

Chapter 6

Modular Programming

Figure 6.1 Function separate

```

1  /*
2  * Separates a number into three parts: a sign (+, -, or blank),
3  * a whole number magnitude, and a fractional part.
4  */
5  void
6  separate(double num, /* input - value to be split          */
7  char *signp, /* output - sign of num                    */
8  int *wholep, /* output - whole number magnitude of num  */
9  double *fracp) /* output - fractional part of num        */
10 {
11     double magnitude; /* local variable - magnitude of num */
12
13     /* Determines sign of num */
14     if (num < 0)
15         *signp = '-';
16     else if (num == 0)
17         *signp = ' ';
18     else
19         *signp = '+';
20
21     /* Finds magnitude of num (its absolute value) and
22     separates it into whole and fractional parts          */
23     magnitude = fabs(num);
24     *wholep = floor(magnitude);
25     *fracp = magnitude - *wholep;
26 }

```

Figure 6.2 Diagram of Function separate with Multiple Results

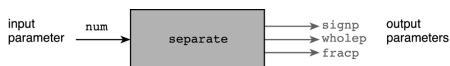


Figure 6.3 Program That Calls a Function with Output Arguments

```

1  /*
2  * Demonstrates the use of a function with input and output parameters.
3  */
4
5  #include <stdio.h>
6  #include <math.h>

```

(continued)

Figure 6.3 Program That Calls a Function with Output Arguments (cont'd)

```

7  void separate(double num, char *signp, int *wholep, double *fracp);
8
9  int
10 main(void)
11 {
12     double value; /* input - number to analyze          */
13     char sig; /* output - sign of value                */
14     int whol; /* output - whole number magnitude of value */
15     double fr; /* output - fractional part of value      */
16
17     /* Data data */
18     printf("Enter a value to analyze: ");
19     scanf("%lf", &value);
20
21     /* Separates data value into three parts
22     separate(value, &sig, &whol, &fr);
23
24     /* Prints results
25     printf("Parts of %.4f\n sign: %c\n", value, sig);
26     printf(" whole number magnitude: %d\n", whol);
27     printf(" fractional part: %.4f\n", fr);
28
29     return (0);
30 }
31
32 /*
33 * separate a number into three parts: a sign (+, -, or blank),
34 * a whole number magnitude, and a fractional part.
35 * Pre: num (as defined) signp, wholep, and fracp contain addresses of memory
36 * cells whose contents are to be stored
37 * Post: function results are stored in cells pointed to by signp, wholep, and
38 * fracp
39 */
40 void
41 separate(double num, /* input - value to be split          */
42 char *signp, /* output - sign of num                    */
43 int *wholep, /* output - whole number magnitude of num  */
44 double *fracp) /* output - fractional part of num        */
45 {
46     double magnitude; /* local variable - magnitude of num */

```

(continued)

Figure 6.3 Program That Calls a Function with Output Arguments (cont'd)

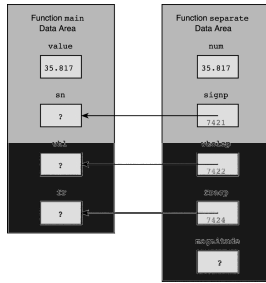
```

47     /* Determines sign of num */
48     if (num < 0)
49         *signp = '-';
50     else if (num == 0)
51         *signp = ' ';
52     else
53         *signp = '+';
54
55     /* Finds magnitude of num (its absolute value) and separates it into
56     whole and fractional parts          */
57     magnitude = fabs(num);
58     *wholep = floor(magnitude);
59     *fracp = magnitude - *wholep;
60 }

```

Enter a value to analyze> 35.817
Parts of 35.8170
sign: +
whole number magnitude: 35
fractional part: 0.8170

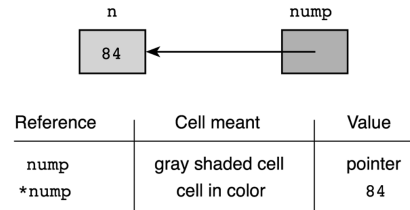
Figure 6.4 Parameter Correspondence for separate(value, &sn, &whl, &fr);



Copyright ©2004 Pearson Addison-Wesley. All rights reserved.

