“Talk is cheap. Show me the code.”

- Linus Torvalds

CSE102
Computer Programming with C

2015-2016 Fall Semester

Modular Programming

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

Functions

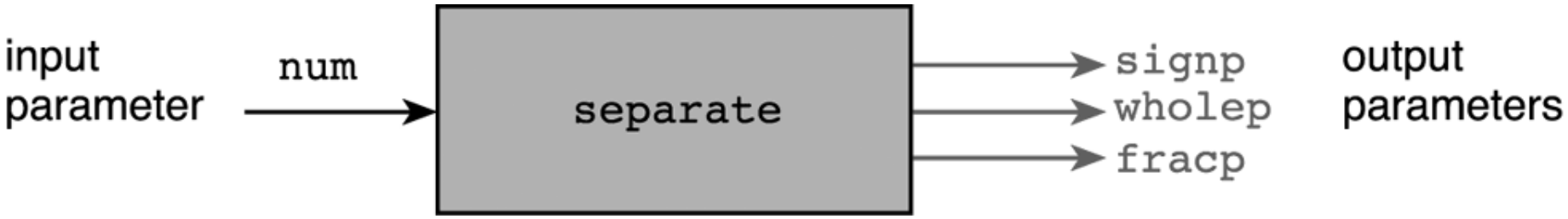
- Functions: Components of a program
- Connect functions to generate a program
- Each function has
  - Inputs
    - Parameters
    - Computes manipulate different data each time it is called
  - Outputs
    - Returns a result with return statement
    - Output parameters to return multiple results

