A First Book of ANSI C

Fourth Edition

Chapter 10

Data Files

Objectives

- Declaring, Opening, and Closing File Streams
- Reading from and Writing to Text Files
- Random File Access
- Passing and Returning Filenames

Objectives (continued)

- Case Study: Creating and Using a Table of Constants
- Writing and Reading Binary Files (Optional)
- Common Programming and Compiler Errors

Declaring, Opening, and Closing File Streams

- To store and retrieve data outside a C program, you need two items:
  - A file
  - A file stream

Files

- **File**: collection of data that is stored together under a common name, usually on a disk, magnetic tape, or CD-ROM
- Each file has a unique filename, referred to as the file's **external name**
  - For example, `prices.dat` and `info.txt`

Files (continued)

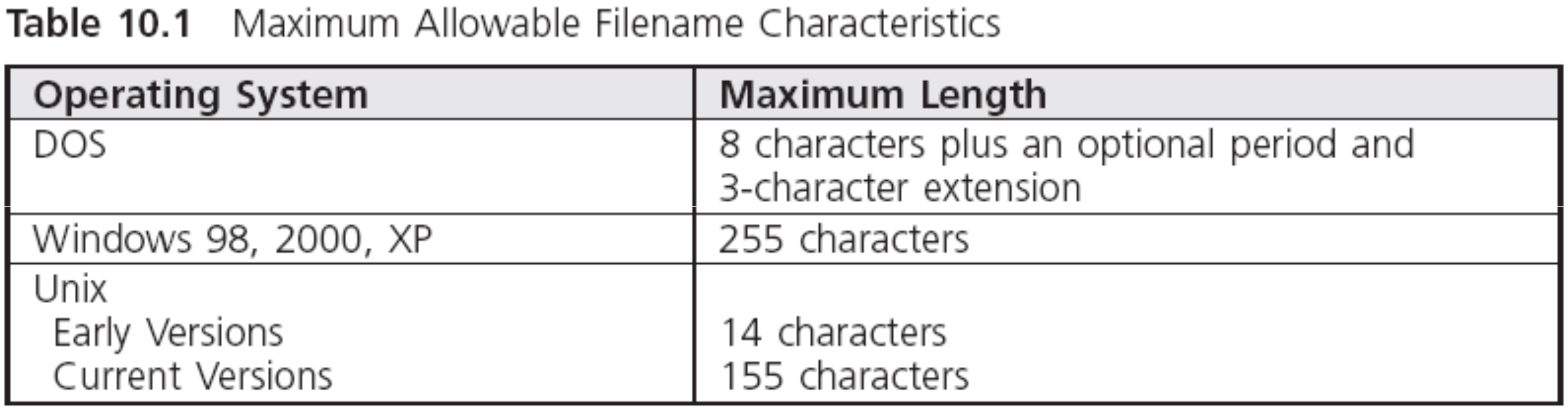


Table 10.1 Maximum Allowable Filename Characteristics

Operating System	Maximum Length
DOS	8 characters plus an optional period and 3-character extension
Windows 98, 2000, XP	255 characters
Unix Early Versions Current Versions	 14 characters 155 characters

Files (continued)

- Most C compilers require a program file to have either the extension `c` or `cpp`
- There are two basic types of files
  - **Text files** (also known as **character-based files**): store each individual character, such as a letter, digit, dollar sign, decimal point, and so on, using an individual character code - store as ASCII code
  - **Binary files**: use the same code as your computer processor uses internally for C's primitive data types
    - Advantage: speed and compactness

\ machine can use it directly

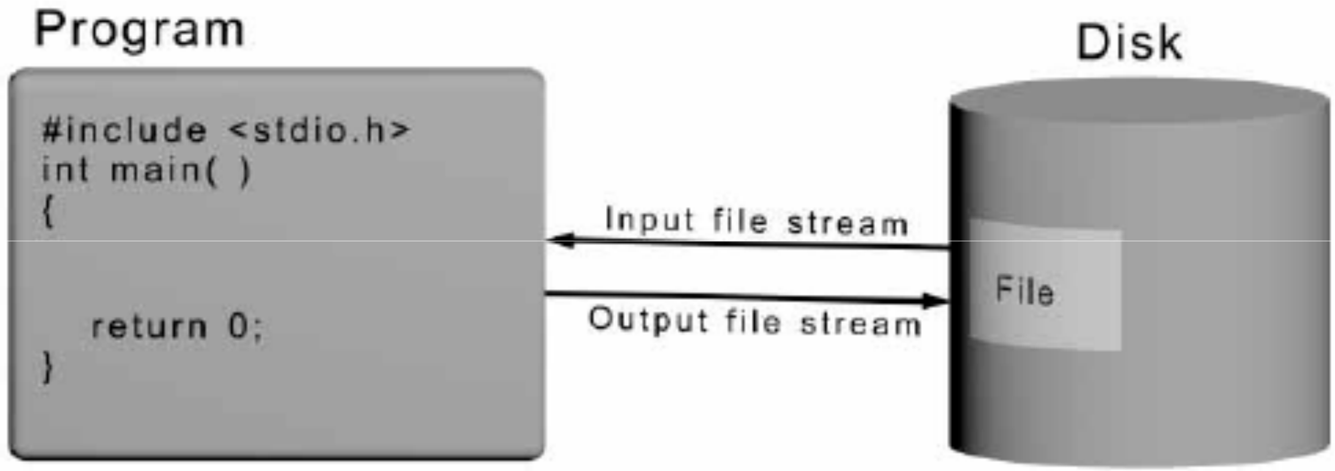
- require 8 byte

File Streams

- **File stream:** one-way transmission path used to connect a file stored on a physical device to a program
- **Input file stream:** receives data from a file into a program - read data from a file
- **Output file stream:** sends data to a file

- writing data

File Streams (continued)



```
#include <stdio.h>
int main( )
{


   return 0;
}
```

Figure 10.1 Input and output file streams

Declaring a File Stream

- For each file that your program uses, a file stream must be named (declared) and created (opened)
- Naming a file stream is accomplished by declaring a variable name to be of type `FILE`
  - `FILE *inFile;`
    - Asterisk is necessary
    - Name is selected by programmer and internal to the program
    - The `FILE` data structure is declared in `stdio.h`

Opening a File Stream

- Opening a file stream (or opening the file):
  - Establishes the physical communication link between the program and the data file
  - Equates a specific external filename to the name declared in the `FILE` declaration statement
- Use `fopen()` (declared in `stdio.h`)

prices.bnd

extension

connect file stream with price.bnd

  - `outFile = fopen("prices.bnd","w");`
  - `fileOut = fopen("prices.dat", "wb");`
  - `inFile = fopen("prices.bnd","r");`

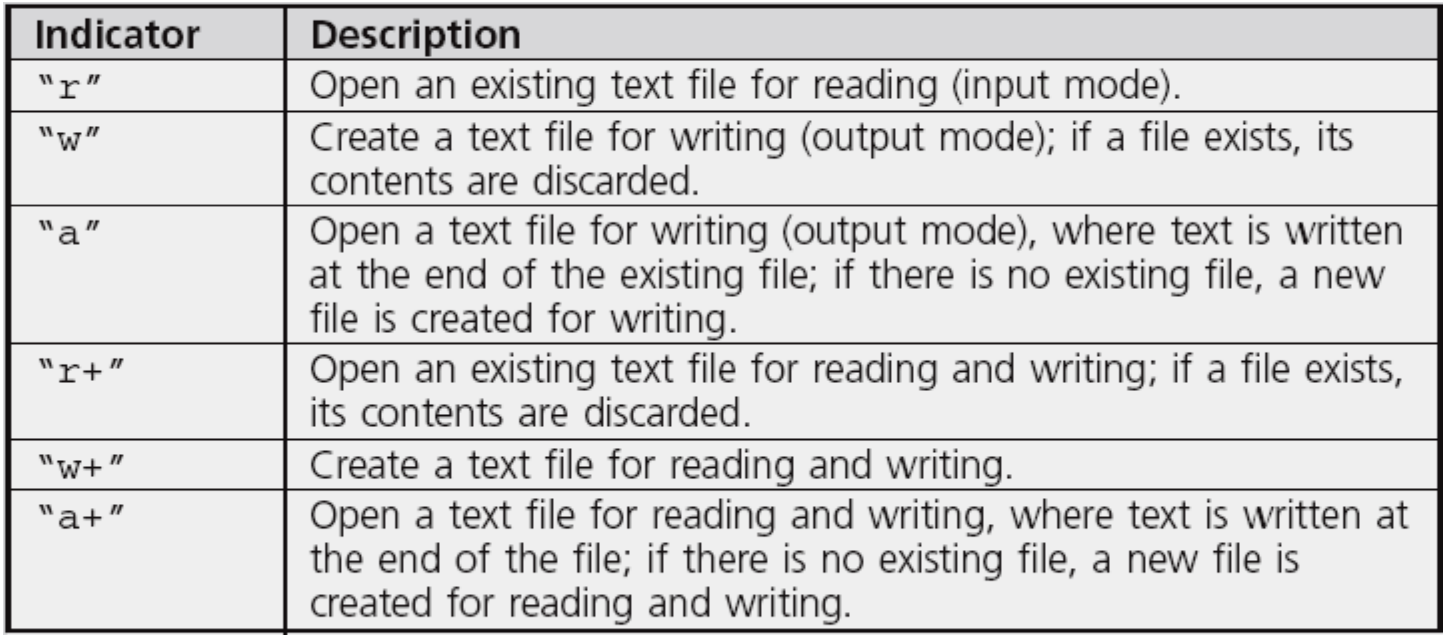
binary

    - If a file opened for reading does not exist, `fopen()` returns the NULL address value

Opening a File Stream (continued)

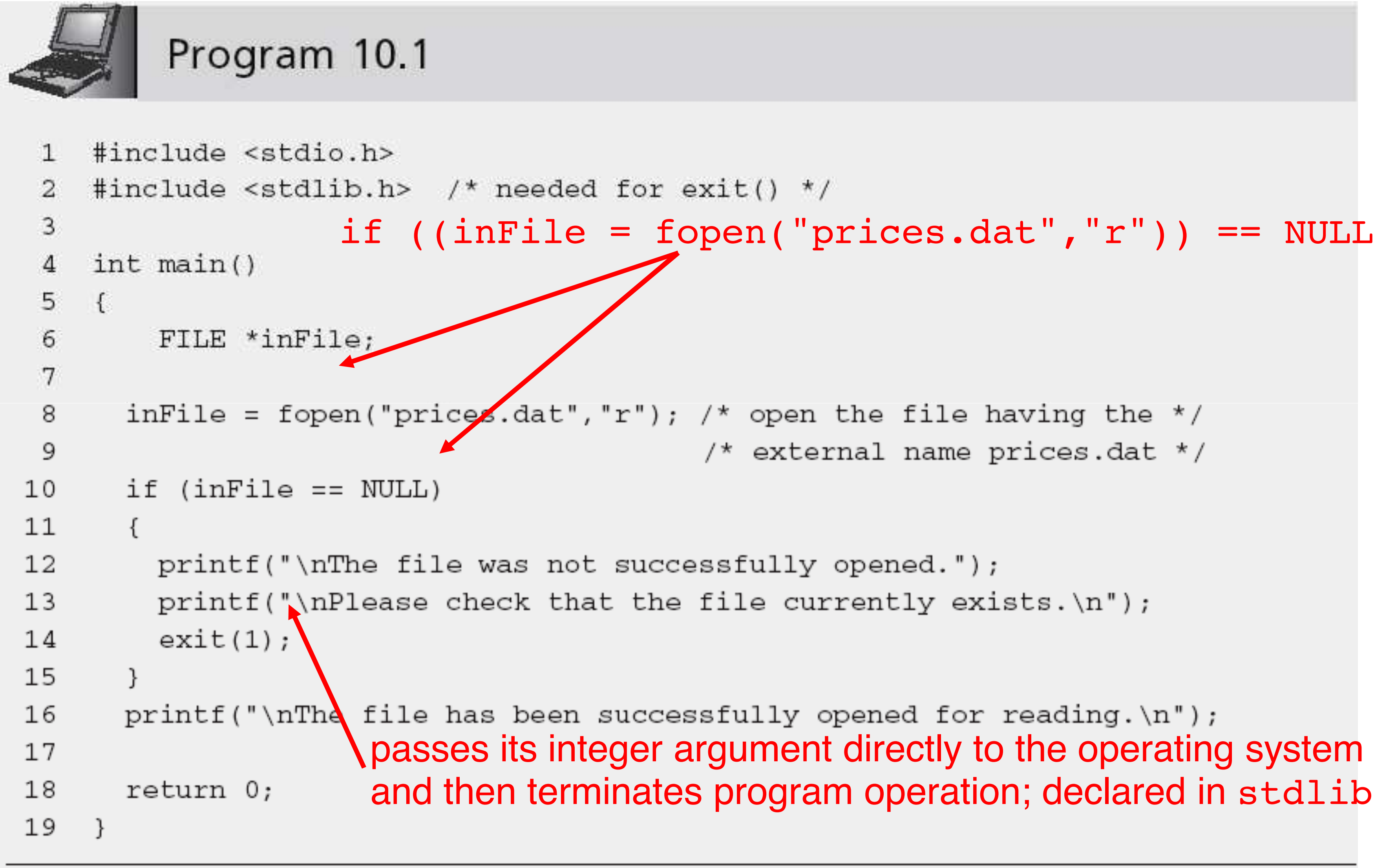
w - if file doesn't exist create new file

- if exist remove everything from that file



Indicator	Description
"r"	Open an existing text file for reading (input mode).
"w"	Create a text file for writing (output mode); if a file exists, its contents are discarded.
"a"	Open a text file for writing (output mode), where text is written at the end of the existing file; if there is no existing file, a new file is created for writing.
"r+"	Open an existing text file for reading and writing; if a file exists, its contents are discarded.
"w+"	Create a text file for reading and writing.
"a+"	Open a text file for reading and writing, where text is written at the end of the file; if there is no existing file, a new file is created for reading and writing.

Opening a File Stream (continued)



Program 10.1

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
    FILE *inFile;

  inFile = fopen("prices.dat","r"); /* open the file having the */
                                    /* external name prices.dat */
  if (inFile == NULL)
  {
    printf("\nThe file was not successfully opened.");
    printf("\nPlease check that the file currently exists.\n");
    exit(1);
  }
  printf("\nThe file has been successfully opened for reading.\n");

