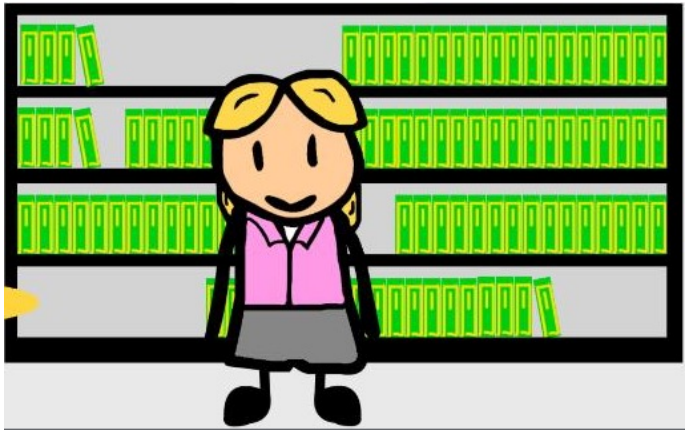


Structures and Unions

Mirza Mohammad Lutfe Elahi

Outline

- Structure data type
- Store data for a structured object or record
- Process individual fields of a structured object
- Use structs as function parameters and return function results
- Understand the relationship between parallel arrays and arrays of structured objects
- Union data type

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

Structure Type Definition

SYNTAX:

```
typedef struct {
        type1  id_list1;
        type2  id_list2;
             .
             .
             .
        typen  id_listn;
} struct_type;
```

EXAMPLE:

```
typedef struct { /* complex number structure */
        double real_pt,
               imag_pt;
} complex_t;
```

(continued)

INTERPRETATION: The identifier *struct_type* is the name of the structure type being defined. Each id_list_i is a list of one or more component names separated by commas; the data type of each component in id_list_i is specified by $type_i$.

NOTE: $type_i$ can be any standard or previously specified user-defined data type.

User-defined Structure Types

```
#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;
```

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

blank_planet

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

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

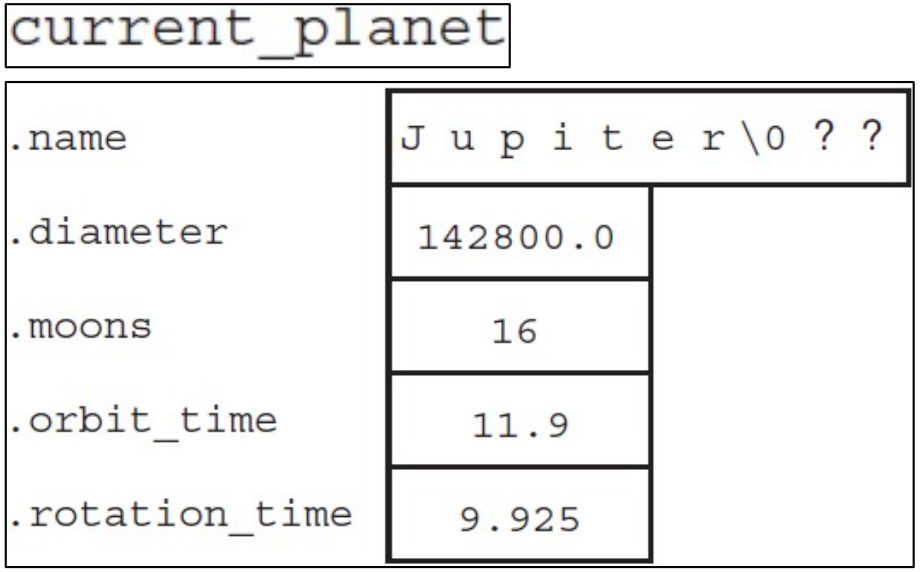
Name: Jupiter

Diameter: 142,800 km

Moons: 16

Orbit time: 11.9 years

Rotation time: 9.925 hours




```
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;
```

Precedence and Associativity of Operators

Precedence	Symbols	Operator Names	Associativity
highest	`a[j] f( ... ) .`	Subscripting, function calls, direct component selection	left
	`++ --`	Postfix increment and decrement	left
	`++ -- ! - + & *`	Prefix increment and decrement, logical not, unary negation and plus, address of, indirection	right
	`(`*type name*`)`	Casts	right
	`* / %`	Multiplicative operators (multiplication, division, remainder)	left
	`+ -`	Binary additive operators (addition and subtraction)	left
	`< > <= >=`	Relational operators	left
	`== !=`	Equality/inequality operators	left
	`&&`	Logical and	left
	`||`	Logical or	left
lowest	`= += -= *= /= %=`	Assignment operators	right