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

```
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. }
```

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

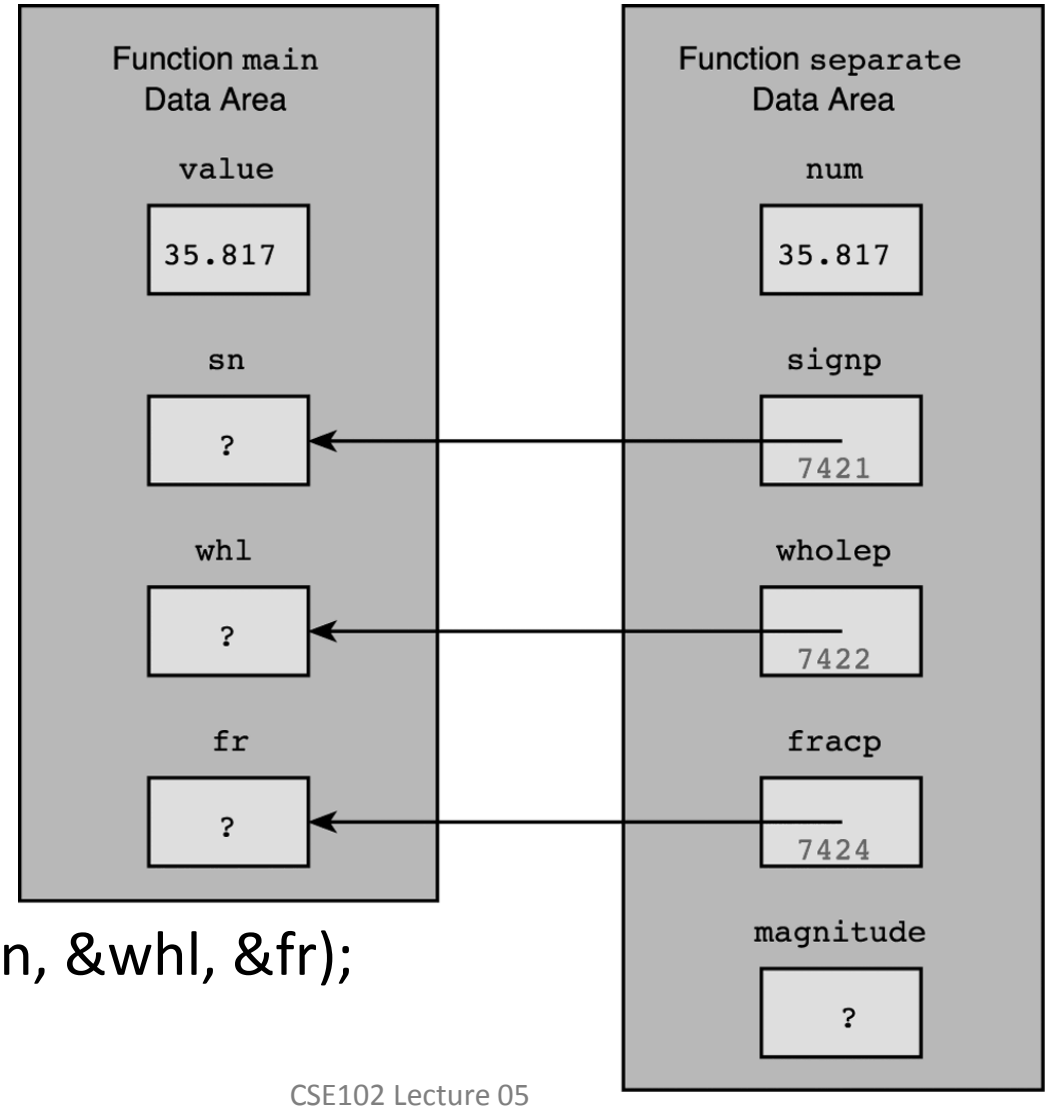
```
5.  #include <stdio.h>
6.  #include <math.h>
7.  void separate(double num, char *signp, int *wholep, double *fracp);
8.
9.  int
10. main(void)
11. {
12.       double value; /* input - number to analyze                    */
13.       char   sn;    /* output - sign of value                       */
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.
```

