

*"C programmers never die. They are just cast into void."*

- Alan Perlis

## CSE102 Computer Programming with C

2016-2018 Spring Semester

### Structures

© 2015-2018 Yakup Genç

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

## Structures

- How to define a structure?
- How to declare a variable?
- How to manipulate individual components?
- How to manipulate whole structures?
  - Assignment

April 2018

CSE102 Lecture 10

3

## Structures

- Defines a new type
  - Represents structured collection of data
    - Different type is possible
- EX: Planet type
  - Name
  - Diameter
  - Number of moons
  - Number of years to complete one solar orbit
  - Number of hours to complete one rotation.

April 2018

CSE102 Lecture 10

2

## Structure definition

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

planet_t  my_planet;
```

April 2018

CSE102 Lecture 10

4

## Structure Definition

- 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};
```

April 2018

CSE102 Lecture 10

5

## Structure Definition (Cont'd)

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

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

April 2018

CSE102 Lecture 10

7

## Structure Definition (Cont'd)

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

<code>.name</code>	\0 ? ? ? ? ? ? ? ?
<code>.diameter</code>	0.0
<code>.moons</code>	0
<code>.orbit_time</code>	0.0
<code>.rotation_time</code>	0.0

April 2018

CSE102 Lecture 10

6

## 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`

<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

April 2018

CSE102 Lecture 10

8

## 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.

April 2018

CSE102 Lecture 10

9

## Structured Input Parameter

```
print_planet(current_planet);
```

```
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. }
```

April 2018

CSE102 Lecture 10

11

## 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.

April 2018

CSE102 Lecture 10

10

## Comparing Two Structured Values

```
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. {
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. }
```

April 2018

CSE102 Lecture 10

12

## 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", (*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. }

```

April 2018

CSE102 Lecture 10

13

## Structured Output Argument (Cont'd)

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

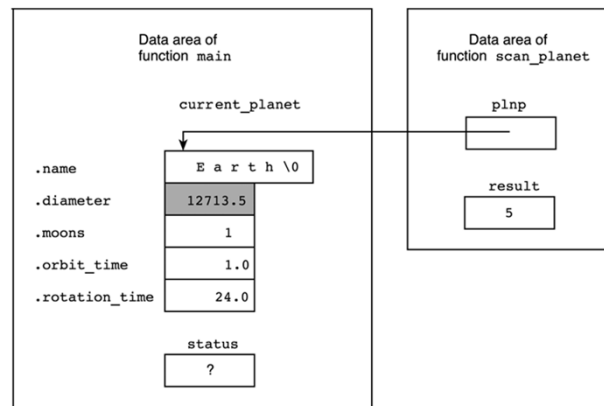
Reference	Type	Value
plnp	planet_t *	address of structure that main refers to as current_planet
*plnp	planet_t	structure that main refers to as current_planet
(*plnp).diameter	double	12713.5
&(*plnp).diameter	double *	address of colored component of structure that main refers to as current_planet

April 2018

CSE102 Lecture 10

15

status = scan\_planet(&current\_planet);



April 2018

CSE102 Lecture 10

14

## 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):
  - Follow the pointer in plnp to the structure.
  - Select the component of interest.
  - Unless this component is an array, get its address to pass to scanf.
- &\*plnp.diameter would attempt step 2 before step 1.

April 2018

CSE102 Lecture 10

16

## 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

April 2018

CSE102 Lecture 10

17

## 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.

```

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.
15.      return (planet);

```

April 2018

CSE102 Lecture 10

19

## Structure as Argument (Cont'd)

- result = scanf("%s%lf%d%lf%lf",
 

plnp->name,  
 &plnp->diameter,  
 &plnp->moons,  
 &plnp->orbit\_time,  
 &plnp->rotation\_time);

April 2018

CSE102 Lecture 10

18

## Compute an Updated Time Value

```

typedef struct {
    int hour, minute, second;
} time_t;
time_now = new_time(time_now, secs);

```

