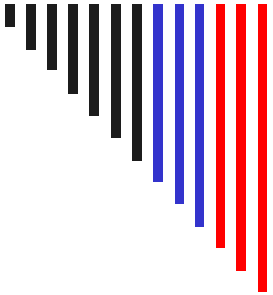

Chapter 7
Text Input/Output

Objectives

- To understand the basic properties and characteristics of external files
- To understand the C implementation of file I/O using streams
- To write programs that read and write text files using the formatting functions
- To write programs that read and write text files using the C character I/O functions
- To write programs that handle simple I/O errors
- To understand and implement basic data validation concepts

7-1 Files

A file is an external collection of related data treated as a unit. The primary purpose of a file is to keep a record of data. Since the contents of primary memory are lost when the computer is shut down, we need files to store our data in a more permanent form.

Topics discussed in this section:

File Name

File Information Table

Note

A file is an external collection of related data treated as a unit.

File Name

- Operating system rules for naming files
- “input.txt”
- “ input.txt”
- “../input.txt”

File Information Table

- File name, the current character position, and so on
- FILE type
  - A structure type to hold information of a file
- stdio.h header file

7-2 Streams

As we briefly discussed in Chapter 2, data is input to and output from a stream. A stream can be associated with a physical device, such as a terminal, or with a file stored in auxiliary memory.

Topics discussed in this section:

Text And Binary Streams
Stream-File Processing
System-Created Streams

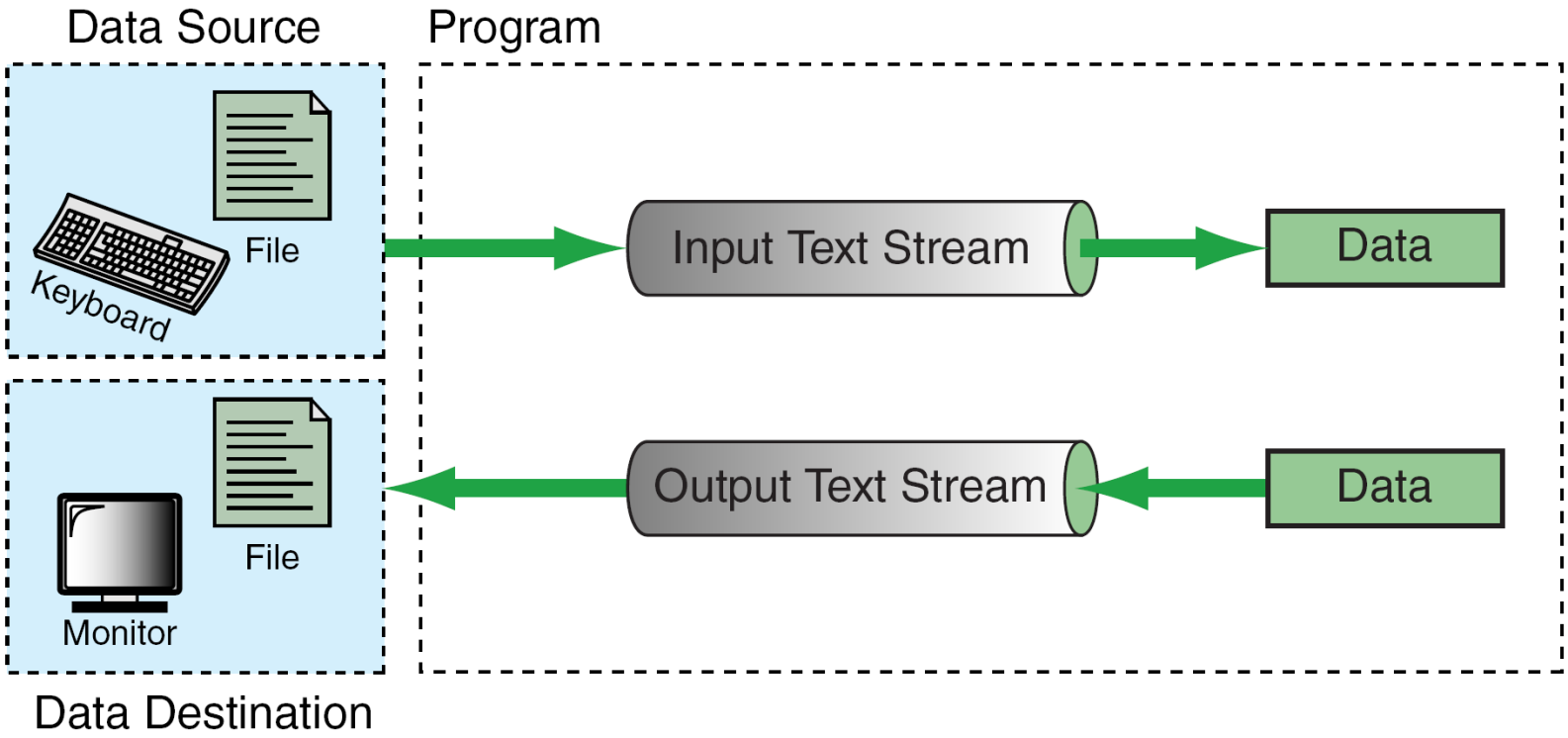

FIGURE 7-1 Streams

Text and Binary Streams

- Text stream
  - A sequence of characters divided into lines with each line terminated by a new line (\ n)
- Binary stream
  - A sequence of data values, such as integer or real, using their memory representations

Stream-File Processing

- Creating a stream
  - FILE* spData;
- Opening a file
- Using the stream name
- Closing the stream

System-Created Streams

- stdio.h
- stdin: the standard input stream
- stout: the standard output stream
- stderr: the standard error stream
- Standard I/O streams are automatically created when the program starts
- What are the defaults?
- How to use them?

Note

Standard stream names have already been declared in the *stdio.h* header file and cannot be declared again in our program.

Note

There is no need to open and close the standard streams. It is done automatically by the operating system.

7-3 Standard Library Input/Output Functions

The stdio.h header file contains several different input/output function declarations. They are grouped into eight different categories, as shown in Figure 7-2. The first three will be discussed in the following sections. Those shown in shaded boxes will be discussed in Chapters 11 and 13.

Topics discussed in this section:

