Structure and Union

User-Defined Structure Types

- A database is a collection of information subdivided into **records**.
  - A **record** is a collection of information of one data object (e.g., ID, name, and age of a student).
- C allows us to define a new data type (called **structure type**) for each category of a structured data object.

Declaring Structure Types (1/2)

- Syntax of the structure type:

```
typedef struct{
    type1 id1;
    type2 id2;
    …
} struct_type;
```

- E.g.,

```
typedef struct{
    char name[20];
    int age;
} student_info;
```

Declaring Structure Types (2/2)

- Declaration:

```
student_info student1,
    student2 = {"Rahim", 18};
```

- A hierarchical structure is a structure containing components which are also structures.

```
typedef struct{
    int NumOfStudents;
    student_info students[20];
} class_info;
```

Manipulating Structure Types (1/2)

- We can reference a component of a structure by the **direct component selection operator(.)**, which is a period.
- E.g.,

```
strcpy(student1.name, "Karim");
student1.age = 18;
printf("%s is in age %d\n",
student1.name, student1.age);
```

Manipulating Structure Types (2/2)

- The **direct component selection operator** has the highest priority in the operator precedence.
  - `student1.age+student2.age+…;`
  - The value of `student1.age` is referenced first.
- The copy of an entire structure can be easily done by the assignment operator.
  - `student1 = student2;`
  - Each component in one structure is copied into the corresponding component in the other structure.

Function with a Structured Input Parameter (1/2)

- Suppose there is a structure defined as follows.
-
```
typedef struct{
    char name[20];
    double diameter;
    int moons;
    double orbit_time,
           rotation_time;
} planet_t;
```

Function with a Structured Input Parameter (2/2)

- When a structure variable is passed as an input argument to a function, all its component values are copied into the local structure variable.

```
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. }
```

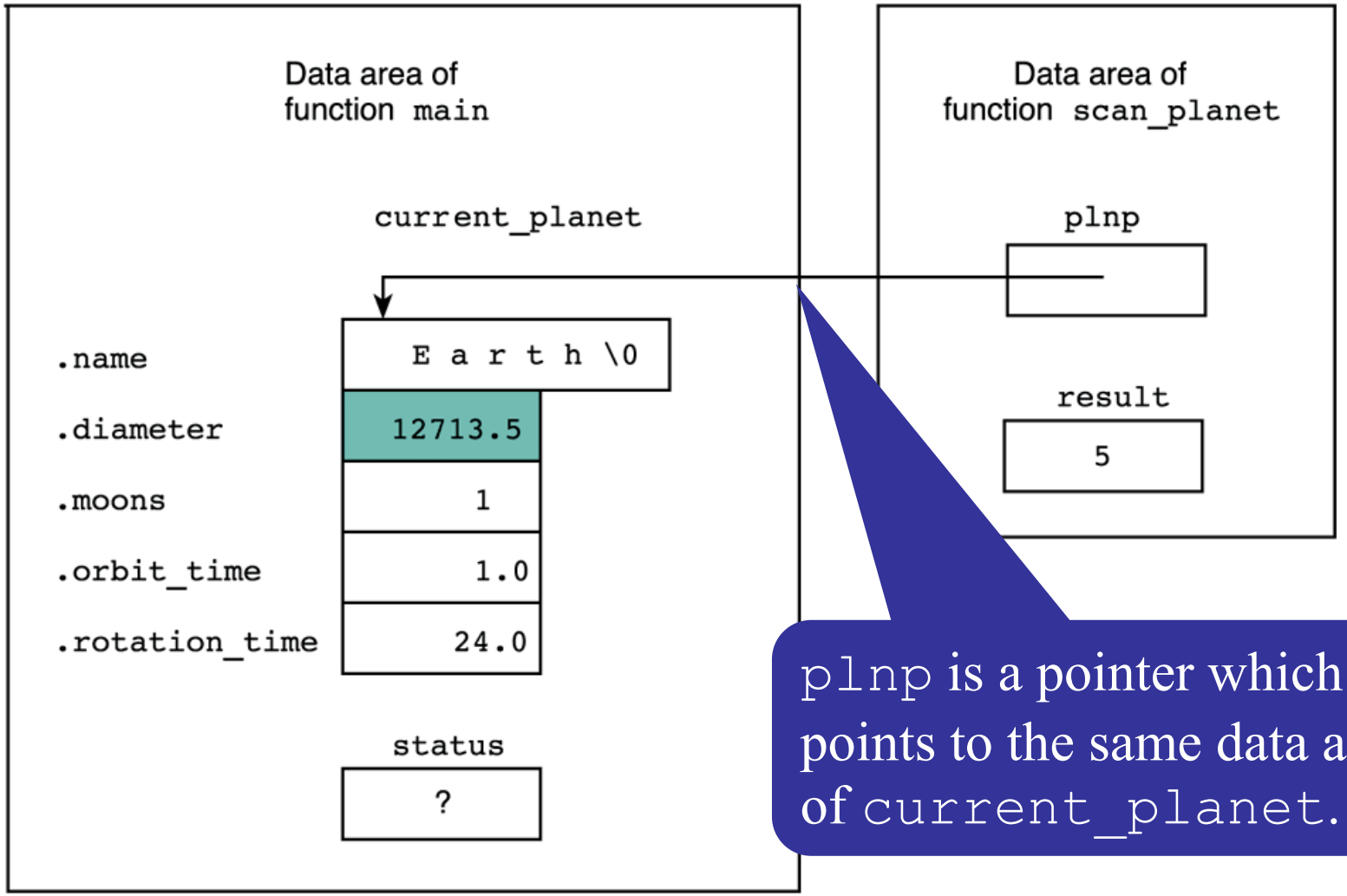
Function with a Structured Input/Output Argument