```

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. }

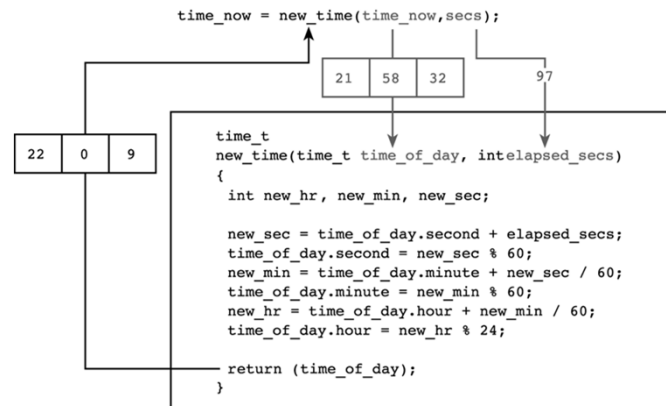
```

April 2018

CSE102 Lecture 10

20

## Structured Values as a Function Result

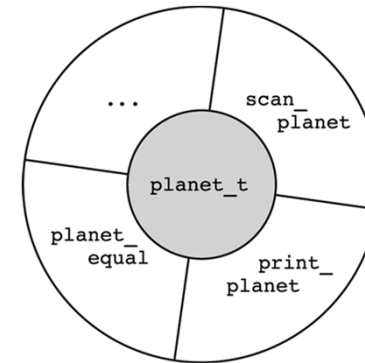


April 2018

CSE102 Lecture 10

21

## Abstract Data Type



April 2018

CSE102 Lecture 10

23

## 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.

April 2018

CSE102 Lecture 10

22

## Type and Operators for Complex Numbers

```

1. /*
2.  * Operators to process complex numbers
3.  */
4. #include <stdio.h>
5. #include <math.h>
6.
7. /* User-defined complex number type */
8. typedef struct {
9.     double real, imag;
10. } complex_t;
11.
12. int scan_complex(complex_t *c);
13. void print_complex(complex_t c);
14. complex_t add_complex(complex_t c1, complex_t c2);
15. complex_t subtract_complex(complex_t c1, complex_t c2);
16. complex_t multiply_complex(complex_t c1, complex_t c2);
17. complex_t divide_complex(complex_t c1, complex_t c2);
18. complex_t abs_complex(complex_t c);
19.

```

April 2018

CSE102 Lecture 10

24

```

21. int
22. main(void)
23. {
24.     complex_t com1, com2;
25.
26.     /* Gets two complex numbers */
27.     printf("Enter the real and imaginary parts of a complex number\n");
28.     printf("separated by a space> ");
29.     scan_complex(&com1);
30.     printf("Enter a second complex number> ");
31.     scan_complex(&com2);
32.
33.     /* Forms and displays the sum */
34.     printf("\n");
35.     print_complex(com1);
36.     printf(" + ");
37.     print_complex(com2);
38.     printf(" = ");
39.     print_complex(add_complex(com1, com2));
40.
41.     /* Forms and displays the difference */
42.     printf("\n\n");

```

(continued)

April 2018

CSE102 Lecture 10

25

```

59. /*
60.  * Complex number input function returns standard scanning error code
61.  * 1 => valid scan, 0 => error, negative EOF value => end of file
62.  */
63. int
64. scan_complex(complex_t *c) /* output - address of complex variable to
65.                             fill */
66. {
67.     int status;
68.
69.     status = scanf("%lf%lf", &c->real, &c->imag);
70.     if (status == 2)
71.         status = 1;
72.     else if (status != EOF)
73.         status = 0;
74.
75.     return (status);
76. }
77.

```

April 2018

CSE102 Lecture 10

27

```

43.     print_complex(com1);
44.     printf(" - ");
45.     print_complex(com2);
46.     printf(" = ");
47.     print_complex(subtract_complex(com1, com2));
48.
49.     /* Forms and displays the absolute value of the first number */
50.     printf("\n\n");
51.     print_complex(com1);
52.     printf("| = ");
53.     print_complex(abs_complex(com1));
54.     printf("\n");
55.
56.     return (0);
57. }
58.

```

April 2018

CSE102 Lecture 10

26

```

78. /*
79.  * Complex output function displays value as (a + bi) or (a - bi),
80.  * dropping a or b if they round to 0 unless both round to 0
81.  */
82. void
83. print_complex(complex_t c) /* input - complex number to display */
84. {
85.     double a, b;
86.     char sign;
87.
88.     a = c.real;
89.     b = c.imag;
90.
91.     printf("(");
92.
93.     if (fabs(a) < .005 && fabs(b) < .005) {
94.         printf("%.2f", 0.0);
95.     } else if (fabs(b) < .005) {
96.         printf("%.2f", a);
97.     } else if (fabs(a) < .005) {
98.         printf("%.2fi", b);
99.     } else {
100.         if (b < 0)
101.             sign = '-';
102.         else
103.             sign = '+';
104.         printf("%.2f %c %.2fi", a, sign, fabs(b));
105.     }
106.

```

April 2018

CSE102 Lecture 10

28