File Open and Close

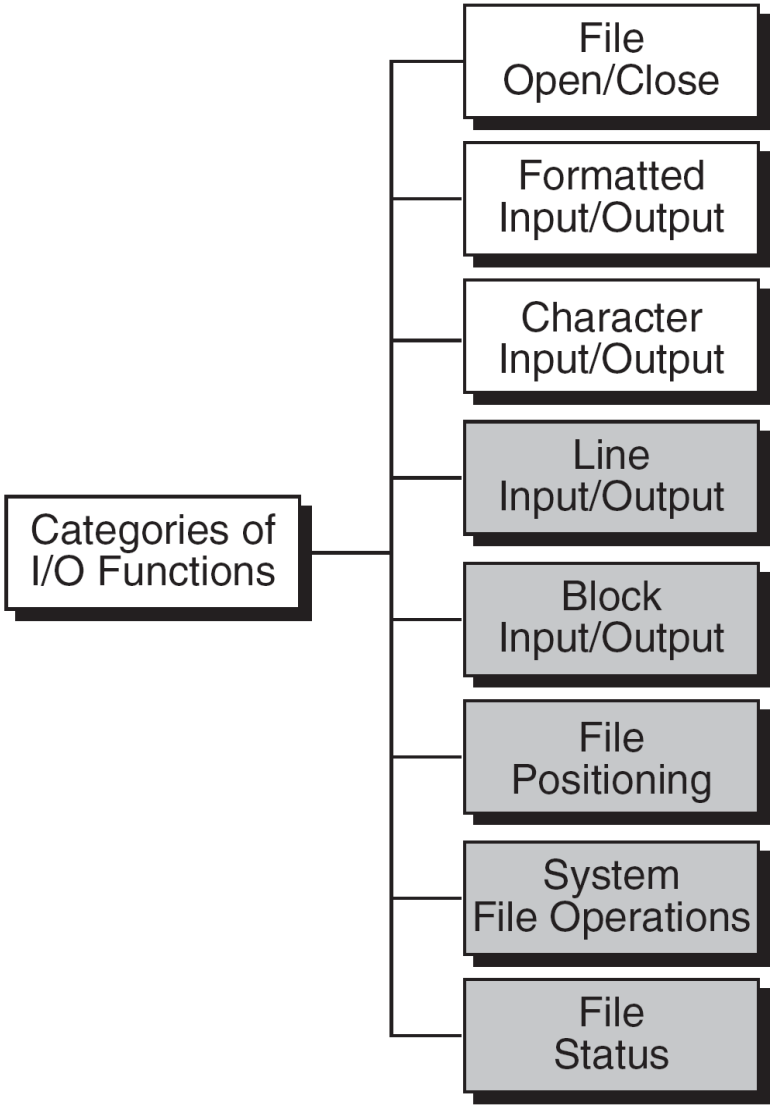

FIGURE 7-2 Categories of Standard Input/Output Functions

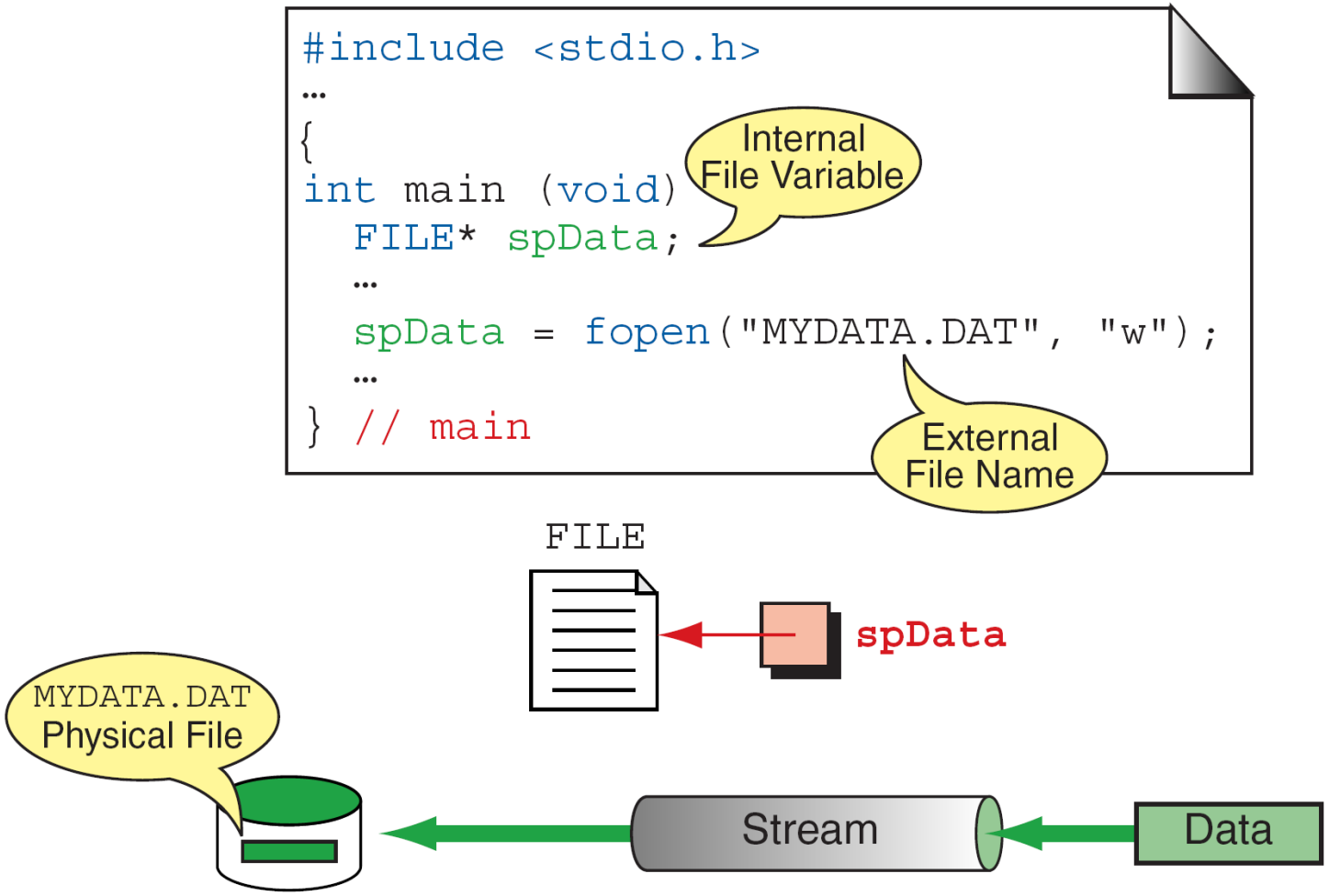

FIGURE 7-3 File Open Results

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

Table 7-1 Text File Modes

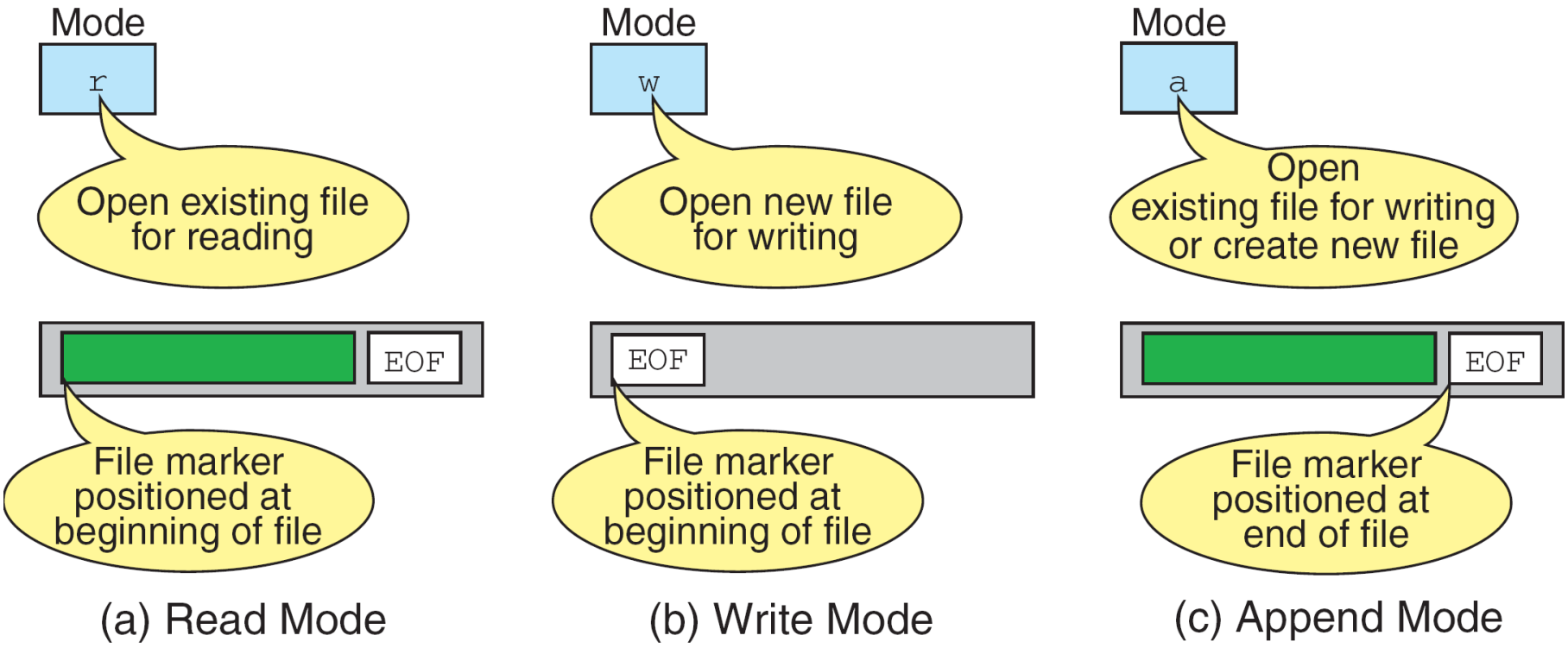

FIGURE 7-4 File-Opening Modes

PROGRAM 7-1 Testing for Open and Close Errors

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  ...
 4
 5  int main (void)
 6  {
 7  // Local Declarations
 8  FILE* spTemps;
 9
10  // Statements
11     ...
12
13     if ((spTemps = fopen("TEMPS.DAT", "r")) == NULL)
14        {
15         printf("\aERROR opening TEMPS.DAT\n");
16         exit (100);
17        } // if open
```

PROGRAM 7-1 Testing for Open and Close Errors