  return 0;
}
```

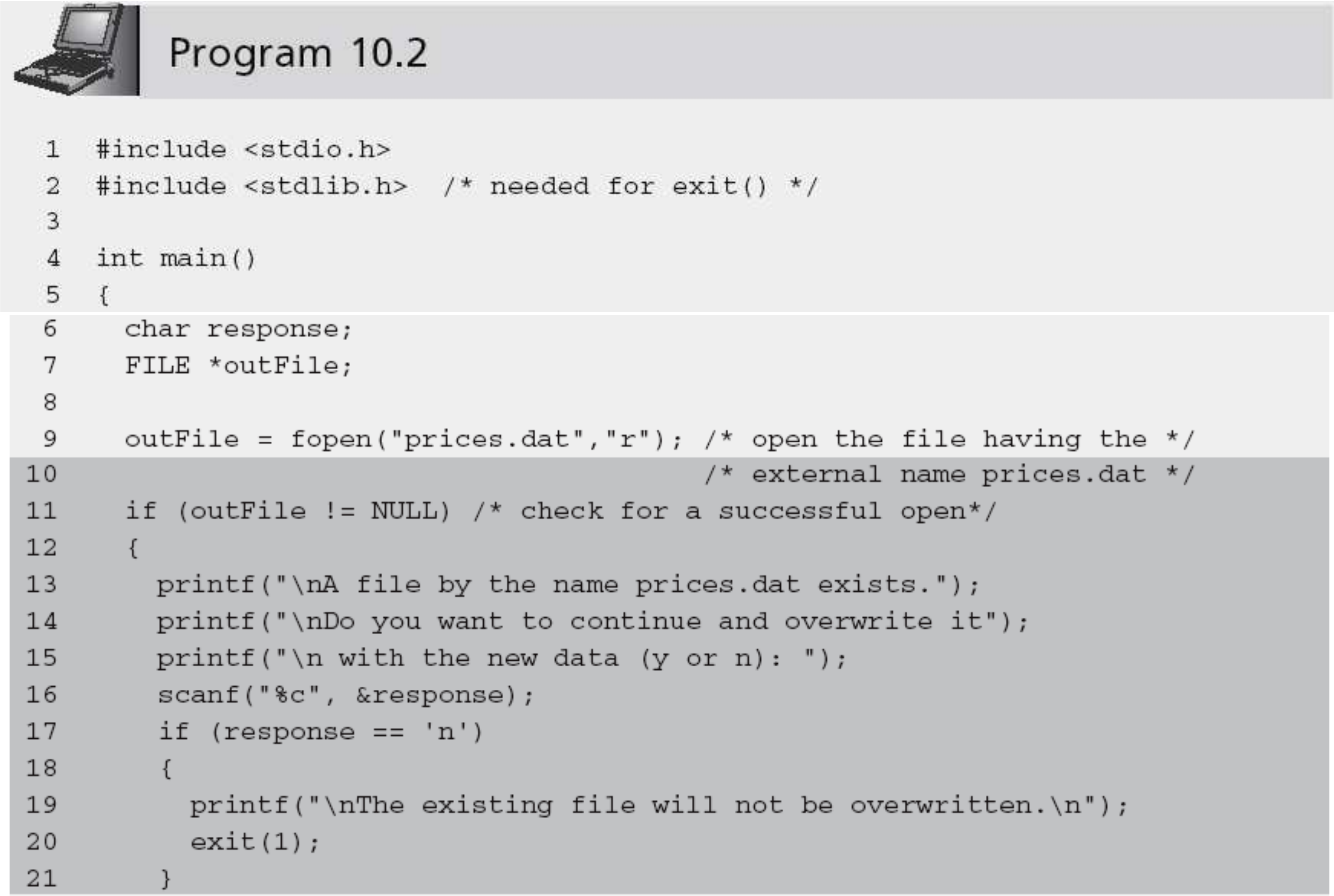
`if ((inFile = fopen("prices.dat","r")) == NULL )`

passes its integer argument directly to the operating system and then terminates program operation; declared in `stdlib.h`

Opening a File Stream (continued)

- Approach in Program 10.1 does not work for output files
  - If a file exists having the same name as the file to be opened for writing, the existing file is erased and all its data is lost
  - The file can first be opened in input mode, simply to see if it exists
    - If it does, the user is given the choice of explicitly permitting it to be overwritten when it is subsequently opened in output mode

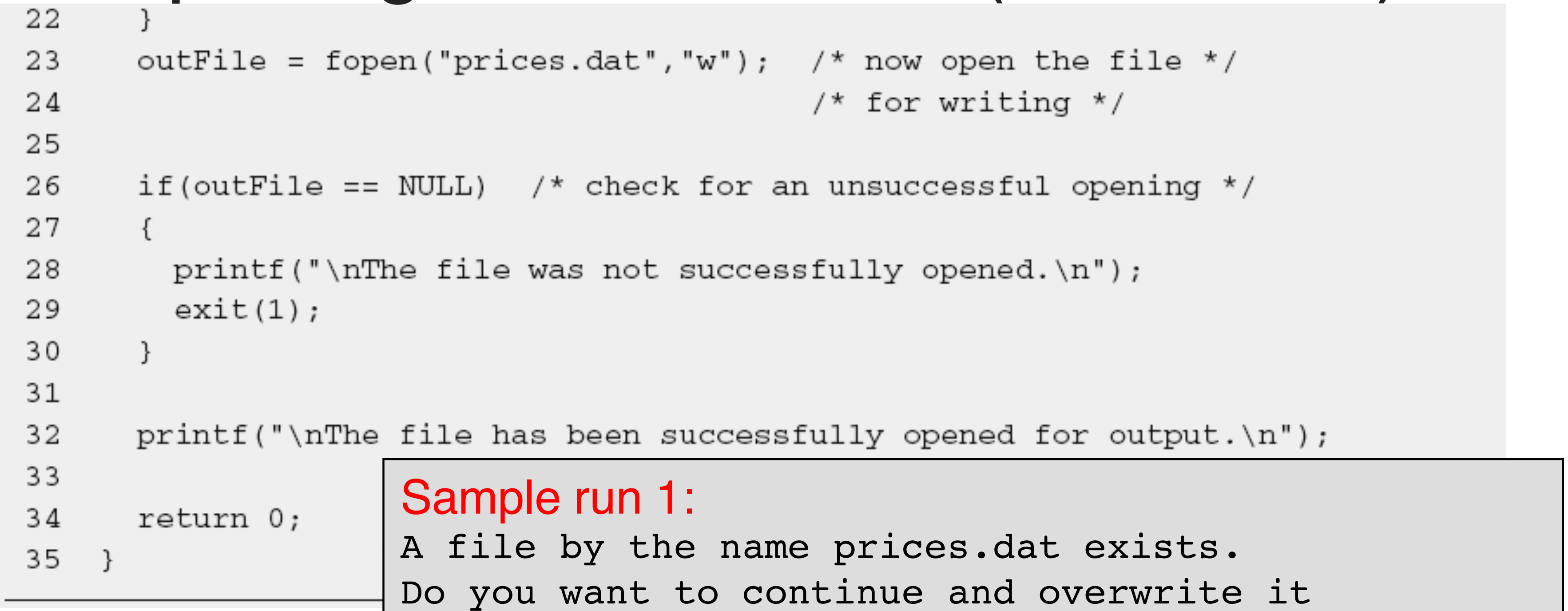
Opening a File Stream (continued)



Program 10.2

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  char response;
  FILE *outFile;

  outFile = fopen("prices.dat","r"); /* open the file having the */
                                     /* external name prices.dat */
  if (outFile != NULL) /* check for a successful open*/
  {
    printf("\nA file by the name prices.dat exists.");
    printf("\nDo you want to continue and overwrite it");
    printf("\n with the new data (y or n): ");
    scanf("%c", &response);
    if (response == 'n')
    {
      printf("\nThe existing file will not be overwritten.\n");
      exit(1);
    }
```

Opening a File Stream (continued)



```
  }
  outFile = fopen("prices.dat","w");    /* now open the file */
                                        /* for writing */

  if(outFile == NULL)  /* check for an unsuccessful opening */
  {
    printf("\nThe file was not successfully opened.\n");
    exit(1);
  }

  printf("\nThe file has been successfully opened for output.\n");

  return 0;
}
```

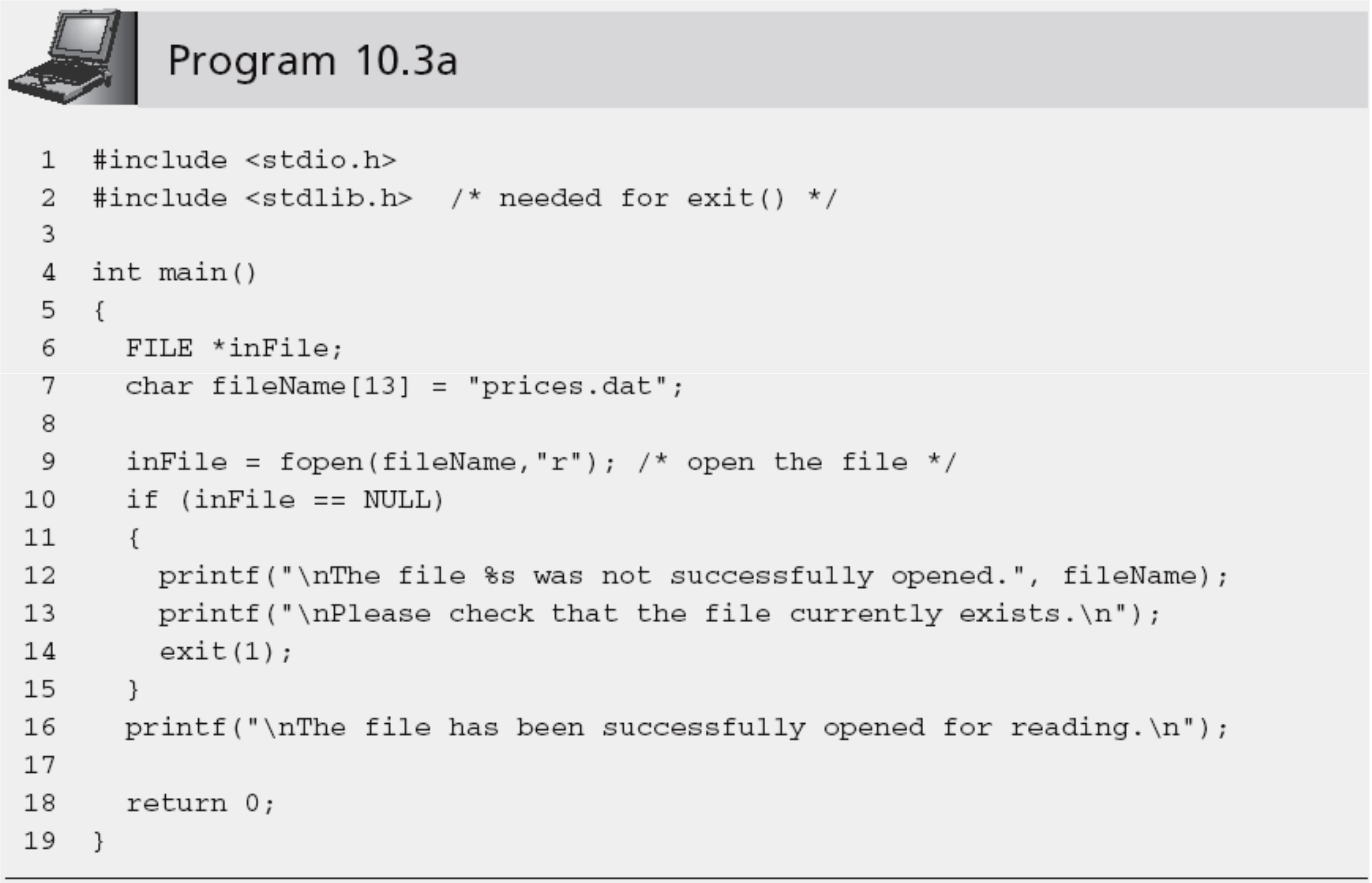
Sample run 1:

```
A file by the name prices.dat exists.
Do you want to continue and overwrite it
with the new data (y or n): n

The existing file will not be overwritten.
```

Sample run 2:

```
A file by the name prices.dat exists.
Do you want to continue and overwrite it
with the new data (y or n): y

The file has been successfully opened for output.
```

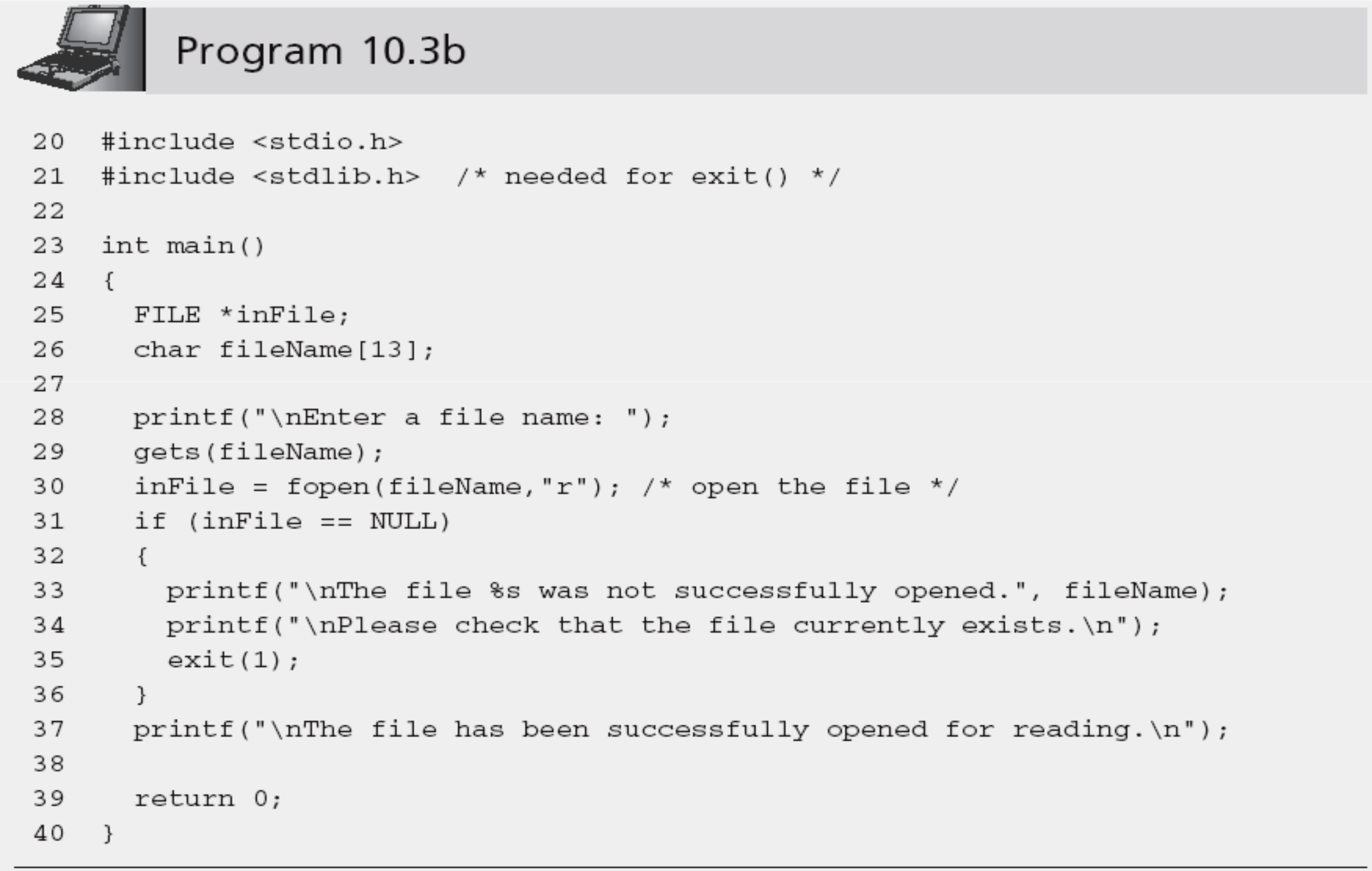
Embedded and Interactive Filenames



Program 10.3a

```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  FILE *inFile;
  char fileName[13] = "prices.dat";

  inFile = fopen(fileName,"r"); /* open the file */
  if (inFile == NULL)
  {
    printf("\nThe file %s was not successfully opened.", fileName);
    printf("\nPlease check that the file currently exists.\n");
    exit(1);
  }
  printf("\nThe file has been successfully opened for reading.\n");

  return 0;
}
```

Embedded and Interactive Filenames (continued)



Program 10.3b

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  FILE *inFile;
  char fileName[13];

  printf("\nEnter a file name: ");
  gets(fileName);
  inFile = fopen(fileName,"r"); /* open the file */
  if (inFile == NULL)
  {
    printf("\nThe file %s was not successfully opened.", fileName);
    printf("\nPlease check that the file currently exists.\n");
    exit(1);
  }
  printf("\nThe file has been successfully opened for reading.\n");