6-7

Figure 6.5 Comparison of Direct and Indirect Reference



Copyright ©2004 Pearson Addison-Wesley. All rights reserved.

6-8

Figure 6.6 Program to Sort Three Numbers

```

1  /*
2  * Tests function order by ordering three numbers
3  */
4  #include <stdio.h>
5  void order(double *amp, double *lpp);
6
7  int
8  main(void)
9  {
10     double num1, num2, num3; /* three numbers to put in order */
11
12     /* Gets test data */
13     printf("Enter three numbers separated by blanks ");
14     scanf("%lf%lf%lf", &num1, &num2, &num3);
15
16     /* Orders the three numbers */
17     order(&num1, &num3);
18     order(&num1, &num3);
19     order(&num2, &num3);
20     order(&num2, &num3);
21
22     /* Displays results */
23     printf("The numbers in ascending order are: %2f %2f %2f\n",
24           num1, num2, num3);
25
26     return (0);
27 }
28

```

(continued)

Copyright ©2004 Pearson Addison-Wesley. All rights reserved.

6-9

Figure 6.6 Program to Sort Three Numbers (cont'd)

```

29  /*
30  * Arranges arguments in ascending order.
31  * Pre:  amp and lpp are addresses of defined type double variables.
32  * Post: variable pointed to by amp contains the smaller of the type
33  *       double values; variable pointed to by lpp contains the larger
34  */
35  void
36  order(double *amp, double *lpp) /* input/output */
37  {
38     double temp; /* temporary variable to hold one number during swap */
39     /* Compares values pointed to by amp and lpp and switches if necessary */
40     if (*amp > *lpp) {
41         temp = *amp;
42         *amp = *lpp;
43         *lpp = temp;
44     }
45 }
46

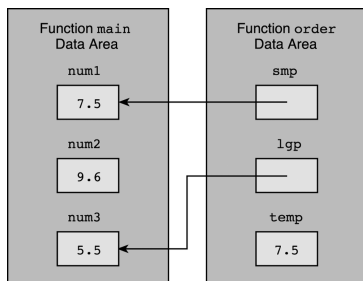
```

Enter three numbers separated by blanks> 7.5 9.6 5.5
The numbers in ascending order are: 5.50 7.50 9.60

Copyright ©2004 Pearson Addison-Wesley. All rights reserved.

6-10

Figure 6.7 Data Areas After temp = *smp; During Call order(&num1, &num3);



Copyright ©2004 Pearson Addison-Wesley. All rights reserved.

6-11

Figure 6.8 Outline of Program for Studying Scope of Names

```

1  #define MAX 950
2  #define LIMIT 200
3
4  void one(int anarg, double second); /* prototype 1 */
5
6  int fun_two(int one, char anarg); /* prototype 2 */
7
8  int
9  main(void)
10 {
11     int localvar;
12     . . .
13 } /* end main */
14
15
16 void
17 one(int anarg, double second) /* header 1 */
18 {
19     int oneLocal; /* local 1 */
20     . . .
21 } /* end one */
22
23
24 int
25 fun_two(int one, char anarg) /* header 2 */
26 {
27     int localvar; /* local 2 */
28     . . .
29 } /* end fun_two */

```

Copyright ©2004 Pearson Addison-Wesley. All rights reserved.

6-12

Figure 6.9 Function scan_fraction (incomplete)

```

1. /*
2.  * Gets and returns a valid fraction as its result
3.  * A valid fraction is of this form: integer/positive integer
4.  * Pre : none
5.  */
6. void
7. scan_fraction(int *nump, int *denomp)
8. {
9.     char slash; /* character between numerator and denominator */
10.    int status; /* status code returned by scanf indicating
11.               number of valid values obtained */
12.    int error; /* flag indicating presence of an error */
13.    char discard; /* unprocessed character from input line */
14.    do {
15.        /* No errors detected yet */
16.        error = 0;
17.

