# STRUCTURE AND UNION TYPES

Algorithms and Programming II

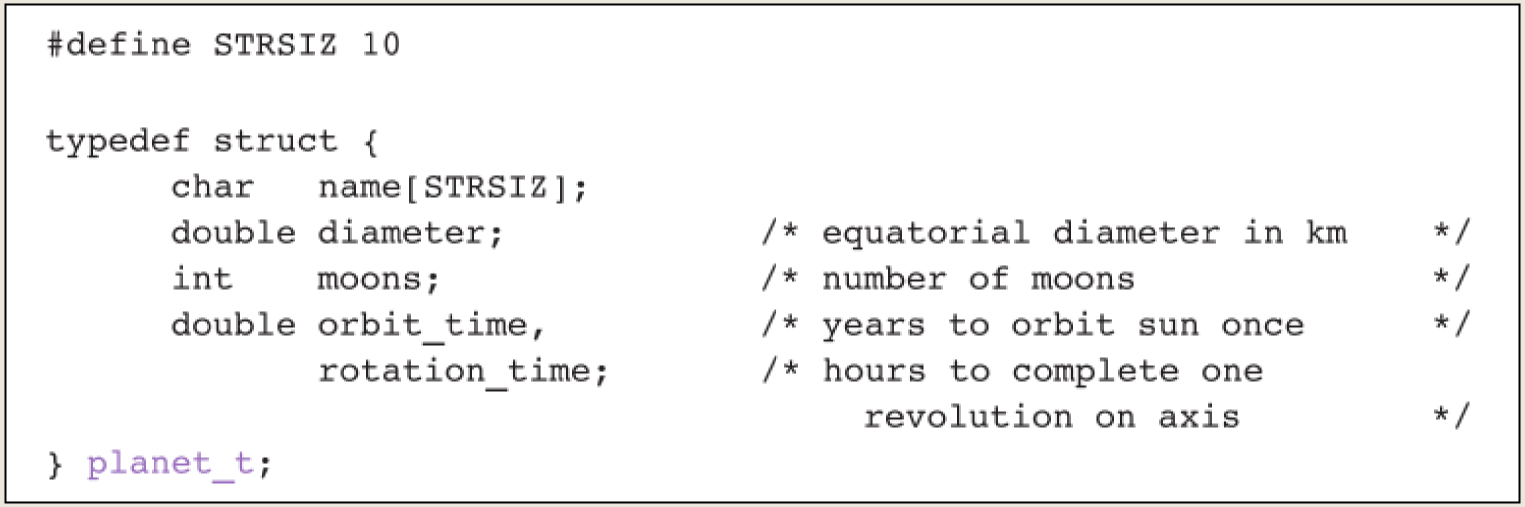
# User-Defined Structure Types

- 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

# User-Defined Structure Types

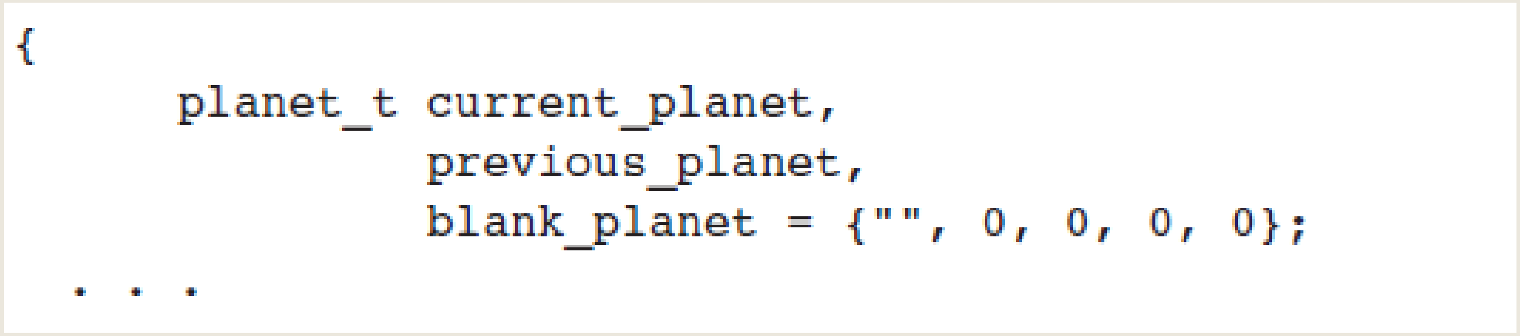


```
#define STRSIZ 10

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;
```

- The reserved word typedef can be used to name many varieties of user-defined types.

# User-Defined Structure Types

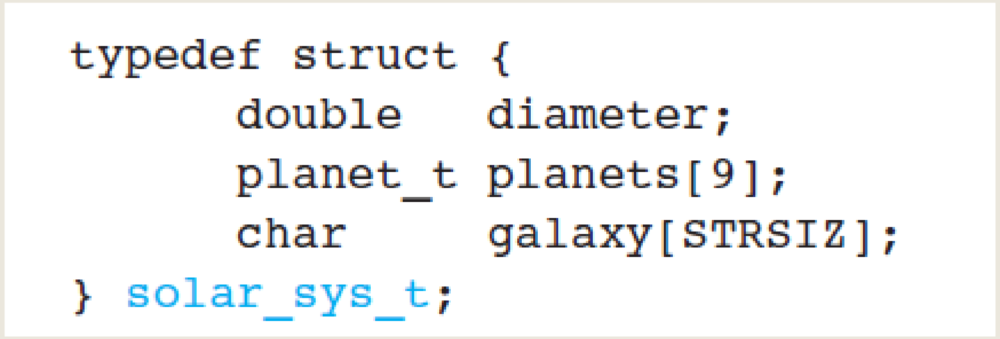
- The typedef statement itself allocates no memory. A variable declaration is required to allocate storage space for a structured data object. The variables current_planet and previous_planet are declared next, and the variable blank_planet is declared and initialized



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

# User-Defined Structure Types

- A user-defined type like `planet_t` can be used to declare both simple and array variables and to declare components in other structure types. A structure containing components that are data structures (arrays or structs) is sometimes called a **hierarchical structure** ..