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

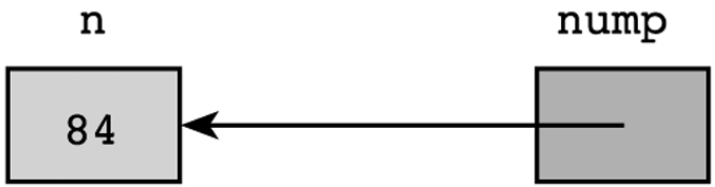
Parameter Correspondence



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

```
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. }
```

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

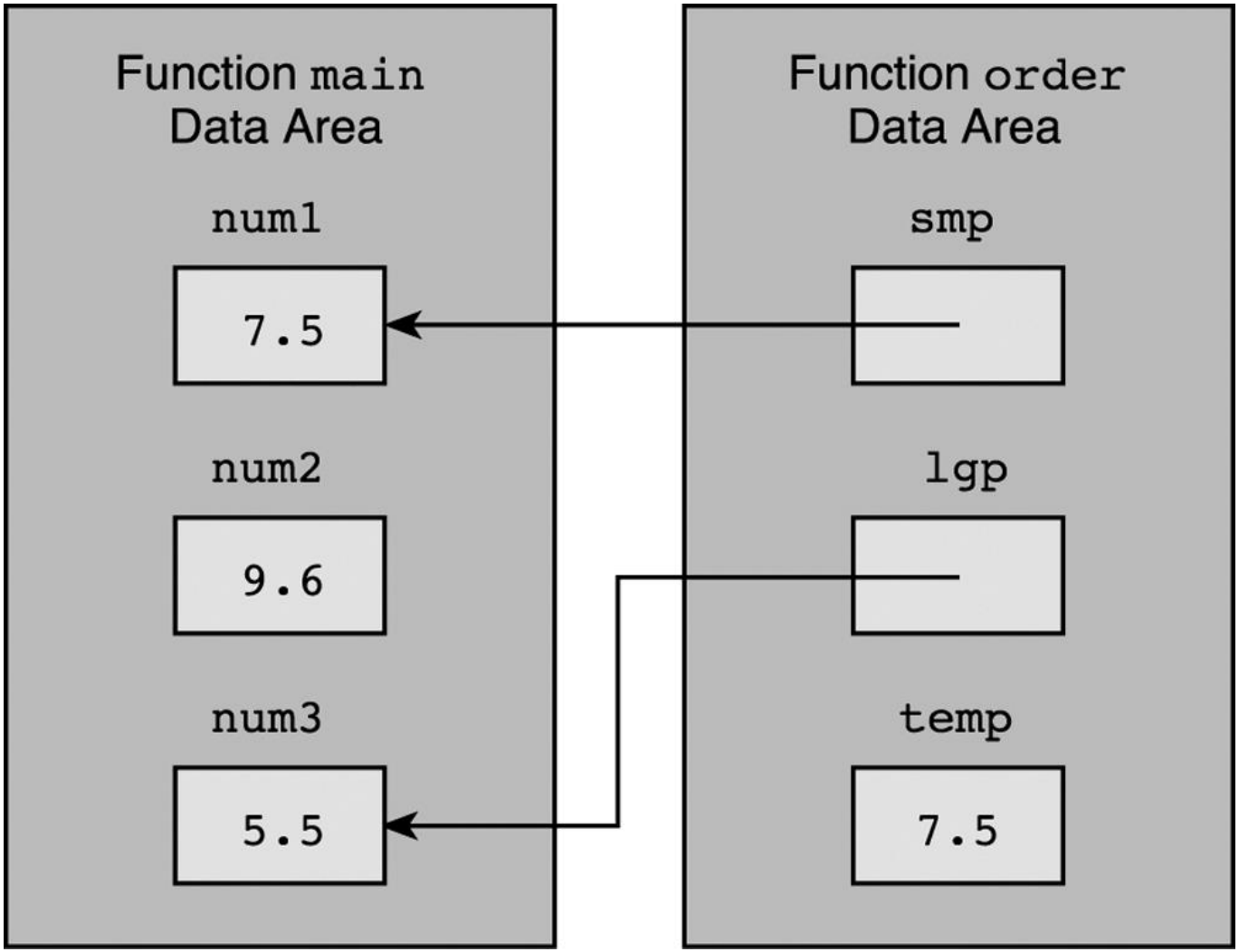
- Ex: Arrange three values in increasing order
- Input:
  - num1, num2, num3
- Output:
  - num1 is the smallest of input values,
  - num2 is the second smallest of input values,
  - num3 is the largest of input values,

- Function order orders two arguments

Program to Sort Three Numbers

```
1.  /*
2.   * Tests function order by ordering three numbers
3.   */
4.  #include <stdio.h>
5.
6.  void order(double *smp, double *lgp);
7.
8.  int
9.  main(void)
10. {
11.         double num1, num2, num3; /* three numbers to put in order              */
12.
13.         /* Gets test data                                                       */
14.         printf("Enter three numbers separated by blanks> ");
15.         scanf("%lf%lf%lf", &num1, &num2, &num3);
16.
17.         /* Orders the three numbers                                            */
18.         order(&num1, &num2);
19.         order(&num1, &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. }
```

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



Program to Sort Three Numbers

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

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

```
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 */
```

