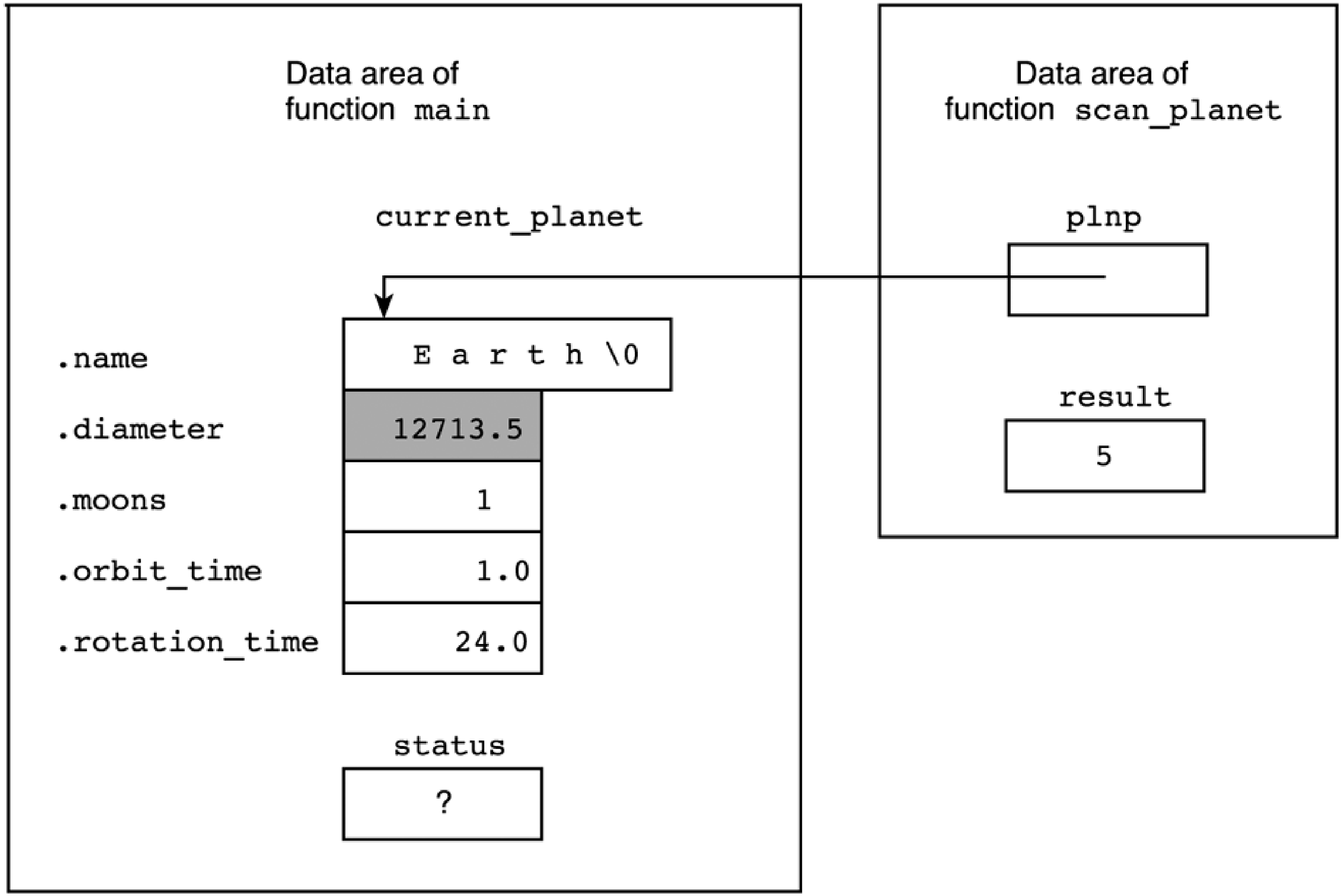


```c
/*
 * Fills a type planet_t structure with input data. Integer returned as
 * function result is success/failure/EOF indicator.
 *     1 => successful input of one planet
 *     0 => error encountered
 *     EOF => insufficient data before end of file
 * In case of error or EOF, value of type planet_t output argument is
 * undefined.
 */
int
scan_planet(planet_t *plnp) /* output - address of planet_t structure
                                        to fill                               */
{
      int result;

      result = scanf("%s%lf%d%lf%lf",  (*plnp).name,
                                       &(*plnp).diameter,
                                       &(*plnp).moons,
                                       &(*plnp).orbit_time,
                                       &(*plnp).rotation_time);
      if (result == 5)
            result = 1;
      else if (result != EOF)
            result = 0;

      return (result);
}
```