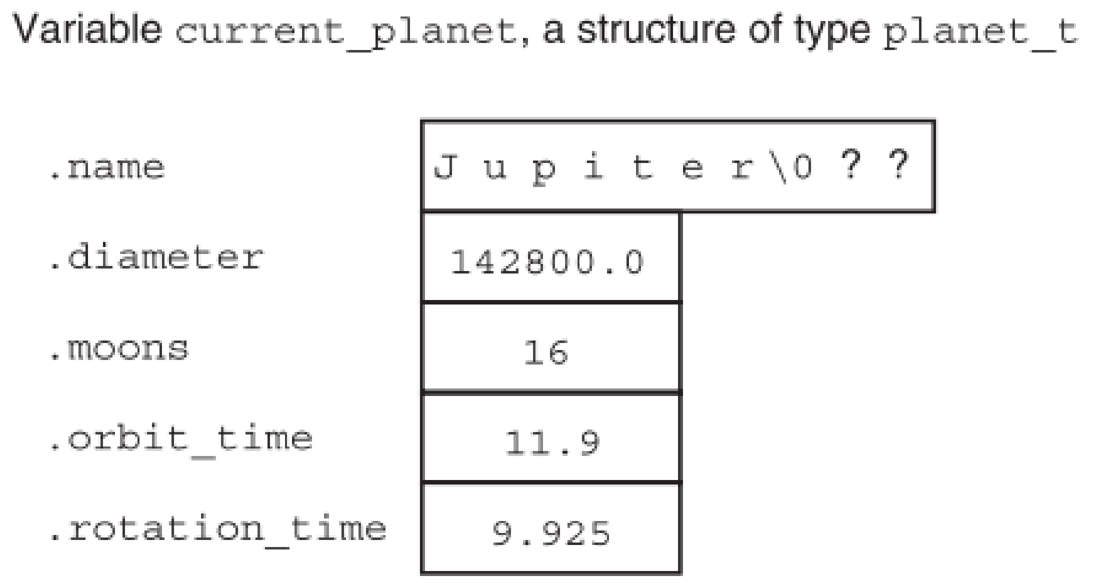


```
typedef struct {
      double   diameter;
      planet_t planets[9];
      char     galaxy[STRSIZ];
} solar_sys_t;
```

## 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*

```
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;
```

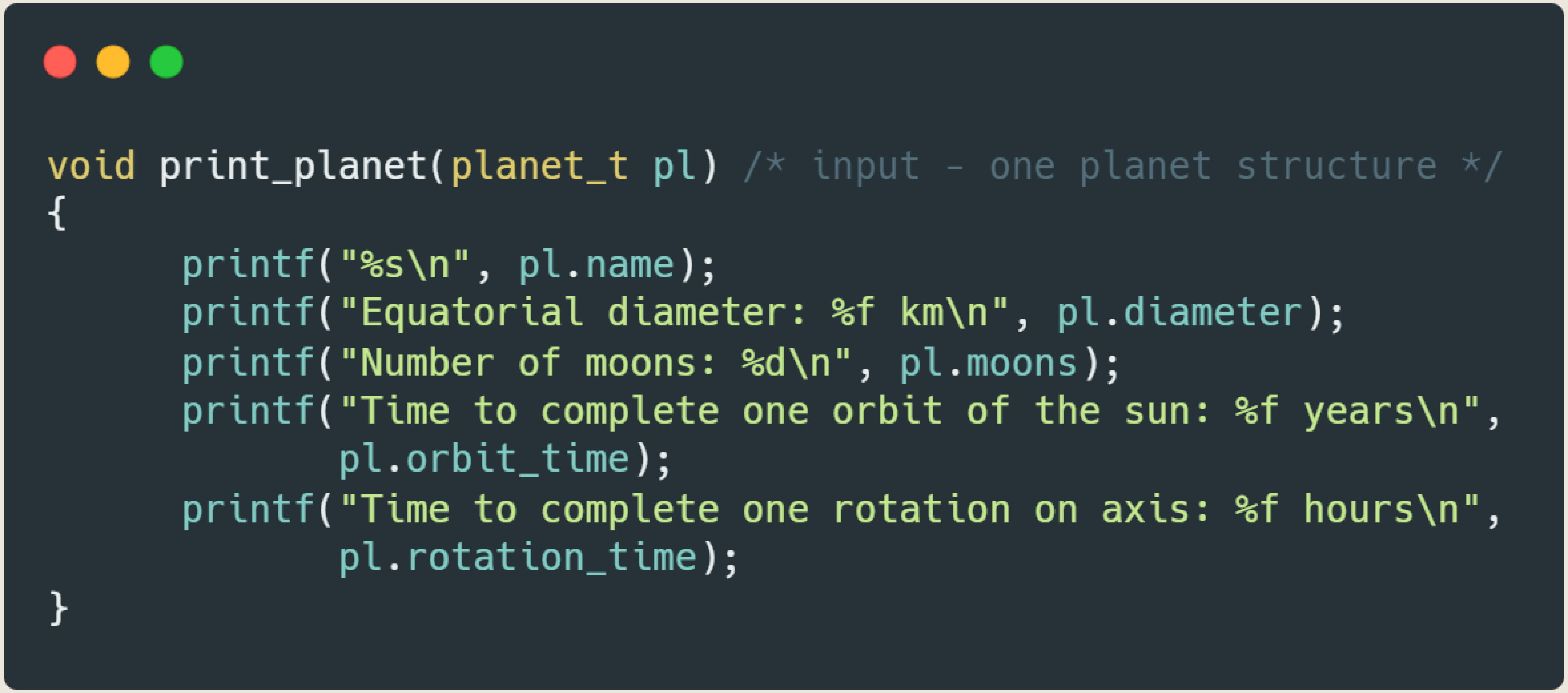

Variable current_planet, a structure of type planet_t

## 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.

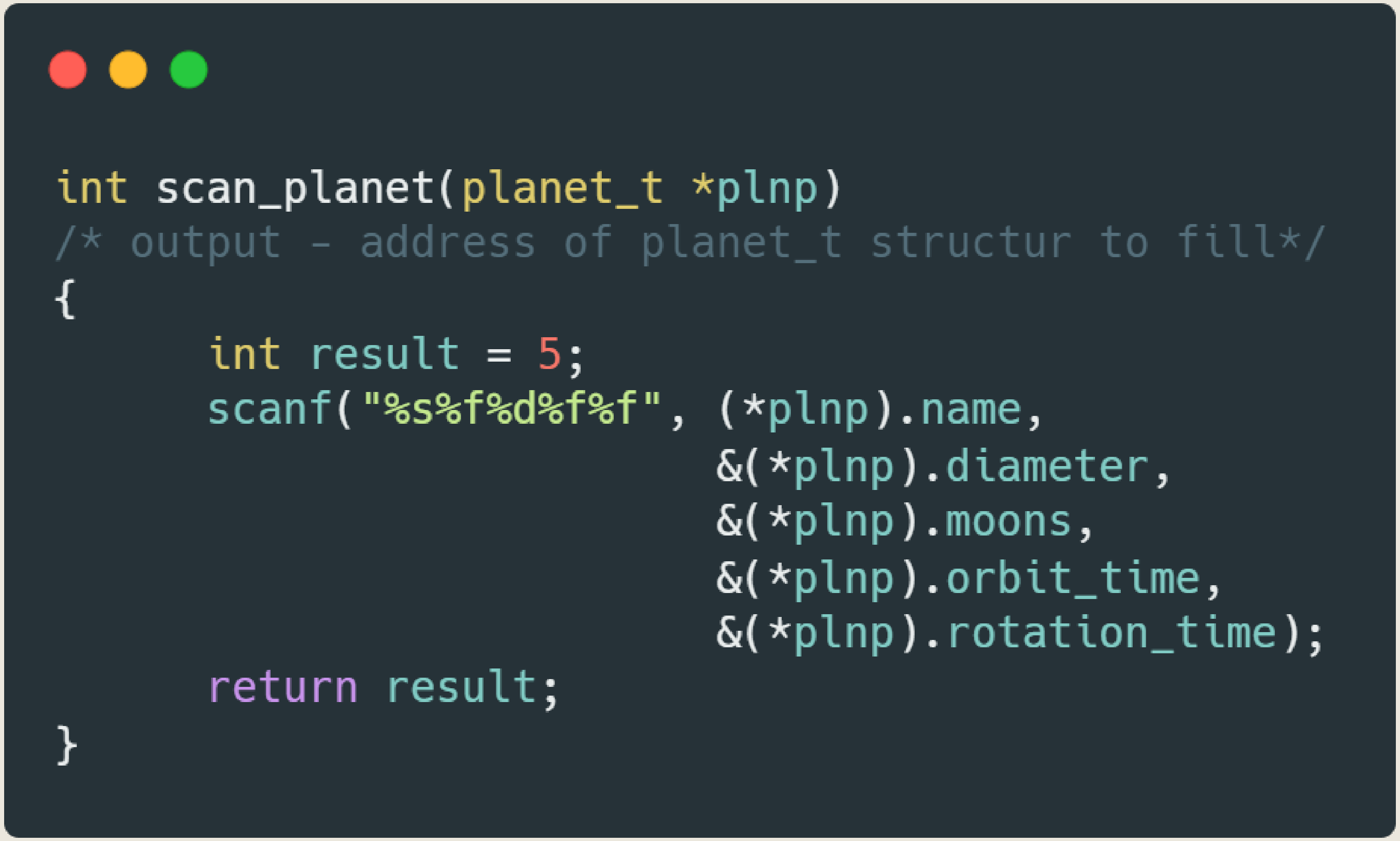
- When such a variable is used as an output argument, the address-of operator must be applied in the same way that we would pass output arguments of the standard types char, int, and double.