```
18      ...
19
20      if (fclose(spTemps) == EOF)
21         {
22          printf("\aERROR closing TEMPS.DAT\n");
23          exit (102);
24         } // if close
25      ...
26
27  }  // main
```

7-4 Formatting Input/Output Functions

In Chapter 2 we introduced two formatting input/output functions, scanf and printf. These two functions can be used only with the keyboard and monitor. The C library defines two more general functions, fscanf and fprintf, that can be used with any text stream.

Topics discussed in this section:

Stream Pointer
Format Control Strings
Input Formatting (*scanf* and *fscanf*)
Output Formatting (*printf* and *fprintf*)

Terminal Input/Output
`scanf ("control string", ...);` `printf("control string", ...);`
General Input/Output
`fscanf (stream_pointer, "control string", ...);` `fprintf(stream_pointer, "control string", ...);`

Table 7-2 Formatting Functions

Stream Pointer

- FILE* spIn;
- FILE* spOut;

- spIn = fopen("file name", "r");
- spOut = fopen("file name", "w");

- fscanf(spIn, "format string", addresses);
- fprintf(spOut, "format string", expressions);

Note

A whitespace character in an input format string causes leading whitespace characters in the input to be discarded. A whitespace character in an output format string is copied to the output stream.

Note

The number, order, and type of the conversion specifications must match the number, order, and type of the parameters in the list. Otherwise, the result will be unpredictable and may terminate the input/output function.

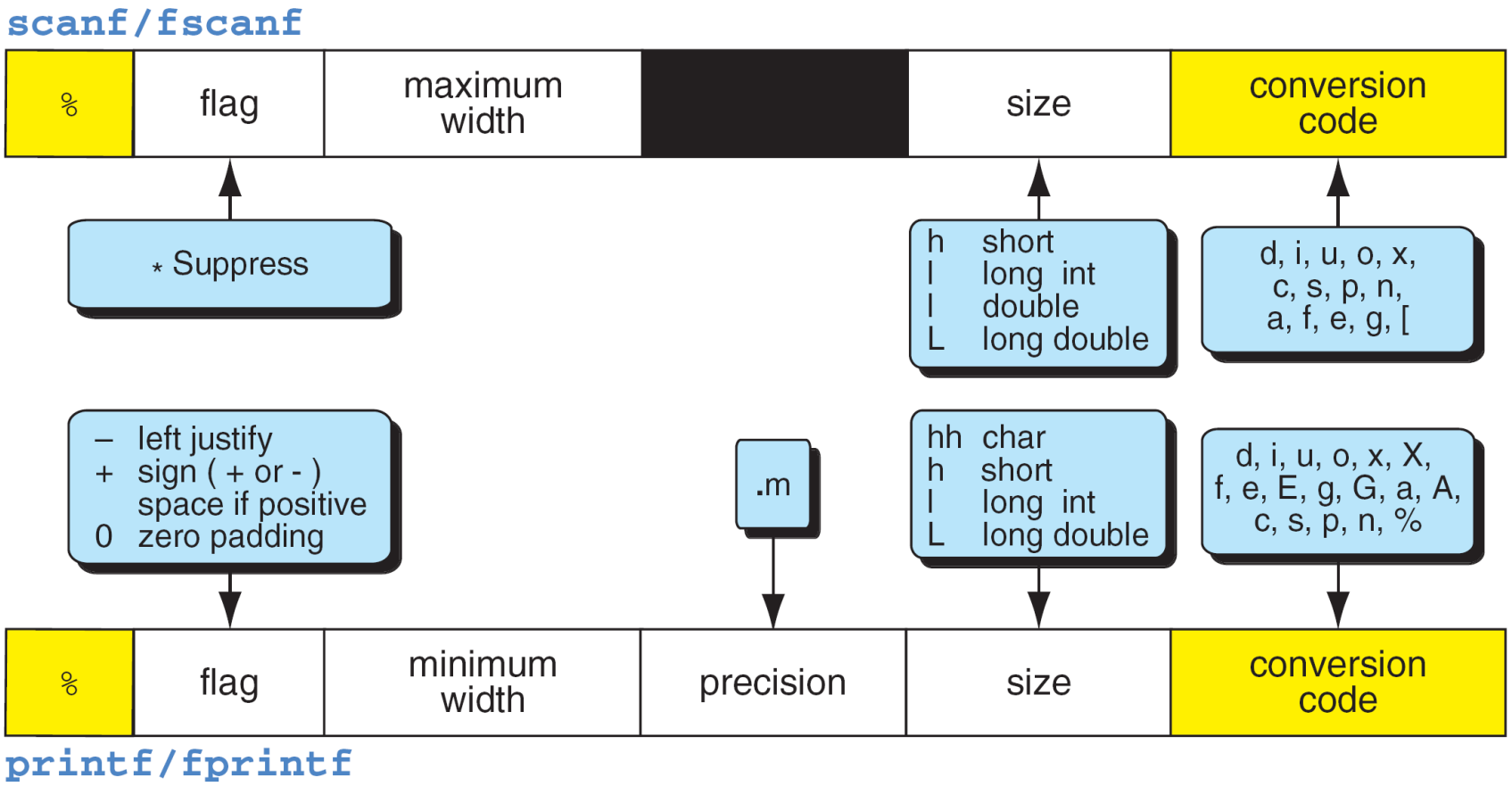

FIGURE 7-5 Conversion Specifications

Note

***scanf* reads from *stdin*;**
***fscanf* reads from a user-specified stream.**

Argument Type	Size Specifier	Code
integral	hh (char), h (short), none (int), l (long), ll (long long)	i
integer	h (short), none (int), l (long). ll (long long)	d
unsigned int	hh (char), h (short), none (int), l (long), ll (long long)	u
character octal	hh (unsigned char)	o
integer hexadecimal	h (short), none (int), l (long), ll (long long)	x
real	none (double), l (double), L (double)	f
real (scientific)	none (double), l (double), L (double)	e
real (scientific)	none (double), l (double), L (double)	g
real (hexadecimal)	none (double), l (double), L (double)	a

Table 7-3 Sizes and Conversion Code for *scanf* Family

Argument Type	Size Specifier	Code
character	none (char), l (wchar_t)	c
string	none (char string), l (wchar_t string)	s
pointer		p
integer (for count)	none (int), hh (char), h (short), l (long), ll (long long)	n
set	none (char), l (wchar_t)	[

Table 7-3 Sizes and Conversion Code for *scanf* Family

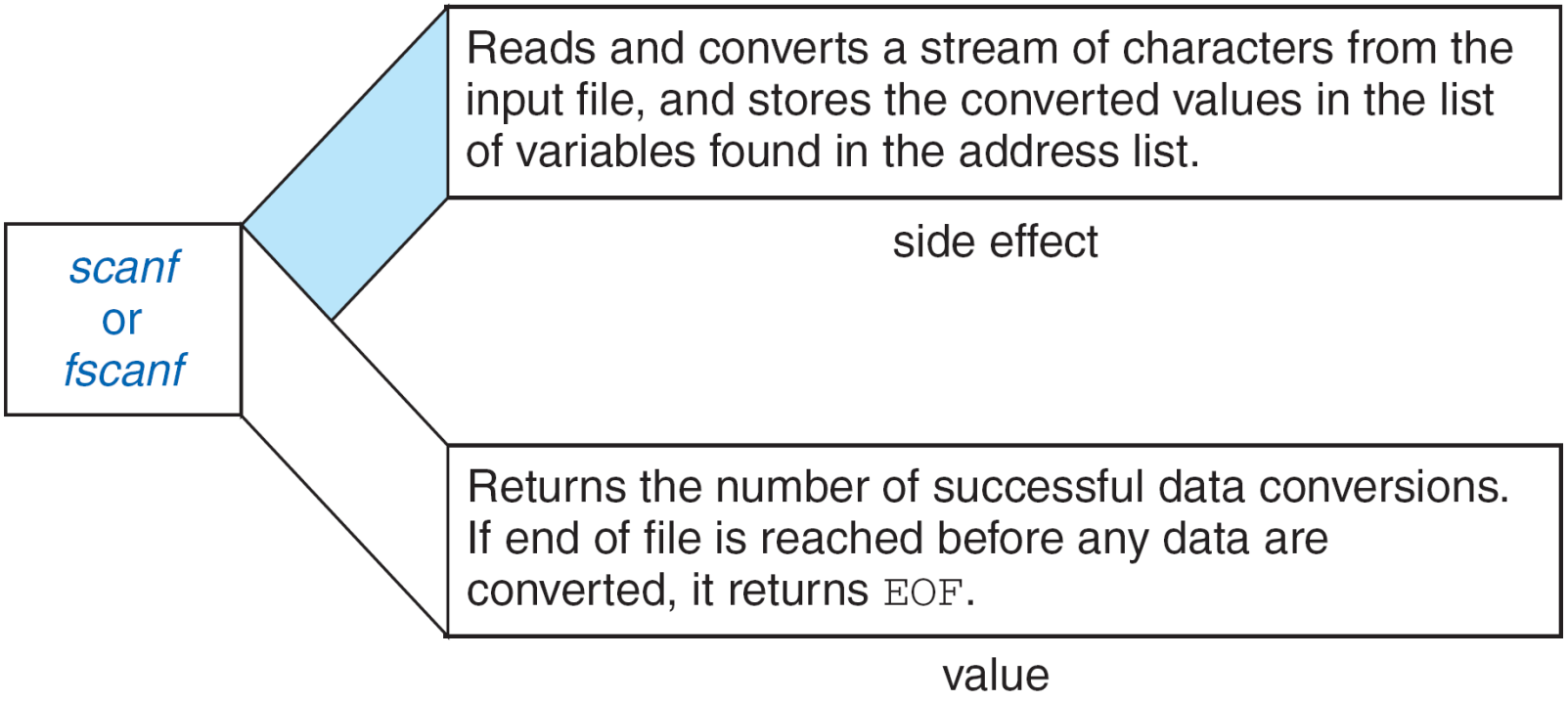

FIGURE 7-6 Side Effect and Value of *scanf* and *fscanf*

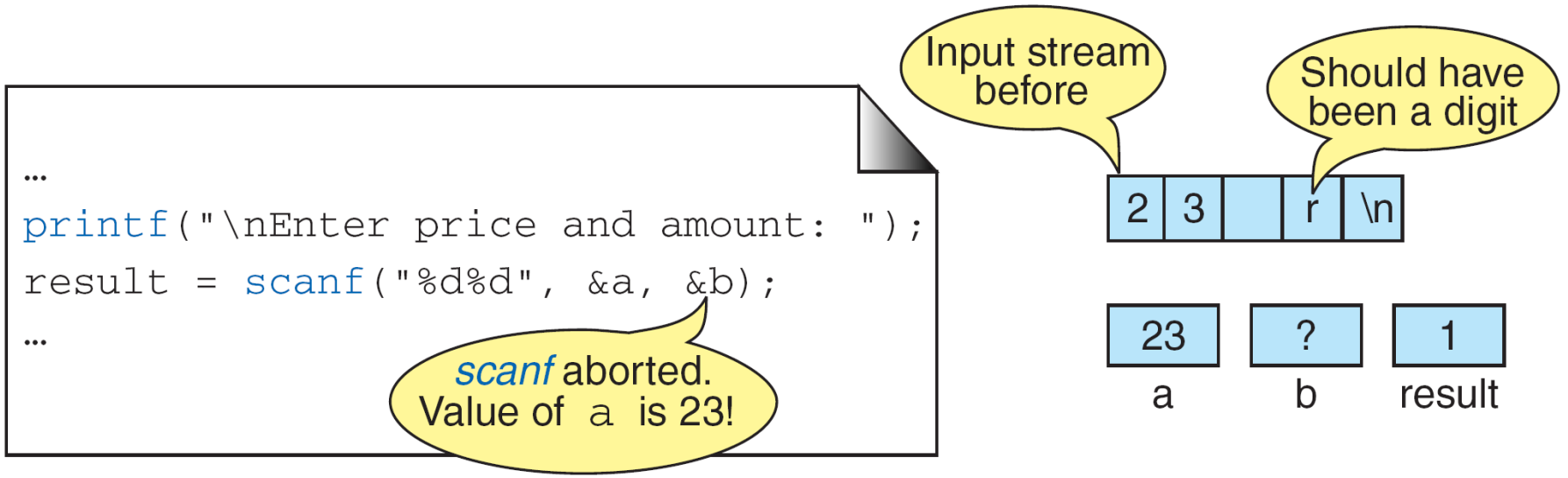

FIGURE 7-7 Another Common Error

PROGRAM 7-2 Checking *scanf* Results