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

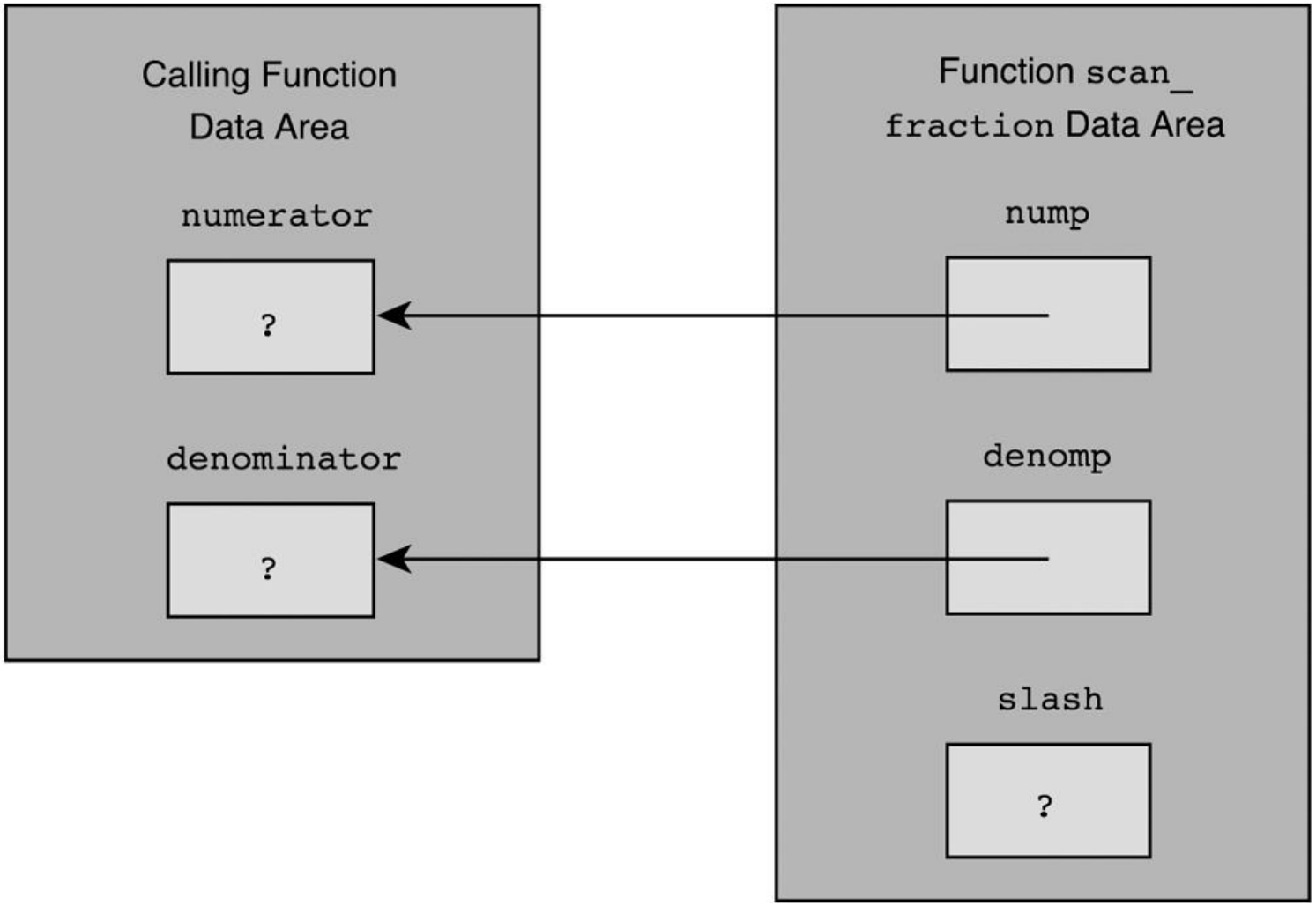
```
1.  /*
2.   * Gets and returns a valid fraction as its result
3.   * A valid fraction is of this form: integer/positive integer
4.   * Pre : none
5.   */
6.  void
7.  scan_fraction(int *nump, int *denomp)
8.  {
9.        char slash;    /* character between numerator and denominator          */
10.       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)

Function scan_fraction