```

110. /*
111.  * Returns sum of complex values c1 and c2
112.  */
113. complex_t
114. add_complex(complex_t c1, complex_t c2) /* input - values to add */
115. {
116.     complex_t csum;
117.
118.     csum.real = c1.real + c2.real;
119.     csum.imag = c1.imag + c2.imag;
120.     return (csum);
121. }
122.

```

April 2018

CSE102 Lecture 10

29

```

137. /* ** Stub **
138.  * Returns product of complex values c1 and c2
139.  */
140. complex_t
141. multiply_complex(complex_t c1, complex_t c2) /* input parameters */
142. {
143.     printf("Function multiply_complex returning first argument\n");
144.     return (c1);
145. }
146.
147. /* ** Stub **
148.  * Returns quotient of complex values (c1 / c2)
149.  */
150. complex_t
151. divide_complex(complex_t c1, complex_t c2) /* input parameters */
152. {
153.     printf("Function divide_complex returning first argument\n");
154.     return (c1);
155. }
156.

```

(continued)

April 2018

CSE102 Lecture 10

31

```

124. /*
125.  * Returns difference c1 - c2
126.  */
127. complex_t
128. subtract_complex(complex_t c1, complex_t c2) /* input parameters */
129. {
130.     complex_t cdiff;
131.     cdiff.real = c1.real - c2.real;
132.     cdiff.imag = c1.imag - c2.imag;
133.
134.     return (cdiff);
135. }
136.

```

April 2018

CSE102 Lecture 10

30

```

157. /*
158.  * Returns absolute value of complex number c
159.  */
160. complex_t
161. abs_complex(complex_t c) /* input parameter */
162. {
163.     complex_t cabs;
164.
165.     cabs.real = sqrt(c.real * c.real + c.imag * c.imag);
166.     cabs.imag = 0;
167.
168.     return (cabs);
169. }

```

April 2018

CSE102 Lecture 10

32



## 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.

April 2018

CSE102 Lecture 10

33

## Universal Measurement Conversion

Data file units.dat:

miles	mi	distance	1609.3
kilometers	km	distance	1000
yards	yd	distance	0.9144
meters	m	distance	1
quarts	qt	liquid_volume	0.94635
liters	l	liquid_volume	1
gallons	gal	liquid_volume	3.7854
milliliters	ml	liquid_volume	0.001
kilograms	kg	mass	1
grams	g	mass	0.001
slugs	slugs	mass	0.14594

April 2018

CSE102 Lecture 10

35

## 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[0].gpa
stulist[1]	512984556 3.09
stulist[2]	232415569 2.98
...	...
stulist[49]	173745903 3.98

April 2018

CSE102 Lecture 10

34

## Universal Measurement Conversion

