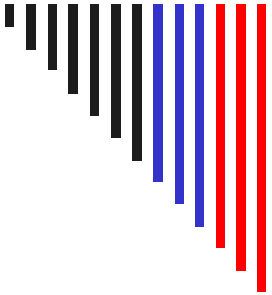

Chapter 12 Enumerated, Structure, and Union Types

Objectives

- To introduce the structure, union, and enumerated types
- To use the type definition statement in programs
- To use enumerated types, including anonymous types.
- To create and use structures in programs
- To be able to use unions in programs
- To understand the software engineering concept of coupling and to be able to evaluate coupling between functions.

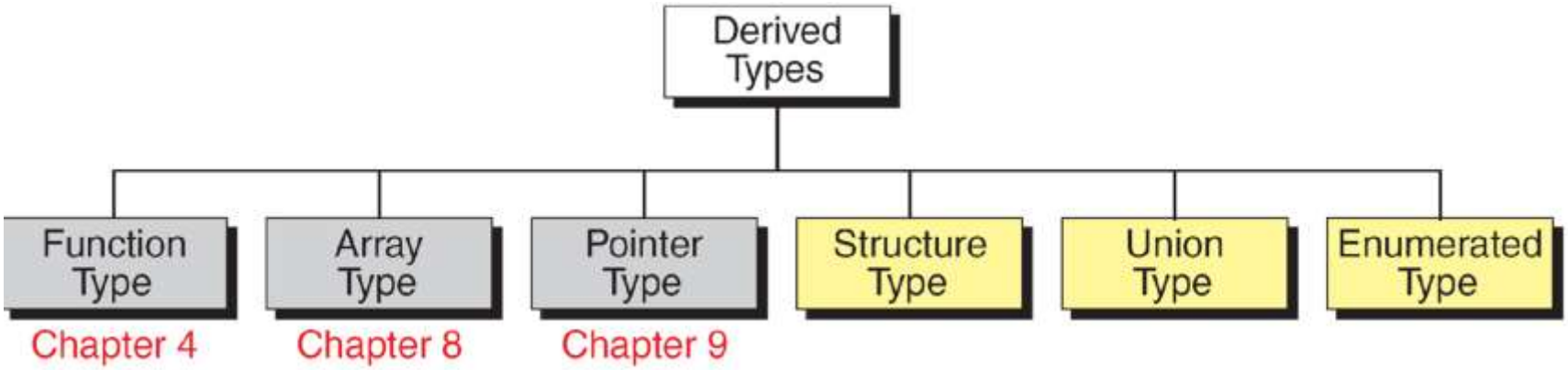

FIGURE 12-1 Derived Types

12-1 The Type Definition (*typedef*)

Before discussing the derived types, let's discuss a C declaration that applies to all of them—the type definition. A type definition, *typedef*, ***gives a name to a data type by creating a new type that can then be used anywhere a type is permitted.***

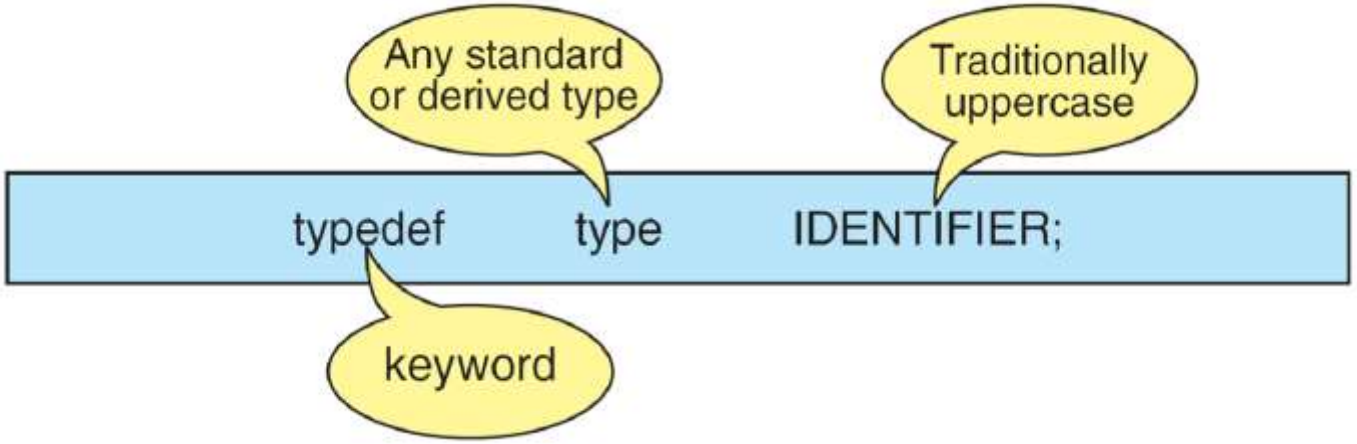

FIGURE 12-2 **Type-definition Format**

12-2 Enumerated Types

The enumerated type is a user-defined type based on the standard integer type. In an enumerated type, each integer value is given an identifier called an enumeration constant.

Topics discussed in this section:

Declaring an Enumerated Type
Operations on Enumerated Types
Enumeration Type Conversion
Initializing Enumerated Constants
Anonymous Enumeration: Constants
Input/Output Operations

PROGRAM 12-1 Print Cable TV Stations

```
1  /* Print selected TV stations for our cable TV system.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9  // Local Declarations
10    enum TV {fox =  2, nbc  =  4, cbs  =  5,
11             abc = 11, hbo  = 15, show = 17,
12             max = 31, espn = 39, cnn  = 51};
13
14 // Statements
15    printf("Here are my favorite cable stations:\n");
16    printf("  ABC: \t%2d\n", abc);
17    printf("  CBS: \t%2d\n", cbs);
18    printf("  CNN: \t%2d\n", cnn);
19    printf("  ESPN:\t%2d\n", espn);
```

PROGRAM 12-1 Print Cable TV Stations

```
20      printf("  Fox: \t%2d\n", fox);
21      printf("  HBO: \t%2d\n", hbo);
22      printf("  Max: \t%2d\n", max);
23      printf("  NBC: \t%2d\n", nbc);
24      printf("  Show:\t%2d\n", show);
25      printf("End of my favorite stations. \n");
26      return 0;
27  }   // main
```

```
Results:
Here are my favorite cable stations:
  ABC:     11
  CBS:      5
  CNN:     51
  ESPN:    39
  Fox:      2
  HBO:     15
  Max:     31
  NBC:      4
  Show:    17
End of my favorite stations.
```

Note

Don't be confused about strings and enumerated types.
"**cnn**" is a string made of three characters;
cnn, as defined in the previous code example, is an enumerated type (identifier) which has the integer value 51.

```
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

```
#include<stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,
          Aug, Sep, Oct, Nov, Dec};

int main()
{
   int i;
   for (i=Jan; i<=Dec; i++)
      printf("%d ", i);

   return 0;
}
```

```
#include <stdio.h>

enum weekdays {Sunday, Monday = 2, Tuesday, Wednesday =
6, Thursday, Friday = 9, Saturday = 12};

int main()

{

printf("%d %d %d %d %d %d %d", Sunday, Monday,
Tuesday,

Wednesday, Thursday, Friday, Saturday);

return 0;