```

(continued)

Figure 6.9 Function scan_fraction (incomplete) (cont'd)

```

18.     /* Get a fraction from the user */
19.     printf("Enter a common fraction as two integers separated ");
20.     printf("by a slash ");
21.     status = scanf("%d %c %d", &num, &discard, &denom);
22.
23.     /* Validate the fraction */
24.     if (status < 3) {
25.         error = 1;
26.         printf("Invalid-please read directions carefully\n");
27.     } else if (slash != '/') {
28.         error = 1;
29.         printf("Invalid-separate numerator and denominator");
30.         printf(" by a slash (/)\n");
31.     } else if (*denomp <= 0) {
32.         error = 1;
33.         printf("Invalid-denominator must be positive\n");
34.     }
35.
36.     /* Discard extra input characters */
37.     do {
38.         scanf("%c", &discard);
39.     } while (discard != '\n');
40.     while (error);
41. }

```

Figure 6.10 Data Areas for scan_fraction and Its Caller

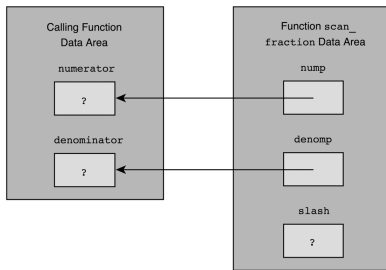


Figure 6.11 Structure Chart for Common Fraction Problem

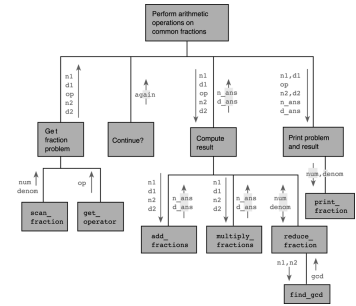


Figure 6.12 Program to Perform Arithmetic Operations on Common Fractions

```

1. /*
2.  * Adds, subtracts, multiplies and divides common fractions, displaying
3.  * results in reduced form.
4.  */
5. #include <stdio.h>
6. #include <stdlib.h> /* provides function abs */
7. /* Function prototypes */
8. void scan_fraction(int *nump, int *denomp);
9. char get_operator(void);
10. void add_fractions(int n1, int d1, int n2, int d2,
11.                  int *nump, int *denomp);
12. void multiply_fractions(int n1, int d1, int n2, int d2,
13.                       int *nump, int *denomp);
14. void find_gcd(int n1, int d2);
15. void reduce_fraction(int *nump, int *denomp);
16. void print_fraction(int num, int denom);
17. int
18. main(void)
19. {
20.     int n1, d1; /* numerator, denominator of first fraction */
21.     int n2, d2; /* numerator, denominator of second fraction */
22.     char op; /* arithmetic operator: +, -, *, / */
23.     char again; /* y or n depending on user's desire to continue */
24.     int num, den; /* numerator, denominator of answer */
25.

```

(continued)

Figure 6.12 Program to Perform Arithmetic Operations on Common Fractions (cont'd)

```

26.     /* While the user wants to continue, gets and solves arithmetic
27.     problem with common fractions */
28.     do {
29.         /* Get fraction problem */
30.         scan_fraction(&n1, &d1);
31.         op = get_operator();
32.         scan_fraction(&n2, &d2);
33.
34.         /* Computes the result */
35.         switch (op) {
36.             case '+':
37.                 add_fractions(&n1, &d1, &n2, &d2, &num, &den);
38.                 break;
39.             case '-':
40.                 subtract_fractions(&n1, &d1, &n2, &d2, &num, &den);
41.                 break;
42.             case '*':
43.                 multiply_fractions(&n1, &d1, &n2, &d2, &num, &den);
44.                 break;
45.             case '/':
46.                 divide_fractions(&n1, &d1, &n2, &d2, &num, &den);
47.                 break;
48.         }
49.         /* Displays problem and result */
50.         printf("\n");
51.         printf(" %d / %d * %d / %d = %d / %d\n", n1, d1, n2, d2, num, den);
52.         printf(" %d / %d + %d / %d = %d / %d\n", n1, d1, n2, d2, num, den);
53.         printf(" %d / %d - %d / %d = %d / %d\n", n1, d1, n2, d2, num, den);
54.         printf(" %d / %d * %d / %d = %d / %d\n", n1, d1, n2, d2, num, den);
55.
56.         /* Asks user about doing another problem */
57.         printf("Would another problem (y/n)? ");
58.         scanf("%c", &again);
59.         while (again == 'y' || again == 'Y')
60.             continue;
61.         return 0;
62.     }

