“Talk is cheap. Show me the code.”

- Linus Torvalds

CSE102
Computer Programming with C

2017-2018 Spring Semester

Modular Programming

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

Functions

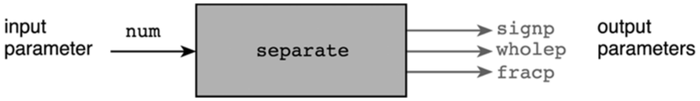
- Functions: Components of a program
- Connect functions to generate a program
- Each function has
  - Inputs
    - Parameters
    - Computes manipulate different data each time it is called
  - Outputs
    - Returns a result with return statement
    - Output parameters to return multiple results

Functions

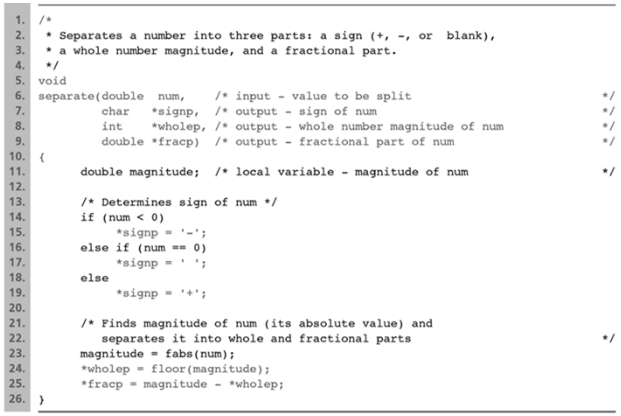
- Function call
  - Allocate memory space for each formal parameter
    - Heap vs Stack
  - Store actual parameter value in the allocated space
  - Execute function code
    - Manipulates the values of formal parameters

Diagram of Function separate

- Ex: Gets a double value, find and return
  - Sign
  - Whole number magnitude
  - Fractional part
- Three output parameters



Function separate

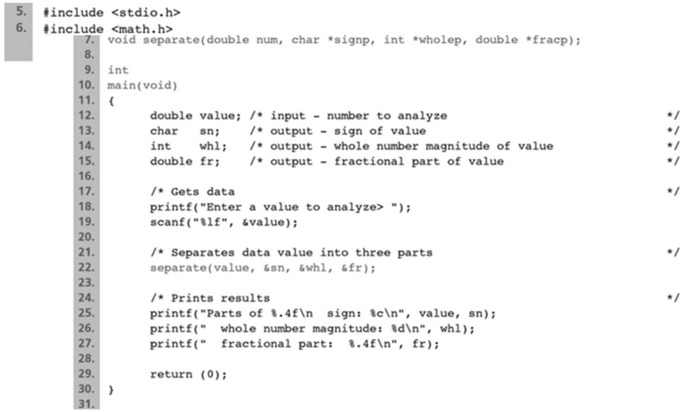


```
/*
 * Separates a number into three parts: a sign (+, -, or  blank),
 * a whole number magnitude, and a fractional part.
 */
void
separate(double  num,    /* input - value to be split                         */
         char   *signp,  /* output - sign of num                              */
         int    *wholep, /* output - whole number magnitude of num            */
         double *fracp)  /* output - fractional part of num                   */
{
      double magnitude;  /* local variable - magnitude of num                 */

      /* Determines sign of num */
      if (num < 0)
           *signp = '-';
      else if (num == 0)
           *signp = ' ';
      else
           *signp = '+';

      /* Finds magnitude of num (its absolute value) and
         separates it into whole and fractional parts                         */
      magnitude = fabs(num);
      *wholep = floor(magnitude);
      *fracp = magnitude - *wholep;
}
```

Function Output Parameters

- Use * in front of the output parameters
  - declaration
    - char *signp,
  - assignment
    - *signp = '-';
- signp : pointer
  - contains address of a char variable
  - "p" is used because it is pointer