# Structure Data Type as Input and Output Parameters



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

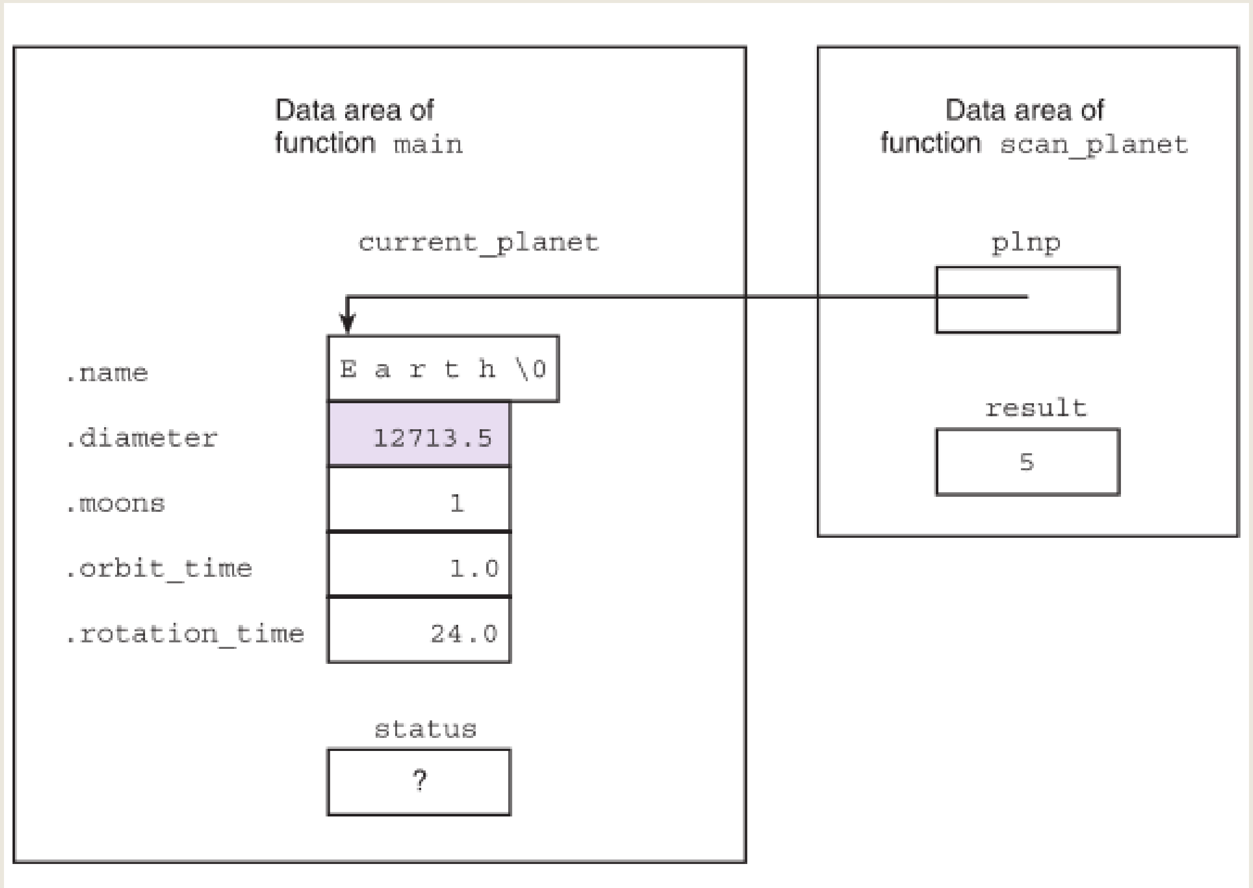
## Structure Data Type as Input and Output Parameters