```

(continued)

Figure 6.12
Program to Perform Arithmetic Operations on Common Fractions (cont'd)

```

75. /* Insert function scan_fraction from Fig. 6.9 here. */
76.
77. /*
78.  * Gets and returns a valid arithmetic operator. Skips over newline
79.  * characters and permits reentry of operator in case of error.
80.  */
81. char
82. get_operator(void)
83. {
84.     char op;
85.
86.     printf("Enter an arithmetic operator (+, -, *, or /)\n");
87.     for (scanf("%c", &op);
88.          op != '+' || op != '-' || op != '*' || op != '/';
89.          scanf("%c", &op))
90.         continue;
91.     if (op != '\n')
92.         printf("Invalid, reenter operator (+, -, *, or /)\n");
93.     return (op);
94. }
95.
96. /*
97.  * Adds fractions represented by pairs of integers.
98.  * Pre: n1, d1, n2, d2 are defined.
99.  * Post: sum of n1/d1 and n2/d2 is stored in variables pointed
100.  * to by n_ansp and d_ansp. Result is not reduced.
101.  */
102. void
103. add_fractions(int n1, int d1, /* input - first fraction */
104.              int n2, int d2, /* input - second fraction */
105.              int *n_ansp, int *d_ansp) /* output - sum of 2 fractions */
106. {
107.     int denom; /* common denominator used for sum (may not be least) */
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. }

```

Figure 6.12
Program to Perform Arithmetic Operations on Common Fractions (cont'd)

```

110. numer; /* numerator of sum */
111. sign_factor = -1 for a negative, 1 otherwise
112.
113. /* Finds a common denominator
114.  * denom = d1 * d2
115.  */
116. int
117. find_common_denom(int d1, int d2)
118. {
119.     return (d1 * d2);
120. }
121.
122. /* Computes numerator
123.  * numer = n1 * d2 + n2 * d1
124.  */
125. int
126. compute_numerator(int n1, int d1, int n2, int d2)
127. {
128.     return (n1 * d2 + n2 * d1);
129. }
130.
131. /* Adjusts sign (at most, numerator should be negative)
132.  * sign_factor = 1;
133.  * sign_factor = -1;
134.  */
135. void
136. adjust_sign(int *numer, int *denom)
137. {
138.     if (*denom < 0)
139.         *numer = -(*numer);
140.     *denom = abs(*denom);
141. }
142.
143. /* Returns result
144.  * n_ansp = numer;
145.  * d_ansp = denom;
146.  */
147. void
148. multiply_fractions(int n1, int d1, /* input - first fraction */
149.                  int n2, int d2, /* input - second fraction */
150.                  int *n_ansp, /* output - */
151.                  int *d_ansp) /* product of 2 fractions */
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. }

```

Figure 6.12
Program to Perform Arithmetic Operations on Common Fractions (cont'd)

```

160. {
161.     /* Displays trace message
162.      * printf("Entering multiply_fractions with\n");
163.      * printf("n1 = %d, d1 = %d, n2 = %d, d2 = %d\n", n1, d1, n2, d2);
164.      */
165.     /* Defines output arguments
166.      * n_ansp = 1;
167.      * d_ansp = 1;
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
```

Figure 6.15 Driver for Function `scan_fraction`

```
1. /* Driver for scan_fraction */
2.
3. int
4. main(void)
5. {
6.     int num, denom;
7.     printf("To quit, enter a fraction with a zero numerator\n");
8.     scan_fraction(&num, &denom);
9.     while (num != 0) {
10.        printf("Fraction is %d/%d\n", num, denom);
11.        scan_fraction(&num, &denom);
12.    }
13.    return (0);
14. }
15.
```