Program That Calls separate

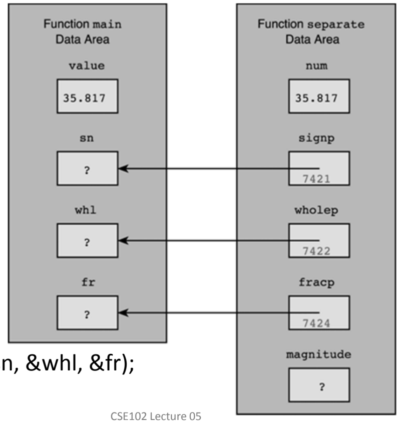


```
#include <stdio.h>
#include <math.h>
void separate(double num, char *signp, int *wholep, double *fracp);

int
main(void)
{
      double value; /* input - number to analyze                              */
      char   sn;    /* output - sign of value                                 */
      int    whl;   /* output - whole number magnitude of value               */
      double fr;    /* output - fractional part of value                      */

      /* Gets data                                                            */
      printf("Enter a value to analyze> ");
      scanf("%lf", &value);

      /* Separates data value into three parts                                */
      separate(value, &sn, &whl, &fr);

      /* Prints results                                                       */
      printf("Parts of %.4f\n  sign: %c\n", value, sn);
      printf("  whole number magnitude: %d\n", whl);
      printf("  fractional part:  %.4f\n", fr);

      return (0);
}
```

Program That Calls separate

- Three variables defined in main function
  - values will be defined by function separate
  - address of sn is stored in output parameter signp
- Use & operator on the actual parameter
  - separate(value, &sn, &whl, &fr);
  - separate knows where sn is in the memory.
    - Like scanf
  - &sn is of type char-pointer

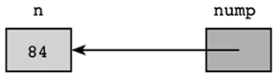
Parameter Correspondence



separate(value, &sn, &whl, &fr);

Use of output parameters

- Indirection operator: *
  - Applied to formal parameter (pointer)
  - Follow the pointer referenced by the formal parameter
  - Indirect reference



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

Function separate

```
/*
 * Separates a number into three parts: a sign (+, -, or  blank),
 * a whole number magnitude, and a fractional part.
 */
void
separate(double  num,    /* input - value to be split                      */
         char   *signp,  /* output - sign of num                           */
         int    *wholep, /* output - whole number magnitude of num         */
         double *fracp)  /* output - fractional part of num                */
{
      double magnitude;  /* local variable - magnitude of num              */

      /* Determines sign of num */
      if (num < 0)
           *signp = '-';
      else if (num == 0)
           *signp = ' ';
      else
           *signp = '+';

      /* Finds magnitude of num (its absolute value) and
         separates it into whole and fractional parts                       */
      magnitude = fabs(num);
      *wholep = floor(magnitude);
      *fracp = magnitude - *wholep;
}
```

Meanings of * Symbol

- Three distinct meanings
  - Multiplication
  - Declaration
    - char *sn : means sn is pointer to char
  - Indirection operator
    - Follow pointer
    - *sn is of type char

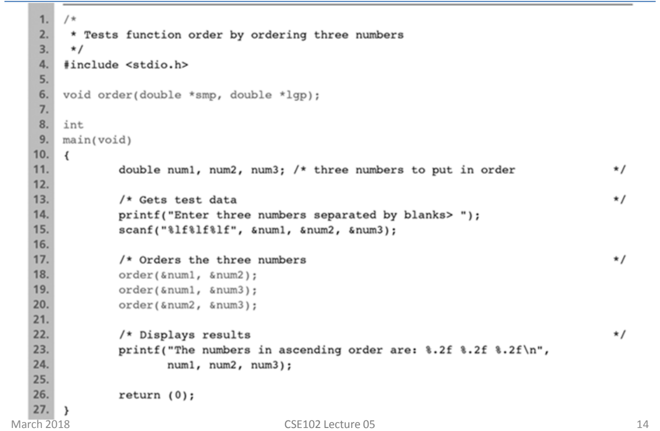
Input/Output Parameters

- Single parameter for
  - Bring data to the function
  - Carry result out of the function

- Ex: Arrange three values in increasing order
- Input:
  - num1, num2, num3
- Output:
  - num1 is the smallest of input values,
  - num2 is the second smallest of input values,
  - num3 is the largest of input values,