  return 0;
}
```

Closing a File Stream

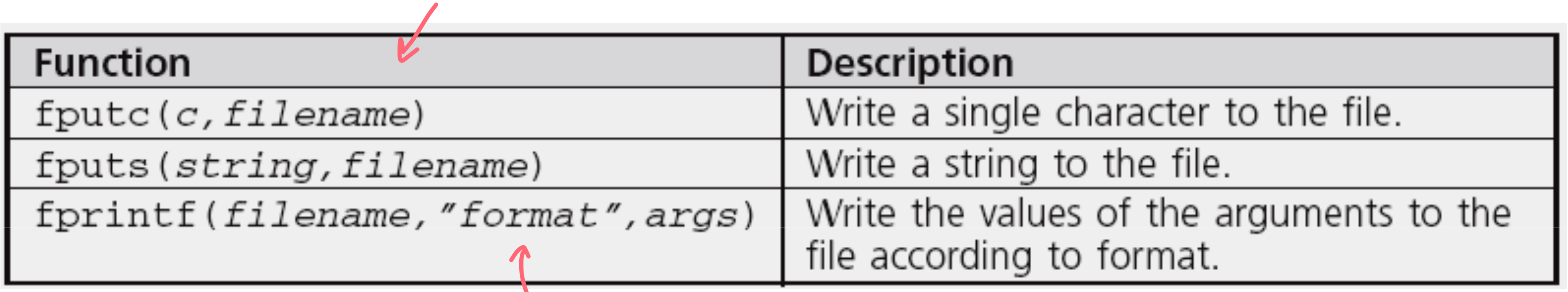
- A file stream is closed using `fclose()`
  - `fclose()` breaks the link between the file's external and internal names, releasing the internal file pointer name, which can then be used for another file
  - `fclose(inFile);`
- Because all computers have a limit on the maximum number of files that can be open at one time, closing files that are no longer needed makes good sense
- Open files existing at the end of normal program execution are closed by the operating system

Reading from and Writing to Text Files

- Prototypes in `stdio.h`
- Examples
  - `fputc('a',outFile);`
  - `fputs("Hello world!",outFile);`
  - `fprintf(outFile,"%s %n",descrip,price);`

Reading from and Writing to Text Files (continued)

send char to

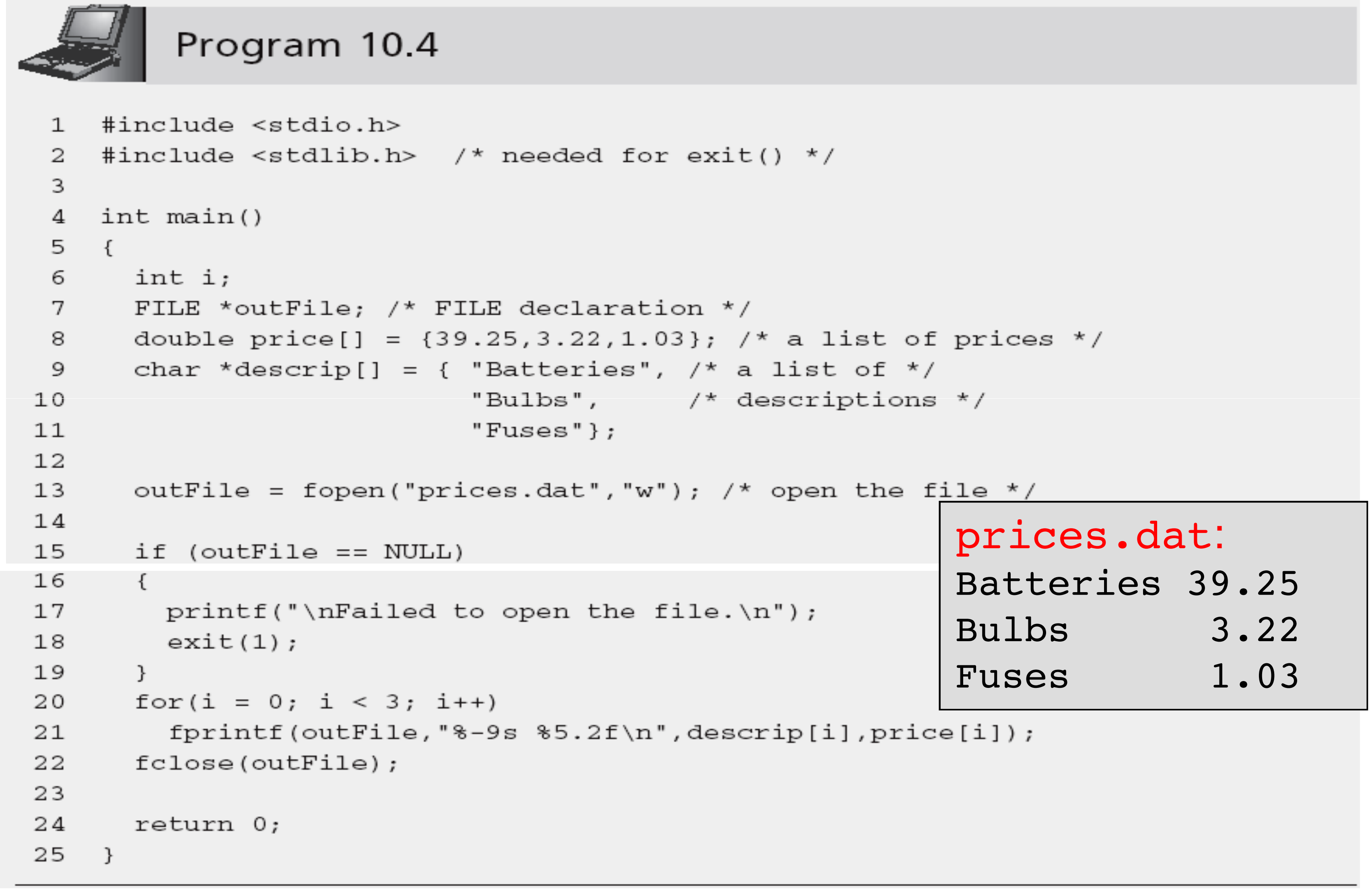


Function	Description
fputc(*c*, *filename*)	Write a single character to the file.
fputs(*string*, *filename*)	Write a string to the file.
fprintf(*filename*, *"format"*, *args*)	Write the values of the arguments to the file according to format.

control string

(%s)

Reading from and Writing to Text Files (continued)



Program 10.4

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  int i;
  FILE *outFile; /* FILE declaration */
  double price[] = {39.25,3.22,1.03}; /* a list of prices */
  char *descrip[] = { "Batteries", /* a list of */
                      "Bulbs",     /* descriptions */
                      "Fuses"};

  outFile = fopen("prices.dat","w"); /* open the file */

  if (outFile == NULL)
  {
    printf("\nFailed to open the file.\n");
    exit(1);
  }
  for(i = 0; i < 3; i++)
    fprintf(outFile,"%-9s %5.2f\n",descrip[i],price[i]);
  fclose(outFile);

  return 0;
}
```

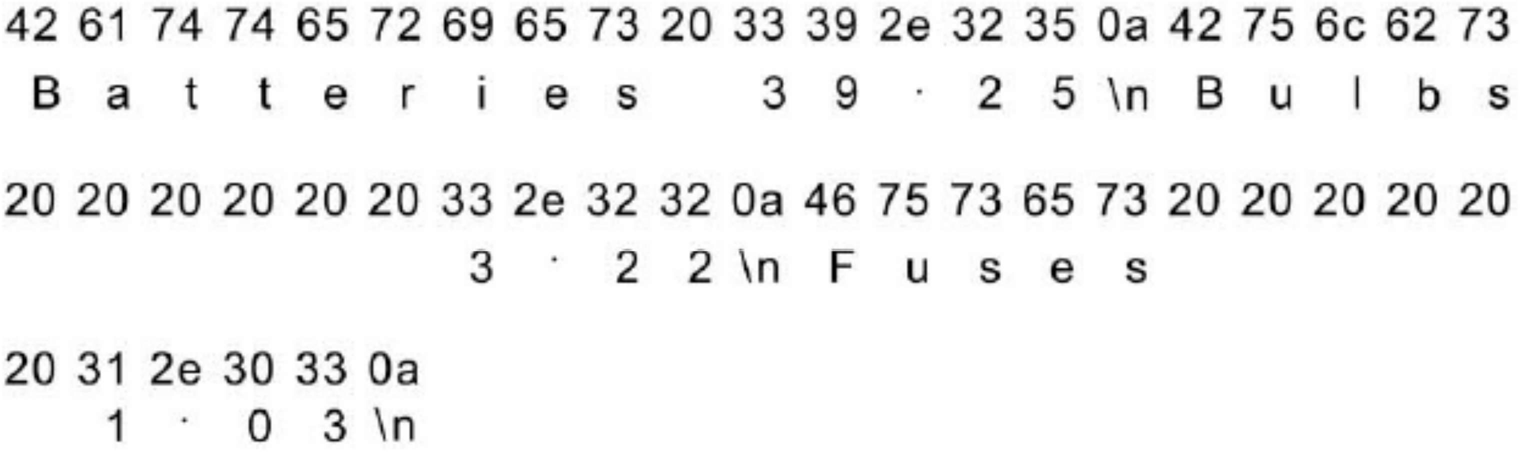
prices.dat:

```
Batteries 39.25
Bulbs      3.22
Fuses      1.03
```

Reading from and Writing to Text Files (continued)

- C appends the low-value hexadecimal byte 0x00 as the end-of-file (EOF) sentinel when the file is closed
- EOF sentinel is never counted as part of the file

Reading from and Writing to Text Files (continued)



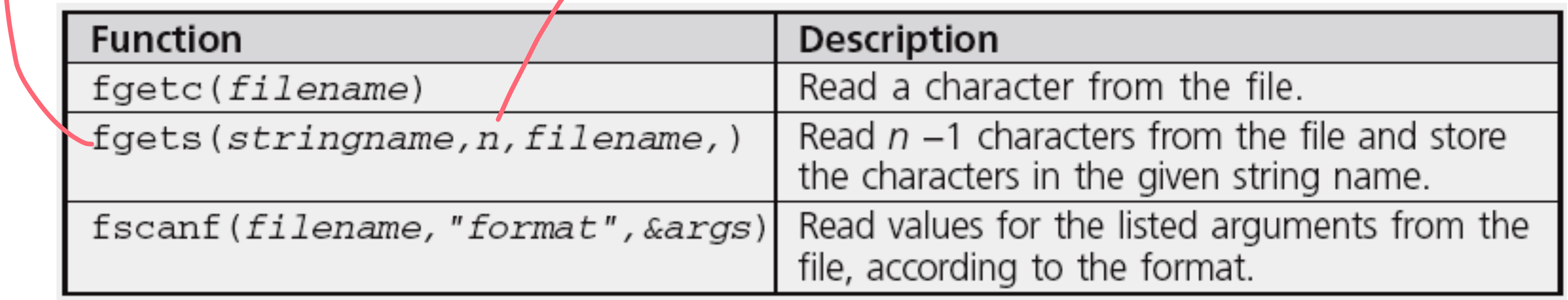
```
42 61 74 74 65 72 69 65 73 20 33 39 2e 32 35 0a 42 75 6c 62 73
 B  a  t  t  e  r  i  e  s     3  9  ·  2  5 \n  B  u  l  b  s

20 20 20 20 20 20 33 2e 32 32 0a 46 75 73 65 73 20 20 20 20 20
                   3  ·  2  2 \n  F  u  s  e  s

20 31 2e 30 33 0a
    1  ·  0  3 \n
```

Figure 10.2 The `prices.dat` file as stored by a typical computer

Read data from file store in first arg

Reading from a Text File

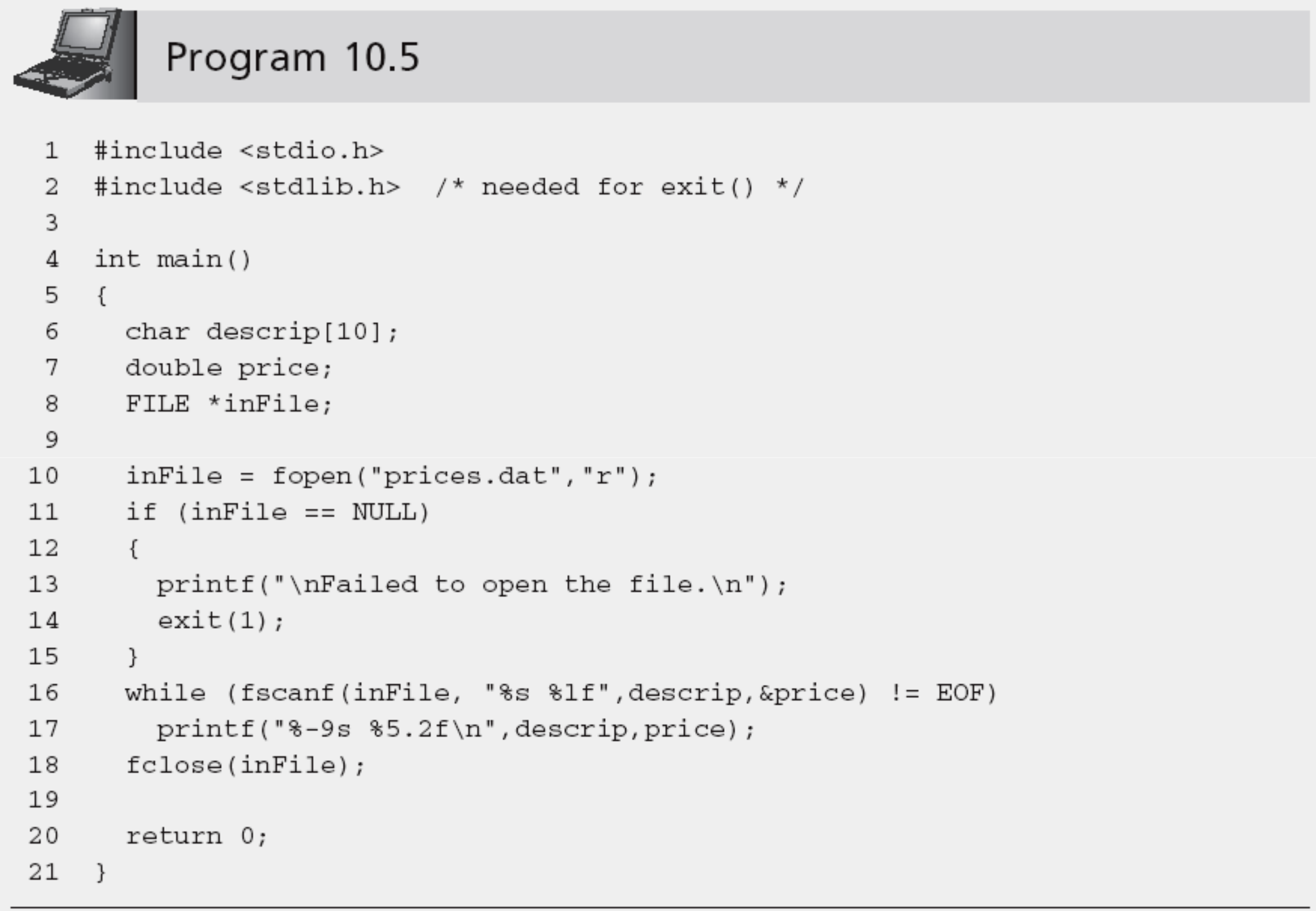
length u want to store



Function	Description
`fgetc(filename)`	Read a character from the file.
`fgets(stringname,n,filename,)`	Read *n* –1 characters from the file and store the characters in the given string name.
`fscanf(filename,"format",&args)`	Read values for the listed arguments from the file, according to the format.

- Prototypes in `stdio.h`
- Examples
  - `fgetc(inFile);`
  - `fgets(message,10,inFile);` up to 10 char
  - `fscanf(inFile,"%lf",&price);` read double store in price
- `fgetc()` and `fscanf()` return EOF when the end-of-file marker is detected
- `fgets()` returns a NULL instead

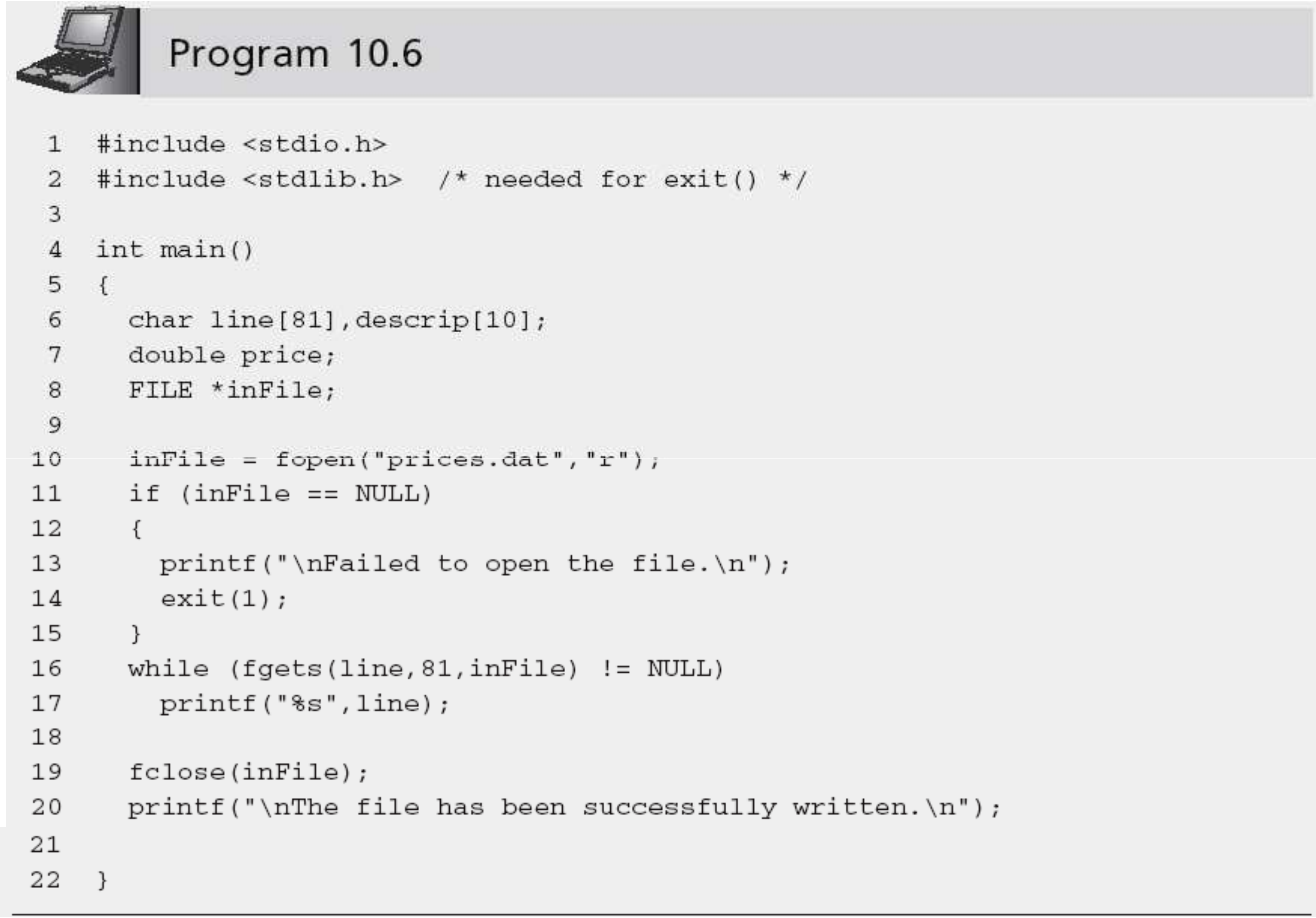
Reading from a Text File (continued)