status = scan_planet(&current_planet);



Structured Output Argument (Cont’d)

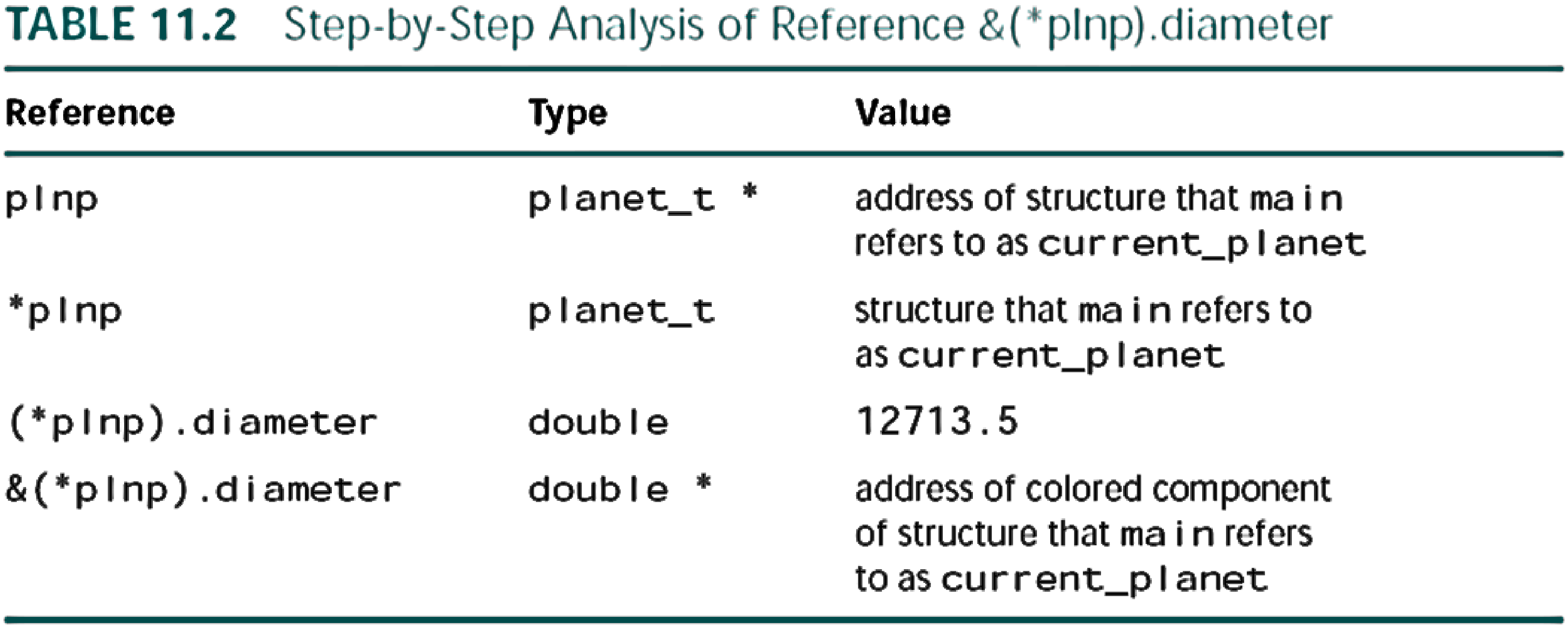


TABLE 11.2 Step-by-Step Analysis of Reference &(*plnp).diameter

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

Structure as Argument

- In order to use scanf to store a value in one component of the structure whose address is in plnp, we must carry out the following steps (in order):
  1. Follow the pointer in plnp to the structure.
  2. Select the component of interest.
  3. Unless this component is an array, get its address to pass to scanf.