```
1. /*
2.  * Converts measurements given in one unit to any other unit of the same
3.  * category that is listed in the database file, units.dat.
4.  * Handles both names and abbreviations of units.
5.  */
6. #include <stdio.h>
7. #include <string.h>
8.
9. #define NAME_LEN 30 /* storage allocated for a unit name */
10. #define ABBREV_LEN 15 /* storage allocated for a unit abbreviation */
11. #define CLASS_LEN 20 /* storage allocated for a measurement class */
12. #define NOT_FOUND -1 /* value indicating unit not found */
13. #define MAX_UNITS 20 /* maximum number of different units handled */
14.
15. typedef struct { /* unit of measurement type */
16.     char name[NAME_LEN]; /* character string such as "milligrams" */
17.     char abbrev[ABBREV_LEN]; /* shorter character string such as "mg" */
18.     char class[CLASS_LEN]; /* character string such as "pressure",
19.                             "distance", "mass" */
20.     double standard; /* number of standard units equivalent
21.                     to this unit */
22. } unit_t;
23.
24. int fscan_unit(FILE *filep, unit_t *unitp);
25. void load_units(int unit_max, unit_t units[], int *unit_sizep);
26. int search(const unit_t units[], const char *target, int n);
27. double convert(double quantity, double old_stand, double new_stand);
28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
110.
111.
112.
113.
114.
115.
116.
117.
118.
119.
120.
121.
122.
123.
124.
125.
126.
127.
128.
129.
130.
131.
132.
133.
134.
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
1000.
1001.
1002.
1003.
1004.
1005.
1006.
1007.
1008.
1009.
1010.
1011.
1012.
1013.
1014.
1015.
1016.
1017.
1018.
1019.
1020.
1021.
1022.
1023.
1024.
1025.
1026.
1027.
1028.
1029.
1030.
1031.
1032.
1033.
1034.
1035.
1036.
1037.
1038.
1039.
1040.
1041.
1042.
1043.
1044.
1045.
1046.
1047.
1048.
1049.
1050.
1051.
1052.
1053.
1054.
1055.
1056.
1057.
1058.
1059.
1060.
1061.
1062.
1063.
1064.
1065.
1066.
1067.
1068.
1069.
1070.
1071.
1072.
1073.
1074.
1075.
1076.
1077.
1078.
1079.
1080.
1081.
1082.
1083.
1084.
1085.
1086.
1087.
1088.
1089.
1090.
1091.
1092.
1093.
1094.
1095.
1096.
1097.
1098.
1099.
1100.
1101.
1102.
1103.
1104.
1105.
1106.
1107.
1108.
1109.
1110.
1111.
1112.
1113.
1114.
1115.
1116.
1117.
1118.
1119.
1120.
1121.
1122.
1123.
1124.
1125.
1126.
1127.
1128.
1129.
1130.
1131.
1132.
1133.
1134.
1135.
1136.
1137.
1138.
1139.
1140.
1141.
1142.
1143.
1144.
1145.
1146.
1147.
1148.
1149.
1150.
1151.
1152.
1153.
1154.
1155.
1156.
1157.
1158.
1159.
1160.
1161.
1162.
1163.
1164.
1165.
1166.
1167.
1168.
1169.
1170.
1171.
1172.
1173.
1174.
1175.
1176.
1177.
1178.
1179.
1180.
1181.
1182.
1183.
1184.
1185.
1186.
1187.
1188.
1189.
1190.
1191.
1192.
1193.
1194.
1195.
1196.
1197.
1198.
1199.
1200.
1201.
1202.
1203.
1204.
1205.
1206.
1207.
1208.
1209.
1210.
1211.
1212.
1213.
1214.
1215.
1216.
1217.
1218.
1219.
1220.
1221.
1222.
1223.
1224.
1225.
1226.
1227.
1228.
1229.
1230.
1231.
1232.
1233.
1234.
1235.
1236.
1237.
1238.
1239.
1240.
1241.
1242.
1243.
1244.
1245.
1246.
1247.
1248.
1249.
1250.
1251.
1252.
1253.
1254.
1255.
1256.
1257.
1258.
1259.
1260.
1261.
1262.
1263.
1264.
1265.
1266.
1267.
1268.
1269.
1270.
1271.
1272.
1273.
1274.
1275.
1276.
1277.
1278.
1279.
1280.
1281.
1282.
1283.
1284.
1285.
1286.
1287.
1288.
1289.
1290.
1291.
1292.
1293.
1294.
1295.
1296.
1297.
1298.
1299.
1300.
1301.
1302.
1303.
1304.
1305.
1306.
1307.
1308.
1309.
1310.
1311.
1312.
1313.
1314.
1315.
1316.
1317.
1318.
1319.
1320.