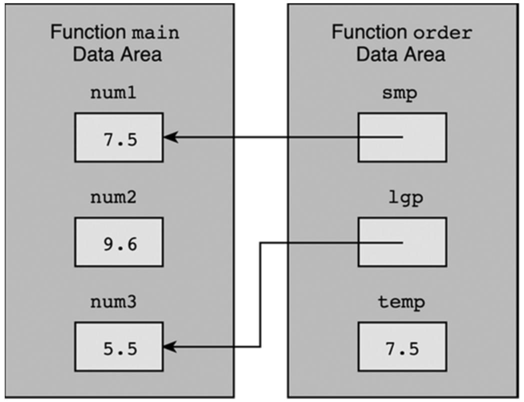
- Function order orders two arguments

Program to Sort Three Numbers

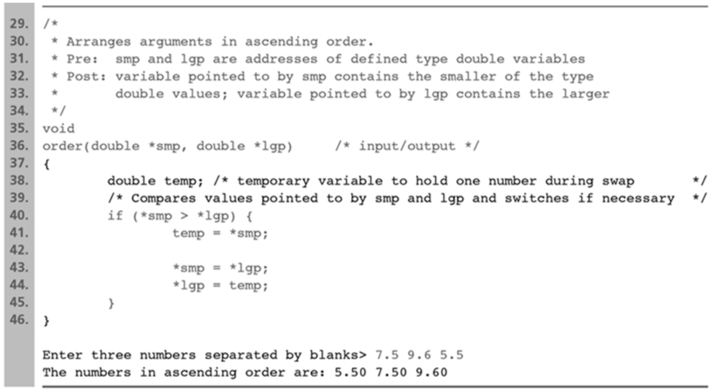


```
/*
 * Tests function order by ordering three numbers
 */
#include <stdio.h>

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);
}
```

Data Areas of order(&num1, &num3);



Program to Sort Three Numbers



```
/*
 * 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
```

Program Style

- Use functions take input parameters and return a value
  - Easier to understand and maintain
    - No indirect reference
    - No address operator
    - Return value is assigned to a variable at caller
- Math function are of this type

Subprograms

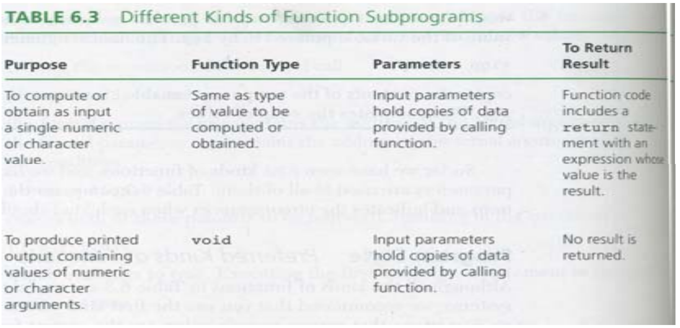
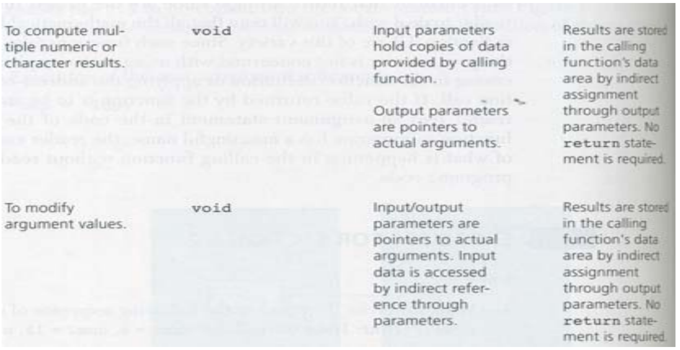


TABLE 6.3 Different Kinds of Function Subprograms

Purpose	Function Type	Parameters	To Return Result
To compute or obtain as input a single numeric or character value.	Same as type of value to be computed or obtained.	Input parameters hold copies of data provided by calling function.	Function code includes a `return` statement with an expression whose value is the result.
To produce printed output containing values of numeric or character arguments.	`void`	Input parameters hold copies of data provided by calling function.	No result is returned.