}
```

12-3 Structure

A structure is a collection of related elements, possibly of different types, having a single name.

Topics discussed in this section:

Structure Type Declaration
Initialization
Accessing Structures
Operations on Structures
Complex Structures
Structures and Functions

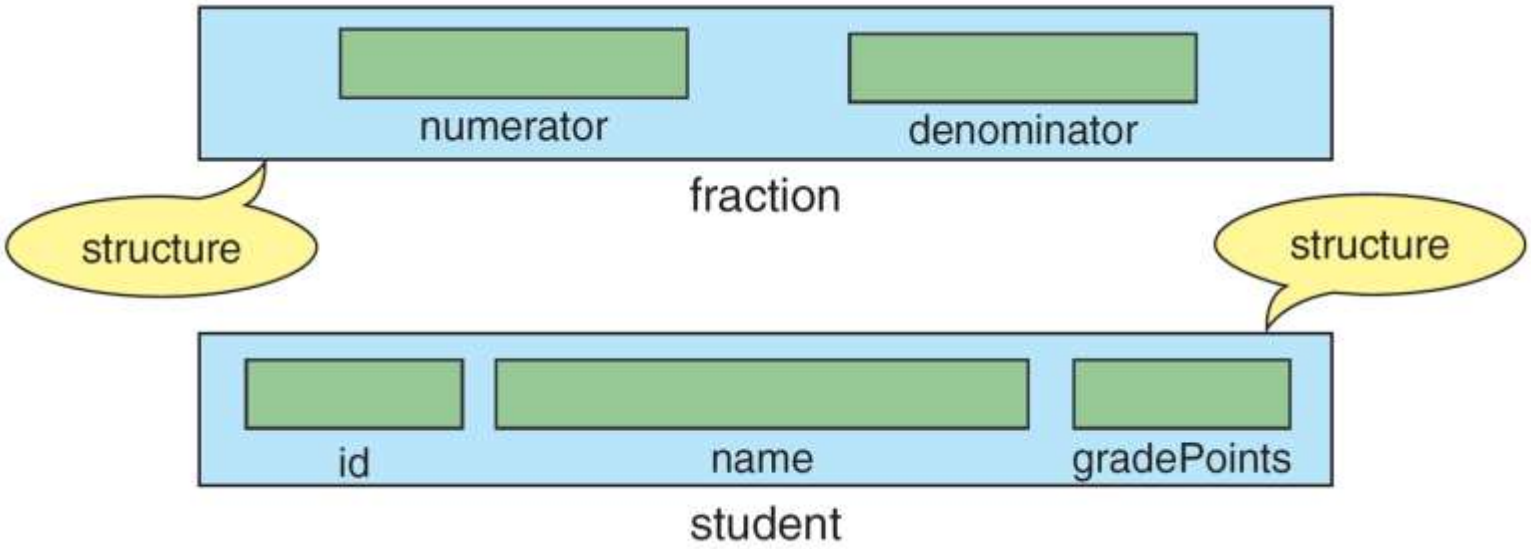

FIGURE 12-3 Structure Examples

Note

Elements in a structure can be of the same or different types. However, all elements in the structure should be logically related.

```
struct TAG
   {

    field list

   } ;
```

Format

```
struct STUDENT
   {
    char id[10];
    char name[26];
    int  gradePts;
   } ; // STUDENT
```

Example

FIGURE 12-4 Tagged Structure Format

```
typedef struct
   {

    field list

   } TYPE;
```

Format

```
typedef struct
   {
    char id[10];
    char name[26];
    int  gradePts;
   } STUDENT;
```

Example

FIGURE 12-5 Structure Declaration with *typedef*

```
// Global Type Declarations
struct STUDENT
   {
    char id[10];
    char name[26];
    int  gradePts;
   } ;

// Local Declarations
struct STUDENT aStudent;
```

```
// Global Type Declarations
typedef struct
   {
    char id[10];
    char name[26];
    int  gradePts;
   } STUDENT;

// Local Declarations
STUDENT aStudent;
```

FIGURE 12-6 Structure Declaration Format and Example

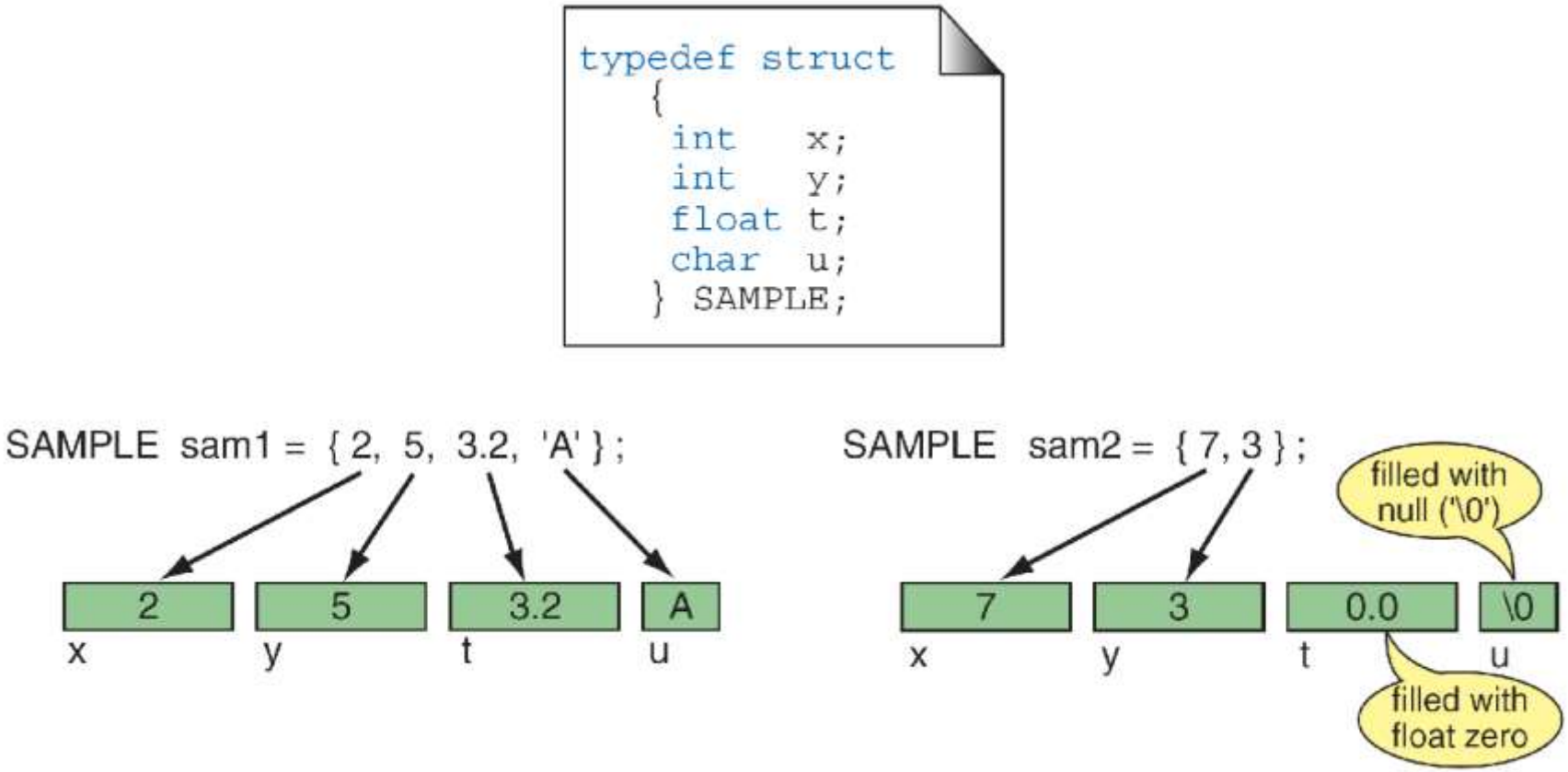

FIGURE 12-7 Initializing Structures

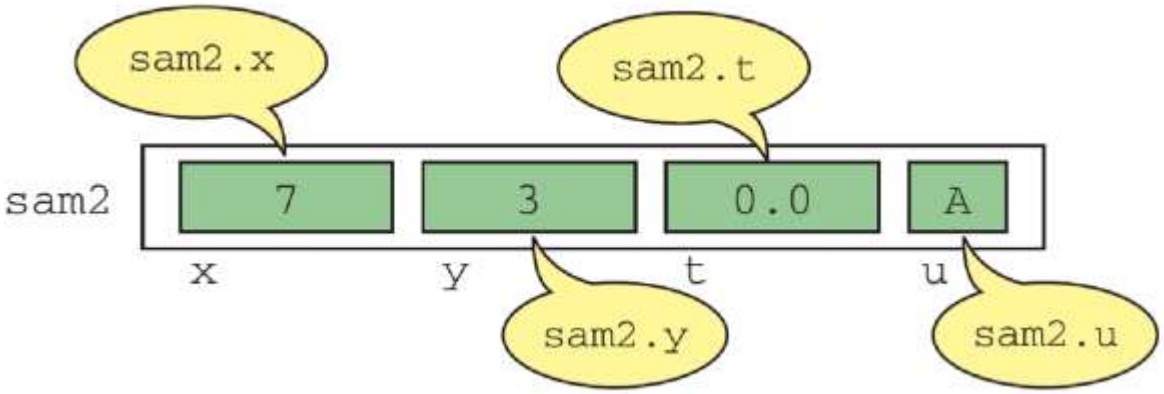

FIGURE 12-8 Structure Direct Selection Operator

PROGRAM 12-2 Multiply Fractions

```
1  /* This program uses structures to simulate the
2     multiplication of fractions.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7
8  // Global Declarations
9     typedef  struct
10       {
11        int numerator;
12        int denominator;
13       } FRACTION;
14
15 int main (void)
16 {
17 // Local Declarations
18    FRACTION  fr1;
19    FRACTION  fr2;
20    FRACTION  res;
21
```

PROGRAM 12-2 Multiply Fractions