- &*plnp.diameter would attempt step 2 before step 1.

Structure as Argument (Cont’d)

- 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
- Two expressions are equivalent.

```
(*structp).component
structp->component
```

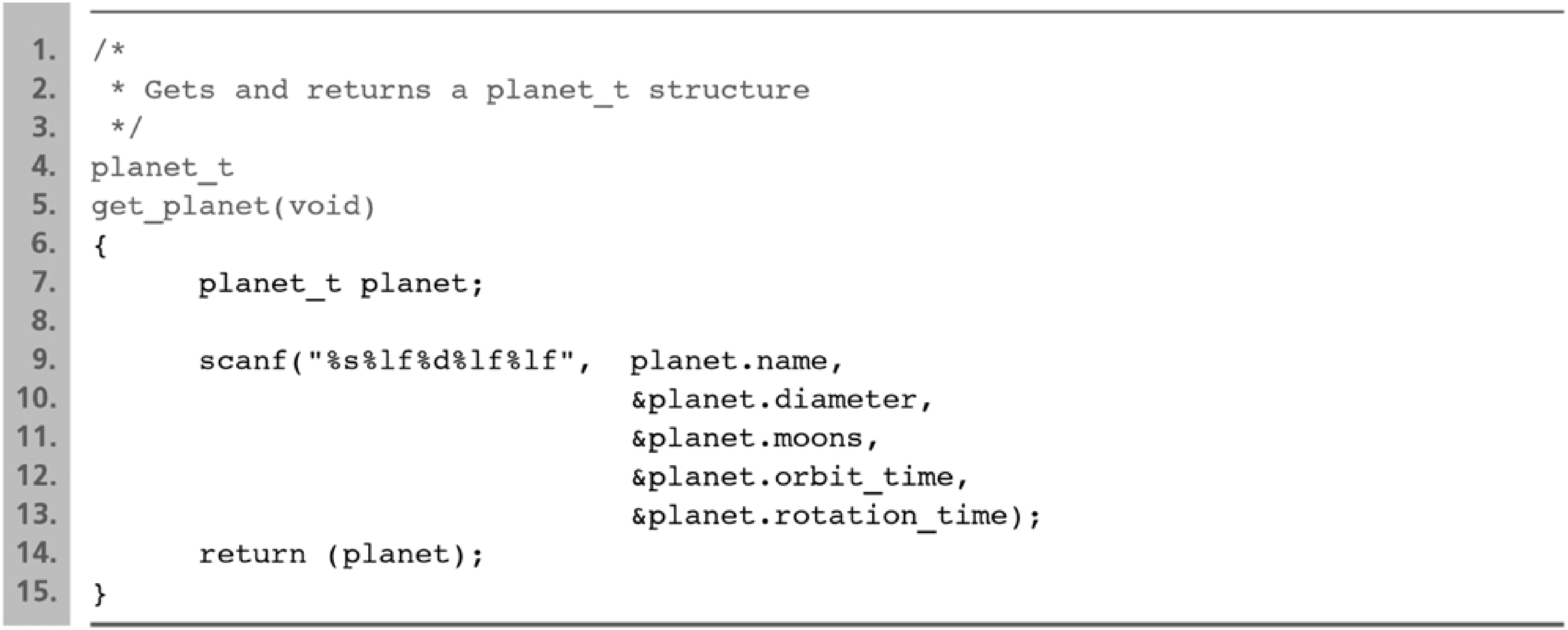
Structure as Argument (Cont’d)

-
  ```
  result = scanf("%s%lf%d%lf%lf",
           plnp->name,
           &plnp->diameter,
           &plnp->moons,
           &plnp->orbit_time,
           &plnp->rotation_time);
  ```