1321.
1322.
1323.
1324.
1325.
1326.
1327.
1328.
1329.
1330.
1331.
1332.
1333.
1334.
1335.
1336.
1337.
1338.
1339.
1340.
1341.
1342.
1343.
1344.
1345.
1346.
1347.
1348.
1349.
1350.
1351.
1352.
1353.
1354.
1355.
1356.
1357.
1358.
1359.
1360.
1361.
1362.
1363.
1364.
1365.
1366.
1367.
1368.
1369.
1370.
1371.
1372.
1373.
1374.
1375.
1376.
1377.
1378.
1379.
1380.
1381.
1382.
1383.
1384.
1385.
1386.
1387.
1388.
1389.
1390.
1391.
1392.
1393.
1394.
1395.
1396.
1397.
1398.
1399.
1400.
1401.
1402.
1403.
1404.
1405.
1406.
1407.
1408.
1409.
1410.
1411.
1412.
1413.
1414.
1415.
1416.
1417.
1418.
1419.
1420.
1421.
1422.
1423.
1424.
1425.
1426.
1427.
1428.
1429.
1430.
1431.
1432.
1433.
1434.
1435.
1436.
1437.
1438.
1439.
1440.
1441.
1442.
1443.
1444.
1445.
1446.
1447.
1448.
1449.
1450.
1451.
1452.
1453.
1454.
1455.
1456.
1457.
1458.
1459.
1460.
1461.
1462.
1463.
1464.
1465.
1466.
1467.
1468.
1469.
1470.
1471.
1472.
1473.
1474.
1475.
1476.
1477.
1478.
1479.
1480.
1481.
1482.
1483.
1484.
1485.
1486.
1487.
1488.
1489.
1490.
1491.
1492.
1493.
1494.
1495.
1496.
1497.
1498.
1499.
1500.
1501.
1502.
1503.
1504.
1505.
1506.
1507.
1508.
1509.
1510.
1511.
1512.
1513.
1514.
1515.
1516.
1517.
1518.
1519.
1520.
1521.
1522.
1523.
1524.
1525.
1526.
1527.
1528.
1529.
1530.
1531.
1532.
1533.
1534.
1535.
1536.
1537.
1538.
1539.
1540.
1541.
1542.
1543.
1544.
1545.
1546.
1547.
1548.
1549.
1550.
1551.
1552.
1553.
1554.
1555.
1556.
1557.
1558.
1559.
1560.
1561.
1562.
1563.
1564.
1565.
1566.
1567.
1568.
1569.
1570.
1571.
1572.
1573.
1574.
1575.
1576.
1577.
1578.
1579.
1580.
1581.
1582.
1583.
1584.
1585.
1586.
1587.
1588.
1589.
1590.
1591.
1592.
1593.
1594.
1595.
1596.
1597.
1598.
1599.
1600.
1601.
1602.
1603.
1604.
1605.
1606.
1607.
1608.
1609.
1610.
1611.
1612.
1613.
1614.
1615.
1616.
1617.
1618.
1619.
1620.
1621.
1622.
1623.
1624.
1625.
1626.
1627.
1628.
1629.
1630.
1631.
1632.
1633.
1634.
1635.
1636.
1637.
1638.
1639.
1640.
1641.
1642.
1643.
1644.
1645.
1646.
1647.
1648.
1649.
1650.
1651.
1652.
1653.
1654.
1655.
1656.
1657.
1658.
1659.
1660.
1661.
1662.
1663.
1664.
1665.
1666.
1667.
1668.
1669.
1670.
1671.
1672.
1673.
1674.
1675.
1676.
1677.
1678.
1679.
1680.
1681.
1682.
1683.
1684.
1685.
1686.
1687.
1688.
1689.
1690.
1691.
1692.
1693.
1694.
1695.
1696.
1697.
1698.
1699.
1700.
1701.
1702.
1703.
1704.
1705.
1706.
1707.
1708.
1709.
1710.
1711.
1712.
1713.
1714.
1715.
1716.
1717.
1718.
1719.
1720.
1721.
1722.
1723.
1724.
1725.
1726.
1727.
1728.
1729.
1730.
1731.
1732.
1733.
1734.
1735.
1736.
1737.
1738.
1739.
1740.
1741.
1742.
1743.
1744.
1745.
1746.
1747.
1748.
1749.
1750.
1751.
1752.
1753.
1754.
1755.
1756.
1757.
1758.
1759.
1760.
1761.
1762.
1763.
1764.
1765.
1766.
1767.
1768.
1769.
1770.
1771.
1772.
1773.
1774.
1775.
1776.
1777.
1778.
1779.
1780.
1781.
1782.
1783.
1784.
1785.
1786.
1787.
1788.
1789.
1790.
1791.
1792.
1793.
1794.
1795.
1796.
1797.
1798.
1799.
1800.
1801.
1802.
1803.
1804.
1805.
1806.
1807.
1808.
1809.
1810.
1811.
1812.
1813.
1814.
1815.
1816.
1817.
1818.
1819.
1820.
1821.
1822.
1823.
1824.
1825.
1826.
1827.
1828.
1829.
1830.
1831.
1832.
1833.
1834.
1835.
1836.
1837.
1838.
1839.
1840.
1841.
1842.
1843.
1844.
1845.
1846.
1847.
1848.
1849.
1850.
1851.
1852.
1853.
1854.
1855.
1856.
1857.
1858.
1859.
1860.
1861.
1862.
1863.
1864.
1865.
1866.
1867.
1868.
1869.
1870.
1871.
1872.
1873.
1874.
1875.
1876.
1877.
1878.
1879.
1880.
1881.
1882.
1883.
1884.
1885.
1886.
1887.
1888.
1889.
1890.
1891.
1892.
18
```

## Universal Measurement Conversion