Subprograms

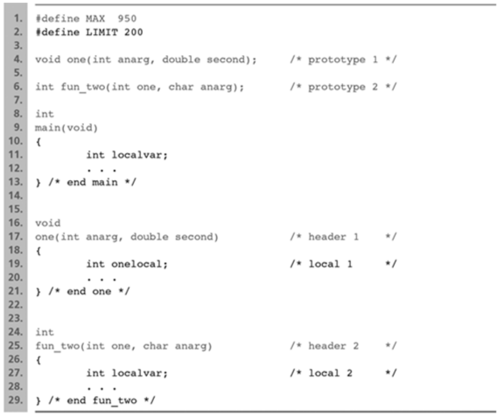


To compute multiple numeric or character results.	`void`	Input parameters hold copies of data provided by calling function. Output parameters are pointers to actual arguments.	Results are stored in the calling function's data area by indirect assignment through output parameters. No `return` statement is required.
To modify argument values.	`void`	Input/output parameters are pointers to actual arguments. Input data is accessed by indirect reference through parameters.	Results are stored in the calling function's data area by indirect assignment through output parameters. No `return` statement is required.

Scope of Names

- Region of program that the name is visible
- Scope of
  - constant macros
    - From definition to the end of source file
  - function names
    - From function prototype to the end of source file
  - variables
    - From declaration to closing brace
- What if an identifier is defined before?

Program for Studying Scope of Names



```c
#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 */
```

Scope of Names

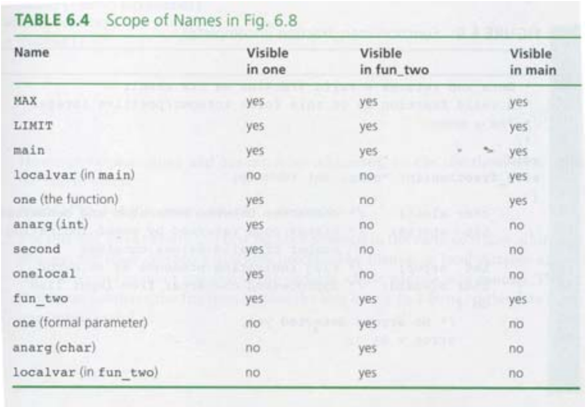


TABLE 6.4 Scope of Names in Fig. 6.8

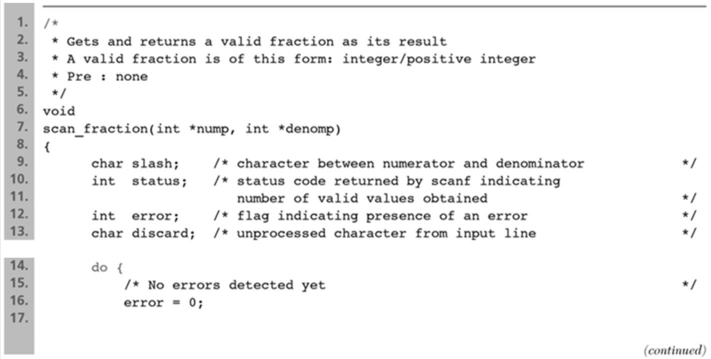
Name	Visible in one	Visible in fun_two	Visible in main
MAX	yes	yes	yes
LIMIT	yes	yes	yes
main	yes	yes	yes
localvar (in main)	no	no	yes
one (the function)	yes	no	yes
anarg (int)	yes	no	no
second	yes	no	no
onelocal	yes	no	no
fun_two	yes	yes	yes
one (formal parameter)	no	yes	no
anarg (char)	no	yes	no
localvar (in fun_two)	no	yes	no

Formal Output Parameters as Actual Arguments

- Passing output parameters to other functions
  - Ex: Reading values into output parameters
- Ex: Write a function to read a common fraction

  numerator / denominator
  - Function scan_fraction
    - Two output parameters
    - Reads a fraction until a valid fraction is entered