Returning a Structured Result

- The function returns the *values* of all components.

  current_planet = get_planet();
- However, **scan_planet** with its ability to return an integer error code is the more generally useful function.



```
/*
 * Gets and returns a planet_t structure
 */
planet_t
get_planet(void)
{
      planet_t planet;

      scanf("%s%lf%d%lf%lf",  planet.name,
                              &planet.diameter,
                              &planet.moons,
                              &planet.orbit_time,
                              &planet.rotation_time);

      return (planet);
}
```

Abstract Data Type

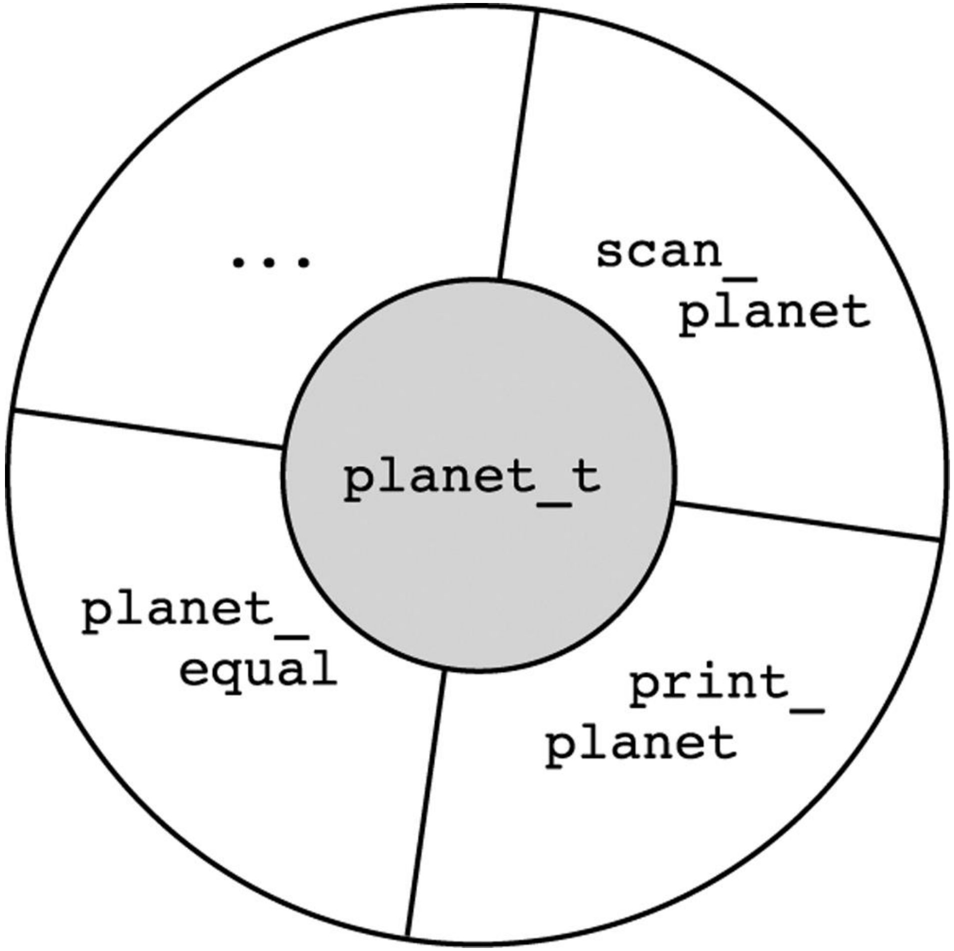
- **Abstract Data Type (ADT)**

  a data type combined with a set of basic operations

- We must also provide basic operations for manipulating our own data types.