Program 10.5

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  char descrip[10];
  double price;
  FILE *inFile;

  inFile = fopen("prices.dat","r");
  if (inFile == NULL)
  {
    printf("\nFailed to open the file.\n");
    exit(1);
  }
  while (fscanf(inFile, "%s %lf",descrip,&price) != EOF)
    printf("%-9s %5.2f\n",descrip,price);
  fclose(inFile);

  return 0;
}
```

Reading from a Text File (continued)



Program 10.6

```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  char line[81],descrip[10];
  double price;
  FILE *inFile;

  inFile = fopen("prices.dat","r");
  if (inFile == NULL)
  {
    printf("\nFailed to open the file.\n");
    exit(1);
  }
  while (fgets(line,81,inFile) != NULL)
    printf("%s",line);

  fclose(inFile);
  printf("\nThe file has been successfully written.\n");

}
```

Standard Device Files

- When a program is run, the keyboard used for entering data is automatically opened and assigned to the internal file pointer name `stdin`
  - `fscanf(stdin,"%d",&num);`

    from std io - connected to keyboard
- The output device used for display is assigned to the file pointer named `stdout`
  - `fprintf(stdout,"Hello World!");`

    monitor
- `stderr` is assigned to the output device used for system error messages
  - `stderr` and `stdout` often refer to the same device

    - connect to monitor

    error

    file stream to keep error message (connect to monitor)

Standard Device Files (continued)

- The character function pairs listed in Table 10.2 can be used as direct replacements for each other
- This is not true for the string-handling functions

Standard Device Files (continued)

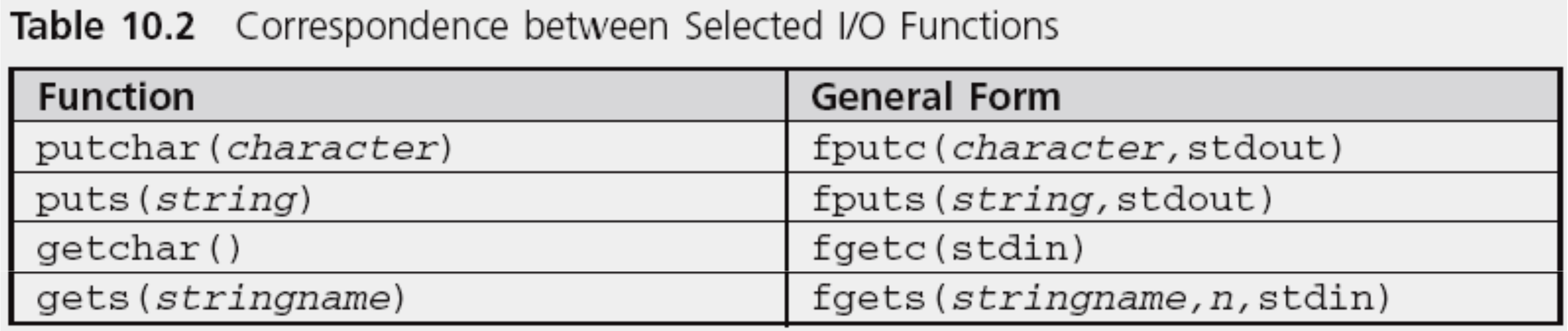


Table 10.2 Correspondence between Selected I/O Functions

Function	General Form
putchar(*character*)	fputc(*character*,stdout)
puts(*string*)	fputs(*string*,stdout)
getchar()	fgetc(stdin)
gets(*stringname*)	fgets(*stringname*,*n*,stdin)

Other Devices

- Most IBM or IBM-compatible personal computers assign the name `prn` to the printer connected to the computer
  - `fprintf("prn","Hello World!");`
- `prn` is not a pointer constant but the actual name of the device; as such, it must be enclosed in double quotes when used in a statement

Random File Access

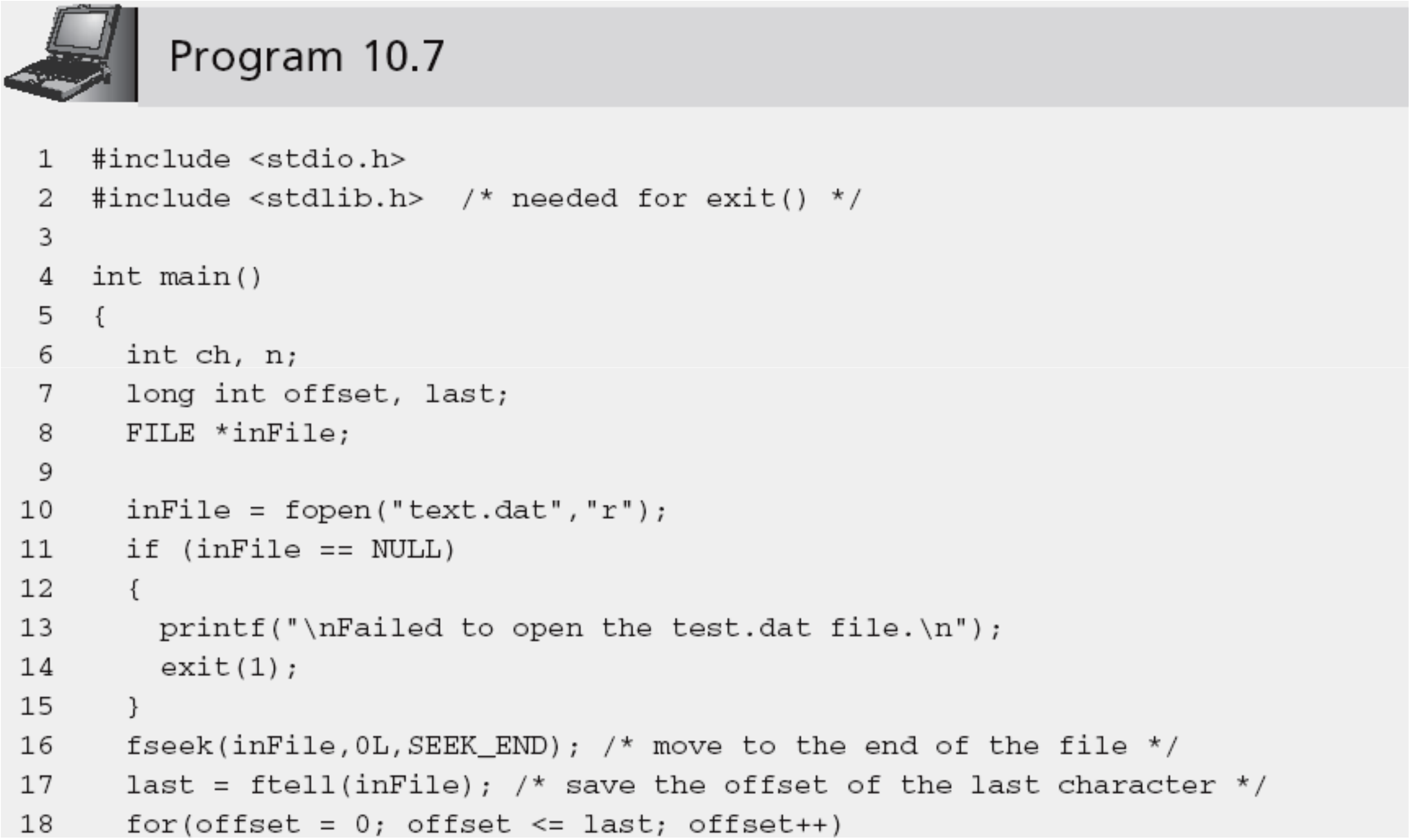
- `rewind()` resets the current position to the start of the file
  - `rewind(inFile)`
- `fseek()` allows the programmer to move to any position in the file
  - `fseek(fileName, offset, origin)`
  - Origin: `SEEK_SET`, `SEEK_CUR`, and `SEEK_END`
- `ftell()` returns the offset value of the next character that will be read or written
  - `ftell(inFile);`

Random File Access (continued)

- Examples of `fseek()` are
  - `fseek(inFile,4L,SEEK_SET);` go to 5th char
  - `fseek(inFile,4L,SEEK_CUR);` move ahead 4 char
  - `fseek(inFile,-4L,SEEK_CUR);` Move back 4 char
  - `fseek(inFile,0L,SEEK_SET);` goto begining of file
  - `fseek(inFile,0L,SEEK_END);` go to end of file
  - `fseek(inFile,-10L,SEEK_END);`

move 10 start from end

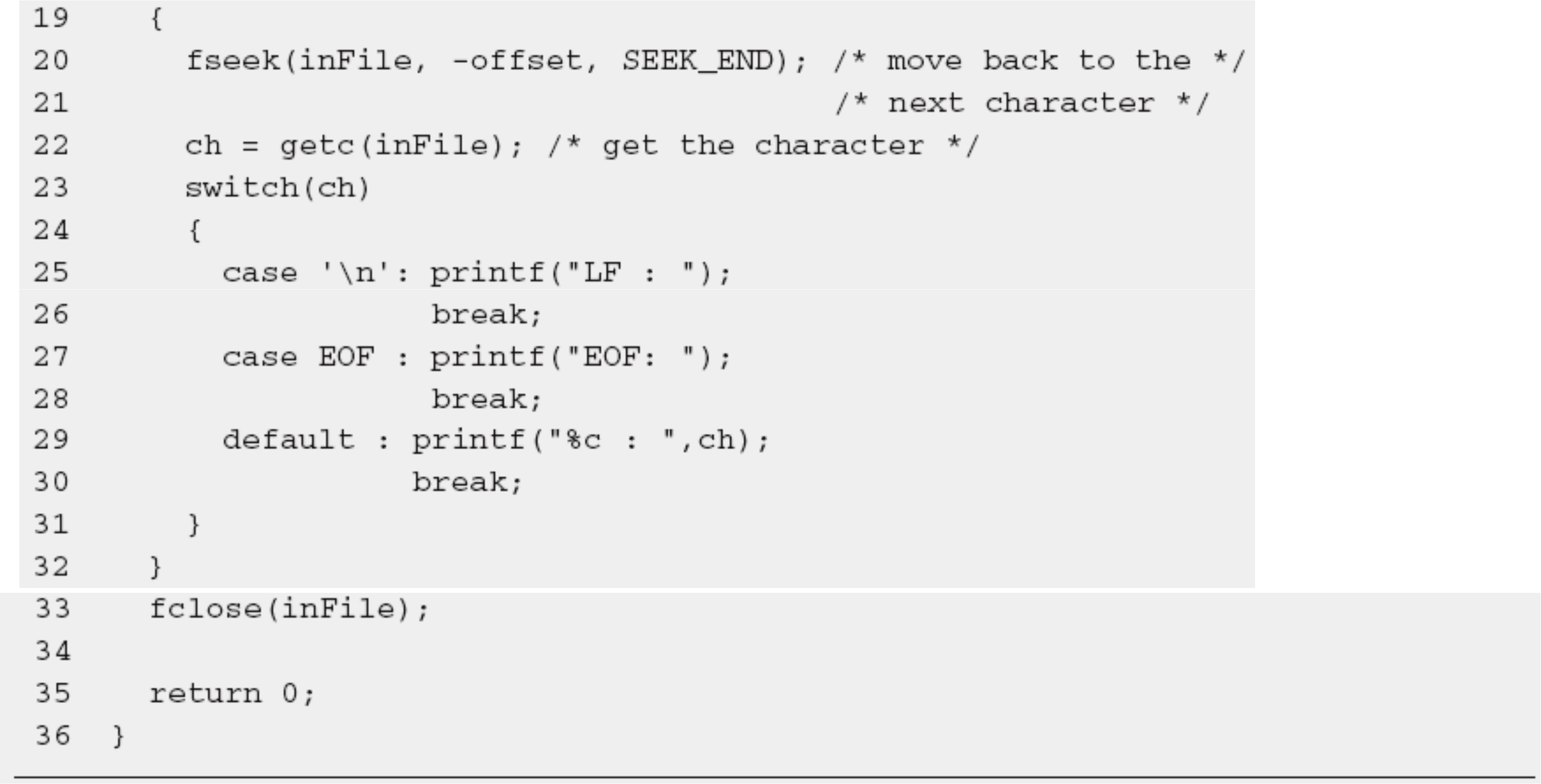
Random File Access (continued)



Program 10.7

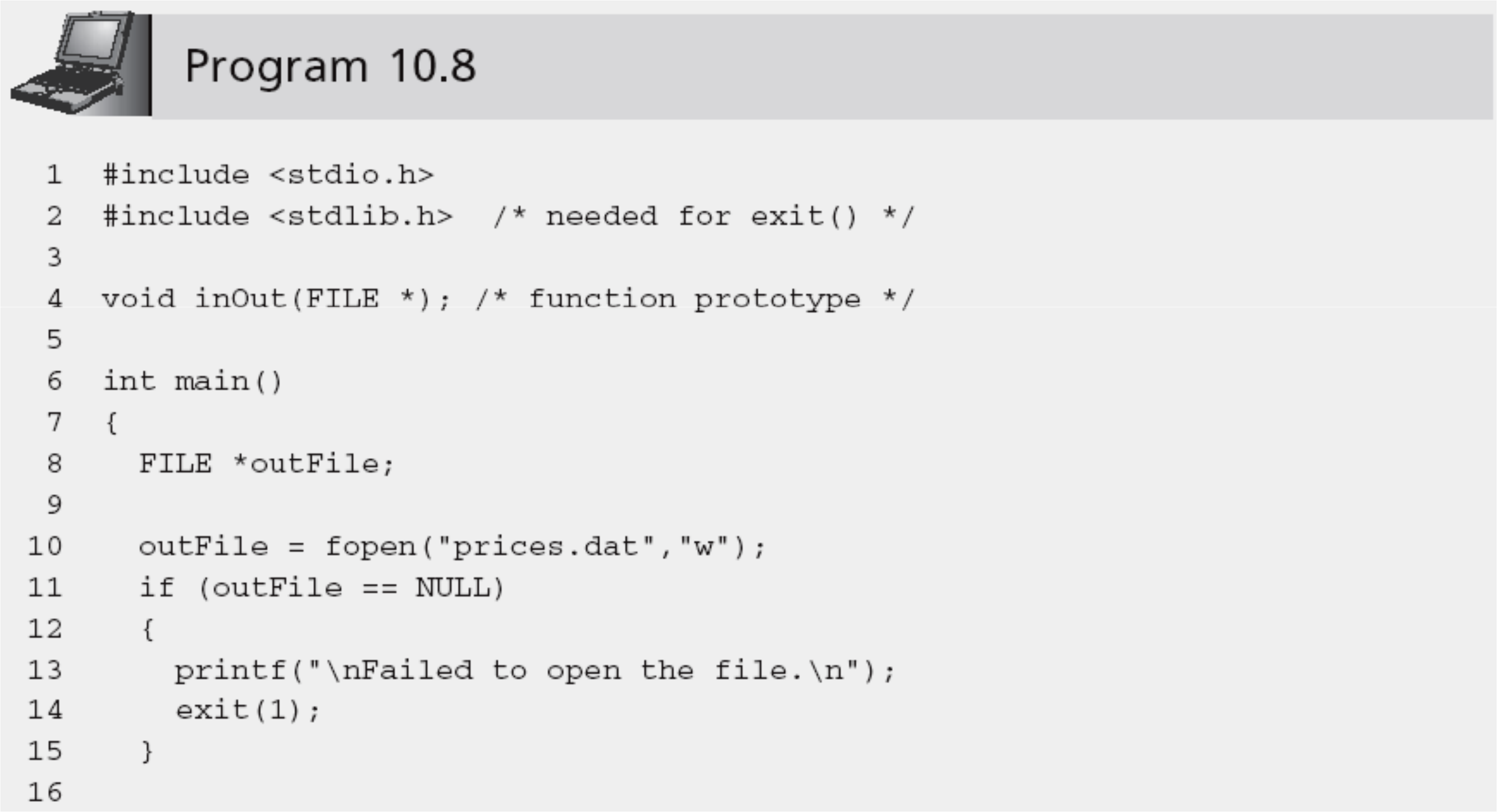
```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  int ch, n;
  long int offset, last;
  FILE *inFile;

  inFile = fopen("text.dat","r");
  if (inFile == NULL)
  {
    printf("\nFailed to open the test.dat file.\n");
    exit(1);
  }
  fseek(inFile,0L,SEEK_END); /* move to the end of the file */
  last = ftell(inFile); /* save the offset of the last character */
  for(offset = 0; offset <= last; offset++)
```

Random File Access (continued)