```
22  // Statements
23     printf("Key first fraction in the form of x/y: ");
24     scanf ("%d /%d", &fr1.numerator, &fr1.denominator);
25     printf("Key second fraction in the form of x/y: ");
26     scanf ("%d /%d", &fr2.numerator, &fr2.denominator);
27
28     res.numerator   = fr1.numerator   * fr2.numerator;
29     res.denominator = fr1.denominator * fr2.denominator;
30
31     printf("\nThe result of %d/%d * %d/%d is %d/%d",
32            fr1.numerator, fr1.denominator,
33            fr2.numerator, fr2.denominator,
34            res.numerator, res.denominator);
35     return 0;
36  }  // main
```

```
Results:
Key first fraction in the form of x/y:     2/6
Key second fraction in the form of x/y:    7/4

The result of 2/6 * 7/4 is 14/24
```

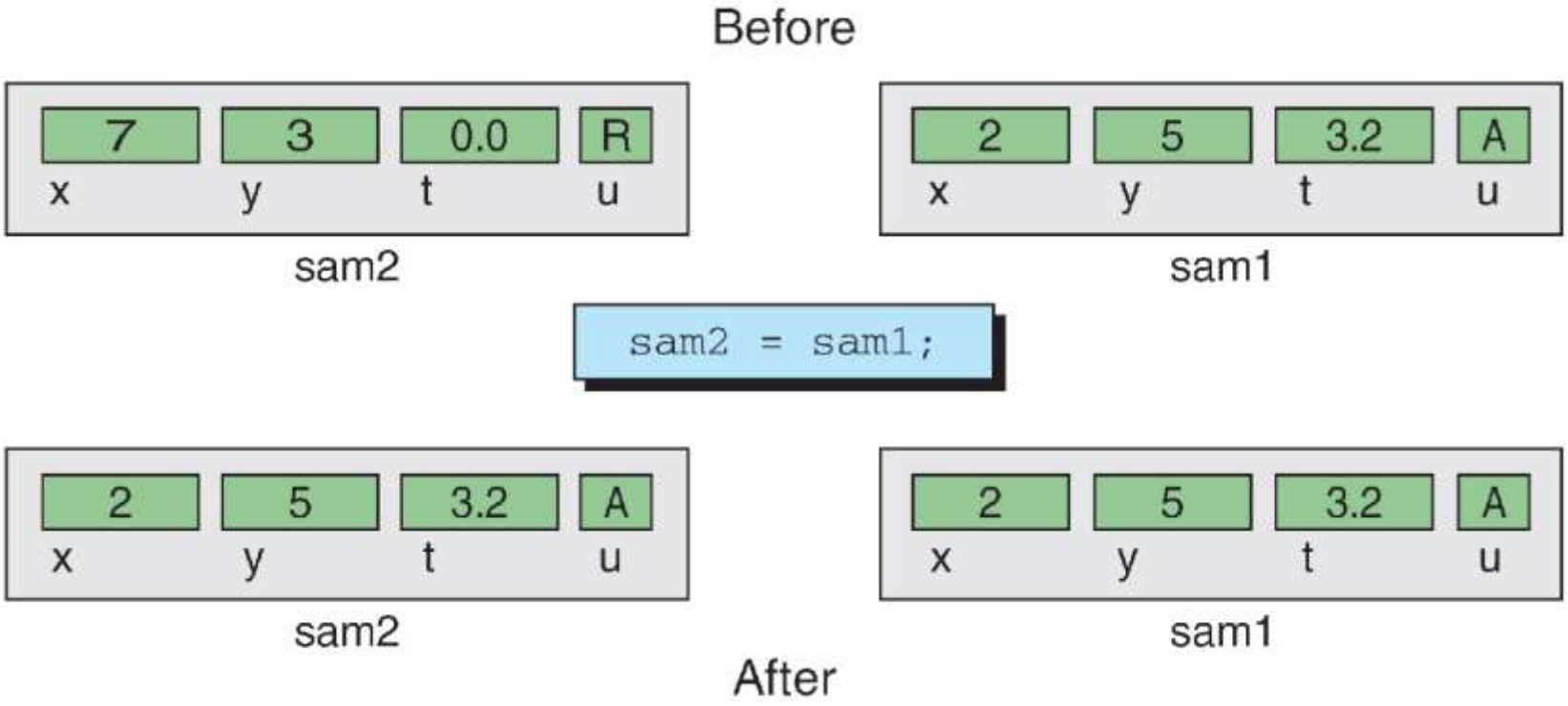

FIGURE 12-9 Copying a Structure

```
typedef struct
   {
    int    x;
    int    y;
    float  t;
    char   u;
   } SAMPLE;
    ...
    SAMPLE  sam1;
    SAMPLE* ptr;
    ...
    ptr = &sam1;
    ...
```

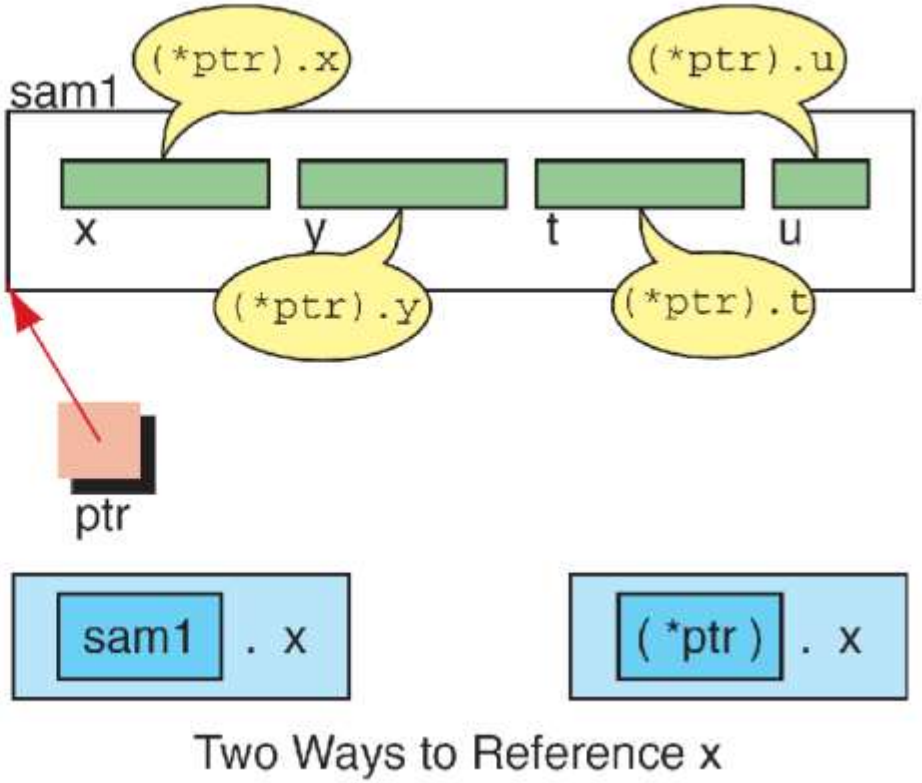

FIGURE 12-10 Pointers to Structures

Note

(*pointerName).fieldName ↔ pointerName->fieldName.

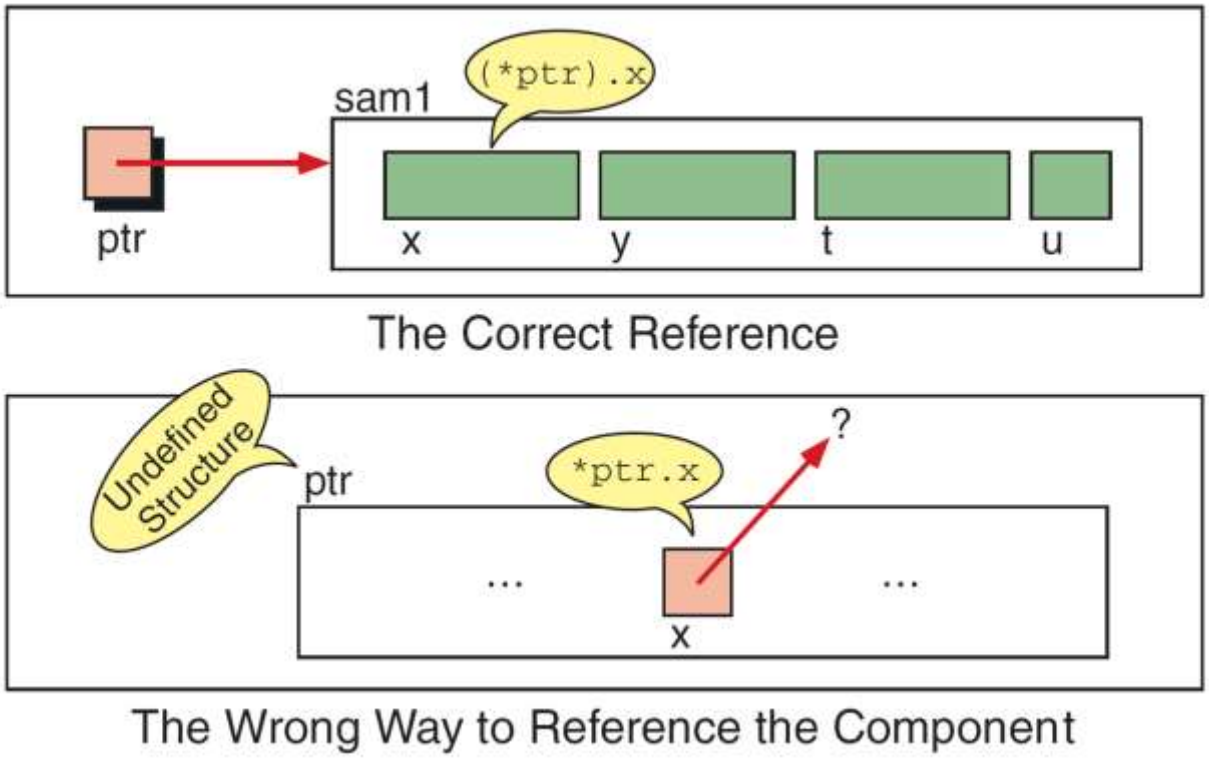

FIGURE 12-11 Interpretation of Invalid Pointer Use

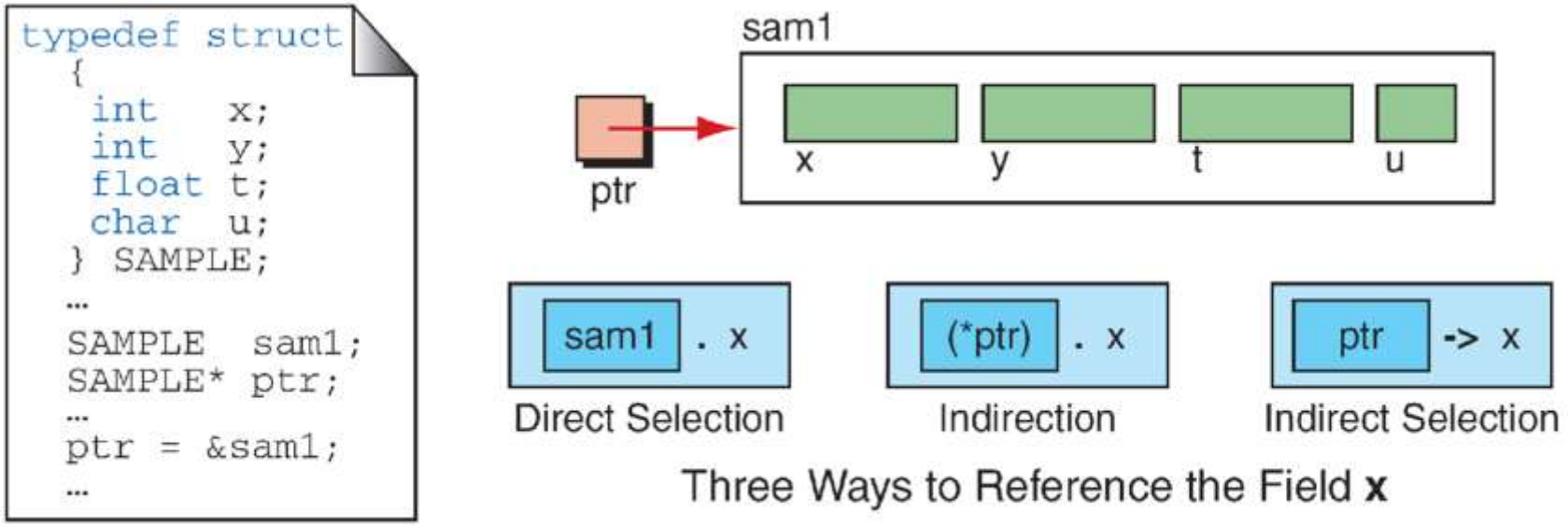

FIGURE 12-12 Indirect Selection Operator

PROGRAM 12-3 Clock Simulation with Pointers