```
1  #define FLUSH while (getchar() != '\n')
2  #define ERR1 "\aPrice incorrect. Re-enter both fields\n"
3  #define ERR2 "\aAmount incorrect. Re-enter both fields\n"
4
5  // Read price and amount
6      do
7        {
8         printf("\nEnter amount and price: ");
9         ioResult = scanf("%d%f", &amount, &price);
10
11        if (ioResult != 2)
12           {
13            FLUSH;
14            if (ioResult == 1)
15                printf(ERR1);
16            else
```

PROGRAM 7-2 Checking *scanf* Results

```
17                printf(ERR2);
18            } // if
19        } while (ioResult != 2);
```

```
Results:
Enter amount and price: ? 15.25
Amount incorrect. Re-enter both fields

Enter amount and price: 100 ?
Price incorrect. Re-enter both fields

Enter amount and price: 100 15.25
```

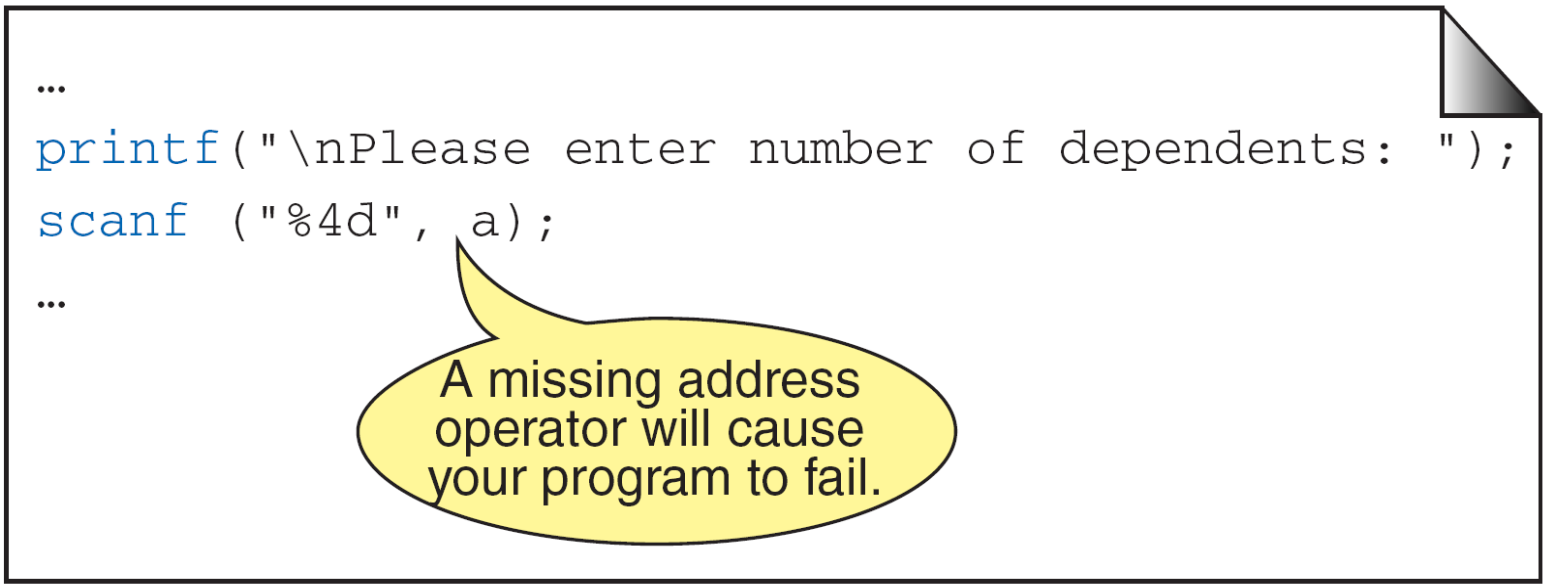

FIGURE 7-8 Missing Address Operator in *scanf*

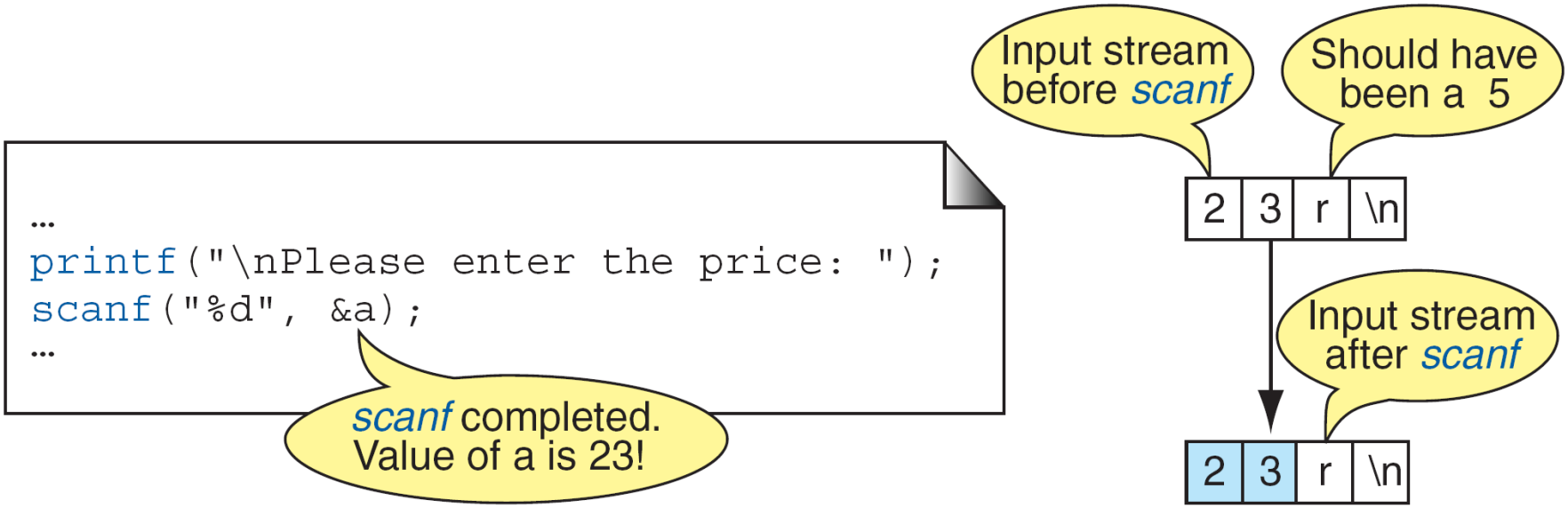

FIGURE 7-9 Data Type Conflict

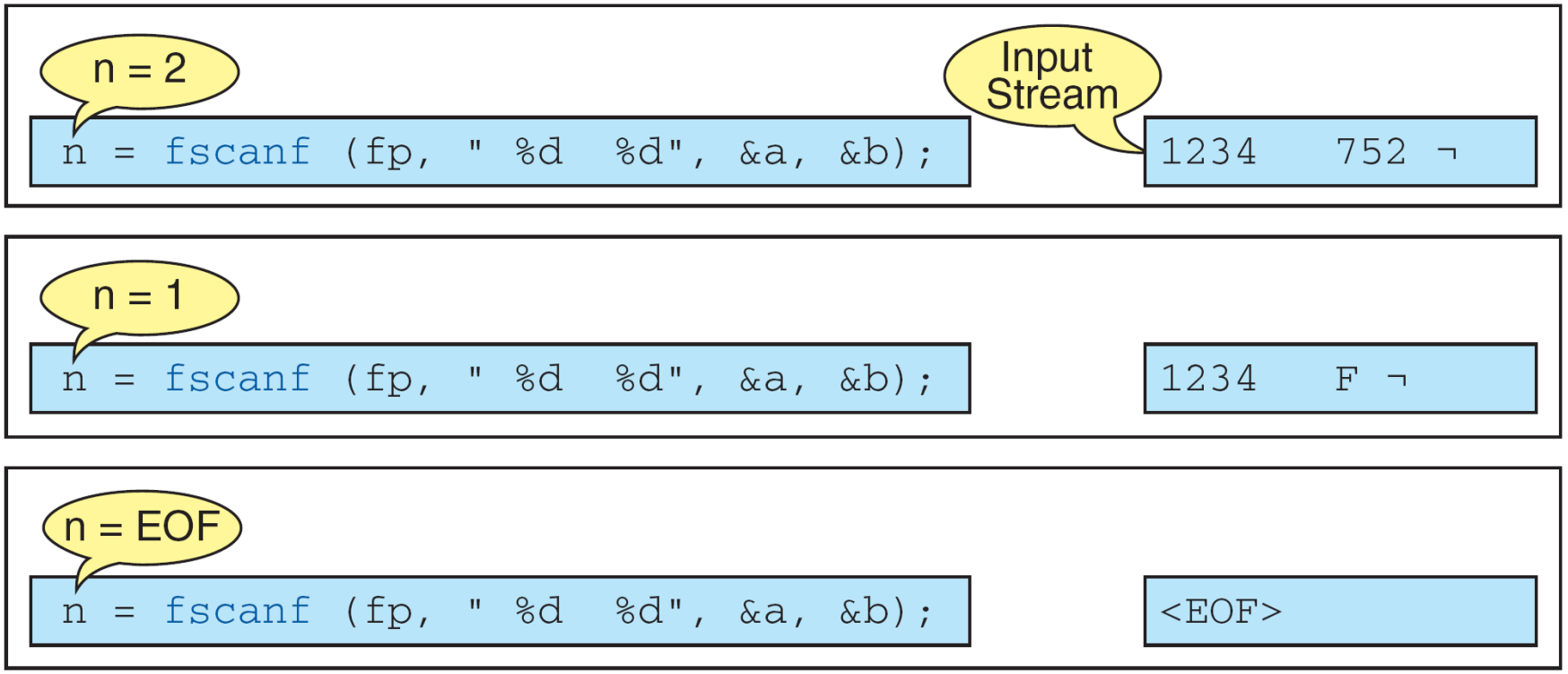

FIGURE 7-10 ***fscanf*** **Examples**

Note

Discarding the return character is different from consuming it.
Discarding can be done by whitespaces in the control string;
consuming can only be done by reading the return character with a %c.

Input Examples & Input Stream Concerns

- Study yourself!

Argument Type	Flag	Size Specifier	code
integer	-, +, 0, space	hh (char), h (short), none (int), l (long), ll (long long)	d, i