```
  {
    fseek(inFile, -offset, SEEK_END); /* move back to the */
                                      /* next character */
    ch = getc(inFile); /* get the character */
    switch(ch)
    {
      case '\n': printf("LF : ");
                 break;
      case EOF : printf("EOF: ");
                 break;
      default : printf("%c : ",ch);
                break;
    }
  }
  fclose(inFile);

  return 0;
}
```

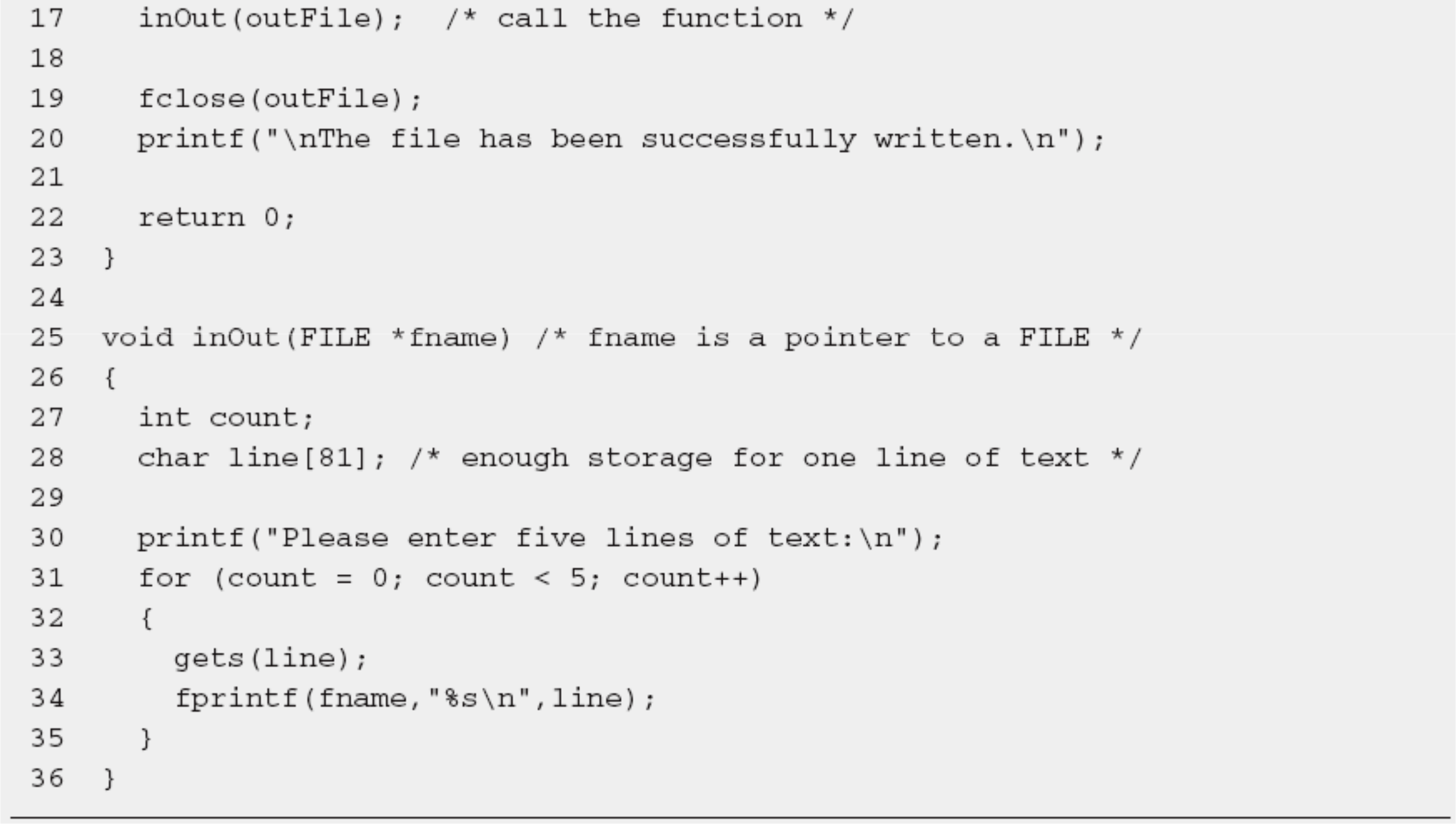
Passing and Returning Filenames



Program 10.8

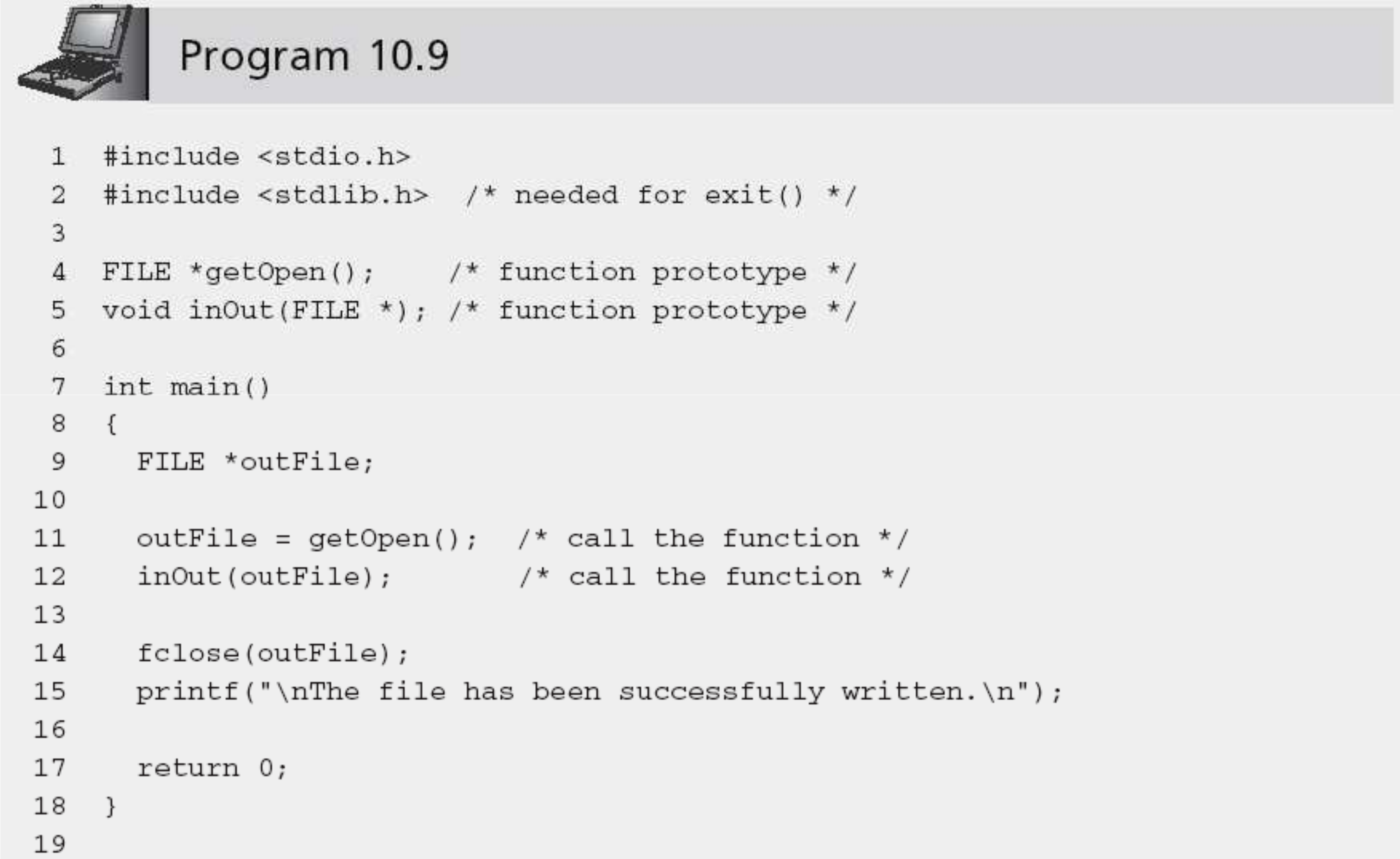
```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

void inOut(FILE *); /* function prototype */

int main()
{
  FILE *outFile;

  outFile = fopen("prices.dat","w");
  if (outFile == NULL)
  {
    printf("\nFailed to open the file.\n");
    exit(1);
  }

```

Passing and Returning Filenames (continued)



```c
  inOut(outFile);  /* call the function */

  fclose(outFile);
  printf("\nThe file has been successfully written.\n");

  return 0;
}

void inOut(FILE *fname) /* fname is a pointer to a FILE */
{
  int count;
  char line[81]; /* enough storage for one line of text */

  printf("Please enter five lines of text:\n");
  for (count = 0; count < 5; count++)
  {
    gets(line);
    fprintf(fname,"%s\n",line);
  }
}
```

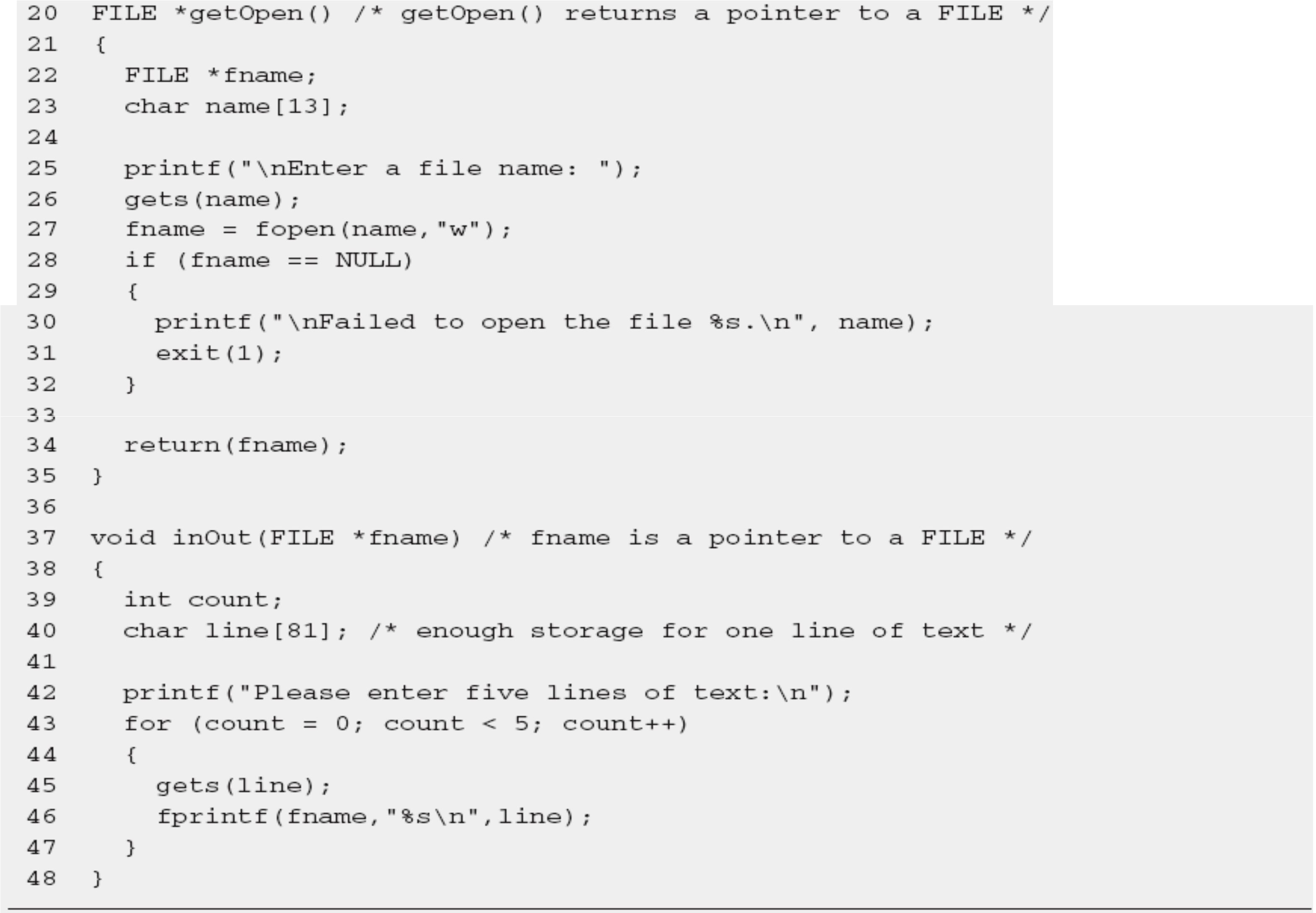
Passing and Returning Filenames (continued)



Program 10.9

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

FILE *getOpen();    /* function prototype */
void inOut(FILE *); /* function prototype */

int main()
{
  FILE *outFile;

  outFile = getOpen();  /* call the function */
  inOut(outFile);       /* call the function */

  fclose(outFile);
  printf("\nThe file has been successfully written.\n");

  return 0;
}

```

Passing and Returning Filenames (continued)



```c
FILE *getOpen() /* getOpen() returns a pointer to a FILE */
{
  FILE *fname;
  char name[13];

  printf("\nEnter a file name: ");
  gets(name);
  fname = fopen(name,"w");
  if (fname == NULL)
  {
    printf("\nFailed to open the file %s.\n", name);
    exit(1);
  }

  return(fname);
}

void inOut(FILE *fname) /* fname is a pointer to a FILE */
{
  int count;
  char line[81]; /* enough storage for one line of text */

  printf("Please enter five lines of text:\n");
  for (count = 0; count < 5; count++)
  {
    gets(line);
    fprintf(fname,"%s\n",line);
  }
}
```

Case Study: Creating and Using a Table of Constants

- A common real-world programming requirement is creating and maintaining a small file of constants, reading and storing these constants into a list, and then providing functions for checking data against the constants in the list
- In financial and scheduling programs, this requirement takes the form of reading a set of holiday dates and then checking a date against each date in the table

Requirements Specification

- Objective: create a set of functions that determines if a given date is a holiday, using concepts that are equally applicable to any program that needs to check data against a list of constants, such as temperatures, densities, or other parameters
- Two functions are developed
  - The first constructs a list of holidays, which is called a **holiday table**, and consists of legal holiday dates that have been previously stored in a file
  - The second compares any given date to the dates in the table and determines if there is a match

Analysis for the First Function

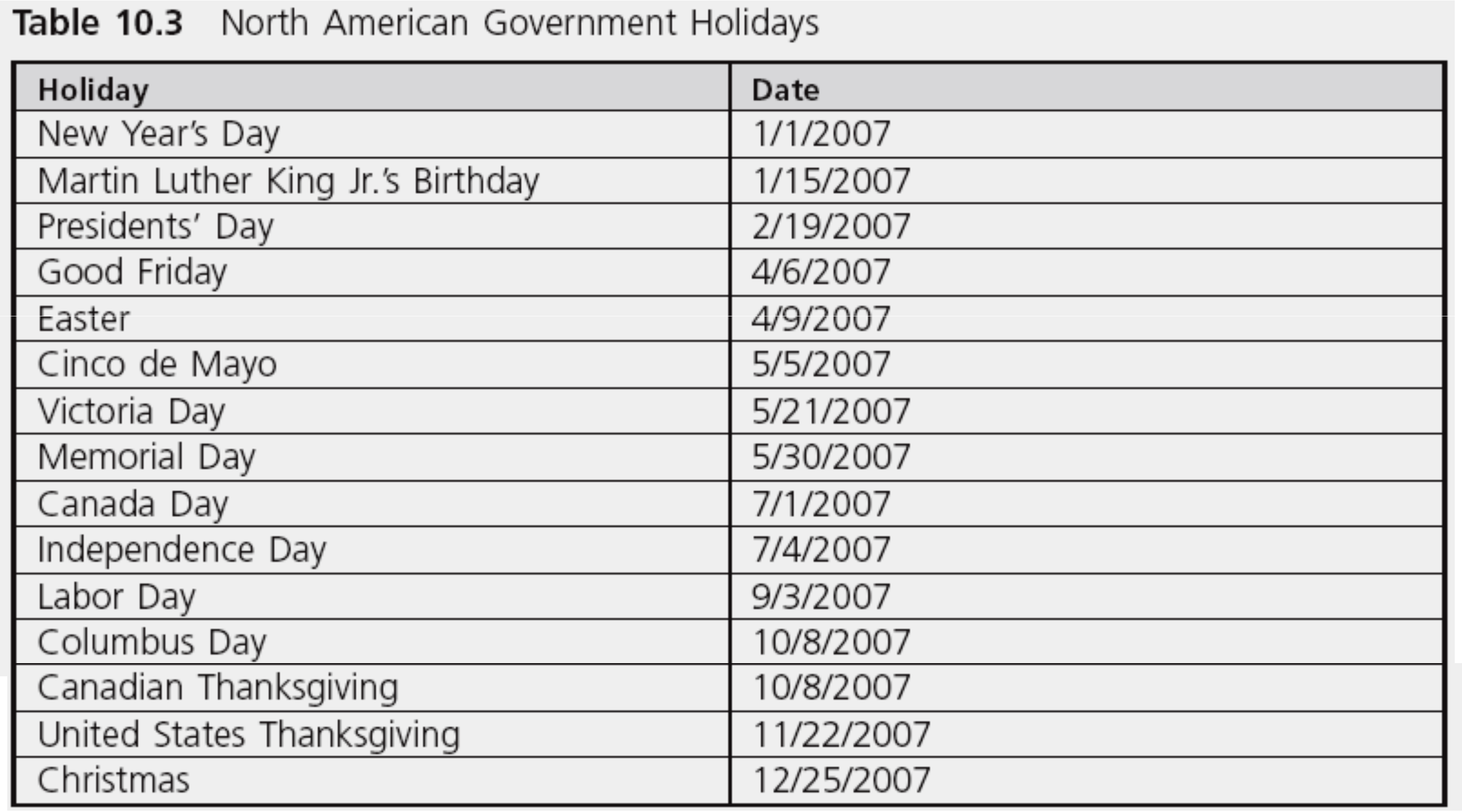
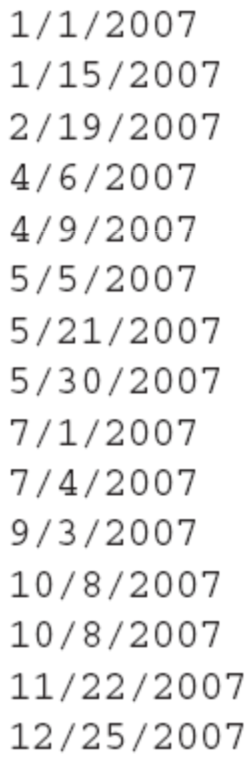