- If we take the time to define enough basic operations for a structure type, we then find it possible to think about a related problem at a higher level of abstraction.

Abstract Data Type



Parallel Arrays & Array of Structures

- **Parallel Arrays**

```
int id[50]; /* id numbers and */
double gpa[50]; /* gpa's of up to 50 students */


double x[NUM_PTS], /* (x,y) coordinates of */,
       y[NUM_PTS]; /* up to NUM_PTS points */
```

- **Array of Structures**

A more natural and convenient organization is to group the information in a structure whose type we define.

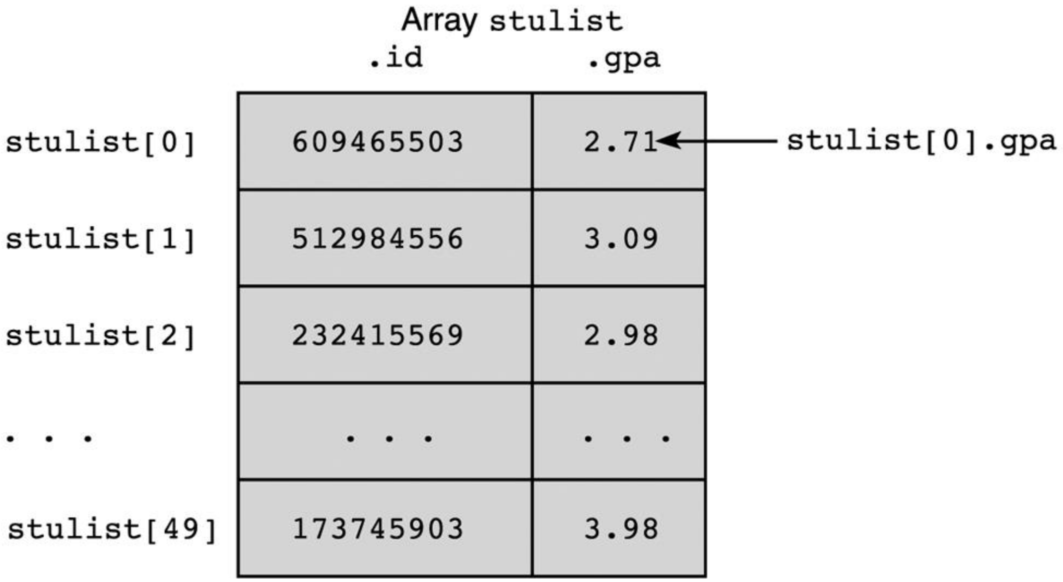
Array of Structures

- Ex. 1

```
#define MAX_STU 50
typedef struct {
   int id;
   double gpa;
} student_t;
. . .
{
   student_t stulist[MAX_STU];
```

- Ex. 2

```
#define NUM_PTS 10
typedef struct {
   double x, y;
} point_t;
. . .
{
   point_t polygon[NUM_PTS];
```



Array stulist

	.id	.gpa
stulist[0]	609465503	2.71
stulist[1]	512984556	3.09
stulist[2]	232415569	2.98
. . .	. . .	. . .
stulist[49]	173745903	3.98

stulist[0].gpa

Union

- A union is a variable that may hold (at different times) objects of different types and sizes, with the compiler keeping track of size and alignment requirement.
- Ex: A compiler symbol table manager. A constant may be an integer, float or a character. The value of a particular constant must be stored in a variable of the proper type, and at the same time it should be stored at the same place regardless of its type.
- This is the purpose of the union – a single variable that can legitimately hold any one of several types, e.g.,

```
union u_tag
{
      int ival;
      float fval;
      char *sval;
} u;
```

Union Types

If the variable utype is used to keep track of the current type stored in **`u`**, then one might see code such as:

```
if (utype == INT)
      printf(“%d\n”, u.ival);
if (utype == FLOAT)
      printf(“%f\n”, u.fval);
if (utype == STRING)
      printf(“%s\n”, u.sval);
```

Union Types

- Another example

```
typedef union {
    int wears_wig;
    char color[20];
} hair_t;

hair_t his_hair;
```

- Memory requirement is determined by the largest component.
- How to determine interpretation?
  - How to determine whether to use wears_wig or color?