unsigned integer	-, +, 0, space	hh (char), h (short), none (int), l (long), ll (long long)	u
integer (octal)	-, +, 0, #, space	hh (char), h (short), none (int), l (long), ll (long long)	o
integer (hex)	-, +, 0, #, space	hh (char), h (short), none (int), l (long), ll (long long)	x, X
real	-, +, 0, #, space	none (double), l (double), L (double)	f
real (scientific)	-, +, 0, #, space	none (double), l (double), L (double)	e, E
real (scientific)	-, +, 0, #, space	none (double), l (double), L (double)	g, G

Table 7-4 Flags, Sizes, and Conversion Codes for *printf* Family

Argument Type	Flag	Size Specifier	code
real (hexadecimal)	-, +, 0, #, space	none (double), l (double), L (double)	a, A
character	-	none (char), l (w-char)	c
string	-	none (char string), l (w-char string)	s
pointer			p
integer (for count)		none (int), h (short), l (long)	n
to print %			%

Table 7-4 Flags, Sizes, and Conversion Codes for *printf* Family

Flag Type	Flag Code	Formatting
Justification	none	right justified
	–	left justified
Padding	none	space padding
	0	zero padding
Sign	none	positive value: no sign negative value: –
	+	positive value: + negative value: –
	space	positive value: space negative value: –
Alternate	#	print alternative format for scientific, hexadecimal, and octal.

Table 7-5 Flag Formatting Options

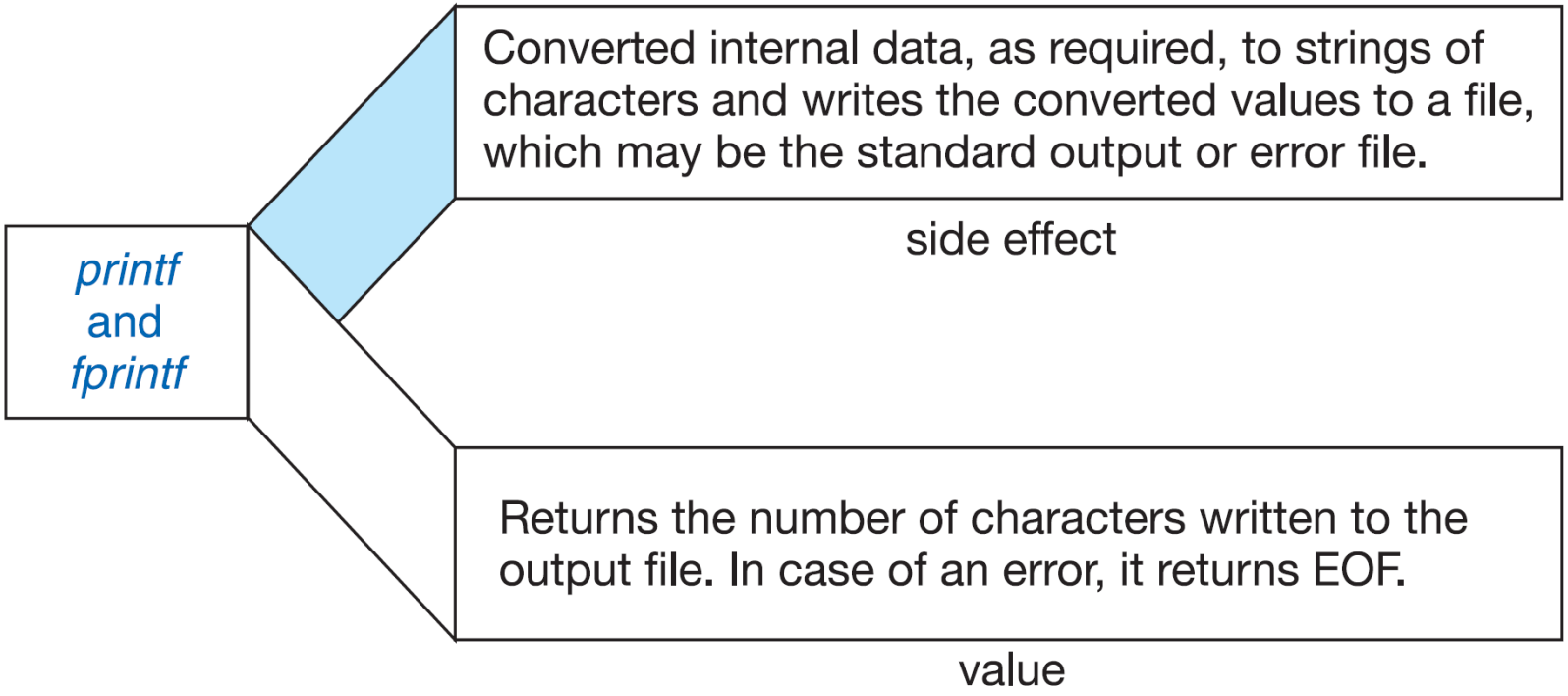

FIGURE 7-11 Side Effect and Value of *printf*

Output Examples

- Study yourself!

PROGRAM 7-3 Read and Print Text File of Integers