Structure Data Type as I/O Parameters

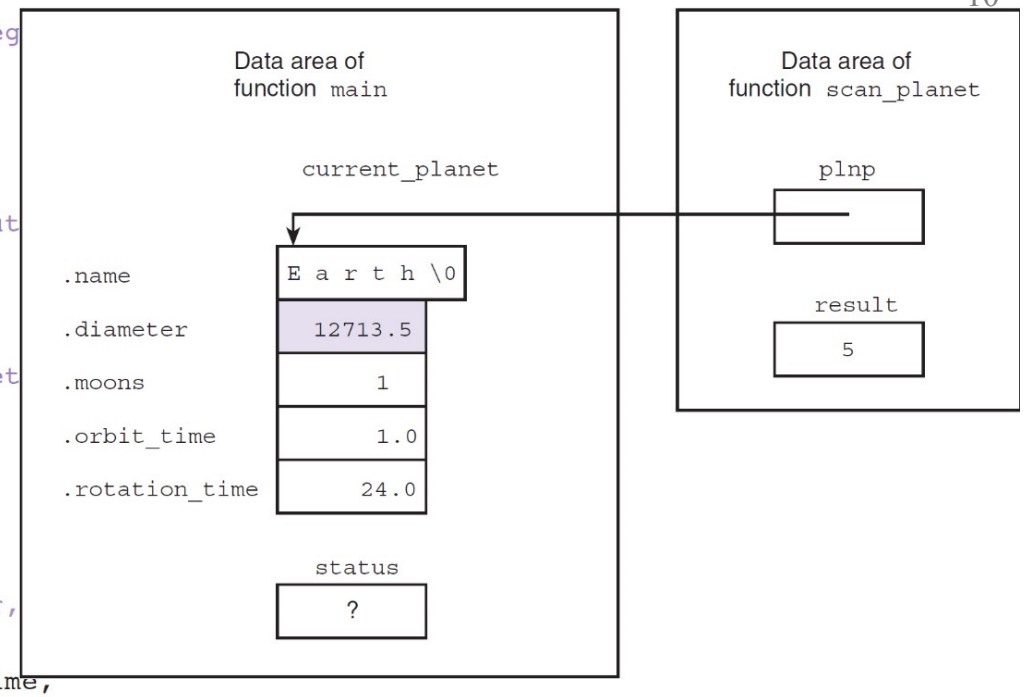
- When a structured variable is passed as an input argument to a function, all of its component **values** are copied into the components of the function's corresponding formal parameter.
- When such a variable is used as an output argument, the address-of operator must be applied in the same way that we would pass output arguments of the standard types **char**, **int**, and **double**.

```c
/*
 * 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);
}
```

```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);
}
```

```
1.  /*
2.   * Fills a type planet_t structure with input data. Integ
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
8.   * undefined.
9.   */
10. int
11. scan_planet(planet_t *plnp) /* output - address of planet
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. }
```



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`

Function Returning Structured Result

- 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.
- 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.

Function Returning Structured Result

```
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. }
```

Function Returning Structured Result

```
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. }
```

Parallel Arrays and Arrays of Structures

- A natural organization of parallel arrays with data that contain items of different types is to group the data into a structure whose type we define.

```
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                */
```

```
#define MAX_STU 50
#define NUM_PTS 10

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

typedef struct {
      double x, y;
} point_t;

. . .

{
      student_t stulist[MAX_STU];
      point_t   polygon[NUM_PTS];
```

Array `stulist`

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

Union Types

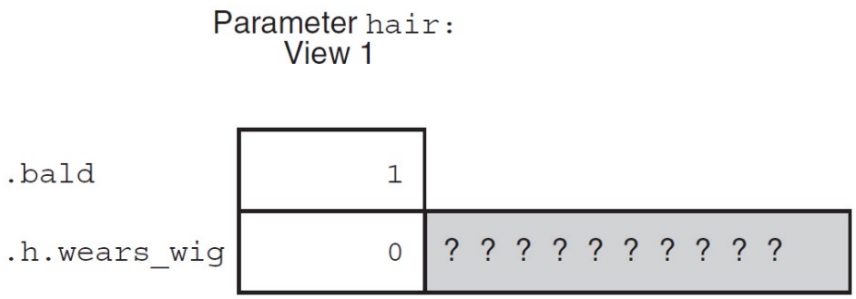
- Union - a data structure that overlays components in memory, allowing one chunk of memory to be interpreted in multiple ways

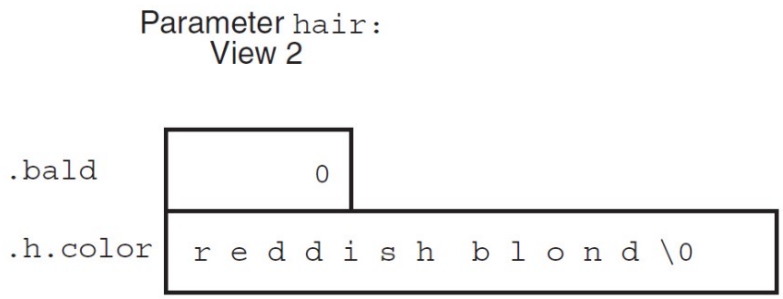
```
typedef union {
        int   wears_wig;
        char  color[20];
} hair_t;
```

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

```
1.  void
2.  print_hair_info(hair_info_t hair) /* input - structure to display          */
3.  {
4.        if (hair.bald) {
5.            printf("Subject is bald");
6.            if (hair.h.wears_wig)
7.                  printf(", but wears a wig.\n");
8.            else
9.                  printf(" and does not wear a wig.\n");
10.       } else {
11.           printf("Subject's hair color is %s.\n", hair.h.color);
12.       }
13. }
```

Union Types


Parameter hair:
View 1
.bald
1
.h.wears_wig
0
? ? ? ? ? ? ? ? ? ?


Parameter hair:
View 2
.bald
0
.h.color
r e d d i s h b l o n d \0

Summary

- C permits the user to define a type composed of multiple named components.
- User-defined structure types can be used in most situations where build-in types are value.
- Structured values can be function arguments and function results and can be copied using the assignment operator.
- Structure types are legitimate in declarations of variables, of structure components, and of arrays.
- In a union type, structure components are overlaid in memory.