Union Types

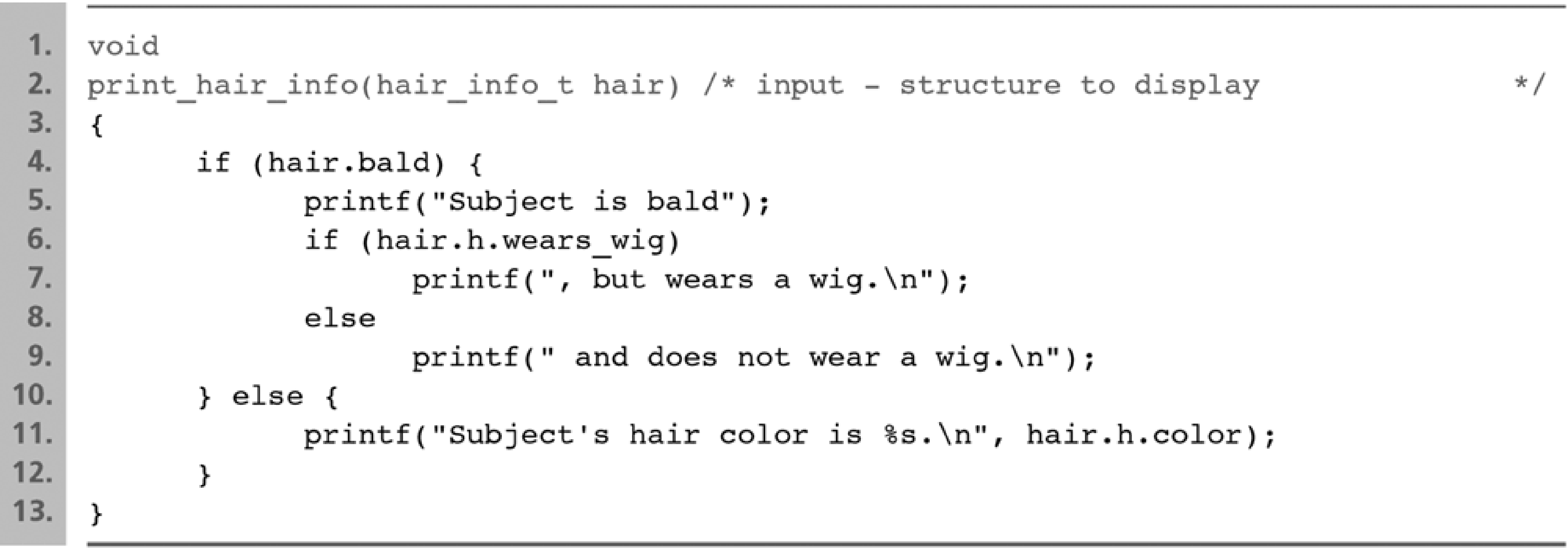
- Data object that can be interpreted in a variety of ways

```
typedef union {
    int wears_wig;
    char color[20];
} hair_t;

typedef struct {
    int bald;
    hair_t h;
} hair_info_t;
hair_info_t his_hair;
```

- Referencing the appropriate union component is *always* the programmer's responsibility; C can do no checking of the validity of such a component reference.

Displays a Structure with a Union



```
void
print_hair_info(hair_info_t hair) /* input - structure to display              */
{
      if (hair.bald) {
            printf("Subject is bald");
            if (hair.h.wears_wig)
                  printf(", but wears a wig.\n");
            else
                  printf(" and does not wear a wig.\n");
      } else {
            printf("Subject's hair color is %s.\n", hair.h.color);
      }
}
```