Pointers and Modular Programming
Chapter 6

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To learn about pointers and indirect addressing
- To see how to access external data files in a program and to be able to read from input file and write to output files using file pointers
- To learn how to return function results through a function's arguments
- To understand the differences between call-by-value and call-by-reference

Chapter Objectives

- To understand the distinction between input, inout, and output parameters and when to use each kind

Pointers

- pointer (pointer variable)
  - a memory cell that stores the address of a data item
  - syntax: *type *variable*

```
int  m  =  25;
int  *itemp;     /* a pointer to an integer */
```

Indirection/indirect reference

accessing the contents of a memory cell through a pointer variable that stores it address

FIGURE 6.1

Referencing a Variable Through a Pointer

TABLE 6.1 References with Pointers

Reference	Cell Referenced	Cell Type (Value)
itemp	gray shaded cell	pointer (1024)
*itemp	cell in color	int (25)

Pointers to Files

- C allows a program to explicitly name a file for input or output.
- Declare file pointers:
  - FILE *inp; /* pointer to input file */
  - FILE *outp; /* pointer to output file */
- Prepare for input or output before permitting access:
  - inp = fopen(“infile.txt”, “r”);
  - outp = fopen(“outfile.txt”, “w”);

Pointers to Files

- fscanf
  - file equivalent of scanf
  - fscanf(inp, “%1f”, &item);
- fprintf
  - file equivalent of printf
  - fprintf(outp, “%.2f\n”, item);
- closing a file when done
  - fclose(inp);
  - fclose(outp);

FIGURE 6.2 Program Using File Pointers

```
1.  /* Inputs each number from an input file and writes it
2.   * rounded to 2 decimal places on a line of an output file.
3.   */
4.  #include <stdio.h>
5.
6.  int
7.  main(void)
8.  {
9.       FILE *inp;          /* pointer to input file */
10.      FILE *outp;         /* pointer to ouput file */
11.      double item;
12.      int input_status;  /* status value returned by fscanf */
13.
14.      /* Prepare files for input or output */
15.      inp = fopen("indata.txt", "r");
16.      outp = fopen("outdata.txt", "w");
17.
18.      /* Input each item, format it, and write it */
19.      input_status = fscanf(inp, "%lf", &item);
20.      while (input_status == 1) {
21.          fprintf(outp, "%.2f\n", item);
22.          input_status = fscanf(inp, "%lf", &item);
23.      }
24.
25.      /* Close the files */
26.      fclose(inp);
27.      fclose(outp);
28.
29.      return (0);
30. }
```

File indata.txt

```
344 55 6.3556 9.4
43.123 47.596
```

File outdata.txt

```
344.00
55.00
6.36
9.40
43.12
47.60
```

Functions with Output Parameters

- We've used the return statement to send back one result value from a function.
- We can also use output parameters to return multiple results from a function.

FIGURE 6.4

Diagram of Function separate with Multiple Results

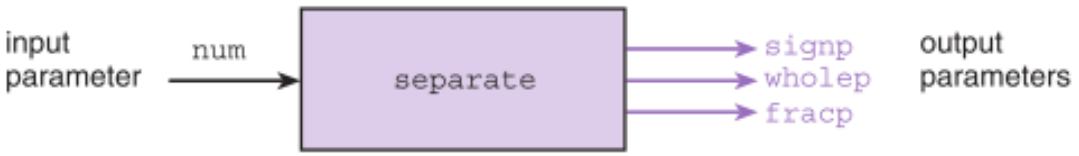

FIGURE 6.3 Function Separate

```
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 6.5 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 6.5 (continued)