Table 10.3 North American Government Holidays

Holiday	Date
New Year's Day	1/1/2007
Martin Luther King Jr.'s Birthday	1/15/2007
Presidents' Day	2/19/2007
Good Friday	4/6/2007
Easter	4/9/2007
Cinco de Mayo	5/5/2007
Victoria Day	5/21/2007
Memorial Day	5/30/2007
Canada Day	7/1/2007
Independence Day	7/4/2007
Labor Day	9/3/2007
Columbus Day	10/8/2007
Canadian Thanksgiving	10/8/2007
United States Thanksgiving	11/22/2007
Christmas	12/25/2007

Analysis for the First Function (continued)

Table 10.4 Holiday Dates Stored in the Holidays.txt File

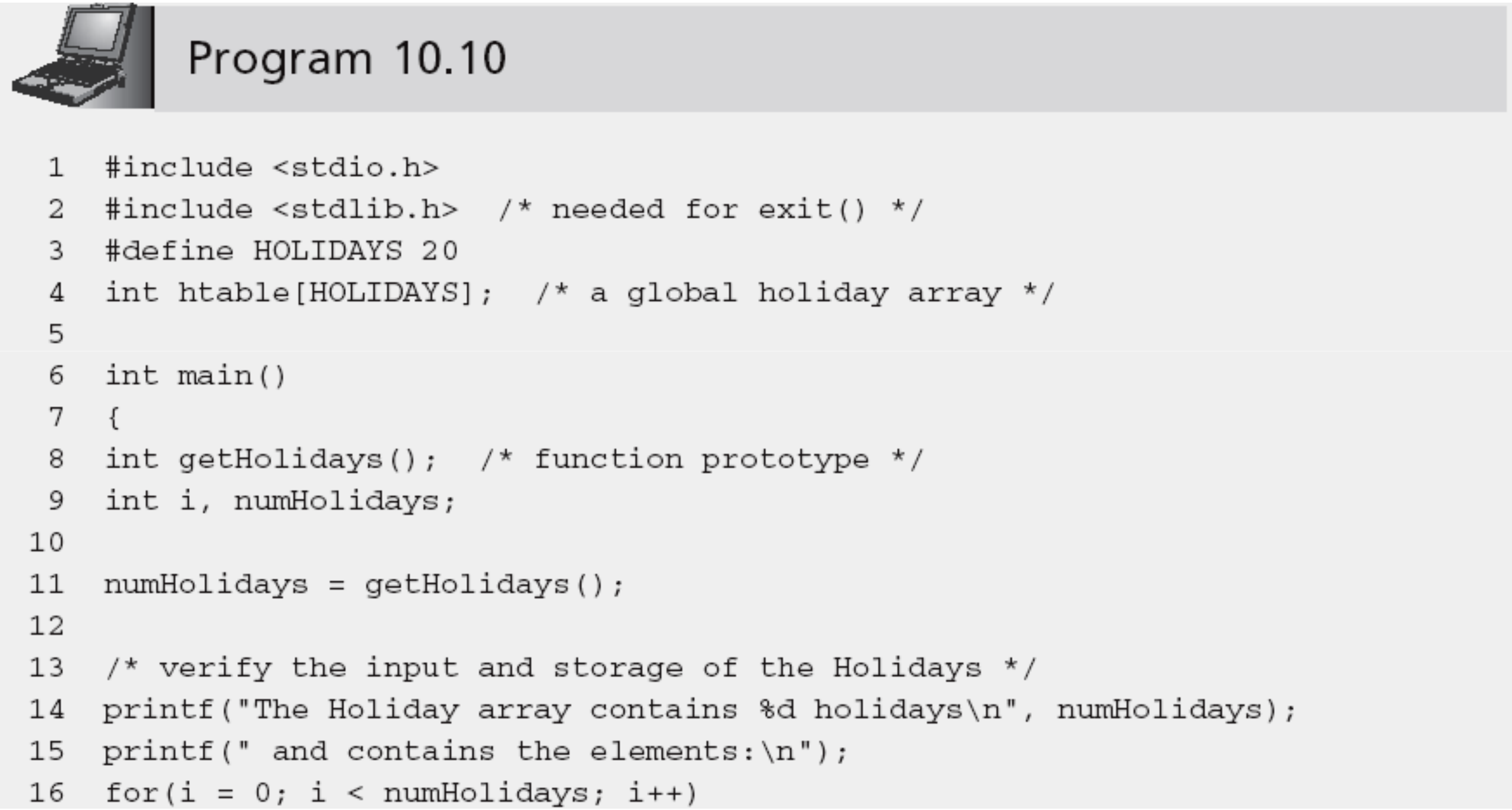


```
1/1/2007
1/15/2007
2/19/2007
4/6/2007
4/9/2007
5/5/2007
5/21/2007
5/30/2007
7/1/2007
7/4/2007
9/3/2007
10/8/2007
10/8/2007
11/22/2007
12/25/2007
```

Code the Function

```
Create an array capable of storing 20 integers
Set a counter to 0
Open the Holidays.txt file, checking that a successful
  open occurred
While there are dates in the file
   Read a date as a month, day, and year
   Convert date to an integer having the form yyyymmdd
   Assign the integer date to the Holiday array
   Add 1 to the counter
EndWhile
Close the Holidays.txt file
Return the value of the counter
```

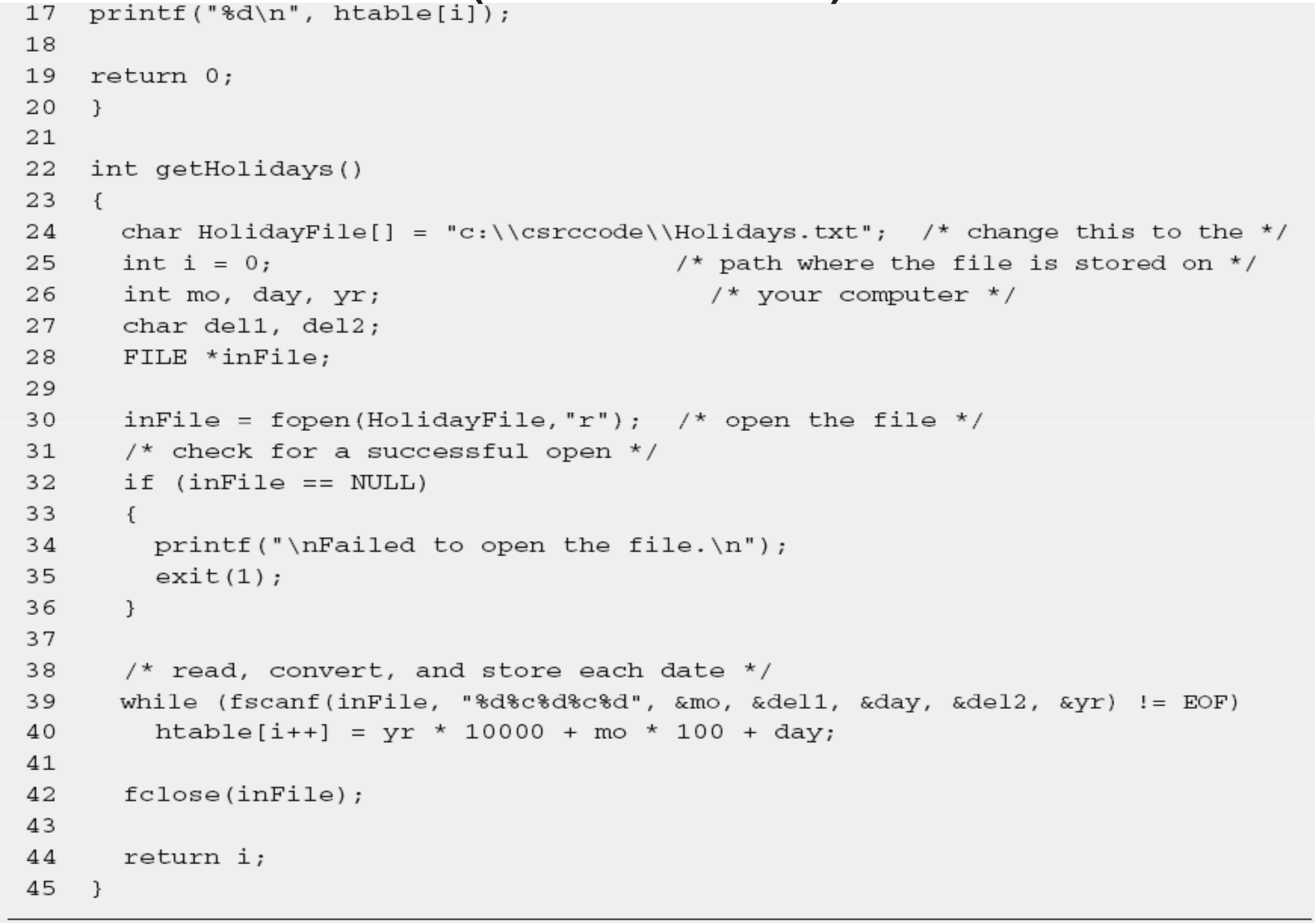
Test and Debug the Function



Program 10.10

```
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */
#define HOLIDAYS 20
int htable[HOLIDAYS];  /* a global holiday array */

int main()
{
int getHolidays();  /* function prototype */
int i, numHolidays;

numHolidays = getHolidays();

/* verify the input and storage of the Holidays */
printf("The Holiday array contains %d holidays\n", numHolidays);
printf(" and contains the elements:\n");
for(i = 0; i < numHolidays; i++)
```

Test and Debug the Function (continued)



```c
printf("%d\n", htable[i]);

return 0;
}

int getHolidays()
{
  char HolidayFile[] = "c:\\csrccode\\Holidays.txt";  /* change this to the */
  int i = 0;                        /* path where the file is stored on */
  int mo, day, yr;                    /* your computer */
  char del1, del2;
  FILE *inFile;

  inFile = fopen(HolidayFile,"r");  /* open the file */
  /* check for a successful open */
  if (inFile == NULL)
  {
    printf("\nFailed to open the file.\n");
    exit(1);
  }

  /* read, convert, and store each date */
  while (fscanf(inFile, "%d%c%d%c%d", &mo, &del1, &day, &del2, &yr) != EOF)
    htable[i++] = yr * 10000 + mo * 100 + day;

  fclose(inFile);

  return i;
}
```

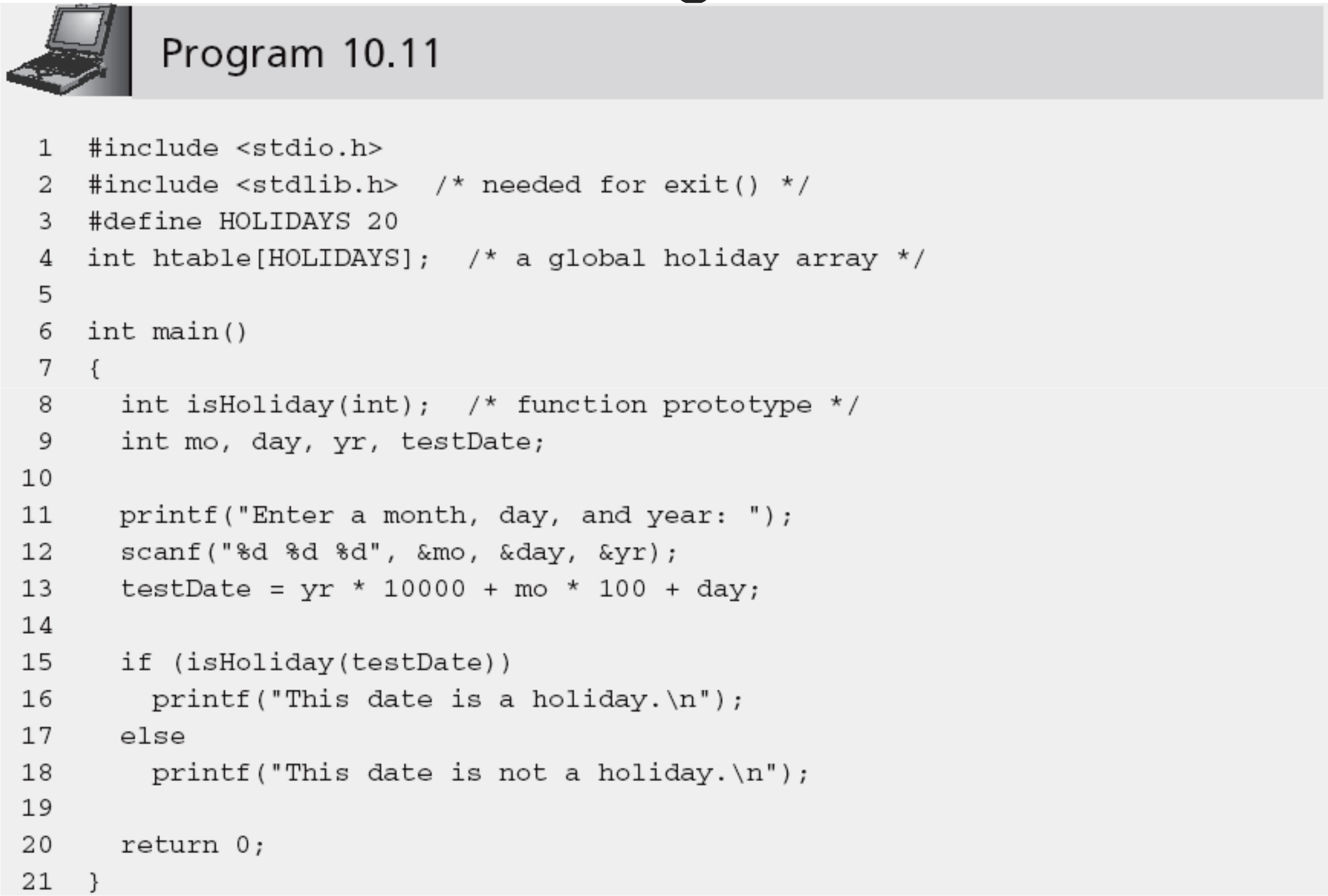
Analysis for the Second Function

```
If the holiday table is empty
   Call getHolidays()
EndIf
For all Holidays in the table
   Retrieve the holiday from the table
   Compare the date being tested to the date
      retrieved from the array
   If there is a match
      Return 1
EndFor
Return 0
```

Code the Function

```
int isHoliday(int testDate)
{
  int getHolidays(); /* function prototype */
  #define TRUE 1
  #define FALSE 0
  int i;

  /* read the Holiday file if the Holiday array is empty */
  if (htable[0] == 0)
    getHolidays();

  /* search the Holiday array for the given date */
  for(i = 0; i < HOLIDAYS; i++)
    if (testDate == htable[i])
      return TRUE;

  return FALSE;
}
```