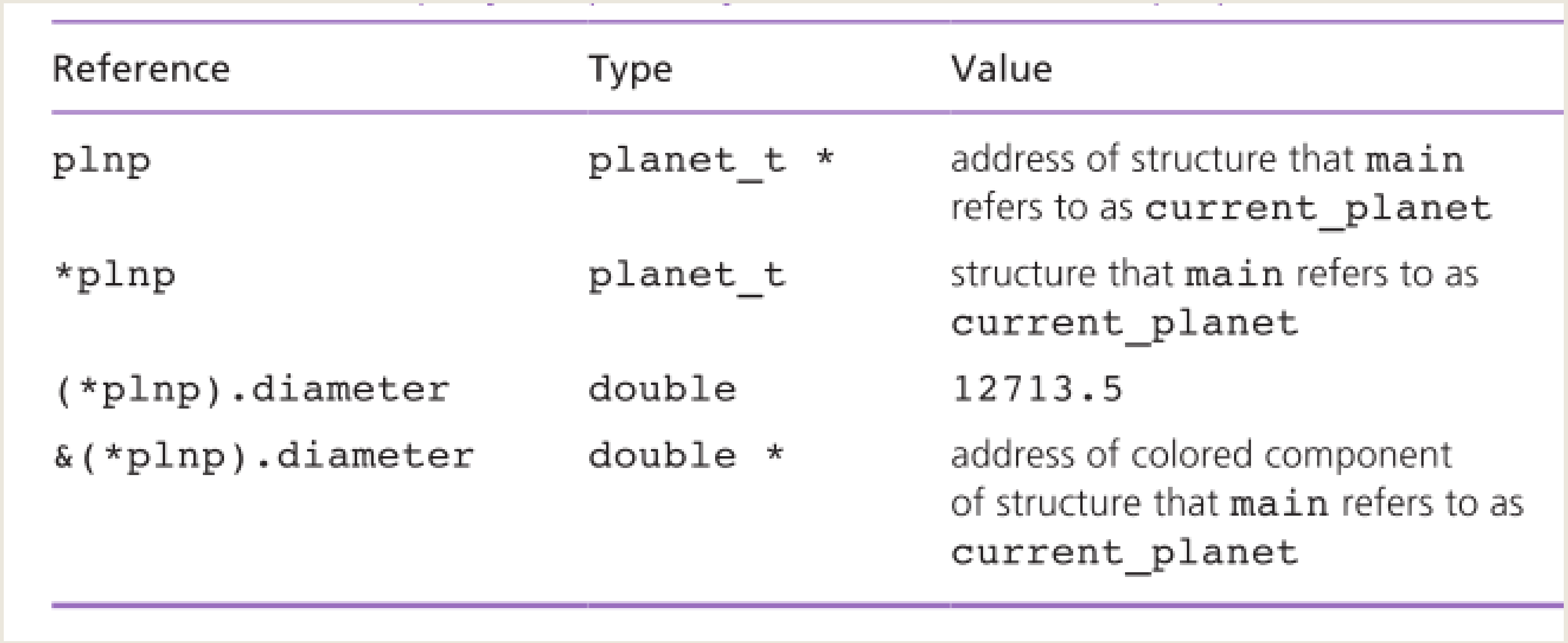
```c
int scan_planet(planet_t *plnp)
/* output - address of planet_t structur to fill*/
{
      int result = 5;
      scanf("%s%f%d%f%f", (*plnp).name,
                          &(*plnp).diameter,
                          &(*plnp).moons,
                          &(*plnp).orbit_time,
                          &(*plnp).rotation_time);
      return result;
}
```

# Structure Data Type as Input and Output Parameters

- `status=scan_planet(&current_planet);`



# Structure Data Type as Input and Output Parameters



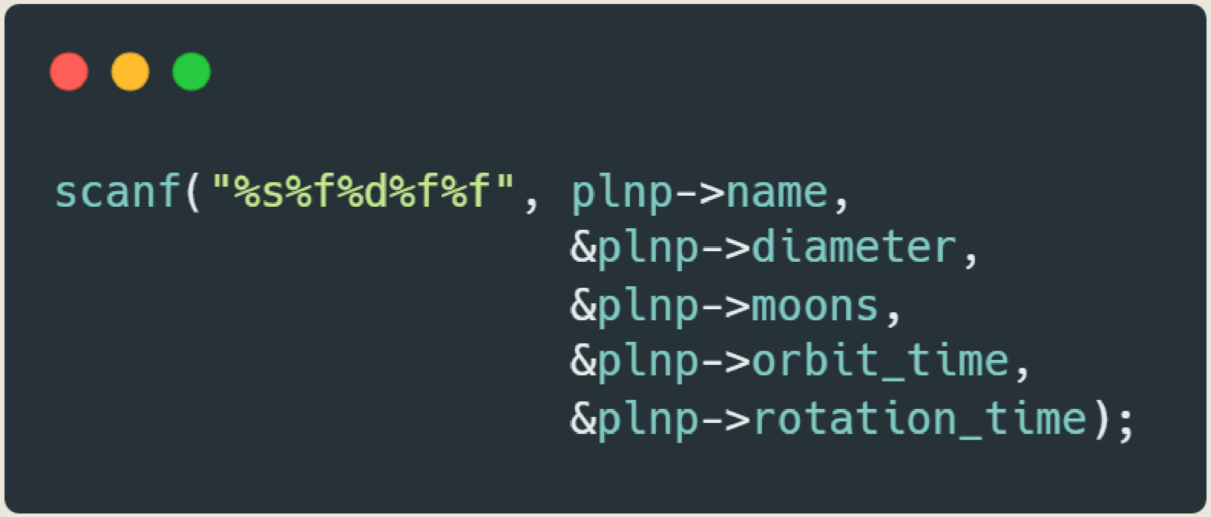
| Reference | Type | Value |
|---|---|---|
| `plnp` | `planet_t *` | address of structure that `main` refers to as `current_planet` |
| `*plnp` | `planet_t` | structure that `main` refers to as `current_planet` |
| `(*plnp).diameter` | `double` | `12713.5` |
| `&(*plnp).diameter` | `double *` | address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

## Structure 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*

## Structure Data Type as Input and Output Parameters

- If we rewrite the `scan_planet` function using the `->` operator, the assignment to result will be

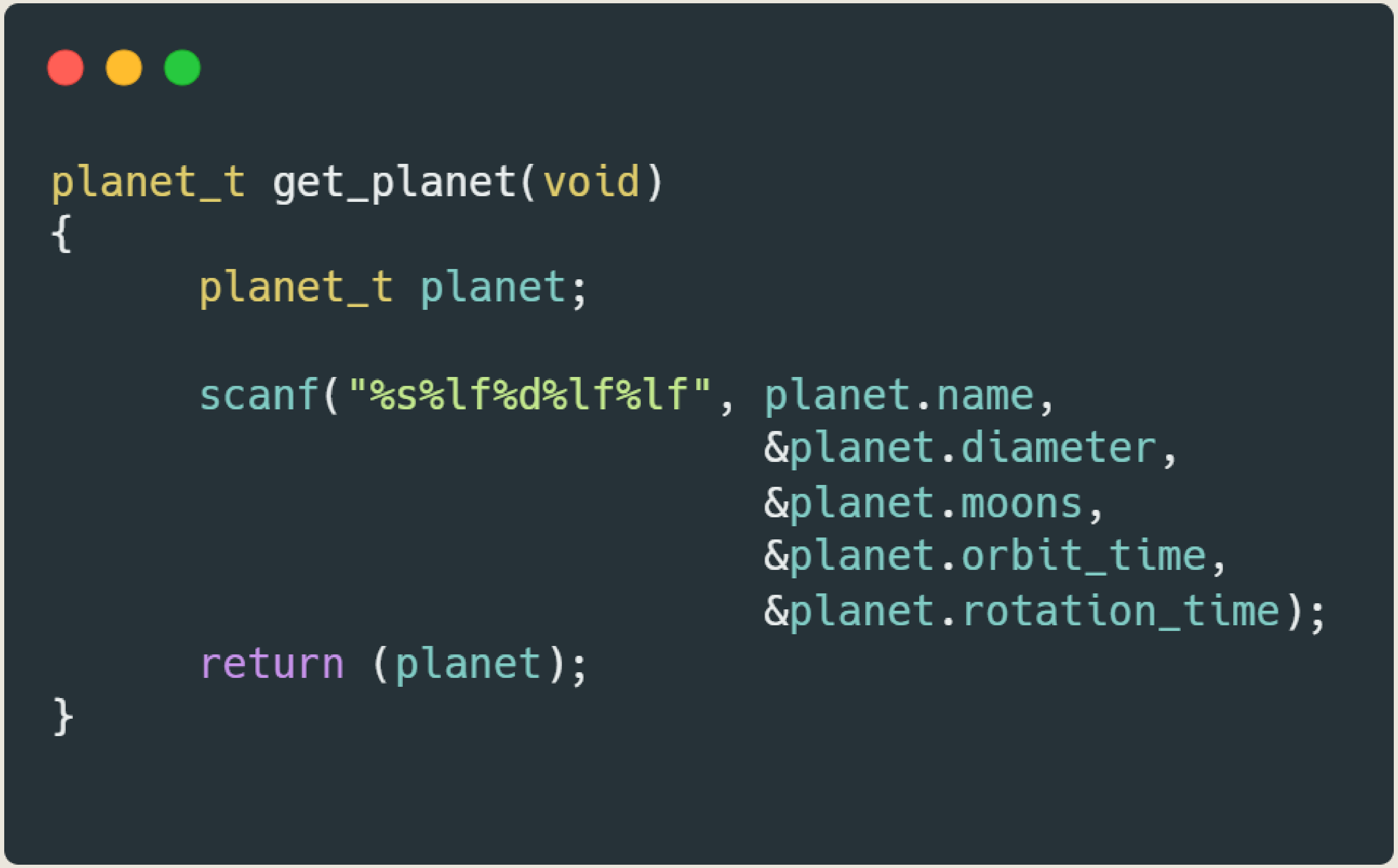


```c
scanf("%s%f%d%f%f", plnp->name,
                    &plnp->diameter,
                    &plnp->moons,
                    &plnp->orbit_time,
                    &plnp->rotation_time);
```

