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   sn;    /* output - sign of value                            */
14.      int    whl;   /* output - whole number magnitude of value          */
15.      double fr;    /* output - fractional part of value                 */
16.
17.      /* Gets data                                                         */
18.      printf("Enter a value to analyze> ");
19.      scanf("%lf", &value);
20.
21.      /* Separates data value into three parts                             */
22.      separate(value, &sn, &whl, &fr);
23.
24.      /* Prints results                                                    */
25.      printf("Parts of %.4f\n  sign: %c\n", value, sn);
26.      printf("  whole number magnitude: %d\n", whl);
27.      printf("  fractional part:  %.4f\n", fr);
28.
29.      return (0);
30. }
31.
32. /*
33.  * Separates a number into three parts: a sign (+, -, or blank),
34.  * a whole number magnitude, and a fractional part.
35.  * Pre:  num is defined; signp, wholep, and fracp contain addresses of memory
36.  *       cells where results 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);

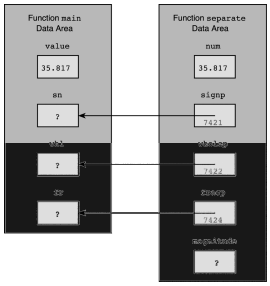
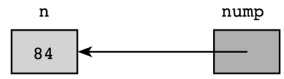


Figure 6.5 Comparison of Direct and Indirect Reference



Reference	Cell meant	Value
nump	gray shaded cell	pointer
*nump	cell in color	84

Figure 6.6 Program to Sort Three Numbers

```
/*
 * Tests function order by ordering three numbers
 */
#include <stdio.h>

void order(double *smp, double *lgp);

int
main(void)
{
        double num1, num2, num3; /* three numbers to put in order              */

        /* Gets test data                                                       */
        printf("Enter three numbers separated by blanks> ");
        scanf("%lf%lf%lf", &num1, &num2, &num3);

        /* Orders the three numbers                                             */
        order(&num1, &num2);
        order(&num1, &num3);
        order(&num2, &num3);

        /* Displays results                                                     */
        printf("The numbers in ascending order are: %.2f %.2f %.2f\n",
               num1, num2, num3);

        return (0);
}
```

(continued)

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

```
/*
 * Arranges arguments in ascending order.
 * Pre:  smp and lgp are addresses of defined type double variables
 * Post: variable pointed to by smp contains the smaller of the type
 *       double values; variable pointed to by lgp contains the larger
 */
void
order(double *smp, double *lgp)     /* input/output */
{
        double temp; /* temporary variable to hold one number during swap       */
        /* Compares values pointed to by smp and lgp and switches if necessary  */
        if (*smp > *lgp) {
                temp = *smp;

                *smp = *lgp;
                *lgp = temp;
        }
}

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


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

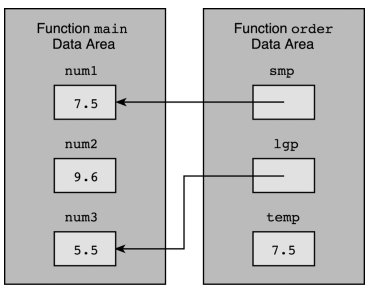


Figure 6.8 Outline of Program for Studying Scope of Names

```
#define MAX  950
#define LIMIT 200

void one(int anarg, double second);     /* prototype 1 */

int fun_two(int one, char anarg);       /* prototype 2 */

int
main(void)
{
        int localvar;
        . . .
} /* end main */


void
one(int anarg, double second)           /* header 1    */
{
        int onelocal;                   /* local 1     */
        . . .
} /* end one */


int
fun_two(int one, char anarg)            /* header 2    */
{
        int localvar;                   /* local 2     */
        . . .
} /* end fun_two */
```

Figure 6.9 Function scan_fraction (incomplete)

```
/*
 * Gets and returns a valid fraction as its result
 * A valid fraction is of this form: integer/positive integer
 * Pre : none
 */
void
scan_fraction(int *nump, int *denomp)
{
      char slash;    /* character between numerator and denominator          */
      int  status;   /* status code returned by scanf indicating
                        number of valid values obtained                      */
      int  error;    /* flag indicating presence of an error                 */
      char discard;  /* unprocessed character from input line                */
      do {
          /* No errors detected yet                                          */
          error = 0;

                                                              (continued)
```


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

```
          /* Get a fraction from the user                                    */
          printf("Enter a common fraction as two integers separated ");
          printf("by a slash> ");
          status = scanf("%d %c%d",_______, _______, _______);

          /* Validate the fraction                                           */
          if (status < 3) {
                error = 1;
                printf("Invalid-please read directions carefully\n");
          } else if (slash != '/') {
                error = 1;
                printf("Invalid-separate numerator and denominator");
                printf(" by a slash (/)\n");
          } else if (*denomp <= 0) {
                error = 1;
                printf("Invalid-denominator must be positive\n");
          }