- For the following function, we have to call it by "`scan_planet(&current_planet);`"
  - The input argument is also used to store the result.

```
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.       if (result == 5)
22.             result = 1;
23.       else if (result != EOF)
24.             result = 0;
25.
26.       return (result);
27. }
```

"`*plnp`" is parenthesized because `&` operator has higher precedence.

Data Areas of call to `scan_planet (&current_planet);`



Step-by-Step Analysis of the Indirect Reference

- "**&(*plnp).diameter**" is evaluated as shown in the following table.

Reference	Type	Value
plnp	planet_t *	Address of structure refers to current_planet
*plnp	planet_t	Real structure of current_planet
(*plnp).diameter	double	12713.5
&(*plnp).diameter	double *	Address of diameter of current_planet structure

Indirect Component Selection Operator

- In the above example, we use direct component selection operator: period.
  - e.g., `&(*plnp).diameter`
- C also provides **indirect component selection operator**: **`->`**.
  - e.g., "`&plnp->diameter`" is the same as "`&(*plnp).diameter`".

Function Returning a Structured Result Type (1/2)

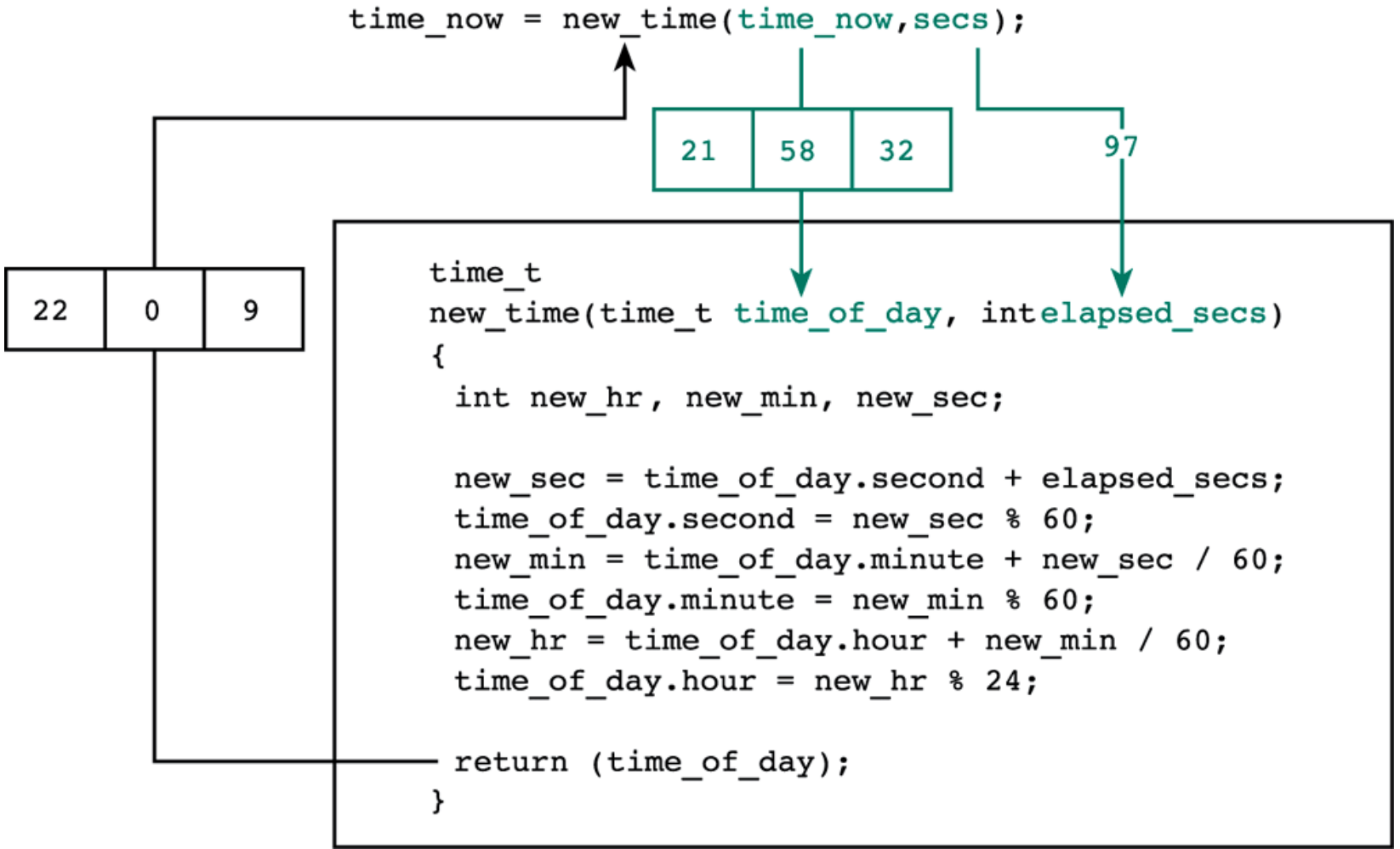
- The structure variable can also be used as the return value of a function.

```
 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 - s
 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. }
```

Use direct component selection operator.

Function Returning a Structured Result Type (2/2)

- Suppose the current time is 21:58:32, and the elapsed time is 97 seconds.



Arrays of Structures (1/2)