## 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.
- 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.

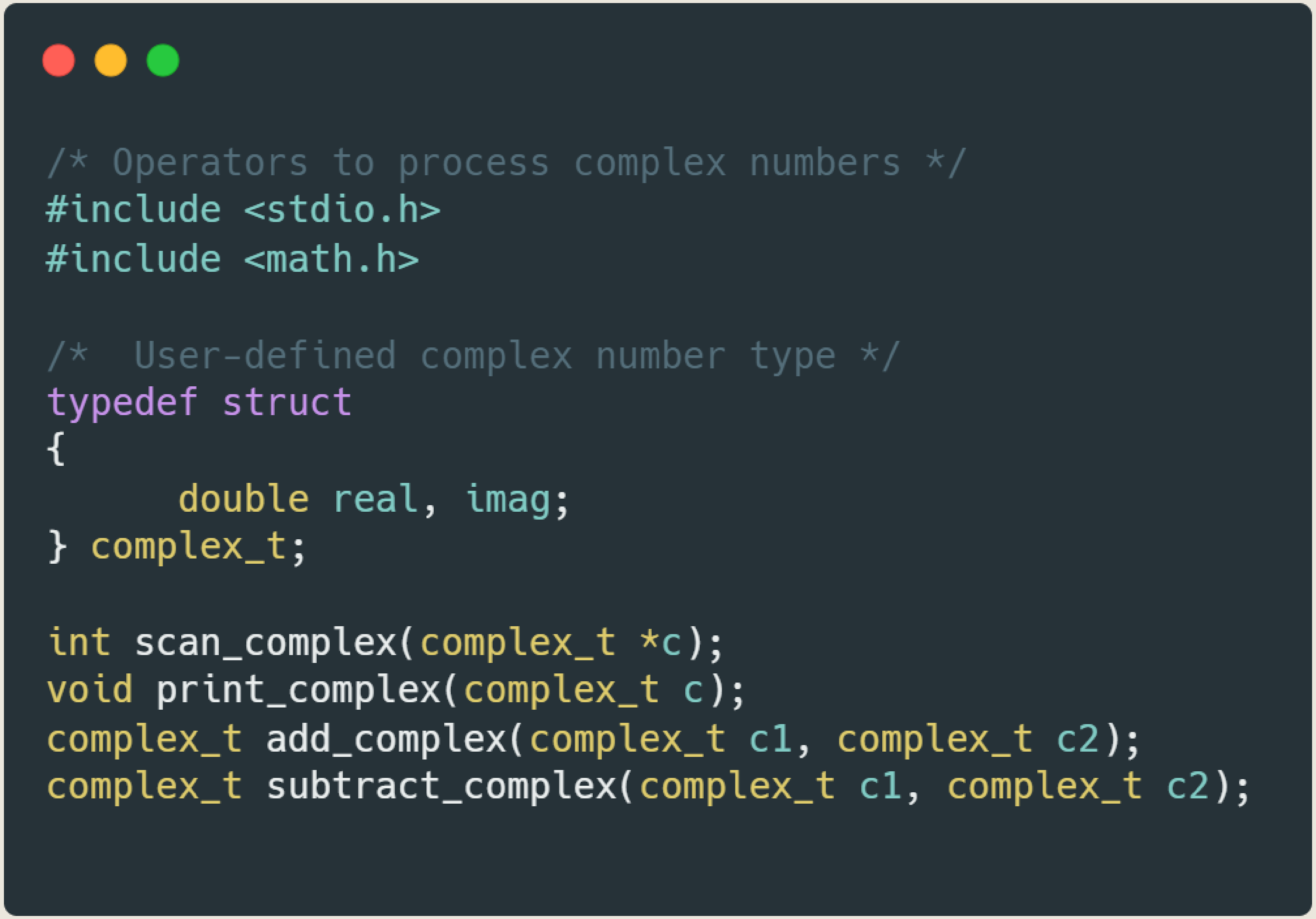
## Functions Whose Result Values are Structured



