Pointers and Modular Programming
Chapter 6

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Chapter Objectives

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- pointer (pointer variable)
  - a memory cell that stores the address of a data item
  - 8 bytes on on server but depends on machine
  - syntax: *type *variable*

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- pointer (pointer variable)
  - a memory cell that stores the address of a data item
  - 8 bytes on on server but depends on machine
  - syntax: *type *variable*

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

& operator (address of)

- Returns the address of a variable

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

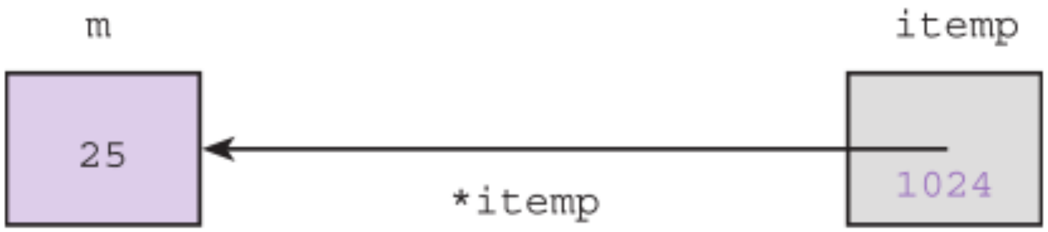


TABLE 6.1 References with Pointers

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

* operator (indirection)

- Follows a pointer to what it points to
- (the thing at the address it stores)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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”);

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers to Files

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Segmentation fault

- Runtime error
- Means you tried to access memory that you weren't allowed to access
- Examples of causes:
  - trying to read from a file that wasn't open
  - following a dangling pointer
  - accessing data beyond array bounds

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Segmentation fault

- Runtime error
- Means you tried to access memory that you weren’t allowed to access
- Examples of causes:
  - trying to read from a file that wasn’t open
  - following a dangling pointer
  - accessing data beyond array bounds

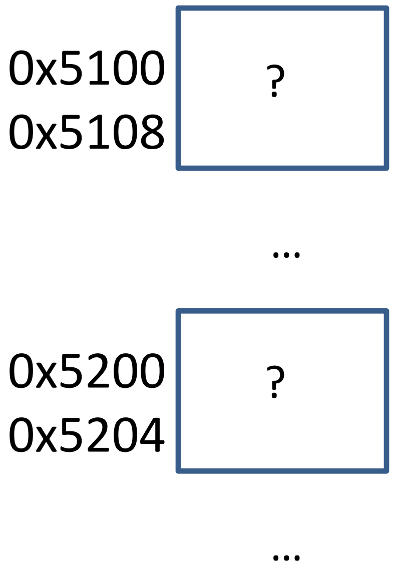
let’s introduce a segmentation fault in read.c

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
  int *b;
  int n;
  n = 5;
  b = &n;
```

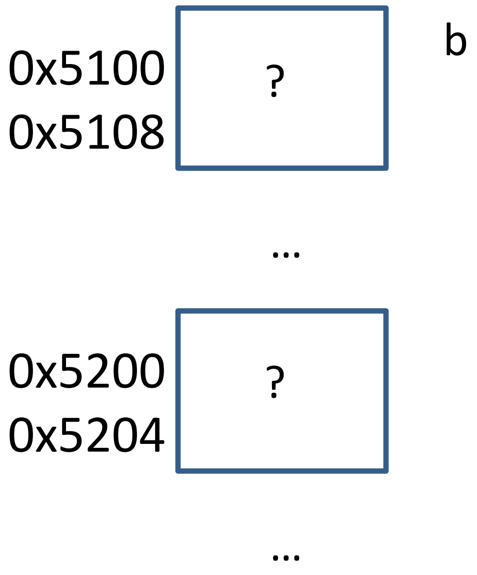


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
```