          /* Discard extra input characters                                  */
          do {
              scanf("%c", &discard);
          } while (discard != '\n');
      } while (error);
}
```


Figure 6.10 Data Areas for scan_fraction and Its Caller

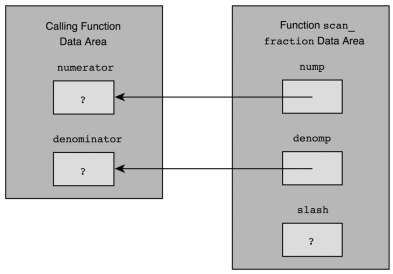


Figure 6.11 Structure Chart for Common Fraction Problem

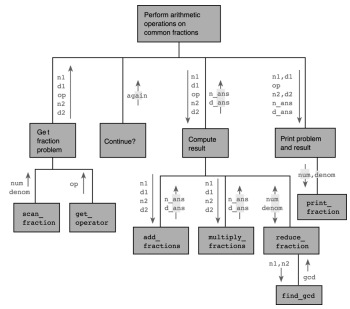


Figure 6.12 Program to Perform Arithmetic Operations on Common Fractions

```
/*
 * Adds, subtracts, multiplies and divides common fractions, displaying
 * results in reduced form.
 */

#include <stdio.h>
#include <stdlib.h>   /* provides function aba */

/* Function prototypes */
void scan_fraction(int *nump, int *denomp);

char get_operator(void);

void add_fractions(int n1, int d1, int n2, int d2,
                   int *n_ansp, int *d_ansp);

void multiply_fractions(int n1, int d1, int n2, int d2,
                        int *n_ansp, int *d_ansp);

int find_gcd (int n1, int n2);

void reduce_fraction(int *nump, int *denomp);

void print_fraction(int num, int denom);

int
main(void)
{
      int  n1, d1;       /* numerator, denominator of first fraction        */
      int  n2, d2;       /* numerator, denominator of second fraction       */
      char op;           /* arithmetic operator + - * or /                  */
      char again;        /* Y or n depending on user's desire to continue   */
      int  n_ans, d_ans; /* numerator, denominator of answer                */
                                                              (continued)
```


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

```
    /* While the user wants to continue, gets and solves arithmetic
       problems with common fractions                                       */
    do {
        /* Gets a fraction problem                                          */
        scan_fraction(&n1, &d1);
        op = get_operator();
        scan_fraction(&n2, &d2);

        /* Computes the result  */
        switch (op) {
        case '+':
              add_fractions(n1, d1, n2, d2, &n_ans, &d_ans);
              break;

        case '-':
              add_fractions(n1, d1, -n2, d2, &n_ans, &d_ans);
              break;

        case '*':
              multiply_fractions(n1, d1, n2, d2, &n_ans, &d_ans);
              break;

        case '/':
              multiply_fractions(n1, d1, d2, n2, &n_ans, &d_ans);
        }
        reduce_fraction(&n_ans, &d_ans);

        /* Displays problem and result                                      */
        printf("\n");
        print_fraction(n1, d1);
        printf(" %c ", op);
        print_fraction(n2, d2);
        printf(" = ");
        print_fraction(n_ans, d_ans);

        /* Asks user about doing another problem                            */
        printf("\nDo another problem? (y/n)> ");
        scanf(" %c", &again);
    } while (again == 'y' || again == 'Y');
    return (0);
}
                                                              (continued)
```

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

```
/* Insert function scan_fraction from Fig. 6.9 here. */

/*
 * Gets and returns a valid arithmetic operator.  Skips over newline
 * characters and permits reentry of operator in case of error.
 */
char
get_operator(void)
{
      char op;

      printf("Enter an arithmetic operator (+,-,*, or /)\n> ");
      for  (scanf("%c", &op);
            op != '+'  &&  op != '-'  &&
            op != '*'  &&  op != '/';
            scanf("%c", &op)) {
          if (op != '\n')
              printf("%c invalid, reenter operator (+,-, *,/)\n> ", op);
      }
      return (op);
}

/*
 * Adds fractions represented by pairs of integers.
 * Pre:  n1, d1, n2, d2 are defined;
 *       n_ansp and d_ansp are addresses of type int variables.
 * Post: sum of n1/d1 and n2/d2 is stored in variables pointed
 *       to by n_ansp and d_ansp.  Result is not reduced.
 */
void
add_fractions(int      n1, int      d1, /* input - first fraction   */
              int      n2, int      d2, /* input - second fraction  */
              int *n_ansp, int *d_ansp) /* output - sum of 2 fractions*/
{
      int denom,     /* common denominator used for sum (may not be least)     */
                                                              (continued)
```


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

```
          numer,       /* numerator of sum                                  */
          sign_factor; /* -1 for a negative, 1 otherwise                    */