```
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",_______, _______, _______);
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. }
```

Data Areas for scan_fraction and Its Caller



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

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

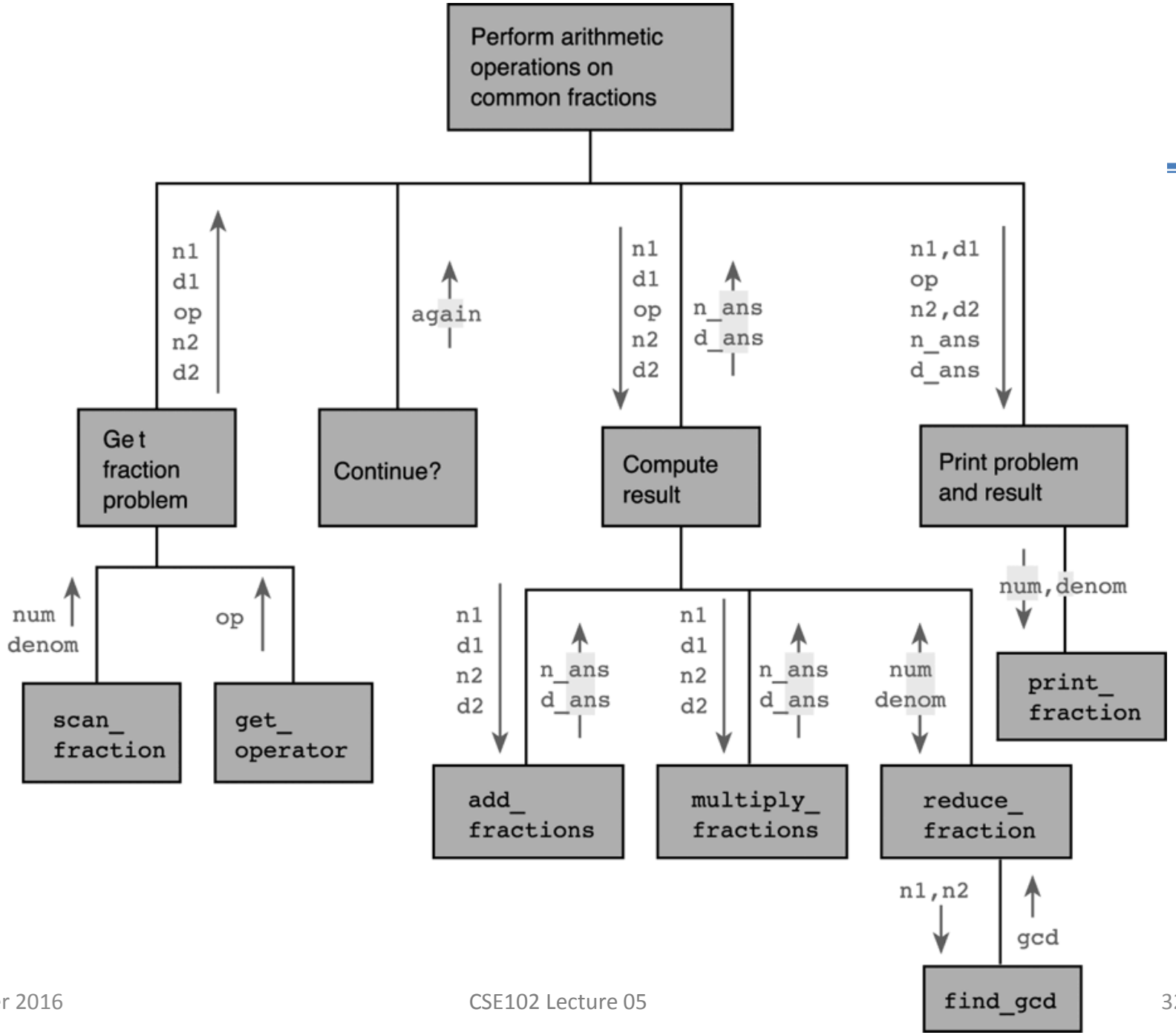
Algorithm

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


Perform arithmetic operations on common fractions
n1
d1
op
n2
d2
again
n1
d1
op
n2
d2
n_ans
d_ans
n1,d1
op
n2,d2
n_ans
d_ans
Get fraction problem
Continue?
Compute result
Print problem and result
num
denom
op
num,denom
n1
d1
n2
d2
n_ans
d_ans
n1
d1
n2
d2
n_ans
d_ans
num
denom
scan_ fraction
get_ operator
add_ fractions
multiply_ fractions
reduce_ fraction
print_ fraction
n1,n2
gcd
find_gcd