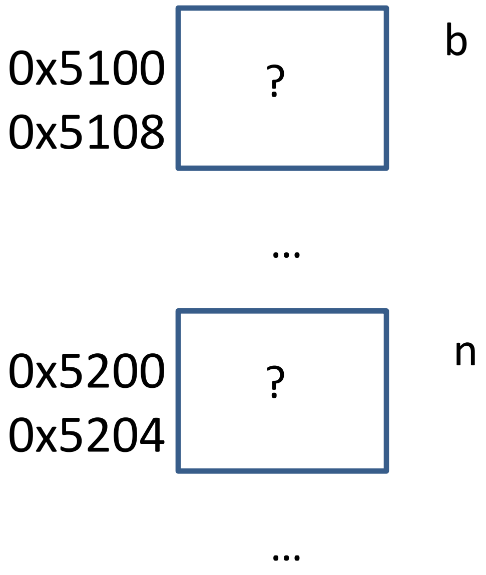


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
```

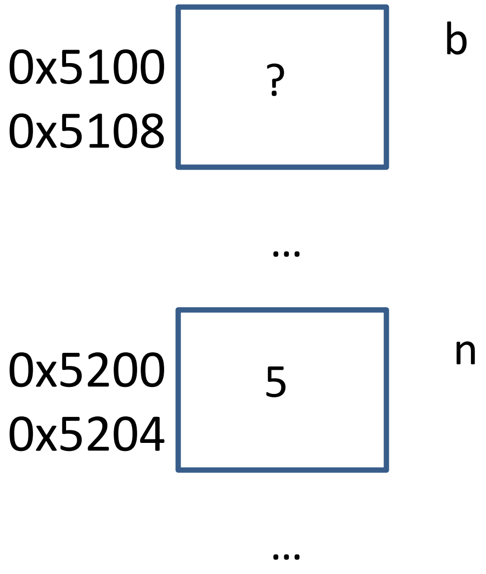


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
```

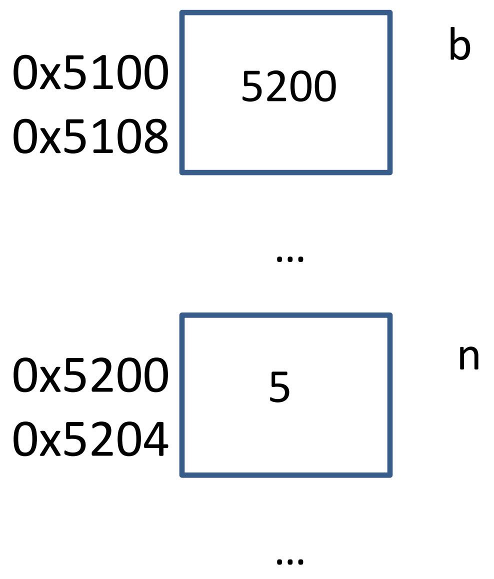


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
```

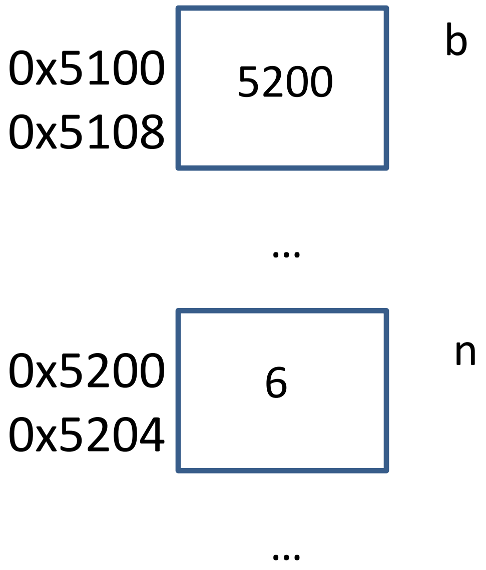


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
   n = 6;
   *b += 1;
   *b = 2 * (*b);
```