```
 1  /* Read a text file of integers, and print the contents.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  int main (void)
 9  {
10  // Local Declarations
11     FILE* spIn;
12     int   numIn;
13
14  // Statements
15     spIn = fopen("P07-03.DAT", "r");
16     if (!spIn)
17        {
```

PROGRAM 7-3 Read and Print Text File of Integers

```
18          printf("Could not open file\a\n");
19          exit  (101);
20         } // if open fail
21
22     while ((fscanf(spIn, "%d", &numIn)) == 1)
23        printf("%d ", numIn);
24
25     return 0;
26  }  // main
```

```
Results:
1 2 3 4 5 6 7 8 9 10
```

PROGRAM 7-4 Copy Text File of Integers

```
 1  /* Copy a text file of integers.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  int main (void)
 9  {
10  // Local Declarations
11     FILE* spIn;
12     FILE* spOut;
13     int   numIn;
14     int   closeResult;
15
16  // Statements
17     printf("Running file copy\n");
```

PROGRAM 7-4 Copy Text File of Integers

```
19      if (!spIn)
20         {
21          printf("Could not open input file\a\n");
22          exit  (101);
23         } // if open fail
24
25      spOut = fopen("P07-04.DAT", "w");
26      if (!spOut)
27         {
28          printf("Could not open output file\a\n");
29          exit  (102);
30         } // if open fail
31
32      while ((fscanf(spIn, "%d", &numIn)) == 1)
33         fprintf(spOut, "%d\n", numIn);
34
35      closeResult = fclose(spOut);
```

PROGRAM 7-4 Copy Text File of Integers

```
36      if (closeResult == EOF)
37         {
38          printf("Could not close output file\a\n");
39          exit  (201);
40         } // if close fail
41
42      printf("File copy complete\n");
43      return 0;
44  }  // main
```

```
Results:
Running file copy
File copy complete
```

PROGRAM 7-5 Append Data to File

```
 1  /* Copy a text file of integers.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  int main (void)
 9  {
10  // Local Declarations
11     FILE* spAppnd;
12     int   numIn;
13     int   closeResult;
14
15  // Statements
16     printf("This program appends data to a file\n");
17     spAppnd = fopen("P07-05.DAT", "a");
18     if (!spAppnd)
```

PROGRAM 7-5 Append Data to File