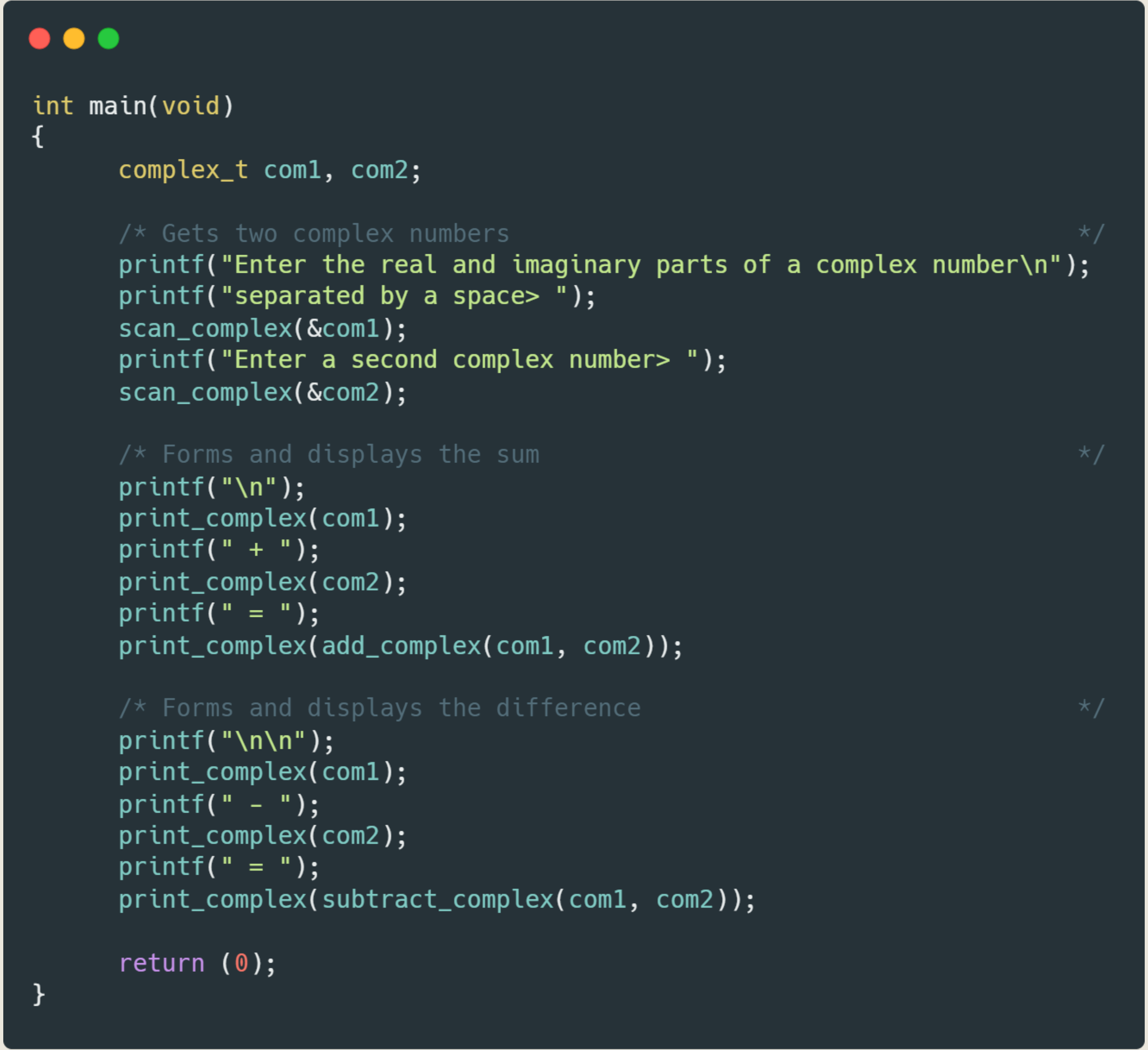
```c
planet_t get_planet(void)
{
      planet_t planet;

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

## A USER-DEFINED TYPE FOR COMPLEX NUMBERS

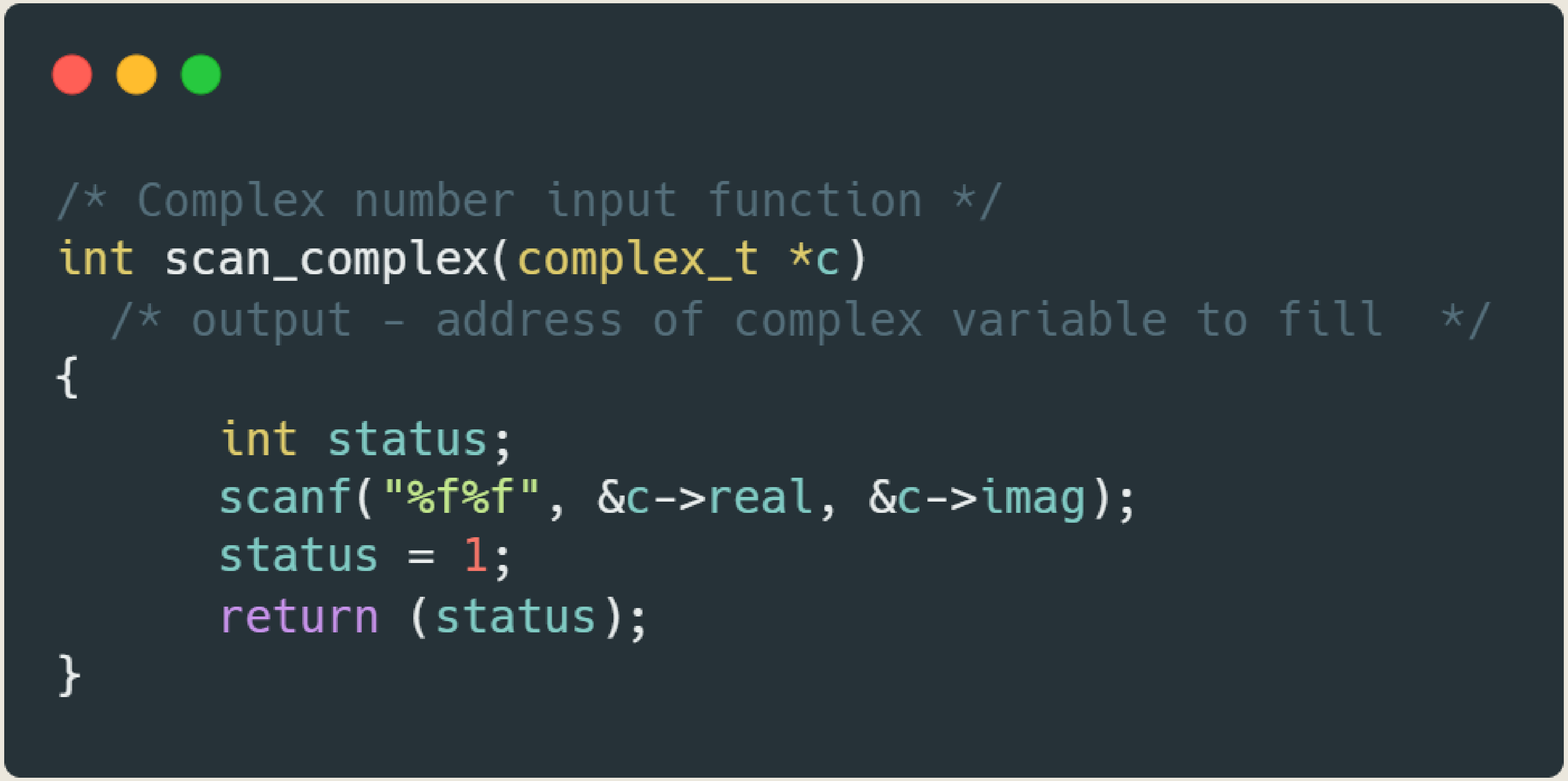
Case Study



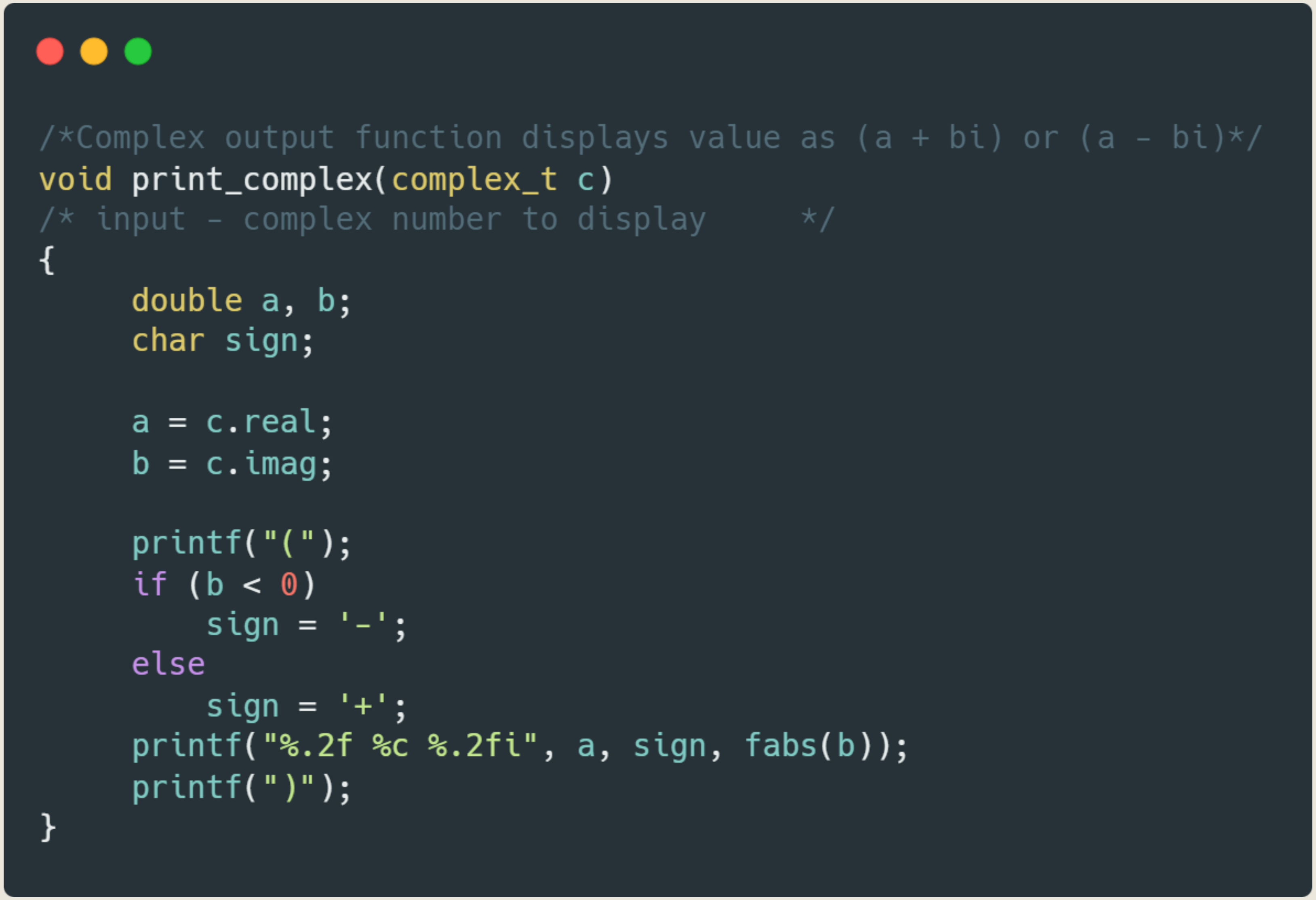
```c
/* Operators to process complex numbers */
#include <stdio.h>
#include <math.h>