```

29. int
30. main(void)
31. {
32.     unit_t units[MAX_UNITS]; /* units classes and conversion factors*/
33.     int num_units; /* number of elements of units in use */
34.     char old_units[NAME_LEN], /* units to convert (name or abbrev) */
35.          new_units[NAME_LEN]; /* units to convert to (name or abbrev)*/
36.     int status; /* input status */
37.     double quantity; /* value to convert */
38.

```

(continued)

April 2018

CSE102 Lecture 10

37

```

54.     for (status = scanf("%lf%s%s", &quantity, old_units, new_units);
55.          status == 3;
56.          status = scanf("%lf%s%s", &quantity, old_units, new_units)) {
57.         printf("Attempting conversion of %.4f %s to %s . . .\n",
58.               quantity, old_units, new_units);
59.         old_index = search(units, old_units, num_units);
60.         new_index = search(units, new_units, num_units);
61.         if (old_index == NOT_FOUND)
62.             printf("Unit %s not in database\n", old_units);
63.         else if (new_index == NOT_FOUND)
64.             printf("Unit %s not in database\n", new_units);
65.         else if (strcmp(units[old_index].class,
66.                         units[new_index].class) != 0)
67.             printf("Cannot convert %s (%s) to %s (%s)\n",
68.                   old_units, units[old_index].class,
69.                   new_units, units[new_index].class);
70.         else
71.             printf("%.4f%s = %.4f %s\n", quantity, old_units,
72.                   convert(quantity, units[old_index].standard,
73.                           units[new_index].standard),
74.                   new_units);
75.         printf("\nEnter a conversion problem or q to quit.\n> ");
76.     }
77.     return (0);
78. }
79.
80.

```

(continued)

April 2018

CSE102 Lecture 10

39

## Universal Measurement Conversion

```

39.     int old_index, /* index of units element where
40.                  old units found */
41.          new_index; /* index where new_units found */
42.
43.     /* Load units of measurement database */
44.     load_units(MAX_UNITS, units, &num_units);
45.
46.     /* Convert quantities to desired units until data format error
47.        (including error code returned when q is entered to quit) */
48.     printf("Enter a conversion problem or q to quit.\n");
49.     printf("To convert 25 kilometers to miles, you would enter\n");
50.     printf("> 25 kilometers miles\n");
51.     printf(" or, alternatively,\n");
52.     printf("> 25 km mi\n> ");
53.

```

April 2018

CSE102 Lecture 10

38

```

81. /*
82.  * Gets data from a file to fill output argument
83.  * Returns standard error code: 1 => successful input, 0 => error,
84.  * negative EOF value => end of file
85.  */
86. int
87. fscan_unit(FILE *filep, /* input - input file pointer */
88.            unit_t *unitp) /* output - unit_t structure to fill */
89. {
90.     int status;
91.
92.     status = fscanf(filep, "%s%s%s%lf", unitp->name,
93.                     unitp->abbrev,
94.                     unitp->class,
95.                     &unitp->standard);
96.
97.     if (status == 4)
98.         status = 1;
99.     else if (status != EOF)
100.        status = 0;
101.
102.     return (status);
103. }
104.

```

April 2018

CSE102 Lecture 10

40