```
1  /* This program uses a structure to simulate the time.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  typedef struct
8     {
9      int hr;
10     int min;
11     int sec;
12    } CLOCK;
13
14 // Function Declaration
15 void increment (CLOCK* clock);
16 void show      (CLOCK* clock);
17
18 int main (void)
19 {
```

PROGRAM 12-3 Clock Simulation with Pointers

```
20  // Local Declaration
21     CLOCK clock = {14, 38, 56};
22
23  // Statements
24     for(int i = 0; i < 6; ++i)
25        {
26         increment (&clock);
27         show (&clock);
28        } // for
29     return 0;
30  }  // main
31
32  /* ==================== increment ====================
33     This function accepts a pointer to clock and
34     increments the time by one second.
35        Pre   previous clock setting
36        Post  clock incremented by one second.
37  */
38  void increment (CLOCK* clock)
39  {
```

PROGRAM 12-3 Clock Simulation with Pointers

```
40  // Statements
41     (clock->sec)++;
42     if (clock->sec == 60)
43        {
44         clock->sec = 0;
45         (clock->min)++;
46         if (clock->min == 60)
47            {
48             clock->min = 0;
49             (clock->hr)++;
50             if (clock->hr == 24)
51                 clock->hr = 0;
52            } // if 60 min
53        } // if 60 sec
54     return;
55  }  // increment
56
```

PROGRAM 12-3 Clock Simulation with Pointers

```
57  /* ======================== show ========================
58     Show the current time in military form.
59        Pre   clock time
60        Post  clock time displayed
61  */
62  void show (CLOCK* clock)
63  {
64  // Statements
65     printf("%02d:%02d:%02d\n",
66              clock->hr, clock->min, clock->sec);
67     return;
68  }  // show
```

```
Results:
14:38:57
14:38:58
14:38:59
14:39:00
14:39:01
14:39:02
```

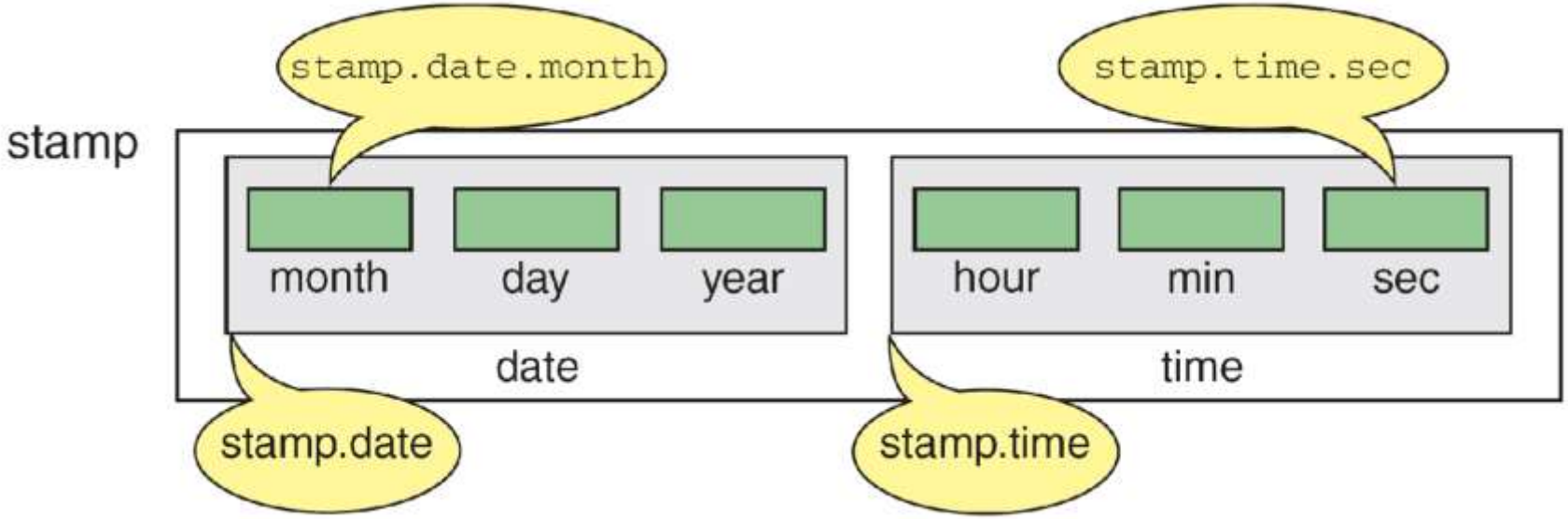

FIGURE 12-13 **Nested Structure**