```
19          {
20           printf("Could not open input file\a\n");
21           exit  (101);
22          } // if open fail
23
24       printf("Please enter first number:  ");
25       while ((scanf("%d", &numIn)) != EOF)
26          {
27           fprintf(spAppnd, "%d\n", numIn);
28           printf("Enter next number or <EOF>: ");
29          } // while
30
31       closeResult = fclose(spAppnd);
32       if (closeResult == EOF)
33          {
34           printf("Could not close output file\a\n");
35           exit  (201);
36          } // if close fail
```

PROGRAM 7-5 Append Data to File

```
37
38      printf("\nFile append complete\n");
39      return 0;
40  }   // main
```

```
Results:
This program appends data to a file
Please enter first number:  1
Enter next number or <EOF>: 2
Enter next number or <EOF>: 3
Enter next number or <EOF>:^d
File append complete
```

PROGRAM 7-6 Student Grades

```
1  /* Create a grades file for transmission to Registrar.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  // Function Declarations
8  int getStu     (FILE* spStu,
9                  int* stuID,  int* exam1,
10                 int*  exam2, int* final);
11 int writeStu   (FILE* spGrades,
12                 int   stuID, int   avrg,  char grade);
13 void calcGrade (int   exam1, int   exam2, int  final,
14                 int*  avrg,  char* grade);
15
16 int main (void)
17 {
18 // Local Declarations
19    FILE* spStu;
20    FILE* spGrades;
```

PROGRAM 7-6 Student Grades

```
21
22      int stuID;
23      int exam1;
24      int exam2;
25      int final;
26      int avrg;
27
28      char grade;
29
30  // Statements
31      printf("Begin student grading\n");
32      if (!(spStu = fopen ("P07-06.DAT", "r")))
33         {
34          printf("\aError opening student file\n");
35          return 100;
36         } // if open input
37
38      if (!(spGrades = fopen ("P07-06Gr.DAT", "w")))
39         {
```

PROGRAM 7-6 Student Grades

```
40          printf("\aError opening grades file\n");
41          return 102;
42        } // if open output
43
44     while (getStu
45           (spStu, &stuID, &exam1, &exam2, &final))
46        {
47          calcGrade (exam1, exam2, final, &avrg, &grade);
48          writeStu  (spGrades, stuID, avrg, grade);
49        } // while
50
51     fclose (spStu);
52     fclose (spGrades);
53
54     printf("End student grading\n");
55     return 0;
56  }  // main
57
```

PROGRAM 7-6 Student Grades

```
58 /*============================getStu============================
59    Reads data from student file.
60       Pre   spStu is an open file.
61             stuID, exam1, exam2, final pass by address
62       Post  reads student ID and exam scores
63             if data read   --returns 1
64             if EOF or error--returns 0
65 */
66 int getStu (FILE* spStu, int* stuID, int* exam1,
67             int*  exam2, int* final)
68 {
69 // Local Declarations
70    int ioResult;
71
72 // Statements
73    ioResult = fscanf(spStu, "%d%d%d%d", stuID,
74                      exam1, exam2, final);
75    if (ioResult == EOF)
76        return 0;
77    else if (ioResult != 4)
```

PROGRAM 7-6 Student Grades

```
78          {
79           printf("\aError reading data\n");
80           return 0;
81          } // if
82       else
83          return 1;
84    } // getStu
85
86    /* ==================== calcGrade ====================
87       Determine student grade based on absolute scale.
88          Pre   exam1, exam2, and final contain scores
89                avrg and grade are addresses of variables
90          Post  Average and grade copied to addresses
91    */
92    void calcGrade (int  exam1, int   exam2, int final,
93                    int* avrg,  char* grade)
94    {
95    // Statements
96       *avrg = (exam1 + exam2 + final) / 3;
```

PROGRAM 7-6 Student Grades

```
 97      if (*avrg >= 90)
 98         *grade = 'A';
 99      else if (*avrg >= 80)
100         *grade = 'B';
101      else if (*avrg >= 70)
102         *grade = 'C';
103      else if (*avrg >= 60)
104         *grade = 'D';
105      else
106         *grade = 'F';
107      return;
108   }  // calcGrade
109
110   /* ====================== writeStu ====================
111      Writes student grade data to output file.
112         Pre   spGrades is an open file
113               stuID, avrg, and grade have values to write
114         Post  Data written to file
115   */
```

PROGRAM 7-6 Student Grades

```
116 int writeStu  (FILE* spGrades, int  stuID,
117                int   avrg,     char grade)
118 {
119 // Statements
120    fprintf(spGrades, "%04d %d %c\n",
121                       stuID, avrg, grade);
122    return 0;
123 }  // writeStu
```

```
Results:
Begin student grading
End student grading

Input--------
0090 90 90 90
0089 88 90 89
0081 80 82 81
0079 79 79 79
0070 70 70 70
0069 69 69 69
0060 60 60 60
0059 59 59 59
```

PROGRAM 7-6 Student Grades

```
Output----
0090 90 A \n
0089 89 B \n
0081 81 B \n
0079 79 C \n
0070 70 C \n
0069 69 D \n
0060 60 D \n
0059 59 F \n
```

7-5 Character Input/Output Functions

Character input functions read one character at a time from a text stream. Character output functions write one character at the time to a text stream.

Topics discussed in this section:

Terminal Character I/O
Terminal and File Character I/O
Character Input/Output Examples

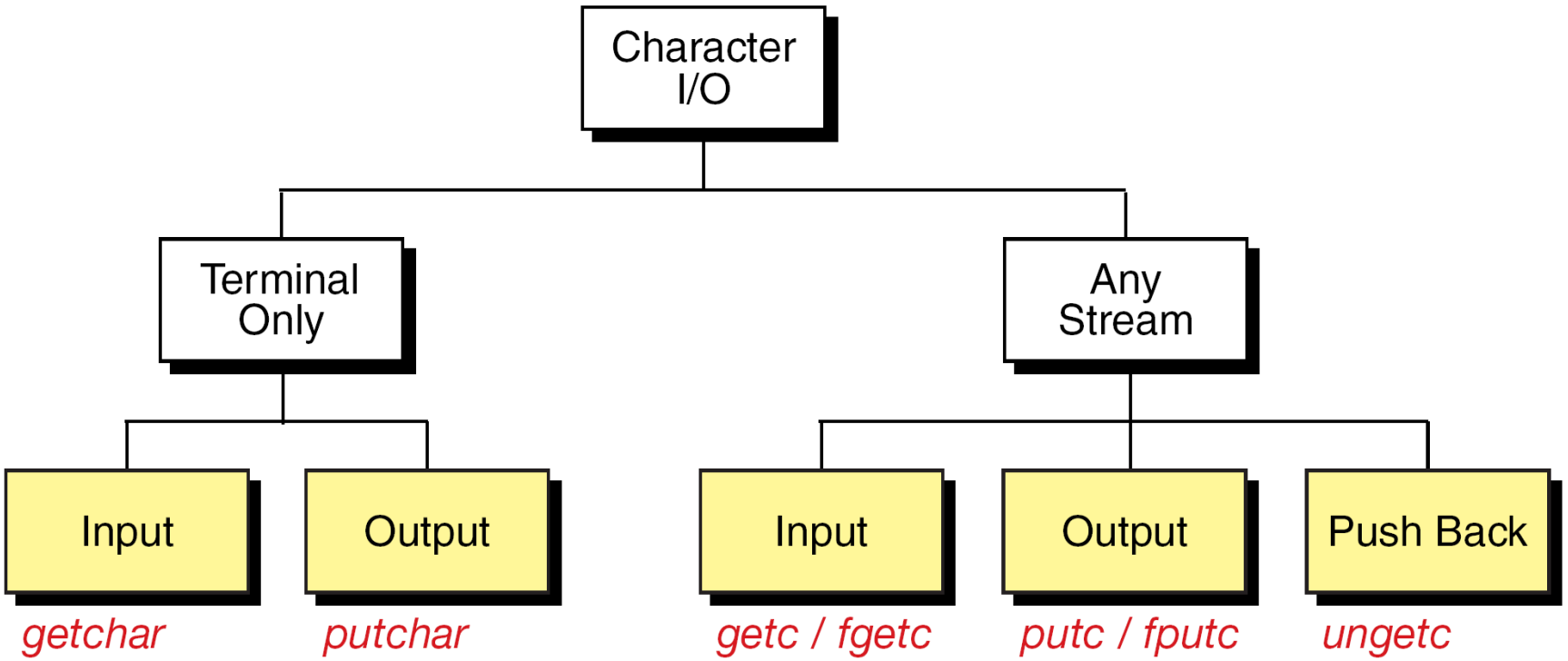

FIGURE 7-12 Character Input/Output Functions

Character I/O

- int getchar (void)
- int putchar (int out_char);
- int fgetc (FILE* spIn);
- int fputc (int oneChar, FILE* spOut);
- int ungetc(int oneChar, FILE* spData);

PROGRAM 7-7 Create Text File