```

105. /*
106.  * Opens database file units.dat and gets data to place in units until end
107.  * of file is encountered. Stops input prematurely if there are more than
108.  * unit_max data values in the file or if invalid data is encountered.
109.  */
110. void
111. load_units(int unit_max, /* input - declared size of units */
112.            unit_t data, /* output - array of data */
113.            int *unit_sizep) /* output - number of data values
114.                               stored in units */
115. {
116.     FILE *inp;
117.     unit_t data;
118.     int i, status;
119.
120.     /* Gets database of units from file */
121.     inp = fopen("units.dat", "r");
122.     i = 0;
123.
124.     for (status = fscanf(inp, "%d %s", &data[i].name, &data[i].abbrev);
125.          status == 1 && i < unit_max;
126.          status = fscanf(inp, "%d %s", &data[i].name, &data[i].abbrev)) {
127.         data[i].name = data[i].name;
128.         data[i].abbrev = data[i].abbrev;
129.         i++;
130.     }
131.     fclose(inp);
132.
133.     /* Issue error message on premature exit */
134.     if (status != 1) {
135.         printf("\n*** Error in data format ***\n");
136.         printf("Using first %d data values ***\n", i);
137.     } else if (status != EOF) {
138.         printf("\n*** Error: too much data in file ***\n");
139.         printf("Using first %d data values ***\n", i);
140.     }
141.
142.     /* Send back size of used portion of array */
143.     *unit_sizep = i;

```

April 2018

CSE102 Lecture 10

41

```

170. /*
171.  * Converts one measurement to another given the representation of both
172.  * in a standard unit. For example, to convert 24 feet to yards given a
173.  * standard unit of inches: quantity = 24, old_stand = 12 (there are 12
174.  * inches in a foot), new_stand = 36 (there are 36 inches in a yard),
175.  * result is 24 * 12 / 36 which equals 8
176.  */
177. double
178. convert(double quantity, /* value to convert */
179.         double old_stand, /* number of standard units in one of
180.                             quantity's original units */
181.         double new_stand) /* number of standard units in 1 new unit */
182. {
183.     return (quantity * old_stand / new_stand);
184. }

```

April 2018

CSE102 Lecture 10

43

```

144. /*
145.  * Searches for target key in name and abbrev components of first n
146.  * elements of array units
147.  * Returns index of structure containing target or NOT_FOUND
148.  */
149. int
150. search(const unit_t units[], /* array of unit_t structures to search */
151.        const char *target, /* key searched for in name and abbrev
152.                             components */
153.        int n) /* number of array elements to search */
154. {
155.     int i,
156.         found = 0, /* whether or not target has been found */
157.         where; /* index where target found or NOT_FOUND */
158.
159.     /* Compare name and abbrev components of each element to target */
160.     i = 0;
161.     while (!found && i < n) {
162.         if (strcmp(units[i].name, target) == 0 ||
163             strcmp(units[i].abbrev, target) == 0)
164.             found = 1;
165.         else
166.             ++i;
167.     }
168.
169.     /* Return index of element containing target or NOT_FOUND */
170.     if (found)
171.         where = i;
172.     else
173.         where = NOT_FOUND;
174.     return (where);
175. }

```

April 2018

CSE102 Lecture 10

42

Sample run:

Enter a conversion problem or q to quit.

To convert 25 kilometers to miles, you would enter

> 25 kilometers miles

or, alternatively,

> 25 km mi

> 450 km miles

Attempting conversion of 450.0000 km to miles . . .

450.0000km = 279.6247 miles

Enter a conversion problem or q to quit.

> 2.5 qt l

Attempting conversion of 2.5000 qt to l . . .

2.5000qt = 2.3659 l

Enter a conversion problem or q to quit.

> 100 meters gallons

Attempting conversion of 100.0000 meters to gallons . . .

Cannot convert meters (distance) to gallons (liquid\_volume)

Enter a conversion problem or q to quit.

> 1234 mg g

Attempting conversion of 1234.0000 mg to g . . .

Unit mg not in database

Enter a conversion problem or q to quit.

> q

April 2018

CSE102 Lecture 10

44

## Union Types

**Union:** Data object that can be interpreted in a variety of ways

- EX: a number can be real number (double) or an integer (int)

- Allows one chunk of memory to be interpreted in multiple ways

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

- *hair\_data* does not contain both *wears\_wig* and *color* components, but *either* a *wears\_wig* component referenced by *hair\_data.wears\_wig*, or a *color* component referenced by *hair\_data.color*.

- The amount of memory is determined by the largest component of the union.

- How to determine interpretation?
  - How to determine whether to use *wears\_wig* or *color*?

April 2018

CSE102 Lecture 10

45

## 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.

April 2018

CSE102 Lecture 10

47

## Union Types

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

```
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*?