- We can also declare an array of structures.
- E.g.,

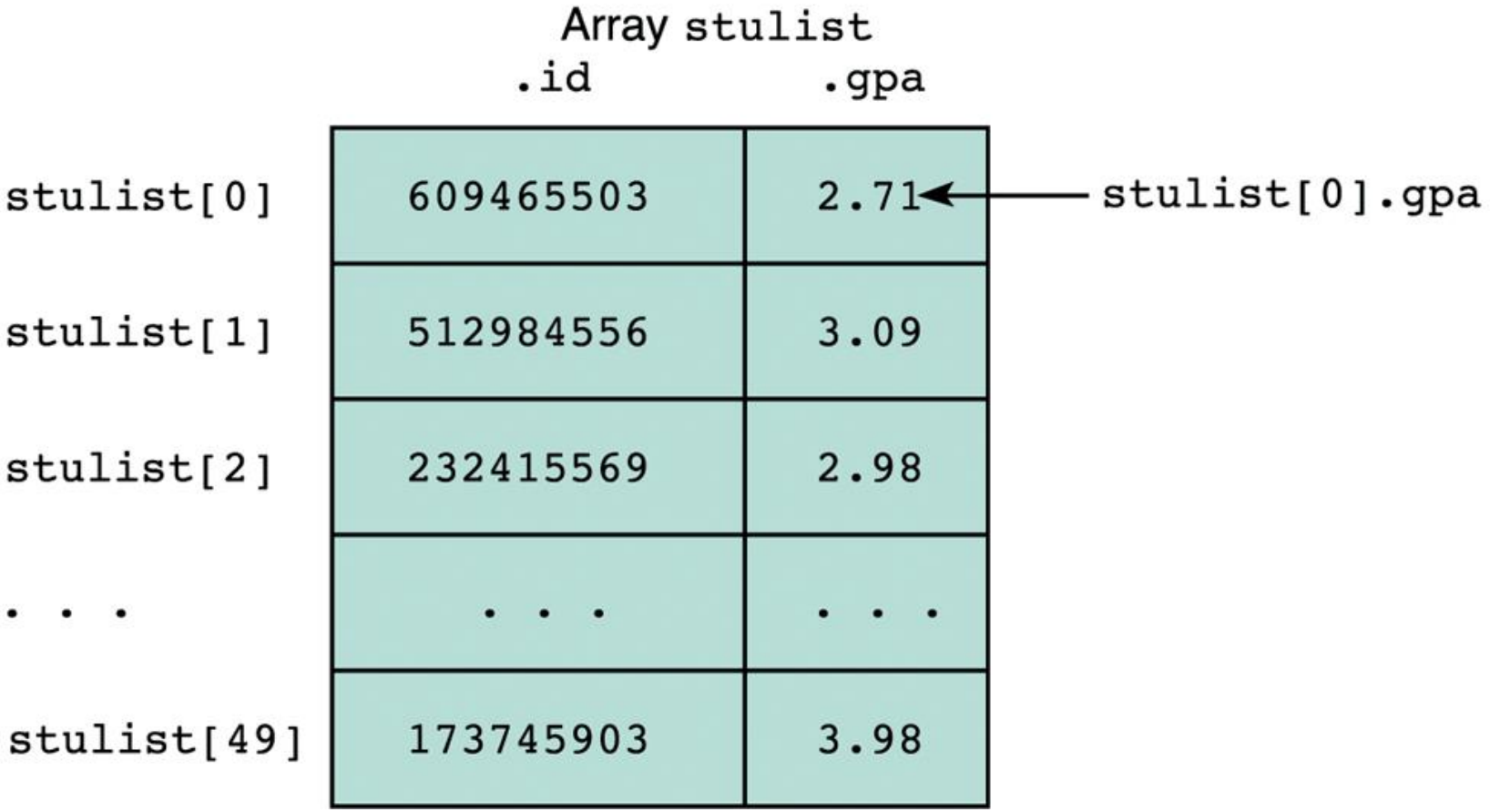
```
typedef struct{
   int id;
   double gpa;
} student_t;
```

- Usage:

```
student_t stulist[50];
stulist[3].id = 92922023;
stulist[3].gpa = 3.0;
```

Arrays of Structures (2/2)

- The array of structures can be simply manipulated as arrays of simple data types.



Unions

- **`union`**
  - Memory that contains a variety of objects over time
  - Only contains one data member at a time
  - Members of an **`union`** share space
  - Size of an union is equal to the size of the largest member.
  - Only the last data member defined can be accessed
- **`union`** declarations
  - Same as struct

```
union Number {
  int x;
  float y;
};
union Number value;
```

Unions

- Valid **`union`** operations
  - Same as **`structure`**
  - Assignment to **`union`** of same type: **`=`**
  - Taking address: **`&`**
  - Accessing union members: **`.`**
  - Accessing members using pointers: **`->`**

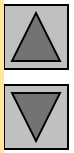
```c
/* Fig. 10.5: fig10_05.c
   An example of a union */
#include <stdio.h>

union number {
   int x;
   double y;
};

int main()
{
   union number value;

   value.x = 100;
   printf( "%s\n%s\n%s%d\n%s%f\n\n",
           "Put a value in the integer member",
           "and print both members.",
           "int:   ", value.x,
           "double:\n", value.y );

   value.y = 100.0;
   printf( "%s\n%s\n%s%d\n%s%f\n",
           "Put a value in the floating member",
           "and print both members.",
           "int:   ", value.x,
           "double:\n", value.y );
   return 0;
}
```



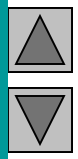
Outline

1. Define union

1.1 Initialize variables

2. Set variables

3. Print



Outline

Program Output

```
Put a value in the integer member
and print both members.
int:   100
double:
-92559592117433136000000000000000000000000000000000000000000000000.00000

Put a value in the floating member
and print both members.
int:   0
double:
100.000000
```

Structure vs Union

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.