```
1  /* This program creates a text file.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  int main (void)
7  {
8  // Local Declarations
9     FILE* spText;
10    int   c;
11    int   closeStatus;
12
13 // Statements
14    printf("This program copies input to a file.\n");
15    printf("When you are through, enter <EOF>.\n\n");
16
17    if (!(spText = fopen("P07-07.DAT","w")))
```

PROGRAM 7-7 Create Text File

```
18          {
19           printf("Error opening P07.07.DAT for writing");
20           return (1);
21          } // if open
22
23       while ((c = getchar()) != EOF)
24          fputc(c, spText);
25
26       closeStatus = fclose(spText);
27       if (closeStatus == EOF)
28          {
29           printf("Error closing file\a\n");
30           return 100;
31          } // if
32
33       printf("\n\nYour file is complete\n");
34       return 0;
35   }   // main
```

PROGRAM 7-7 Create Text File

```
Results:
This program copies input to a file.
When you are through, enter <EOF>.

Now is the time for all good students
To come to the aid of their college.^d

Your file is complete
```

PROGRAM 7-8 Copy Text File

```
1  /* This program copies one text file into another.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  int main (void)
7  {
8  // Local Declarations
9     int   c;
10    int   closeStatus;
11    FILE* sp1;
12    FILE* sp2;
13
14 // Statements
15    printf("Begin file copy\n");
16
17    if (!(sp1 = fopen ("P07-07.DAT", "r")))
18       {
19        printf("Error opening P07-07.DAT for reading");
20        return (1);
```

PROGRAM 7-8 Copy Text File

```
21          } // if open input
22       if (!(sp2 = fopen ("P07-08.DAT", "w")))
23          {
24           printf("Error opening P07-08.DAT for writing");
25           return (2);
26          } // if open output
27
28       while ((c = fgetc(sp1)) != EOF)
29          fputc(c, sp2);
30
31       fclose(sp1);
32       closeStatus = fclose(sp2);
33       if (closeStatus == EOF)
34          {
35           printf("File close error.\a\n");
36           return 201;
37          } // if close error
38       printf("File successfully created\n");
39       return 0;
40    }  // main
```

PROGRAM 7-9 Count Characters and Lines

```
 1  /* This program counts characters and lines in a program.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  int main (void)
 7  {
 8  // Local Declarations
 9     int   curCh;
10     int   preCh;
11     int   countLn = 0;
12     int   countCh = 0;
13     FILE* sp1;
14
15  // Statements
16     if (!(sp1 = fopen("P07-07.DAT", "r")))
17        {
18         printf("Error opening P07-07.DAT for reading");
19         return (1);
20        } // if open error
21
```

PROGRAM 7-9 Count Characters and Lines

```
22    while ((curCh = fgetc(sp1)) != EOF)
23        {
24         if (curCh != '\n')
25             countCh++;
26         else
27             countLn++;
28         preCh = curCh;
29        } // while
30
31    if (preCh != '\n')
32         countLn++;
33
34    printf("Number of characters: %d\n", countCh);
35    printf("Number of lines     : %d\n", countLn);
36    fclose(sp1);
37   return 0;
38  }  // main
```

```
Results:
Number of characters: 74
Number of lines:      2
```

PROGRAM 7-10 Count Words

```
 1  /* Count number of words in file. Words are separated by
 2     whitespace characters: space, tab, and newline.
 3        Written by:
 4        Date:
 5  */
 6  #include <stdio.h>
 7
 8  #define WHT_SPC\
 9           (cur == ' ' || cur == '\n' || cur ==  '\t')
10  int main (void)
11  {
12  //Local Declarations
13     int   cur;
14     int   countWd = 0;
15     char  word = 'O';        // O out of word: I in word
16     FILE* sp1;
17
18  // Statements
19     if (!(sp1 = fopen("P07-07.DAT", "r")))
20        {
21         printf("Error opening P07-07.DAT for reading");
```

PROGRAM 7-10 Count Words

```
22          return (1);
23        } // if file open error
24     while ((cur = fgetc(sp1)) != EOF)
25        {
26         if (WHT_SPC)
27             word = 'O';
28         else
29             if (word ==  'O')
30                {
31                 countWd++;
32                 word = 'I';
33                } // else
34        } // while
35     printf("The number of words is: %d\n", countWd);
36
37     fclose(sp1);
38     return 0;
39  }  // main
```

```
Results:
The number of words is: 15
```

7-6 Software Engineering

In this section, we discuss some software engineering issues related to files.

Topics discussed in this section:

Testing Files
Data Terminology

PROGRAM 7-11 Handling Errors—the Right Way

```
1  #define FLUSH while (getchar() != '\n')
2  …
3     printf("\nPlease enter Number of Units Sold: ");
4     while (scanf("%d", &unitsSold) != 1)
5        {
6         // scanf returns 1 if number read corrrectly
7         FLUSH;
8         printf("\aInvalid number. Please re-enter: ");
9        } // while
```

PROGRAM 7-12 Handling Errors with Explanations

```
1  /* This function reads the units sold from the keyboard
2     and verifies the data with the user.
3        Pre   nothing
4        Post  units Sold read, verified, and returned
5  */
6  int getUnitsSold (void)
7  {
8  // Local Declarations
9     int  unitsSold;
10    bool valid;
11
12 // Statements
13 do
14    {
15     printf("\nPlease enter Number of Units Sold: ");
16     while (scanf("%d", &unitsSold) != 1)
17           {
```

PROGRAM 7-12 Handling Errors with Explanations

```
18              FLUSH;
19             printf("\aInvalid number. Please re-enter: ");
20             }  // while
21          printf("\nVerify Units Sold: %d: ", unitsSold);
22          printf("<Y> correct: <N> not correct: \n");
23          FLUSH;
24          if (toupper(getchar ()) ==  'Y')
25              valid = true;
26          else
27             {
28              FLUSH;
29              printf("\nYou responded 'no.' ");
30              printf("Please re-enter Units Sold\n");
31              valid = false;
32              } // if
33       } while (!valid);
34   return unitsSold;
35   } // getUnitsSold
```

Data Terminology

- Field (data item)
  - A smallest named unit of data in a file
- Record
  - A collection of related data
- Key
  - One or more fields that uniquely identify the record

State	Capital	No.	Sq. Miles	Population	No. Counties
Arizona	Phoenix	48	113,508	3,447,100	15
California	Sacramento	31	156,299	33,871,648	58
Colorado	Denver	38	103,595	4,301,251	63
Idaho	Boise	43	82,412	1,293,953	44
Montana	Helena	41	145,388	902,195	56
Nevada	Carson City	36	109,894	1,998,257	16
New Mexico	Santa Fe	47	121,335	1,819,046	33
Oregon	Salem	33	96,184	3,421,399	36
Washington	Olympia	42	66,511	5,894,121	39
Wyoming	Cheyenne	44	96,989	493,782	23

Table 7-6 Ten Western States (2000 Census)