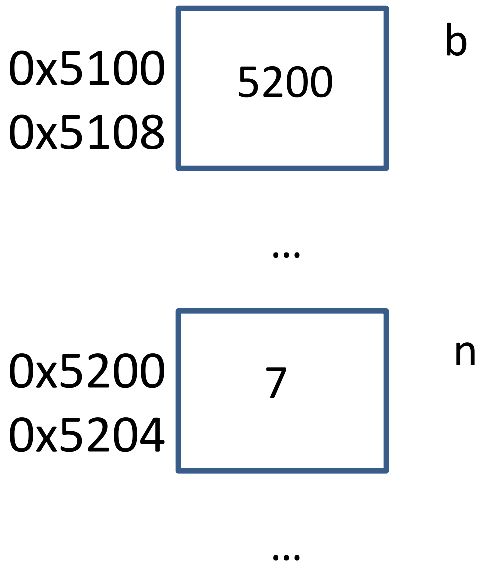


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
```

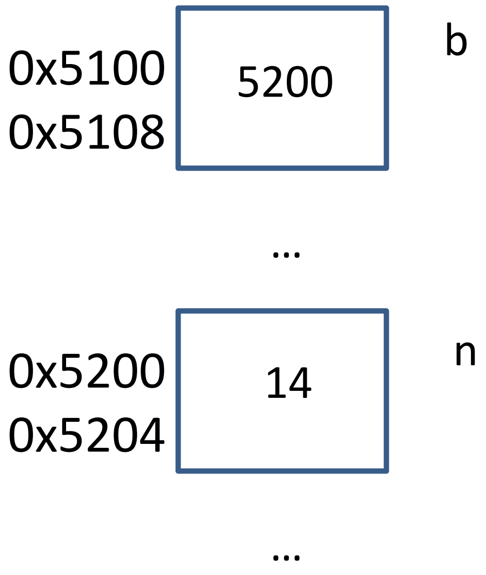


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
```

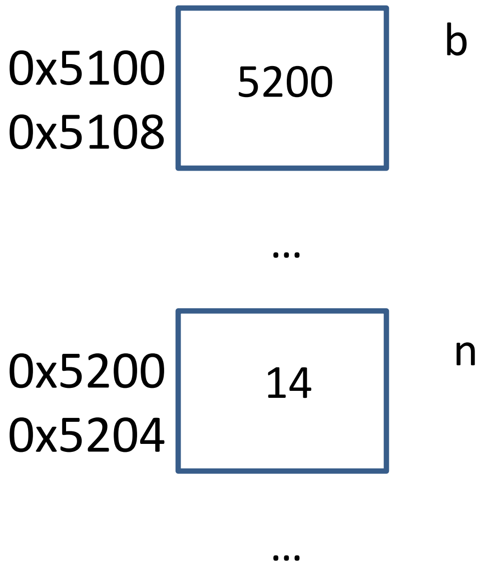


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
    b = 2 * (*b);
```

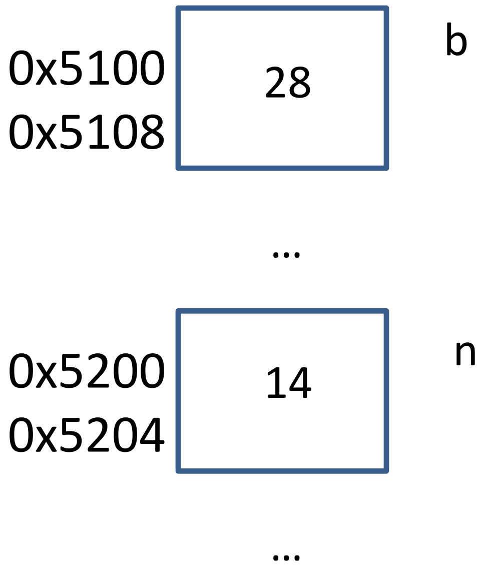


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
   n = 6;
   *b += 1;
   *b = 2 * (*b);
   b = 2 * (*b);
```