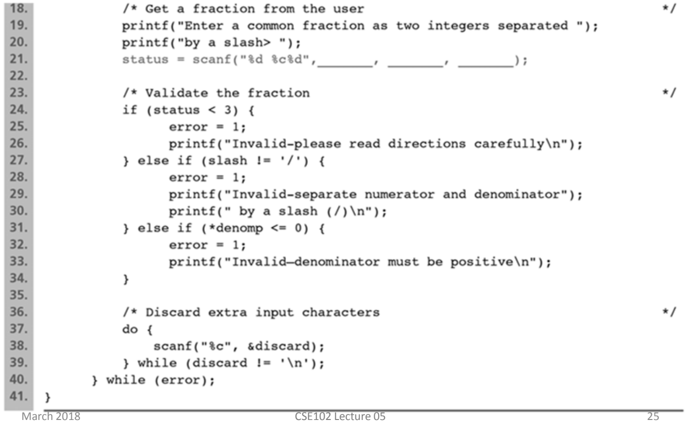
Function scan_fraction



```c
/*
 * 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)

Function scan_fraction

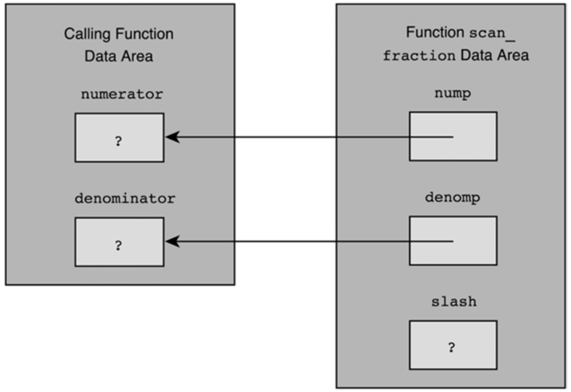


```
        /* Get a fraction from the user                                  */
        printf("Enter a common fraction as two integers separated ");
        printf("by a slash> ");
        status = scanf("%d %c%d",_______, _______, _______);