/*  User-defined complex number type */
typedef struct
{
      double real, imag;
} complex_t;

int scan_complex(complex_t *c);
void print_complex(complex_t c);
complex_t add_complex(complex_t c1, complex_t c2);
complex_t subtract_complex(complex_t c1, complex_t c2);
```



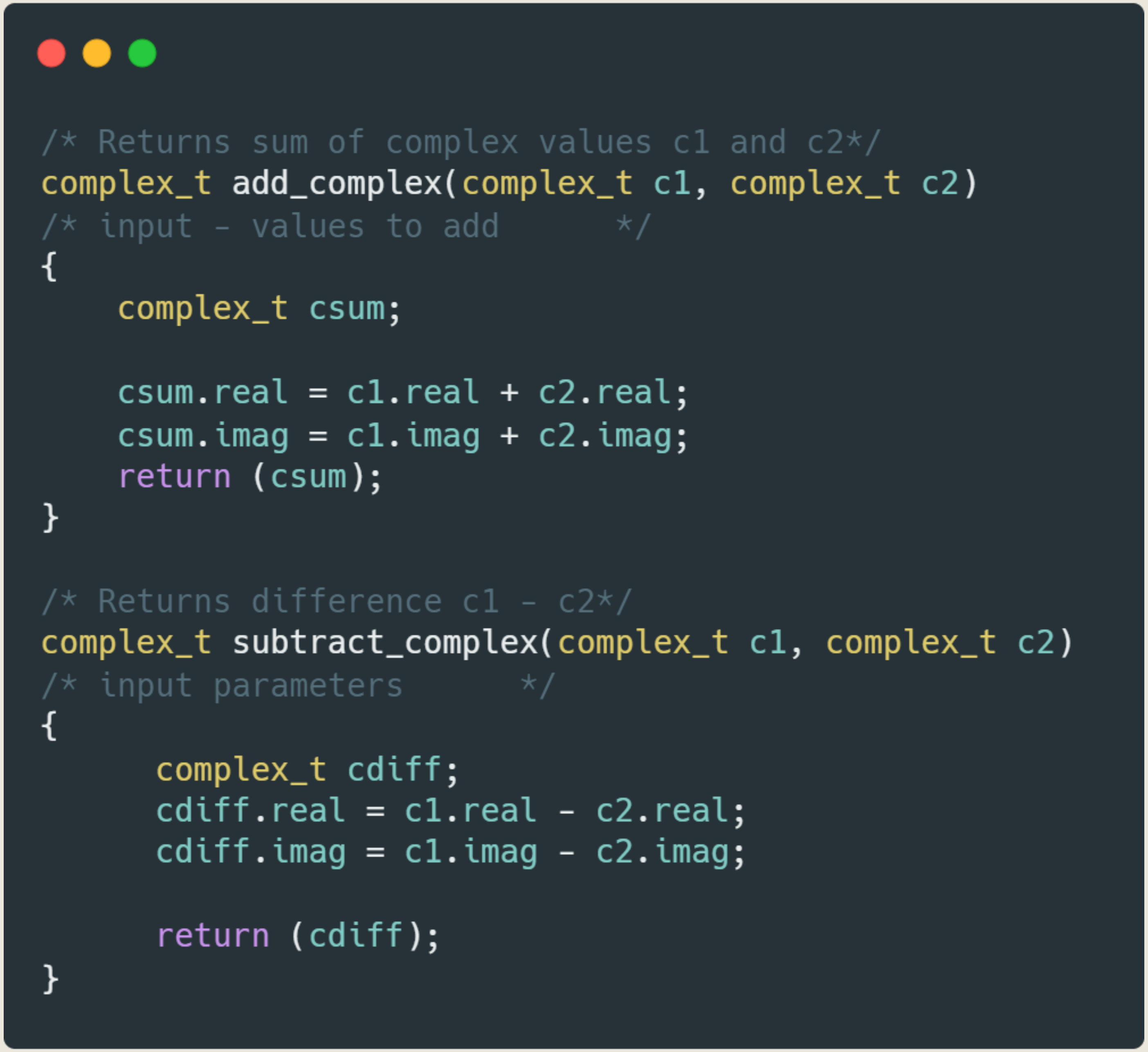
```c
int main(void)
{
      complex_t com1, com2;