```c
/*
 * 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                   */
```

```
34.       /* While the user wants to continue, gets and solves arithmetic
35.          problems with common fractions                                */
36.       do {
37.           /* Gets a fraction problem                                   */
38.           scan_fraction(&n1, &d1);
39.           op = get_operator();
40.           scan_fraction(&n2, &d2);
41.
42.           /* Computes the result  */
43.           switch (op) {
44.           case '+':
45.                 add_fractions(n1, d1, n2, d2, &n_ans, &d_ans);
46.                 break;
47.
48.           case '-':
49.                 add_fractions(n1, d1, -n2, d2, &n_ans, &d_ans);
50.                 break;
51.
52.           case '*':
53.                 multiply_fractions(n1, d1, n2, d2, &n_ans, &d_ans);
54.                 break;
55.
56.           case '/':
57.                 multiply_fractions(n1, d1, d2, n2, &n_ans, &d_ans);
58.           }
59.           reduce_fraction(&n_ans, &d_ans);
60.
61.           /* Displays problem and result                               */
62.           printf("\n");
63.           print_fraction(n1, d1);
64.           printf(" %c ", op);
65.           print_fraction(n2, d2);
66.           printf(" = ");
67.           print_fraction(n_ans, d_ans);
68.
69.           /* Asks user about doing another problem                     */
70.           printf("\nDo another problem? (y/n)> ");
71.           scanf(" %c", &again);
72.       } while (again == 'y'  ||  again == 'Y');
73.       return (0);
74. }
```

```
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 != '-'  &&
89.              op != '*'  &&  op != '/';
90.              scanf("%c", &op)) {
91.            if (op != '\n')
92.                printf("%c invalid, reenter operator (+,-, *,/)\n> ", op);
93.        }
94.        return (op);
95.  }
96.
```

```
97.   /*
98.    * Adds fractions represented by pairs of integers.
99.    * Pre:  n1, d1, n2, d2 are defined;
100.   *       n_ansp and d_ansp are addresses of type int variables.
101.   * Post: sum of n1/d1 and n2/d2 is stored in variables pointed
102.   *       to by n_ansp and d_ansp.  Result is not reduced.
103.   */
104.  void
105.  add_fractions(int      n1, int      d1, /* input - first fraction   */
106.                int      n2, int      d2, /* input - second fraction  */
107.                int *n_ansp, int *d_ansp) /* output - sum of 2 fractions*/
108.  {
109.        int denom,     /* common denominator used for sum (may not be least)     */
```

```
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.        /* Computes numerator                                          */
117.        numer = n1 * d2 + n2 * d1;
118.
119.        /* Adjusts sign (at most, numerator should be negative)        */
120.        if (numer * denom >= 0)
121.              sign_factor = 1;
122.        else
123.              sign_factor = -1;
124.
125.        numer = sign_factor * abs(numer);
126.        denom = abs(denom);
127.
128.        /* Returns result                                              */
129.        *n_ansp = numer;
130.        *d_ansp = denom;
131.  }
```

```
133. /*
134.  ***** STUB *****
135.  * Multiplies fractions represented by pairs of integers.
136.  * Pre:  n1, d1, n2, d2 are defined;
137.  *       n_ansp and d_ansp are addresses of type int variables.
138.  * Post: product of n1/d1 and n2/d2 is stored in variables pointed
139.  *       to by n_ansp and d_ansp.  Result is not reduced.
140.  */
141. void
142. multiply_fractions(int      n1, int   d1, /*  input - first fraction          */
143.                     int      n2, int   d2, /*  input - second fraction         */
144.                     int *n_ansp,           /*  output -                       */
145.                     int *d_ansp)           /*  product of 2 fractions         */
146. {
147.       /* Displays trace message                                               */
148.       printf("\nEntering multiply_fractions with\n");
149.       printf("n1 = %d, d1 = %d, n2 = %d, d2 = %d\n", n1, d1, n2, d2);
150.       /* Defines output arguments                                             */
151.       *n_ansp = 1;
152.       *d_ansp = 1;
153. }
154.
```