        /* 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);
}
```

Case Study: Common Fraction Problem

- Problem: Write a program to add, subtract, multiply and divide pairs of common fractions
- Inputs:
  - First fraction: numerator and denominator
  - Second fraction: numerator and denominator
  - Operator
- Output:
  - Resulting fraction

Data Areas for scan_fraction and Its Caller



Case Study: Common Fraction Problem

Algorithm

1. Repeat as long as user wants to continue
2. Get a fraction problem
3. Compute the result
4. Display the problem and result
5. Check if user wants to continue

Case Study: Common Fraction Problem

Algorithm

1. Repeat as long as user wants to continue
2. Get a fraction problem
    1. Get first fraction (scan_fraction)
    2. Get operator (get_operator)
    3. Get second fraction (scan_fraction)
3. Compute the result
4. Display the problem and result
5. Check if user wants to continue

Case Study: Common Fraction Problem

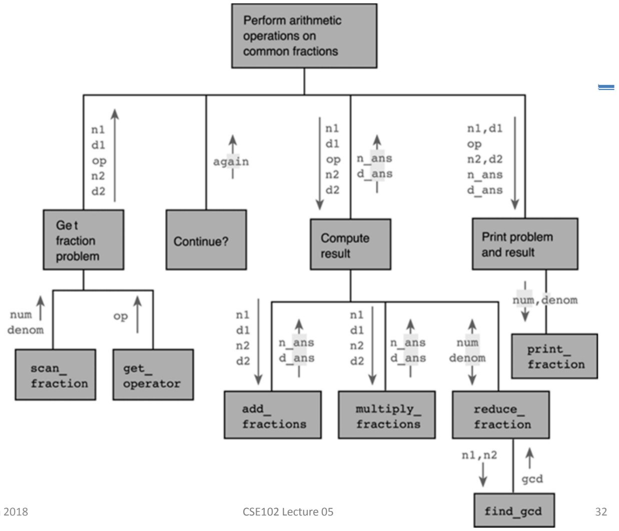
Algorithm

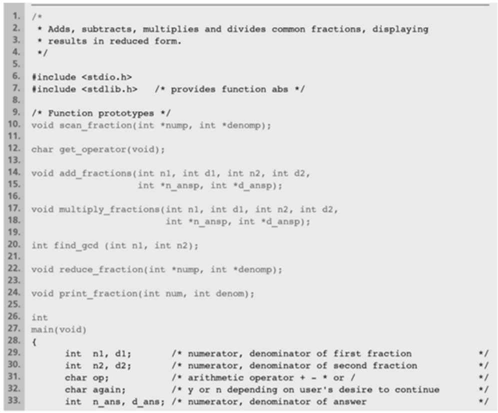
1. Repeat as long as user wants to continue
2. Get a fraction problem
3. Compute the result
    1. Select and perform task based on operator
        - Add the fractions (add_fractions)
        - Add the first fraction and the negation of the second fraction
        - Multiply the fractions (multiply_fractions)
        - Multiply the first fraction with reciprocal of the second fraction
    2. Put the result fraction in reduced form
4. Display the problem and result
5. Check if user wants to continue

Case Study: Common Fraction Problem

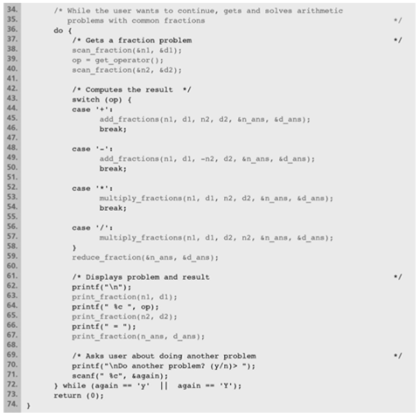
Algorithm

1. Repeat as long as user wants to continue
2. Get a fraction problem
3. Compute the result
    1. Select and perform task based on operator
    2. Put the result fraction in reduced form
        - Find the GCD of the numerator and denominator (find_gcd)
        - Divide numerator and denominator by the GCD (reduce_fraction)
4. Display the problem and result (print_fraction)
5. Check if user wants to continue

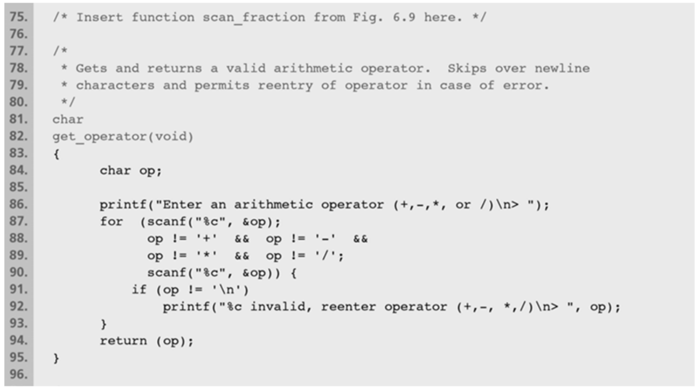




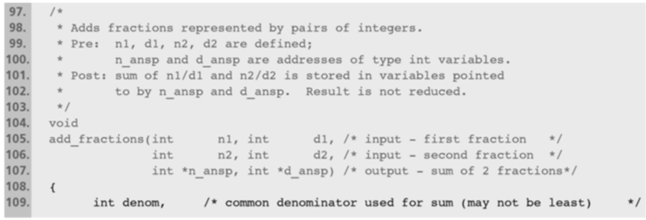
```
/*
 * Adds, subtracts, multiplies and divides common fractions, displaying
 * results in reduced form.
 */

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

/* 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                   */
```



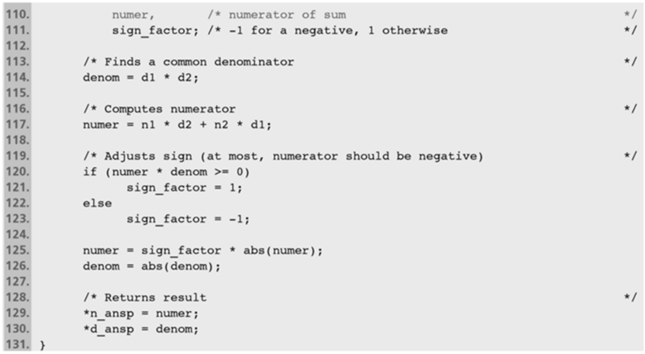
```
      /* While the user wants to continue, gets and solves arithmetic
         problems with common fractions                                      */
      do {
          /* Gets a fraction problem                                         */
          scan_fraction(&n1, &d1);
          op = get_operator();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



```
/* 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)     */
```



```
            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;
}
```