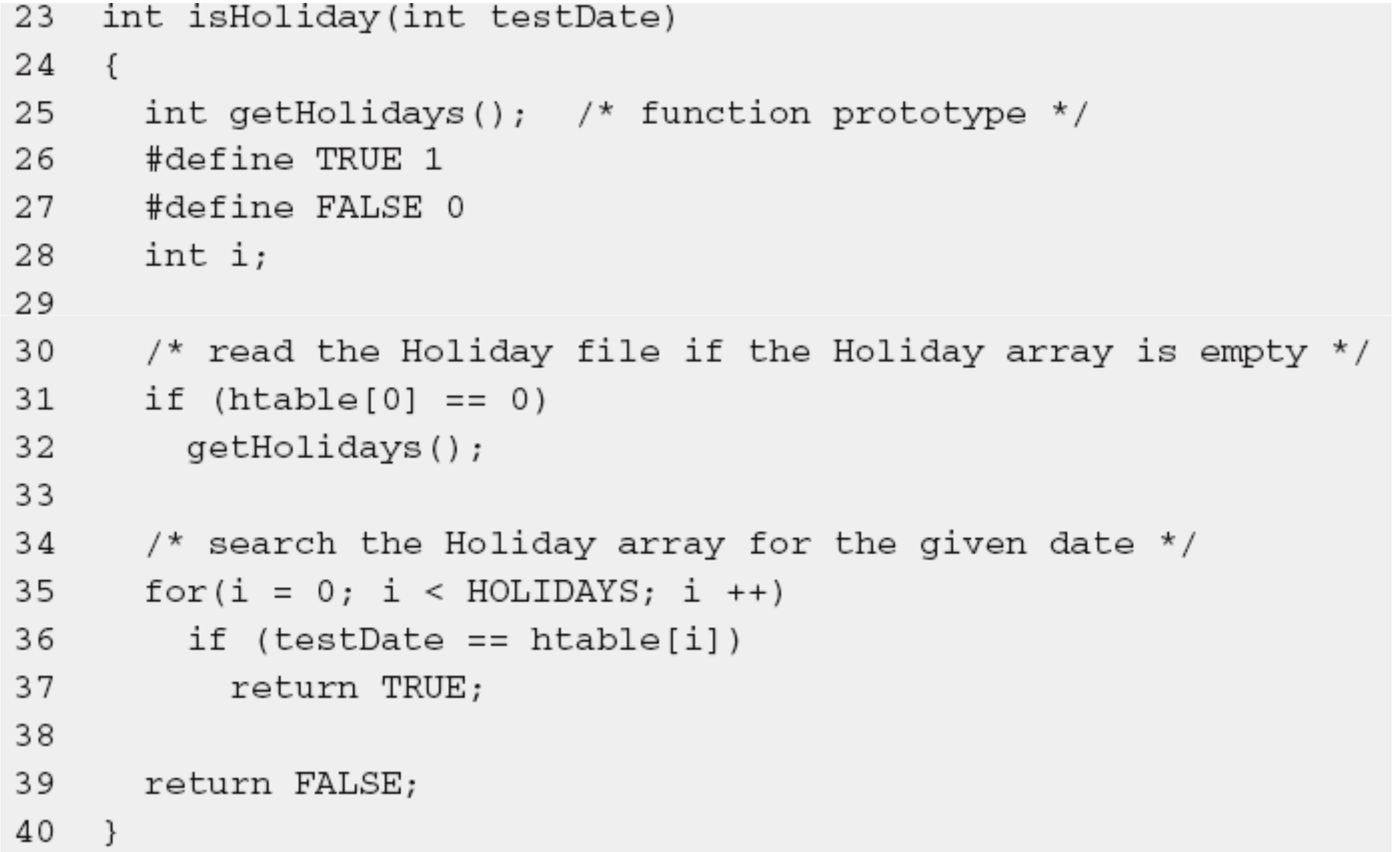
Test and Debug the Function



Program 10.11

```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */
#define HOLIDAYS 20
int htable[HOLIDAYS];  /* a global holiday array */

int main()
{
  int isHoliday(int);  /* function prototype */
  int mo, day, yr, testDate;

  printf("Enter a month, day, and year: ");
  scanf("%d %d %d", &mo, &day, &yr);
  testDate = yr * 10000 + mo * 100 + day;

  if (isHoliday(testDate))
    printf("This date is a holiday.\n");
  else
    printf("This date is not a holiday.\n");

  return 0;
}
```

Test and Debug the Function (continued)



```c
int isHoliday(int testDate)
{
  int getHolidays();  /* function prototype */
  #define TRUE 1
  #define FALSE 0
  int i;

  /* read the Holiday file if the Holiday array is empty */
  if (htable[0] == 0)
    getHolidays();

  /* search the Holiday array for the given date */
  for(i = 0; i < HOLIDAYS; i ++)
    if (testDate == htable[i])
      return TRUE;

  return FALSE;
}
```

Writing and Reading Binary Files

- **Binary files** store numerical values using the computer's internal numerical code
- No number-to-character conversion when writing a number to a file, and no character-to-number conversion when a value is read from the file
  - Resulting file frequently requires less storage space than its character-based counterpart

Writing and Reading Binary Files (continued)

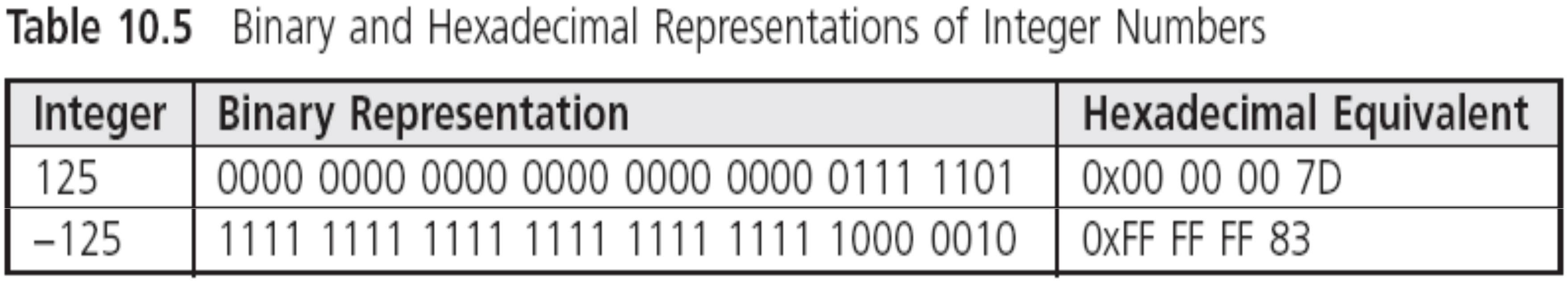
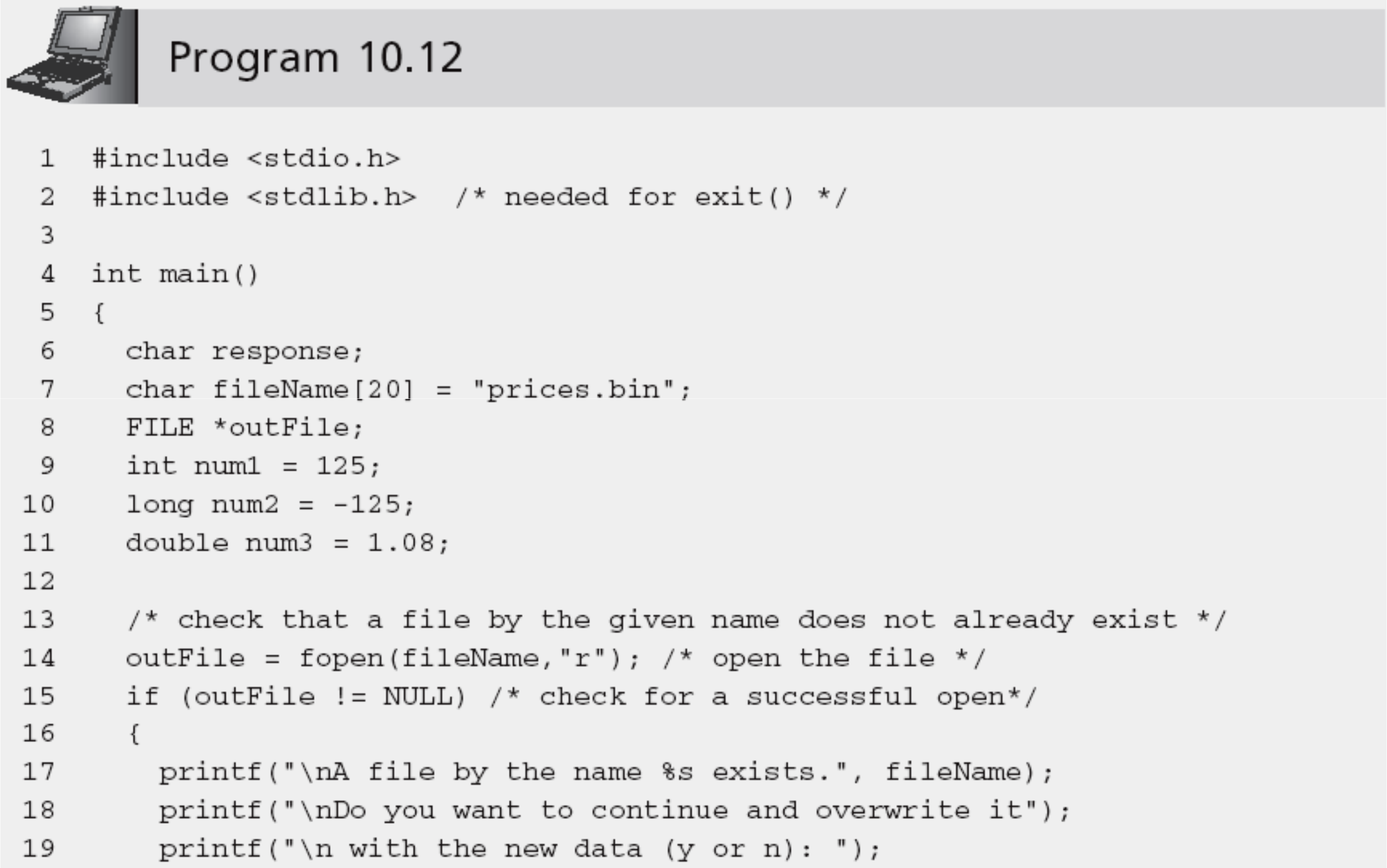


Table 10.5 Binary and Hexadecimal Representations of Integer Numbers

Integer	Binary Representation	Hexadecimal Equivalent
125	0000 0000 0000 0000 0000 0000 0111 1101	0x00 00 00 7D
–125	1111 1111 1111 1111 1111 1111 1000 0010	0xFF FF FF 83

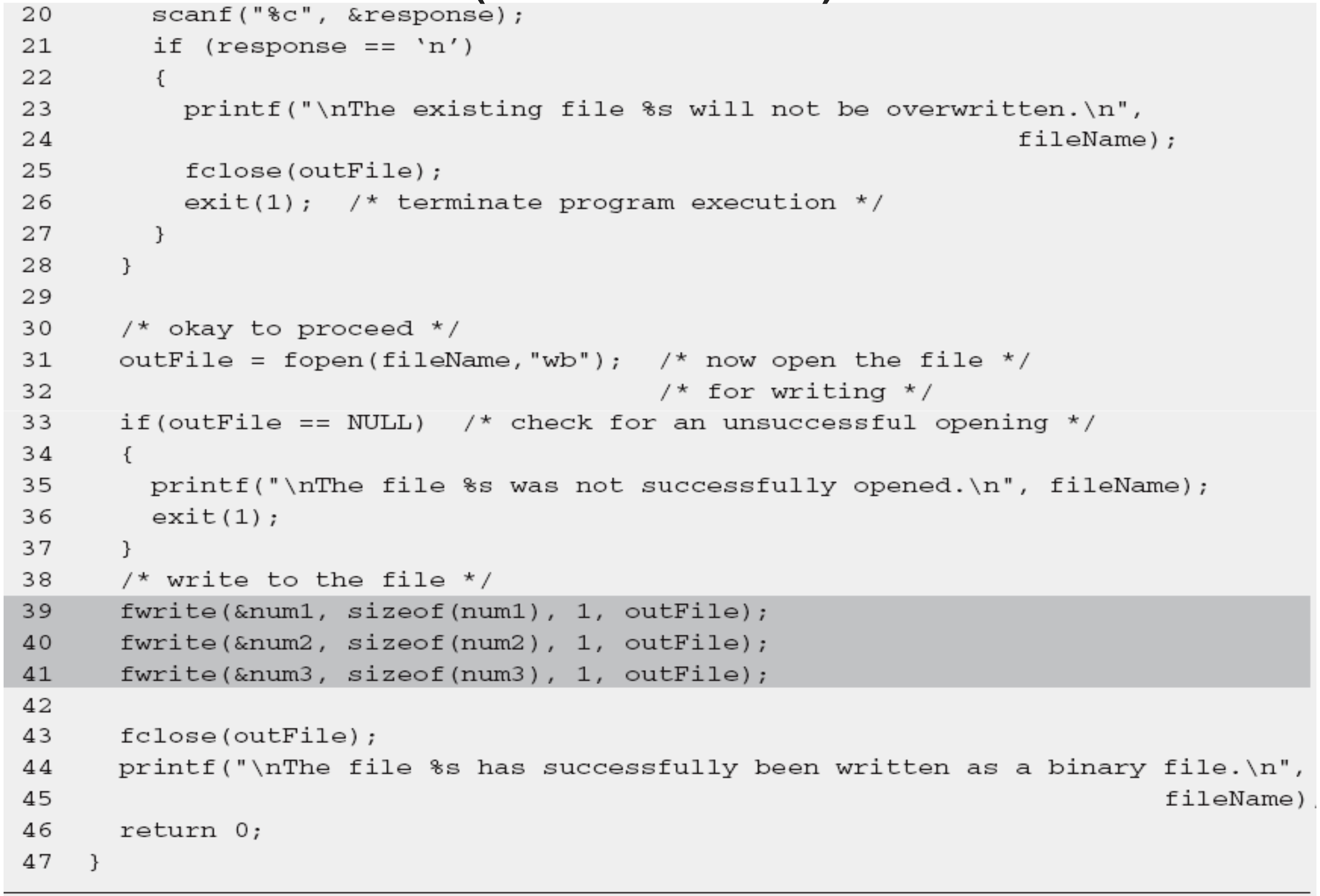
Writing and Reading Binary Files (continued)



Program 10.12

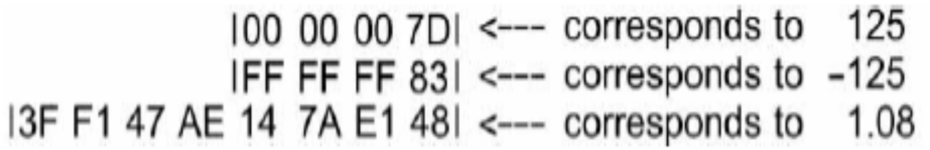
```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  char response;
  char fileName[20] = "prices.bin";
  FILE *outFile;
  int num1 = 125;
  long num2 = -125;
  double num3 = 1.08;

  /* check that a file by the given name does not already exist */
  outFile = fopen(fileName,"r"); /* open the file */
  if (outFile != NULL) /* check for a successful open*/
  {
    printf("\nA file by the name %s exists.", fileName);
    printf("\nDo you want to continue and overwrite it");
    printf("\n with the new data (y or n): ");
```

Writing and Reading Binary Files (continued)



```c
    scanf("%c", &response);
    if (response == 'n')
    {
      printf("\nThe existing file %s will not be overwritten.\n",
                                                         fileName);
      fclose(outFile);
      exit(1);  /* terminate program execution */
    }
  }

  /* okay to proceed */
  outFile = fopen(fileName,"wb");  /* now open the file */
                                   /* for writing */
  if(outFile == NULL)  /* check for an unsuccessful opening */
  {
    printf("\nThe file %s was not successfully opened.\n", fileName);
    exit(1);
  }
  /* write to the file */
  fwrite(&num1, sizeof(num1), 1, outFile);
  fwrite(&num2, sizeof(num2), 1, outFile);
  fwrite(&num3, sizeof(num3), 1, outFile);

  fclose(outFile);
  printf("\nThe file %s has successfully been written as a binary file.\n",
                                                                  fileName);
  return 0;
}
```

Writing and Reading Binary Files (continued)



```
        |00 00 00 7D| <--- corresponds to   125
        |FF FF FF 83| <--- corresponds to  -125
|3F F1 47 AE 14 7A E1 48| <--- corresponds to   1.08
```

Figure 10.3 The stored binary data in the `prices.bin` file and their decimal equivalents

Writing and Reading Binary Files (continued)