```
// Global Declarations
typedef struct
   {
    char name[26];
    int  midterm[3];
    int  final;
   } STUDENT ;
// Local Declarations
STUDENT student;
```

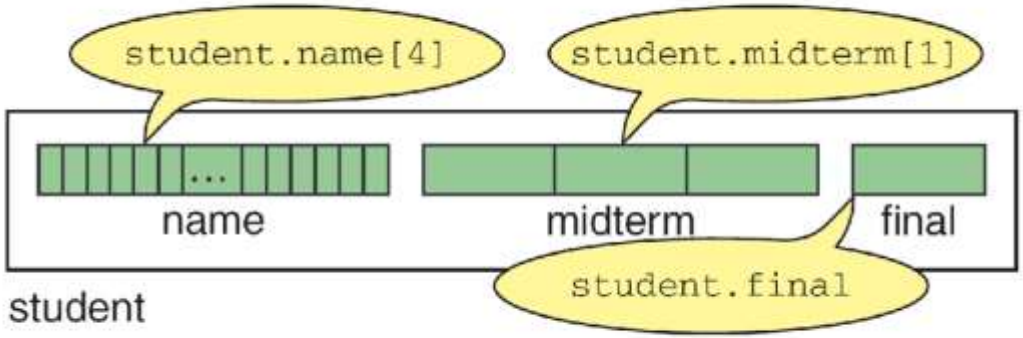

FIGURE 12-14 Arrays in Structures

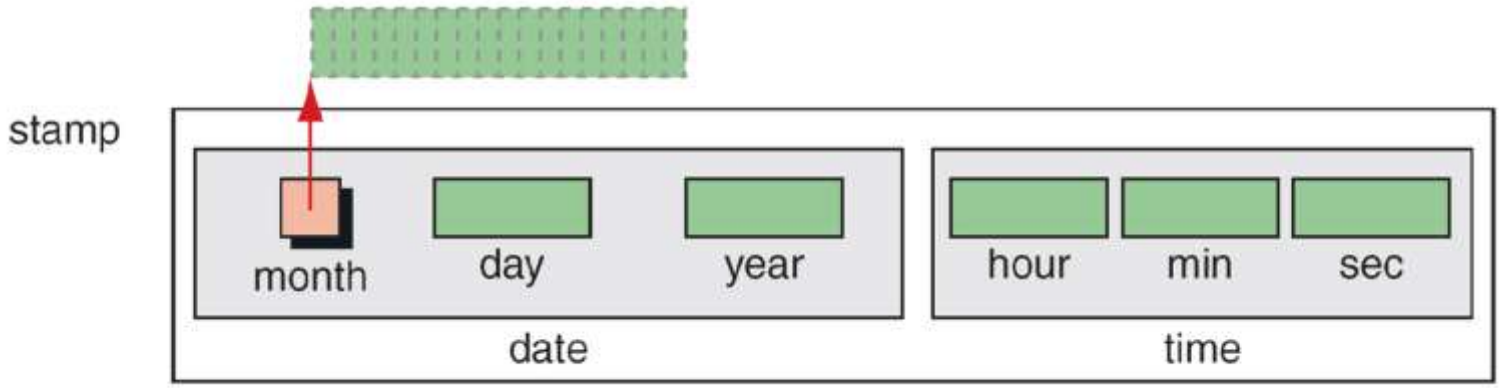

FIGURE 12-15 Pointers in Structures

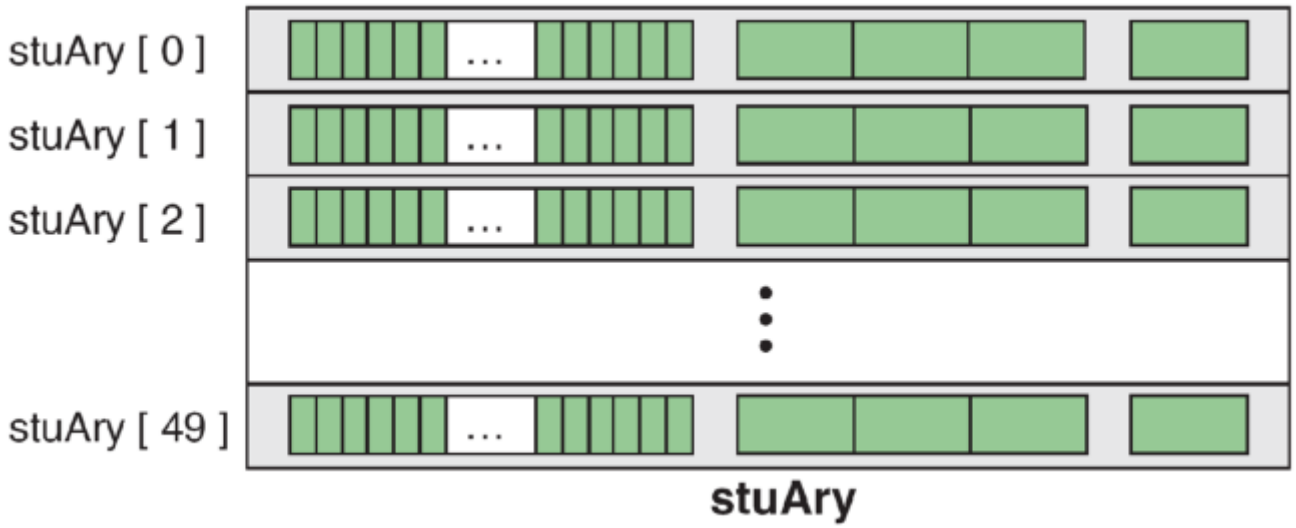

FIGURE 12-16 Array of Structures

```
#include <stdio.h>
struct student {
    char firstName[50];
    int roll;
    float marks;
} s[5];
```

```
int main() {
    int i;
    printf("Enter information of students:\n");

    // storing information
    for (i = 0; i < 5; ++i) {
        s[i].roll = i + 1;
        printf("\nFor roll number%d,\n", s[i].roll);
        printf("Enter first name: ");
        scanf("%s", s[i].firstName);
        printf("Enter marks: ");
        scanf("%f", &s[i].marks);
    }
```

```
 printf("Displaying Information:\n\n");

    // displaying information
    for (i = 0; i < 5; ++i) {
        printf("\nRoll number: %d\n", i + 1);
        printf("First name: ");
        puts(s[i].firstName);
        printf("Marks: %.1f", s[i].marks);
        printf("\n");
    }
    return 0;
}
```

```
...
res.numerator =
   multiply(fr1.numerator, fr2.numerator)
res.denominator =
   multiply(fr1.denominator, fr2.denominator);
...
```