April 2018

CSE102 Lecture 10

46

## Displays a Structure with a Union

```
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. }
```

April 2018

CSE102 Lecture 10

48

## Two Interpretations of Parameter hair

	Parameter hair: View 1			Parameter hair: View 2	
.bald	1		.bald	0	
.h.wears_wig	0	????????	.h.color	r e d d i s h b l o n d \0	

April 2018

CSE102 Lecture 10

49

## Compute Area and Perimeter

```

28. /* Type of a structure that can be interpreted a different way for
29.    each shape
30.    typedef union {
31.        circle_t    circle;
32.        rectangle_t rectangle;
33.        square_t    square;
34.    } figure_data_t;
35.
36. /* Type containing a structure with multiple interpretations along with
37.    * a component whose value indicates the current valid interpretation
38.    typedef struct {
39.        char        shape;
40.        figure_data_t fig;
41.    } figure_t;
42.
43. figure_t get_figure_dimensions(void);
44. figure_t compute_area(figure_t object);
45. figure_t compute_perim(figure_t object);
46. void print_figure(figure_t object);
47.

```

April 2018

CSE102 Lecture 10

51

## Compute Area and Perimeter

```

1. /*
2.  * Computes the area and perimeter of a variety of geometric figures.
3.  */
4.
5. #include <stdio.h>
6. #define PI 3.14159
7.
8. /* Types defining the components needed to represent each shape.
9.  */
10. typedef struct {
11.     double area,
12.     double circumference,
13.     double radius;
14. } circle_t;
15.
16. typedef struct {
17.     double area,
18.     double perimeter,
19.     double width,
20.     double height;
21. } rectangle_t;
22.
23. typedef struct {
24.     double area,
25.     double perimeter,
26.     double side;
27. } square_t;

```

April 2018

CSE102 Lecture 10

50

```

48. int
49. main(void)
50. {
51.     figure_t onefig;
52.
53.     printf("Area and Perimeter Computation Program\n");
54.
55.     for (onefig = get_figure_dimensions();
56.          onefig.shape != 'Q';
57.          onefig = get_figure_dimensions()) {
58.         onefig = compute_area(onefig);
59.         onefig = compute_perim(onefig);
60.         print_figure(onefig);
61.     }
62.
63.     return (0);
64. }
65.
66. /*
67.  * Prompts for and stores the dimension data necessary to compute a
68.  * figure's area and perimeter. Figure returned contains a 'Q' in the
69.  * shape component when signaling end of data.
70.  */
71. figure_t
72. get_figure_dimensions(void)
73. {
74.     figure_t object;

```

(continued)

April 2018

CSE102 Lecture 10

52

```

75.     printf("Enter a letter to indicate the object shape or Q to quit.\n");
76.     printf("C (circle), R (rectangle), or S (square)> ");
77.     object.shape = getchar();
78.
79.     switch (object.shape) {
80.     case 'C':
81.     case 'c':
82.         printf("Enter radius> ");
83.         scanf("%lf", &object.fig.circle.radius);
84.         break;
85.
86.     case 'R':
87.     case 'r':
88.         printf("Enter height> ");
89.         scanf("%lf", &object.fig.rectangle.height);
90.         printf("Enter width> ");
91.         scanf("%lf", &object.fig.rectangle.width);
92.         break;
93.
94.     case 'S':
95.     case 's':
96.         printf("Enter length of a side> ");
97.         scanf("%lf", &object.fig.square.side);
98.         break;
99.
100.    default: /* Error is treated as a QUIT */
101.        object.shape = 'Q';
102.    }
103.
104.    return (object);
105. }

```

April 2018

CSE102 Lecture 10

53

```

107. /*
108.  * Computes the area of a figure given relevant dimensions. Returns
109.  * figure with area component filled.
110.  * Pre: value of shape component is one of these letters: CcRrSs
111.  *       necessary dimension components have values
112.  */
113. figure_t
114. compute_area(figure_t object)
115. {
116.     switch (object.shape) {
117.     case 'C':
118.     case 'c':
119.         object.fig.circle.area = PI * object.fig.circle.radius *
120.                                     object.fig.circle.radius;
121.         break;
122.
123.     case 'R':
124.     case 'r':
125.         object.fig.rectangle.area = object.fig.rectangle.height *
126.                                     object.fig.rectangle.width;
127.         break;
128.
129.     case 'S':
130.     case 's':
131.         object.fig.square.area = object.fig.square.side *
132.                                    object.fig.square.side;
133.         break;
134.
135.     default:
136.         printf("Error in shape code detected in compute_area\n");
137.     }
138.
139.     return (object);
140. }

```

April 2018

CSE102 Lecture 10

54