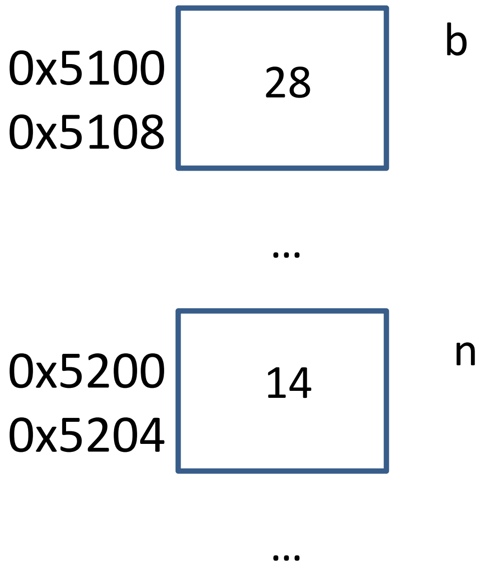


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Pointers

- Create an integer pointer variable and set it

```
int main(void) {
   int *b;
   int n;
   n = 5;
   b = &n;
   n = 6;
   *b += 1;
   *b = 2 * (*b);
   b = 2 * (*b);
```



ptr0.c shows seg fault accessing *b

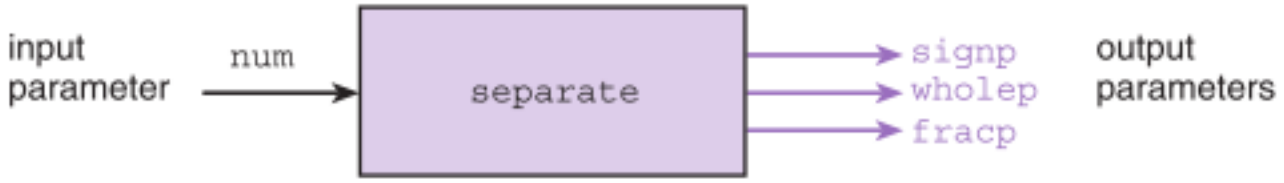
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

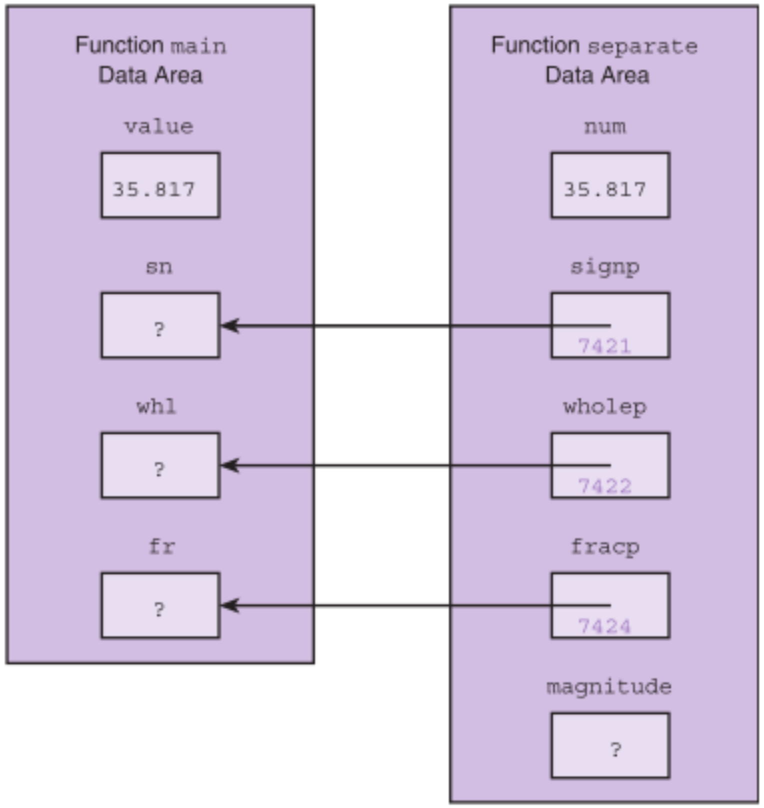
Diagram of Function separate with Multiple Results



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 6.6

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



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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)`

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Multiple Calls to a Function with Input/Output Parameters

An example of sorting data

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 6.7 Program to Sort Three Numbers

```c
/*
 * 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);
}
```