```
// =============== multiply ===============
multiply (int x, int y)
{
   return x * y;
}  // multiply
```

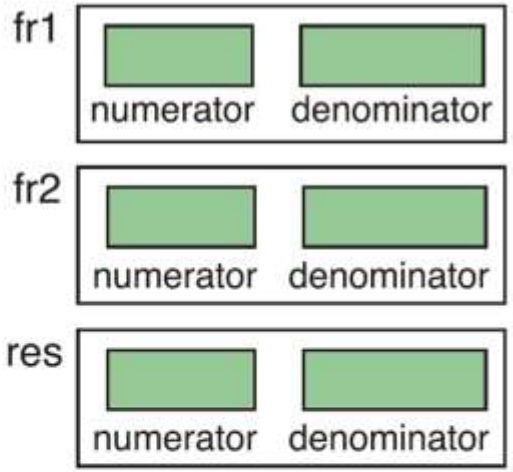

FIGURE 12-17 Passing Structure Members to Functions

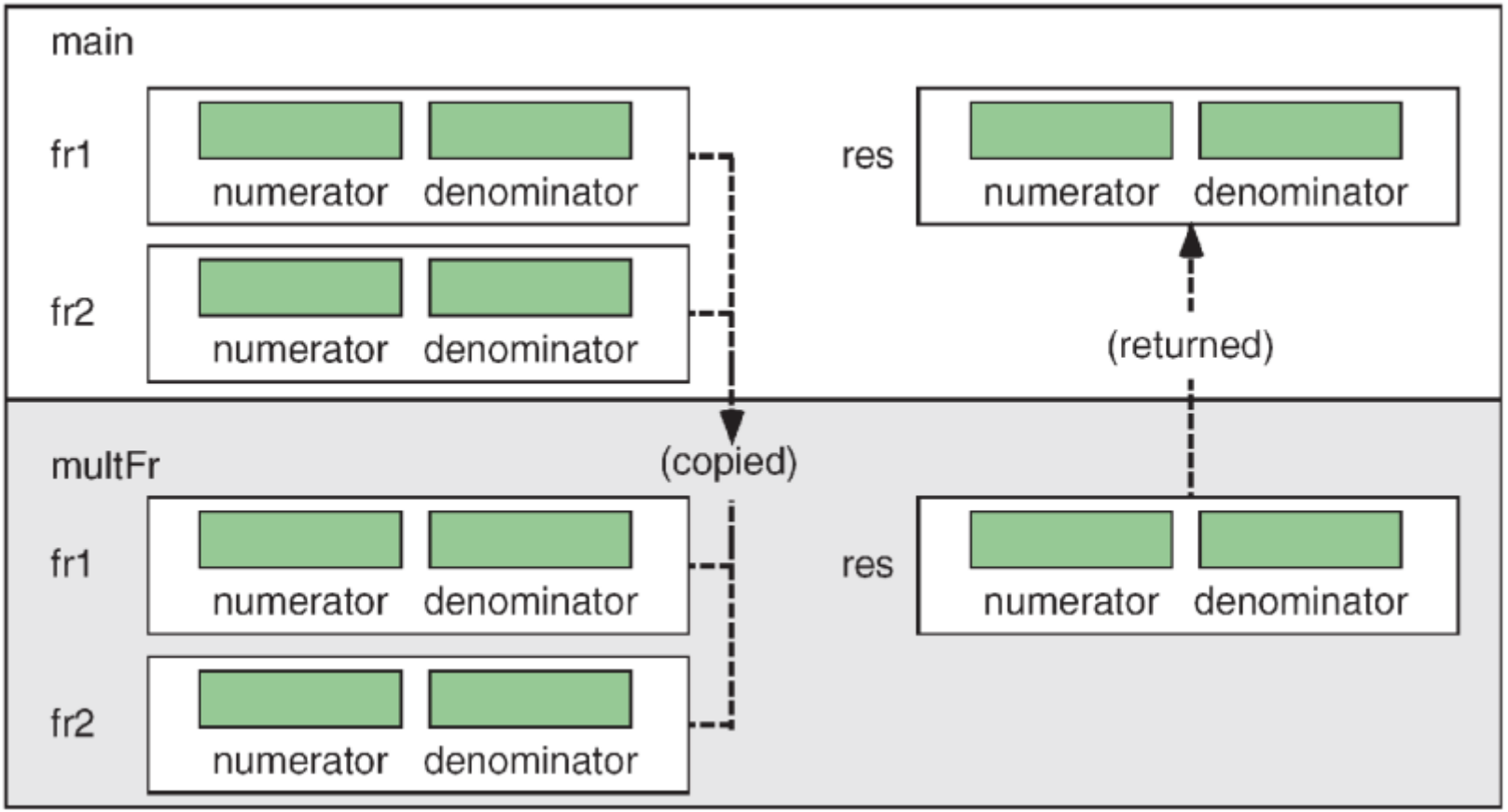

FIGURE 12-18 Passing and returning structures

PROGRAM 12-5 Passing and Returning Structures

```
 1  /* This program uses structures to multiply fractions.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  // Global Declarations
 8  typedef  struct
 9       {
10        int numerator;
11        int denominator;
12       } FRACTION;
13
14  // Function Declarations
15  FRACTION getFr   (void);
16  FRACTION multFr  (FRACTION fr1, FRACTION fr2);
17  void     printFr (FRACTION fr1, FRACTION fr2,
18                    FRACTION result);
19
```

PROGRAM 12-5 Passing and Returning Structures

```
20  int main (void)
21  {
22  // Local Declarations
23     FRACTION fr1;
24     FRACTION fr2;
25     FRACTION res;
26
27  // Statements
28     fr1 = getFr  ();
29     fr2 = getFr  ();
30     res = multFr (fr1, fr2);
31     printFr (fr1, fr2, res);
32     return 0;
33  }  // main
34
```

PROGRAM 12-5 Passing and Returning Structures

```
35 /* ==================== getFr ====================
36    Get two integers from the keyboard, make & return
37    a fraction to the main program.
38       Pre   nothing
39       Post  returns a fraction
40 */
41 FRACTION getFr (void)
42 {
43 // Local Declarations
44    FRACTION fr;
45
46 // Statements
47    printf("Write a fraction in the form of x/y: ");
48    scanf ("%d/%d", &fr.numerator, &fr.denominator);
49    return fr;
50 }  // getFraction
51
```

PROGRAM 12-5 Passing and Returning Structures

```
52  /* ==================== multFr ====================
53     Multiply two fractions and return the result.
54        Pre   fr1 and fr2 are fractions
55        Post  returns the product
56  */
57  FRACTION multFr (FRACTION fr1, FRACTION fr2)
58  {
59  // Local Declaration
60     FRACTION res;
61
62  // Statements
63     res.numerator   = fr1.numerator   * fr2.numerator;
64     res.denominator = fr1.denominator * fr2.denominator;
65     return res;
66  } // multFr
67
```

PROGRAM 12-5 Passing and Returning Structures

```
68  /* ===================== printFr ====================
69     Prints the value of the fields in three fractions.
70        Pre   two original fractions and the product
71        Post  fractions printed
72  */
73  void  printFr  (FRACTION fr1,  FRACTION fr2,
74                  FRACTION res)
75  {
76  // Statements
77     printf("\nThe result of %d/%d * %d/%d is %d/%d\n",
78               fr1.numerator, fr1.denominator,
79               fr2.numerator, fr2.denominator,
80               res.numerator, res.denominator);
81     return;
82  }  // printFractions
83  // ================== End of Program ================
```

```
Results:
Write a fraction in the form of x/y: 4/3
Write a fraction in the form of x/y: 6/7

The result of 4/3 * 6/7 is 24/21
```

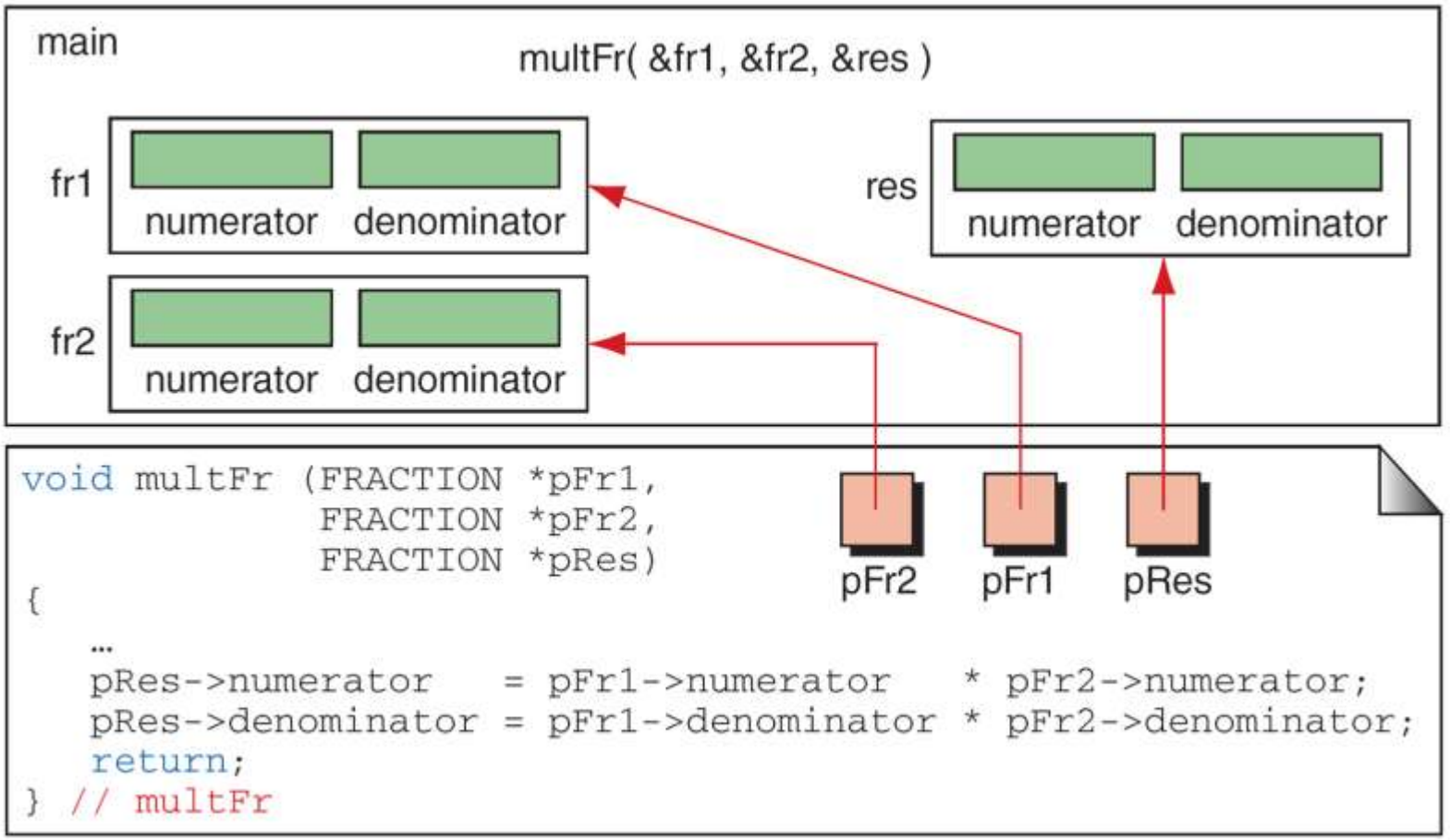

FIGURE 12-19 Passing Structures Through Pointers

PROGRAM 12-6 Passing Structures through Pointers