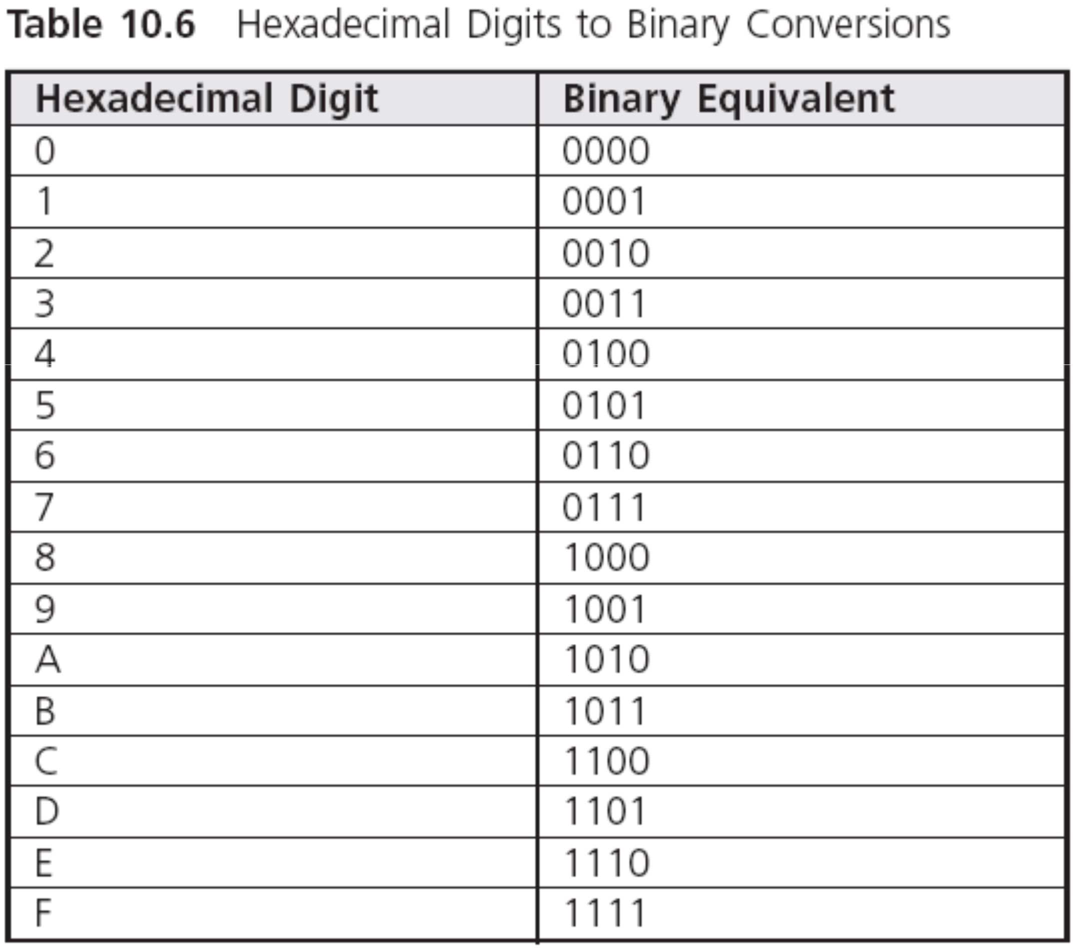
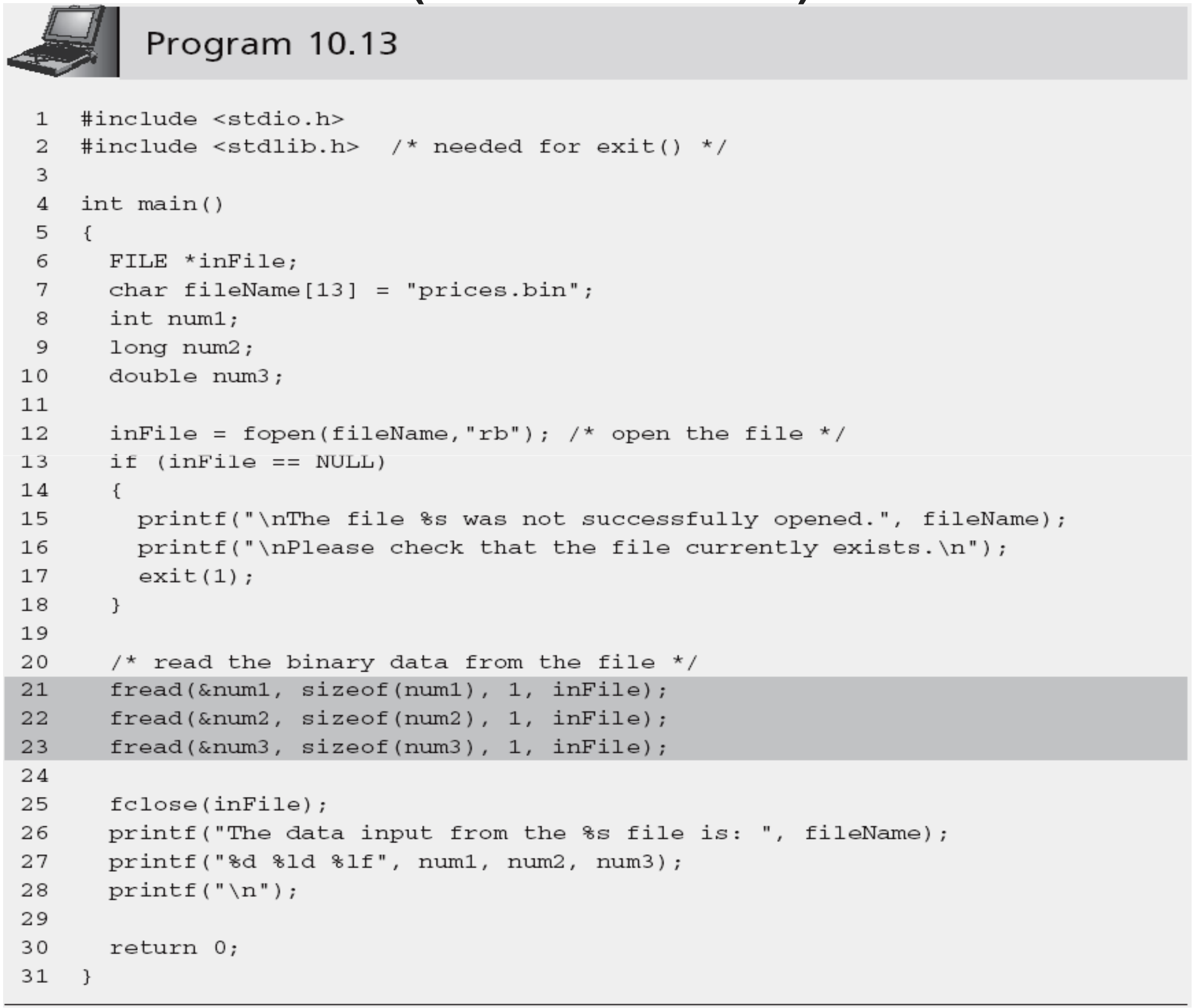


Table 10.6 Hexadecimal Digits to Binary Conversions

Hexadecimal Digit	Binary Equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Writing and Reading Binary Files (continued)



Program 10.13

```c
#include <stdio.h>
#include <stdlib.h>  /* needed for exit() */

int main()
{
  FILE *inFile;
  char fileName[13] = "prices.bin";
  int num1;
  long num2;
  double num3;

  inFile = fopen(fileName,"rb"); /* open the file */
  if (inFile == NULL)
  {
    printf("\nThe file %s was not successfully opened.", fileName);
    printf("\nPlease check that the file currently exists.\n");
    exit(1);
  }

  /* read the binary data from the file */
  fread(&num1, sizeof(num1), 1, inFile);
  fread(&num2, sizeof(num2), 1, inFile);
  fread(&num3, sizeof(num3), 1, inFile);

  fclose(inFile);
  printf("The data input from the %s file is: ", fileName);
  printf("%d %ld %lf", num1, num2, num3);
  printf("\n");

  return 0;
}
```

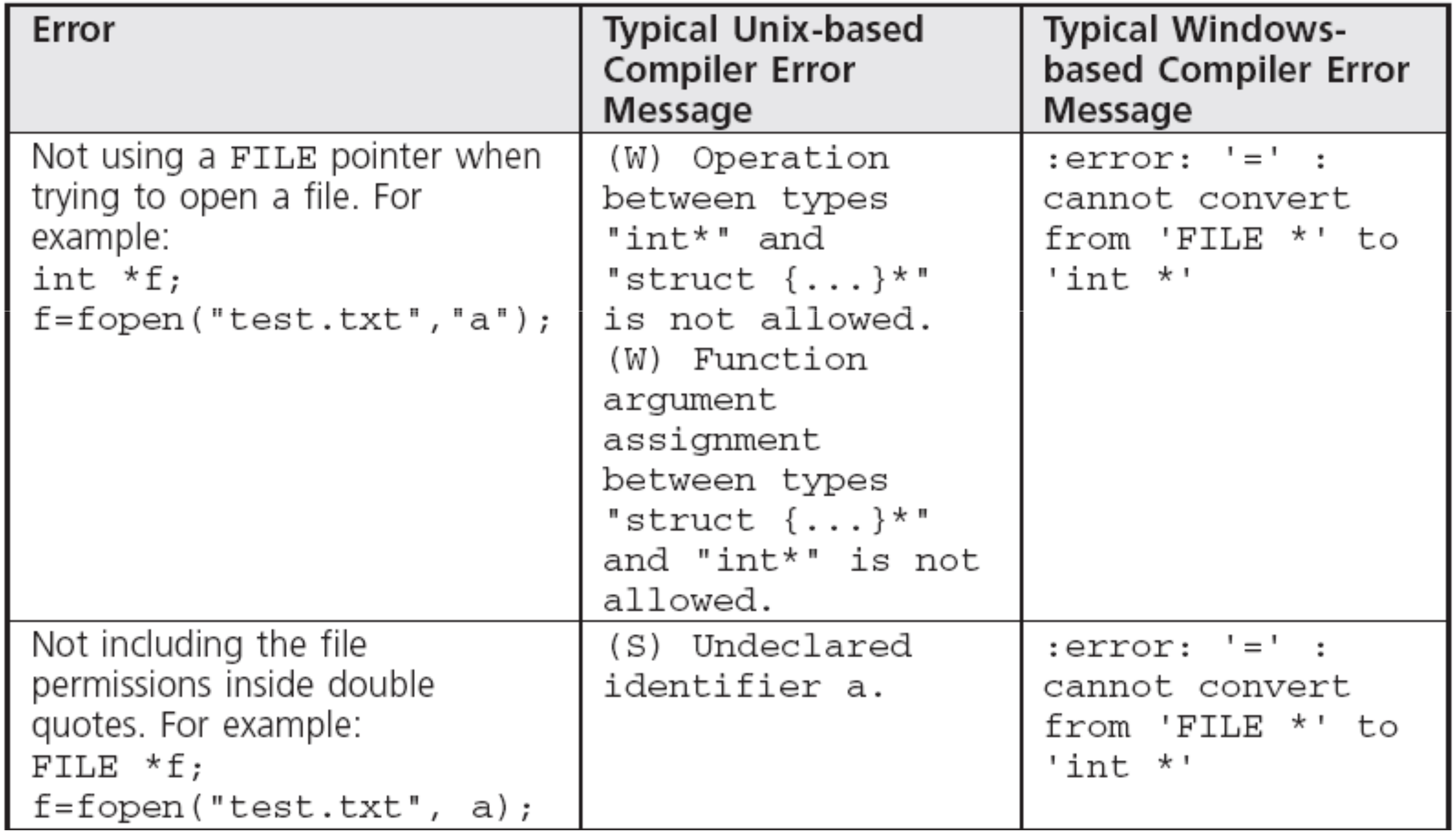
Common Programming Errors

- Using a file’s external name in place of the internal file pointer variable name when accessing the file
- Omitting the file pointer name altogether
- Opening a file for output without first checking that a file with the given name already exists
- Not understanding the end of a file is only detected until after the EOF sentinel has either been read or passed over

Common Programming Errors (continued)

- Attempting to detect the end of a file using character variable for the EOF marker
- Supplying an integer argument offset to the `seekg()` and `seekp()` functions
- Not using the `sizeof()` operator when specifying the number of bytes to be written when writing a binary file
- Not using the `sizeof()` operator when specifying the number of bytes to be read when reading a binary file

Common Compiler Errors

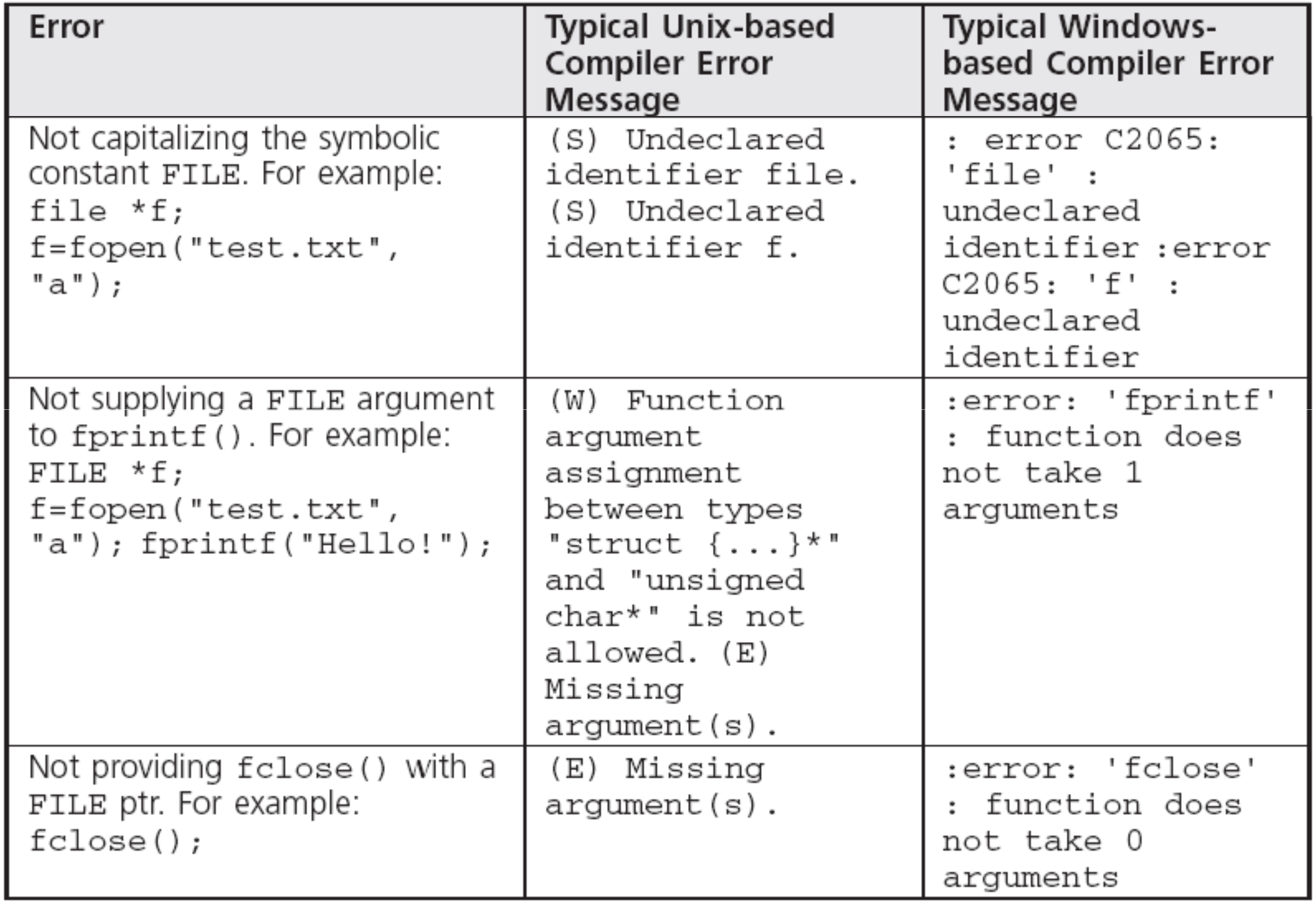


Error	Typical Unix-based Compiler Error Message	Typical Windows-based Compiler Error Message

Not using a `FILE` pointer when trying to open a file. For example:
`int *f;`
`f=fopen("test.txt","a");`
	`(W) Operation between types "int*" and "struct {...}*" is not allowed.`
	`(W) Function argument assignment between types "struct {...}*" and "int*" is not allowed.`
		`:error: '=' : cannot convert from 'FILE *' to 'int *'`

Not including the file permissions inside double quotes. For example:
`FILE *f;`
`f=fopen("test.txt", a);`
	`(S) Undeclared identifier a.`	`:error: '=' : cannot convert from 'FILE *' to 'int *'`

Common Compiler Errors (continued)



Error	Typical Unix-based Compiler Error Message	Typical Windows-based Compiler Error Message

Not capitalizing the symbolic constant FILE. For example:
file *f;
f=fopen("test.txt", "a");
	(S) Undeclared identifier file.
	(S) Undeclared identifier f.
		: error C2065: 'file' : undeclared identifier :error C2065: 'f' : undeclared identifier

Not supplying a FILE argument to fprintf(). For example:
FILE *f;
f=fopen("test.txt", "a"); fprintf("Hello!");
	(W) Function argument assignment between types "struct {...}*" and "unsigned char*" is not allowed. (E) Missing argument(s).	:error: 'fprintf' : function does not take 1 arguments

Not providing fclose() with a FILE ptr. For example: fclose();	(E) Missing argument(s).	:error: 'fclose' : function does not take 0 arguments

Summary

- A data file is any collection of data stored together in an external storage medium under a common name
- Data files can be stored as either character-based or binary files
- A data file is opened using the `fopen()` standard library function
- A file can be opened for reading, writing, or appending

Summary (continued)

- An internal filename must be declared as a pointer to a `FILE`
- In addition to any files opened within a function, the standard files `stdin`, `stdout`, and `stderr` are automatically opened when a program is run
- Data files can be accessed randomly using `rewind()`, `fseek()`, and `ftell()`
- Table 10.7 lists the standard file library functions