```
156.  ***** STUB *****
157.  * Finds greatest common divisor of two integers
158.  */
159. int
160. find_gcd (int n1, int n2) /* input - two integers                    */
161. {
162.       int gcd;
163.
164.       /* Displays trace message                                      */
165.       printf("\nEntering find_gcd with n1 = %d, n2 = %d\n", n1, n2);
166.
167.       /* Asks user for gcd                                           */
168.       printf("gcd of %d and %d?> ", n1, n2);
169.       scanf("%d", &gcd);
170.
171.       /* Displays exit trace message                                 */
172.       printf("find_gcd returning %d\n", gcd);
173.       return (gcd);
174. }
175.
```

```
176. /*
177.  * Reduces a fraction by dividing its numerator and denominator by their
178.  * greatest common divisor.
179.  */
180. void
181. reduce_fraction(int *nump,    /* input/output -                              */
182.                  int *denomp) /* numerator and denominator of fraction       */
183. {
184.       int gcd;   /* greatest common divisor of numerator & denominator     */
185.
186.       gcd = find_gcd(*nump, *denomp);
187.       *nump = *nump / gcd;
188.       *denomp = *denomp / gcd;
189. }
190.
191. /*
192.  * Displays pair of integers as a fraction.
193.  */
194. void
195. print_fraction(int num, int denom)  /* input - numerator & denominator      */
196. {
197.       printf("%d/%d", num, denom);
198. }
```

Sample Run

```
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
```

Program Style

- Keep the functions to a manageable size
  - Less error
  - Easier to read and test

Testing

- Top-down testing
  - Test general flow of control
    - **stubs**

- Stubs
  - Used instead of functions not yet written
  - Team work!..
    - Enables testing and debugging
  - Displays an identification message
  - Assign values to output parameters

Stub for Function multiply_fractions

```
1.  /*
2.   ***** STUB *****
3.   * Multiplies fractions represented by pairs of integers.
4.   * Pre:  n1, d1, n2, d2 are defined;
5.   *       n_ansp and d_ansp are addresses of type int variables.
6.   * Post: product of n1/d1 and n2/d2 is stored in variables pointed
7.   *       to by n_ansp and d_ansp.  Result is not reduced.
8.   */
9.  void
10. multiply_fractions(int      n1, int   d1, /*  input - first fraction          */
11.                    int      n2, int   d2, /*  input - second fraction         */
12.                    int *n_ansp,           /*  output -                       */
13.                    int *d_ansp)           /*     product of 2 fractions      */
14. {
15.       /* Displays trace message                                             */
16.       printf("\nEntering multiply_fractions with\n");
17.       printf("n1 = %d, d1 = %d, n2 = %d, d2 = %d\n", n1, d1, n2, d2);
18.
19.       /* Defines output arguments`                                          */
20.       *n_ansp = 1;
21.       *d_ansp = 1;
22. }
```

Testing

- Bottom-up Testing
  - First test individual functions
    - **Unit test**
  - Test entire system later
    - **System integration test**

- Unit Test
  - Preliminary test of a function separate from the whole program
  - Using driver program
    - Driver gives values to input parameters
    - Calls the function
    - Display and check function results

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. }
```

Debugging

- Good documentation is essential
  - Function's purpose, parameters, local variables
- Debug each function as you write them
- Create a trace
  - Display the function name as you enter it
  - Display and verify the input parameters as you enter a function
  - Display and verify return values after function returns
  - After it works fine do not erase display statements, comment them out. You may need them later
- Use debugger
  - First execute a function as a single statement
  - If the result is incorrect step in its statements