```
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 6.5 (continued)

```
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 6.6

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

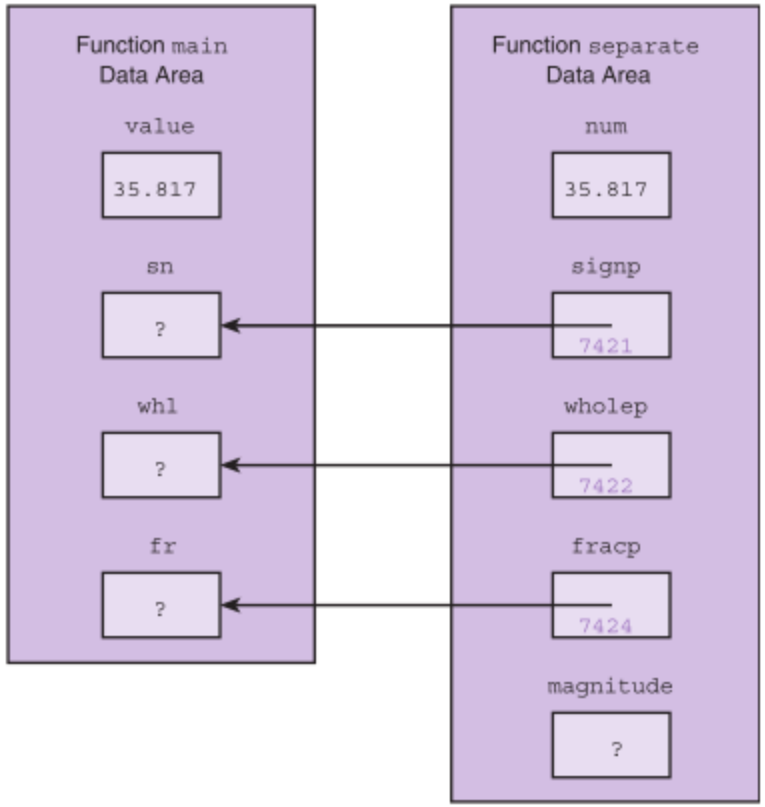

TABLE 6.2 Effect of & Operator on the Data Type of a Reference

Declaration	Data Type of x	Data Type of &x
`char x`	`char`	`char * (pointer to char)`
`int x`	`int`	`int * (pointer to int)`
`double x`	`double`	`double * (pointer to double)`

Meaning of Symbol *

- binary operator for multiplication
- “pointer to” when used when declaring a function’s formal parameters
- unary indirection operator in a function body

Multiple Calls to a Function with Input/Output Parameters

An example of sorting data

FIGURE 6.7 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. }
```

```
28.
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    */
```

(continued)

FIGURE 6.7 (continued)

```
39.         /* Compares values pointed to by smp and lgp and switches if necessary    */
40.         if (*smp > *lgp) {
41.                 temp = *smp;
42.                 *smp = *lgp;
43.                 *lgp = temp;
44.         }
45. }

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

TABLE 6.3 Trace of Program to Sort Three Numbers

Statement	num1	num2	num3	Effect
`scanf("...", &num1, &num2, &num3);`	7.5	9.6	5.5	Enters data
`order(&num1, &num2);`				No change
`order(&num1, &num3);`	5.5	9.6	7.5	Switches num1 and num3
`order(&num2, &num3);`	5.5	7.5	9.6	Switches num2 and num3
`printf("...", num1, num2, num3);`				Displays 5.5 7.5 9.6

FIGURE 6.8

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

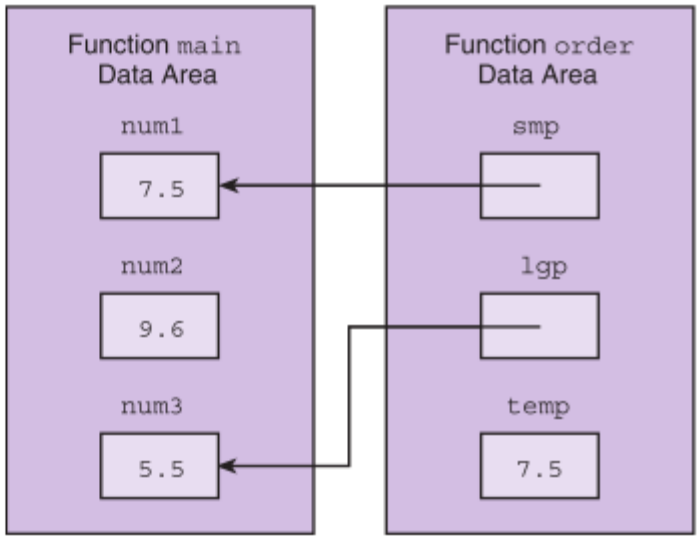

TABLE 6.4 Different Kinds of Function Subprograms

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.
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

- The scope of a name is the region in a program where a particular meaning of a name is visible.

FIGURE 6.9 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;
```

(continued)

FIGURE 6.9 (continued)

```
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 */
```

TABLE 6.5 Scope of Names in Fig. 6.9

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

- A function may need to pass its own output parameter as an argument when it calls another function.