```
1  /* This program uses structures to multiply fractions.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  // Global Declarations
8  typedef struct
9           {
10           int numerator;
11           int denominator;
12         } FRACTION;
13
14 // Function Declarations
15 void getFr   (FRACTION* pFr);
16 void multFr  (FRACTION* pFr1, FRACTION* pFr2,
17               FRACTION* pRes2);
18 void printFr (FRACTION* pFr1, FRACTION* pFr2,
19               FRACTION* pRes);
20
```

PROGRAM 12-6 Passing Structures through Pointers

```
21  int   main (void)
22  {
23  // Local Declarations
24     FRACTION fr1;
25     FRACTION fr2;
26     FRACTION res;
27
28  // Statements
29     getFr   (&fr1);
30     getFr   (&fr2);
31     multFr  (&fr1, &fr2, &res);
32     printFr (&fr1, &fr2, &res);
33     return 0;
34  }  // main
35
36  /* ==================== getFr ====================
37     Get two integers from the keyboard, make & return a
38     fraction to the main program.
39        Pre   pFr is pointer to fraction structure
40        Post  fraction stored at pFr.
41  */
```

PROGRAM 12-6 Passing Structures through Pointers

```
42  void getFr (FRACTION* pFr)
43  {
44  // Statements
45     printf("Write a fraction in the form of x/y: ");
46     scanf ("%d/%d", &pFr->numerator,
47            &(*pFr).denominator);
48     return;
49  }  // getFr
50
51  /* ======================== multFr ====================
52     Multiply two fractions and return the result.
53        Pre   fr1, fr2, pRes are pointers to fractions
54        Post  product stored at pRes
55  */
56  void multFr  (FRACTION* pFr1, FRACTION* pFr2,
57                FRACTION* pRes)
58  {
59  // Statements
60     pRes->numerator    =
61           pFr1->numerator * pFr2->numerator;
```

PROGRAM 12-6 Passing Structures through Pointers

```
62      pRes->denominator =
63            pFr1->denominator * pFr2->denominator;
64      return;
65  }   // multFr
66
67  /* ====================== printFr ====================
68      Prints the value of the fields in three fractions.
69         Pre   pointers to two fractions and their product
70         Post  fractions printed
71  */
72  void printFr (FRACTION* pFr1, FRACTION* pFr2,
73                FRACTION* pRes)
74  {
75  // Statements
76      printf("\nThe result of %d/%d * %d/%d is %d/%d\n",
77             pFr1->numerator, pFr1->denominator,
78             pFr2->numerator, pFr2->denominator,
79             pRes->numerator, pRes->denominator);
80      return;
81  }   // printFr
82  // ================= End of Program =================
```

File Handling

Storage seen so far

- All variables stored in memory
- Problem: the contents of memory are wiped out when the computer is powered off
- Example: Consider keeping students' records
  - 100 students records are added in array of structures
  - Machine is then powered off after sometime
  - When the machine is powered on, the 100 records entered earlier are all gone!
  - Have to enter again if they are needed

Solution: Files

- A named collection of data, stored in secondary storage like disk, CD-ROM, USB drives etc.
- Persistent storage, not lost when machine is powered off
- Save data in memory to files if needed (file write)
- Read data from file later whenever needed (file read)

Organization of a file

- Stored as sequence of bytes, logically contiguous
  - May not be physically contiguous on disk, but you do not need to worry about that
- The last byte of a file contains the end-of-file character (**`EOF`**), with ASCII code 1A (hex).
  - While reading a text file, the EOF character can be checked to know the end
- Two kinds of files:
  - Text : contains ASCII codes only
  - Binary : can contain non-ASCII characters
    - Example: Image, audio, video, executable, etc.
    - EOF cannot be used to check end of file

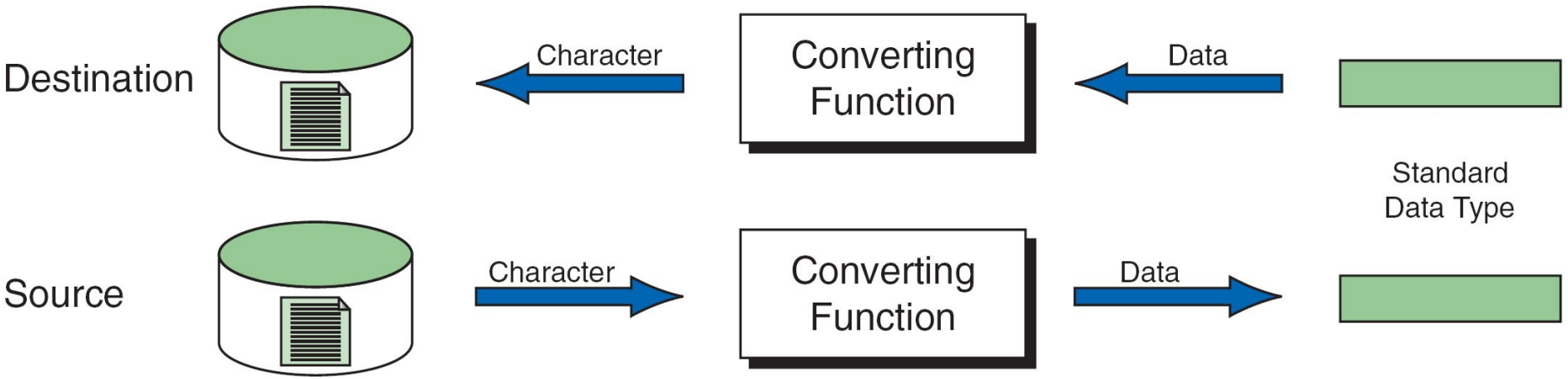

FIGURE 13-1 Reading and Writing Text Files

Note

Formatted input/output, character input/output, and string input/output functions can be used only with text files.

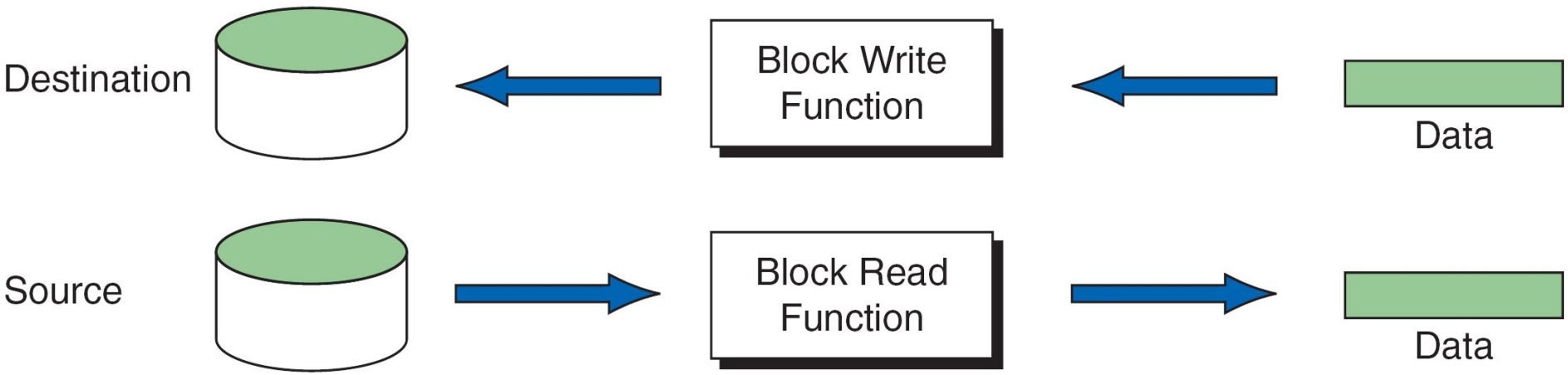

FIGURE 13-2 Block Input and Output

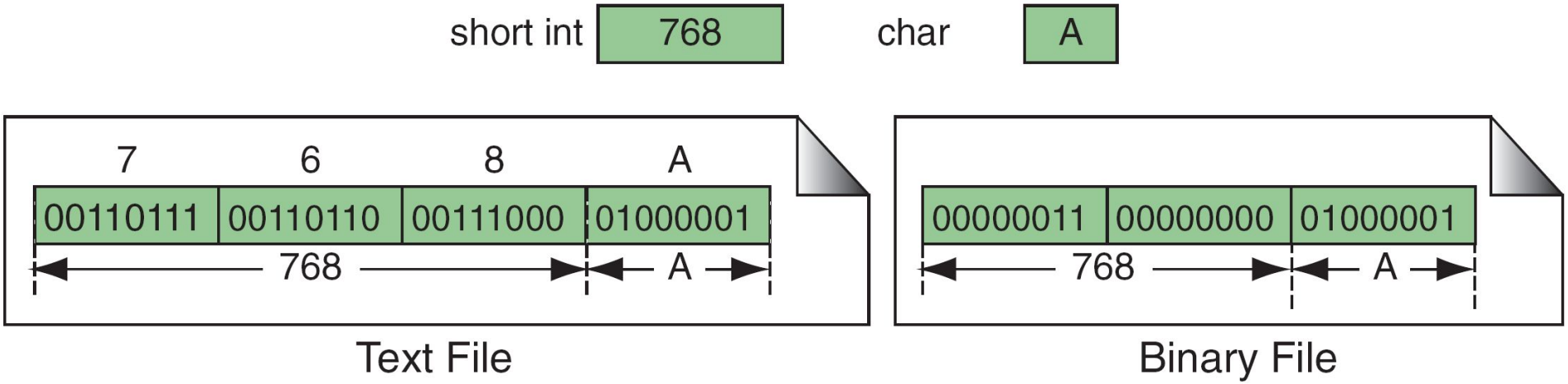

FIGURE 13-3 Binary and Text Files

Text File	Binary File
Its Bits represent character.	Its Bits represent a custom data.
Less prone to get corrupt as change reflects as soon as made and can be undone.	Can easily get corrupted, corrupt on even single bit change
Store only plain text in a file.	Can store different types of data (audio, text,image) in a single file.
Widely used file format and can be opened in any text editor.	Developed for an application and can be opened in that application only.
Mostly .txt,.rtf are used as extensions to text files.	Can have any application defined extension.

Note

Text files store data as a sequence of characters; binary files store data as they are stored in primary memory.

Basic operations on a file

- Open
- Read
- Write
- Close
- Mainly we want to do read or write, but a file has to be opened before read/write, and should be closed after all read/write is over

Opening a File: fopen()

- FILE * is a datatype used to represent a pointer to a file
- fopen takes two parameters, the name of the file to open and the mode in which it is to be opened
- It returns the pointer to the file if the file is opened successfully, or NULL to indicate that it is unable to open the file

Example: opening file.dat for write