      /* Gets two complex numbers                                      */
      printf("Enter the real and imaginary parts of a complex number\n");
      printf("separated by a space> ");
      scan_complex(&com1);
      printf("Enter a second complex number> ");
      scan_complex(&com2);

      /* Forms and displays the sum                                    */
      printf("\n");
      print_complex(com1);
      printf(" + ");
      print_complex(com2);
      printf(" = ");
      print_complex(add_complex(com1, com2));

      /* Forms and displays the difference                             */
      printf("\n\n");
      print_complex(com1);
      printf(" - ");
      print_complex(com2);
      printf(" = ");
      print_complex(subtract_complex(com1, com2));

      return (0);
}
```



```c
/* Complex number input function */
int scan_complex(complex_t *c)
  /* output - address of complex variable to fill  */
{
      int status;
      scanf("%f%f", &c->real, &c->imag);
      status = 1;
      return (status);
}
```



```c
/*Complex output function displays value as (a + bi) or (a - bi)*/
void print_complex(complex_t c)
/* input - complex number to display     */
{
     double a, b;
     char sign;

     a = c.real;
     b = c.imag;

     printf("(");
     if (b < 0)
         sign = '-';
     else
         sign = '+';
     printf("%.2f %c %.2fi", a, sign, fabs(b));
     printf(")");
}
```



```c
/* Returns sum of complex values c1 and c2*/
complex_t add_complex(complex_t c1, complex_t c2)
/* input - values to add      */
{
    complex_t csum;

    csum.real = c1.real + c2.real;
    csum.imag = c1.imag + c2.imag;
    return (csum);
}

/* Returns difference c1 - c2*/
complex_t subtract_complex(complex_t c1, complex_t c2)
/* input parameters      */
{
      complex_t cdiff;
      cdiff.real = c1.real - c2.real;
      cdiff.imag = c1.imag - c2.imag;

      return (cdiff);
}
```

```
Enter the real and imaginary parts of a complex number
separated by a space> 3.5 5.2
Enter a second complex number> 2.5 1.2
(3.50 + 5.20i) + (2.50 + 1.20i) = (6.00 + 6.40i)
(3.50 + 5.20i) - (2.50 + 1.20i) = (1.00 + 4.00i)
```

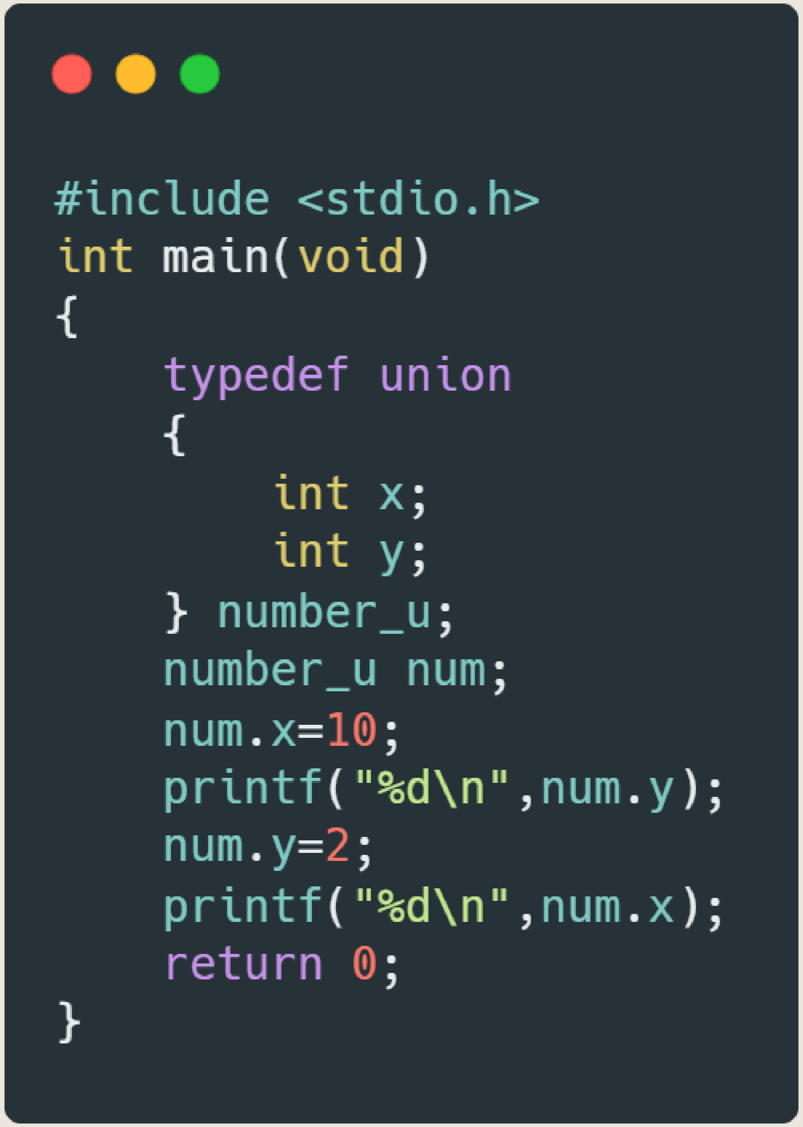
## Unions

- A union is a derived data type—like a structure—with members that share the same storage space.
- For different situations in a program, some variables may not be relevant, but other variables are—so a union shares the space instead of wasting storage on variables that are not being used.
- The members of a union can be of any data type.

## Unions

- The number of bytes used to store a union must be at least enough to hold the largest member.
- In most cases, unions contain two or more data types.
- Only one member, and thus one data type, can be referenced at a time.
- It’s your responsibility to ensure that the data in a union is referenced with the proper data type.

## Unions



```c
#include <stdio.h>
int main(void)
{
    typedef union
    {
        int x;
        int y;
    } number_u;
    number_u num;
    num.x=10;
    printf("%d\n",num.y);
    num.y=2;
    printf("%d\n",num.x);
    return 0;
}
```

# References

- Problem Solving and Program Design in C —7th ed, Jeri R. Hanly, Elliot B. Koffman
- C How to Program, 8ed, by Paul Deitel and Harvey Deitel, Pearson, 2016.