      /* Finds a common denominator                                         */
      denom = d1 * d2;

      /* Computes numerator                                                 */
      numer = n1 * d2 + n2 * d1;

      /* Adjusts sign (at most, numerator should be negative)               */
      if (numer * denom >= 0)
          sign_factor = 1;
      else
          sign_factor = -1;

      numer = sign_factor * abs(numer);
      denom = abs(denom);

      /* Returns result                                                     */
      *n_ansp = numer;
      *d_ansp = denom;
}

/*
 ***** STUB *****
 * Multiplies fractions represented by pairs of integers.
 * Pre:  n1, d1, n2, d2 are defined;
 *       n_ansp and d_ansp are addresses of type int variables.
 * Post: product of n1/d1 and n2/d2 is stored in variables pointed
 *       to by n_ansp and d_ansp.  Result is not reduced.
 */
void
multiply_fractions(int      n1, int   d1, /*  input - first fraction          */
                   int      n2, int   d2, /*  input - second fraction         */
                   int *n_ansp,           /*  output -                       */
                   int *d_ansp)           /*  product of 2 fractions         */
                                                              (continued)
```


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

```
{
      /* Displays trace message                                             */
      printf("\nEntering multiply_fractions with\n");
      printf("n1 = %d, d1 = %d, n2 = %d, d2 = %d\n", n1, d1, n2, d2);
      /* Defines output arguments                                           */
      *n_ansp = 1;
      *d_ansp = 1;
}
/*
 ***** STUB *****
 * Finds greatest common divisor of two integers
 */
int
find_gcd (int n1, int n2) /* input - two integers                           */
{
      int gcd;

      /* Displays trace message                                             */
      printf("\nEntering find_gcd with n1 = %d, n2 = %d\n", n1, n2);

      /* Asks user for gcd                                                  */
      printf("gcd of %d and %d?> ", n1, n2);
      scanf("%d", &gcd);

      /* Displays exit trace message                                        */
      printf("find_gcd returning %d\n", gcd);
      return (gcd);
}
/*
 * Reduces a fraction by dividing its numerator and denominator by their
 * greatest common divisor.
 */
void
reduce_fraction(int *nump,   /* input/output -                             */
                int *denomp) /* numerator and denominator of fraction      */
{
      int gcd;   /* greatest common divisor of numerator & denominator     */

      gcd = find_gcd(*nump, *denomp);
                                                              (continued)
```


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

```
      *nump = *nump / gcd;
      *denomp = *denomp / gcd;
}

/*
 * Displays pair of integers as a fraction.
 */
void
print_fraction(int num, int denom)  /* input - numerator & denominator        */
{
      printf("%d/%d", num, denom);
}
```


Figure 6.13 Sample Run of a Partially Complete Program Containing Stubs

```
Enter a common fraction as two integers separated by a slash> 3/-4
Input invalid--denominator must be positive
Enter a common fraction as two integers separated by a slash> 3/4
Enter an arithmetic operator (+,-,*, or /)
> *
Enter a common fraction as two integers separated by a slash> 5/8
Entering find_gcd with n1 = 44, n2 = 32
gcd of 44 and 32?> 4
find_gcd returning 4

3/4 * 5/8 = 11/8
Do another problem? (y/n)> y
Enter a common fraction as two integers separated by a slash> 1/2
Enter an arithmetic operator (+,-,*, or /)
> 5
5 invalid, reenter operator (+,-,*,/)
> *
Enter a common fraction as two integers separated by a slash> 5/7
Entering multiply_fractions with
n1 = 1, d1 = 2, n2 = 5, d2 = 7
Entering find_gcd with n1 = 1, n2 = 1
gcd of 1 and 1?> 1
find_gcd returning 1

1/2 * 5/7 = 1/1
Do another problem? (y/n)> n
```


Figure 6.14 Stub for Function multiply_fractions

```
/*
 ***** STUB *****
 * Multiplies fractions represented by pairs of integers.
 * Pre:  n1, d1, n2, d2 are defined;
 *       n_ansp and d_ansp are addresses of type int variables.
 * Post: product of n1/d1 and n2/d2 is stored in variables pointed
 *       to by n_ansp and d_ansp.  Result is not reduced.
 */
void
multiply_fractions(int      n1, int   d1, /*  input - first fraction          */
                   int      n2, int   d2, /*  input - second fraction         */
                   int *n_ansp,           /*  output -                       */
                   int *d_ansp)           /*      product of 2 fractions     */
{
      /* Displays trace message                                             */
      printf("\nEntering multiply_fractions with\n");
      printf("n1 = %d, d1 = %d, n2 = %d, d2 = %d\n", n1, d1, n2, d2);

      /* Defines output arguments                                           */
      *n_ansp = 1;
      *d_ansp = 1;
}
```

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.
14.       return (0);
15. }
```