FIGURE 6.10 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.
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. }
```

FIGURE 6.11

Data Areas for scan_fraction and Its Caller

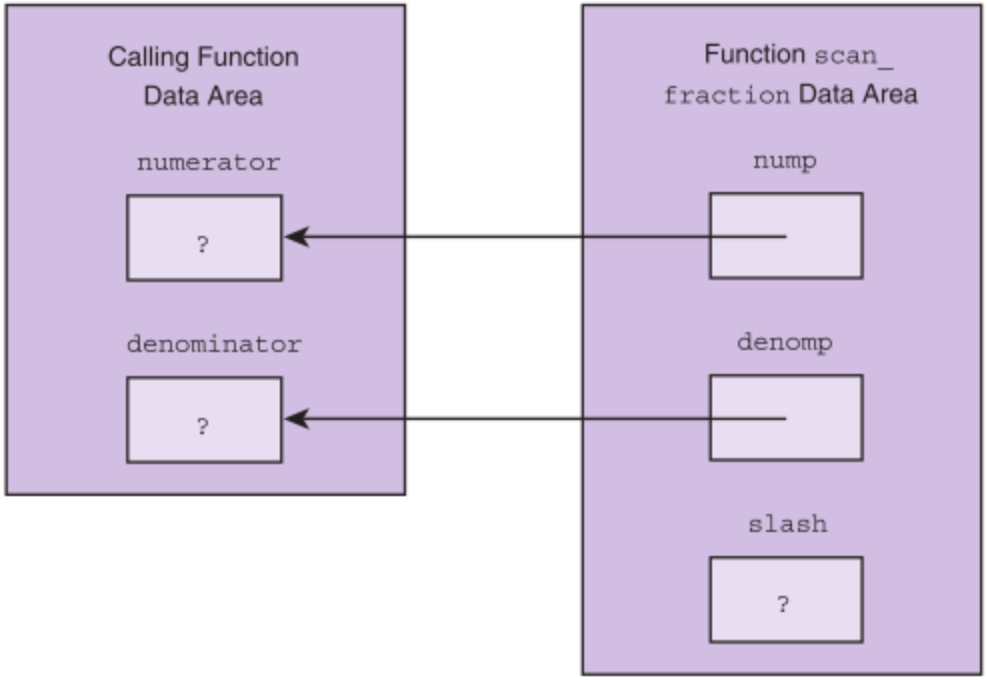

TABLE 6.6 Passing an Argument x to Function some_fun

Actual Argument Type	Use in Calling Function	Purpose in Called Function (some_fun)	Formal Parameter Type	Call to some_fun	Example

`int`
`char`
`double`
	local variable or input parameter	input parameter	`int`
			`char`
			`double`
				`some_fun(x)`	Fig. 6.5, `main: separate(value, &sn, &whl, &fr);` (1st argument)

`int`
`char`
`double`
	local variable	output or input/ output parameter	`int *`
			`char *`
			`double *`
				`some_fun(&x)`	Fig. 6.5, `main: separate(value, &sn, &whl, &fr);` (2nd–4th arguments)

`int *`
`char *`
`double *`
	output or input/ output parameter	output or input/ output parameter	`int *`
			`char *`
			`double *`
				`some_fun(x)`	Fig. 6.10 completed, `scanf(. . .,nump, &slash, denomp);` (2nd and 4th arguments)

`int *`
`char *`
`double *`
	output or input/ output parameter	input parameter	`int`
			`char`
			`double`
				`some_fun(*x)`	Self-Check Ex. 2 in Section 6.6, `trouble: double_trouble(y, *x);` (2nd argument)

Debugging and Testing a Program System

- Unit Testing
  - testing the smallest testable piece of the software, a single function.
  - write a short driver function to call the function tested
  - the driver should give values to all input and inout/output parameters
  - after calling the function, the driver should display the function results

FIGURE 6.17 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 and Testing a Program System

- Integration Testing
  - testing the interactions among functions
  - testing functions that are dependent on other functions whose unit tests may not be complete requires a temporary function called a stub
  - a stub has the same header as the function it replaces but its body displays only a message indicating that the stub was called
  - the stub may provide temporary values for any output arguments or returned data

FIGURE 6.18 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. }
```

Debugging and Testing a Program System

- System Testing
  - testing the whole program in the context in which it will be used
  - a program may need to be tested with other programs and hardware

Debugging and Testing a Program System

- Acceptance Testing
  - system testing designed to show that the program meets its functional requirements
  - typically involves use of the system in the real environment or in a close approximation to the real environment

Wrap Up

- a program can declare pointers to variables of a specified type
- C allows a program to explicitly name a file for input or output
- parameters enable a programmer to pass data to functions and to return multiple results from functions
- a function can use parameters declared as pointers to return values
- the scope of an identifier dictates where it can be referenced