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```

/*
 * 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    */
```

(continued)

FIGURE 6.7 (continued)

```
        /* 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
```

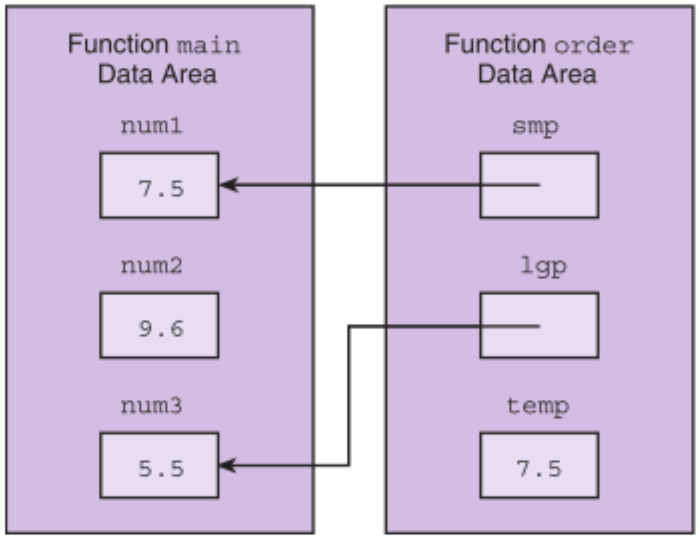
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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);`



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Scope of Names

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 6.9 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;
```

(continued)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 6.9 (continued)

```
        . . .
} /* 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 */
```

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Formal Output Parameters as Actual Arguments

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

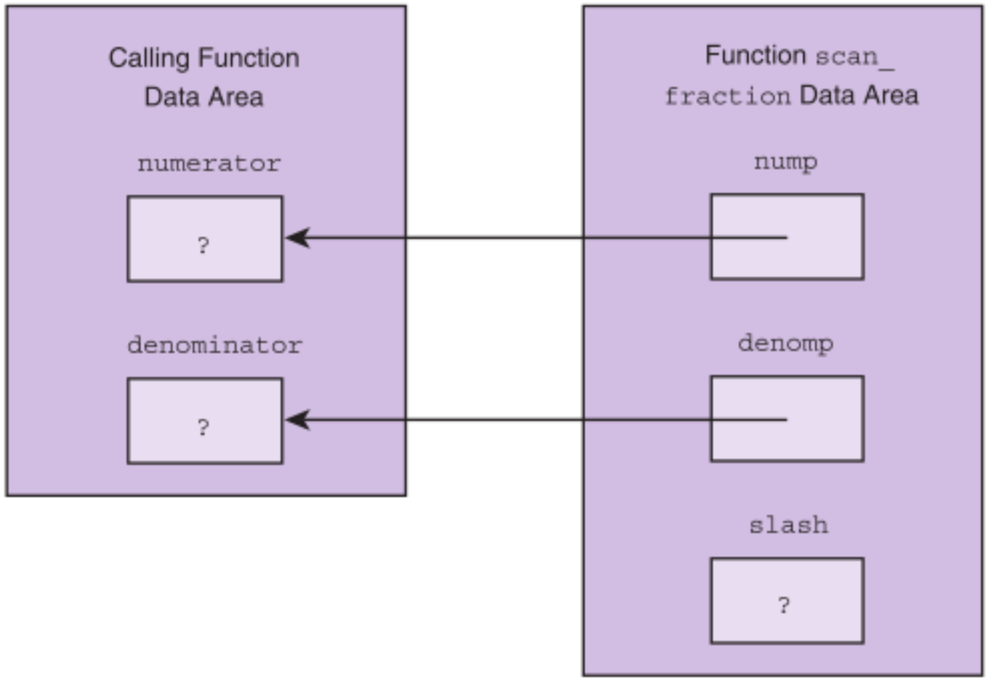
FIGURE 6.10 Function scan_fraction (incomplete)

```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;

          /* 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);
}
```

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 6.11

Data Areas for scan_fraction and Its Caller



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.