```
FILE *fptr;
char filename[ ]= "file2.dat";
fptr = fopen (filename,"w");
if (fptr == NULL) {
     printf (“ERROR IN FILE CREATION”);
     /* DO SOMETHING */
}
```

Modes for opening files

- The second argument of fopen is the mode in which we open the file.

Modes for opening files

- The second argument of fopen is the mode in which we open the file.
  - "r" : opens a file for reading (can only read)
    - Error if the file does not already exists
    - "r+" : allows write also

Modes for opening files

- The second argument of fopen is the mode in which we open the file.
  - "r" : opens a file for reading (can only read)
    - Error if the file does not already exists
    - "r+" : allows write also
  - "w" : creates a file for writing (can only write)
    - Will create the file if it does not exist
    - Caution: writes over all previous contents if the flle already exists
    - "w+" : allows read also

Modes for opening files

- The second argument of fopen is the mode in which we open the file.
  - "r" : opens a file for reading (can only read)
    - Error if the file does not already exists
    - "r+" : allows write also
  - "w" : creates a file for writing (can only write)
    - Will create the file if it does not exist
    - Caution: writes over all previous contents if the flle already exists
    - "w+" : allows read also
  - "a" : opens a file for appending (write at the end of the file)
    - "a+" : allows read also

Mode	r	w	a	r+	w+	a+
Open State	read	write	write	read	write	write
Read Allowed	yes	no	no	yes	yes	yes
Write Allowed	no	yes	yes	yes	yes	yes
Append Allowed	no	no	yes	no	no	yes
File Must Exist	yes	no	no	yes	no	no
Contents of Existing File Lost	no	yes	no	no	yes	no

Table 13-1 File Modes

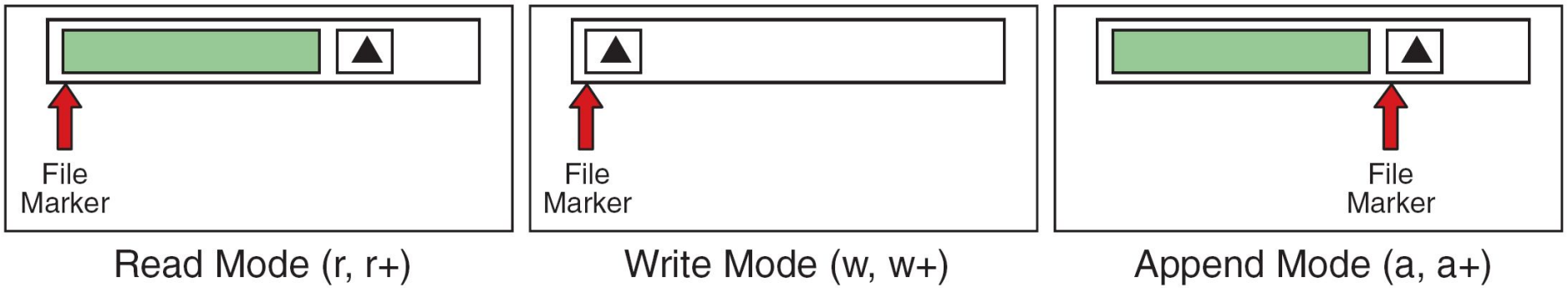

FIGURE 13-5 File-Opening Modes

The exit() function

- Sometimes error checking means we want an emergency exit from a program
- Can be done by the exit() function
- The exit() function, called from anywhere in your C program, will terminate the program at once

Usage of exit( )

```
FILE *fptr;
char filename[]= "file2.dat";
fptr = fopen (filename,"w");
if (fptr == NULL) {
    printf (“ERROR IN FILE CREATION”);
   /* Do something */
   exit(-1);
}
…….rest of the program……
```

Writing to a file: fprintf( )

- fprintf() works exactly like printf(), except that its first argument is a file pointer. The remaining two arguments are the same as printf
- The behaviour is exactly the same, except that the writing is done on the file instead of the display

```
FILE *fptr;
fptr = fopen ("file.dat","w");
fprintf (fptr, "Hello World!\n");
fprintf (fptr, “%d %d”, a, b);
```

Reading from a file: fscanf( )

- fscanf() works like scanf(), except that its first argument is a file pointer. The remaining two arguments are the same as scanf
- The behaviour is exactly the same, except
  - The reading is done from the file instead of from the keyboard (think as if you typed the same thing in the file as you would in the keyboard for a scanf with the same arguments)
  - The end-of-file for a text file is checked differently (check against special character EOF)

Reading from a file: fscanf( )

```
FILE *fptr;
fptr = fopen ("input.dat", "r");
/* Check it's open */
if (fptr == NULL)
  {
    printf("Error in opening file \n");
    exit(-1);
  }
fscanf (fptr, "%d %d",&x, &y);
```

EOF checking in a loop

```
char ch;
while (fscanf(fptr, "%c", &ch) != EOF)
{
   /* not end of file; read */
}
```

Reading lines from a file: fgets()

- Takes three parameters
  - a character array str, maximum number of characters to read size, and a file pointer fp
- Reads from the file fp into the array str until any one of these happens
  - No. of characters read = size - 1
  - \n is read (the char \n is added to str)
  - EOF is reached or an error occurs
- '\0' added at end of str if no error
- Returns NULL on error or EOF, otherwise returns pointer to str

Reading lines from a file: fgets()

```
FILE *fptr;
char line[1000];
/* Open file and check it is open */
while (fgets(line,1000,fptr) != NULL)
{
   printf ("Read line %s\n",line);
}
```

Writing lines to a file: fputs()

- Takes two parameters
  - A string str (null terminated) and a file pointer fp
- Writes the string pointed to by str into the file
- Returns non-negative integer on success, EOF on error

Reading/Writing a character: fgetc(), fputc()

- Equivalent of getchar(), putchar() for reading/writing char from/to keyboard
- Exactly same, except that the first parameter is a file pointer
- Equivalent to reading/writing a byte (the char)

```
int fgetc(FILE *fp);
int fputc(int c, FILE *fp);
```

- Example:

```
char c;
c = fgetc(fp1); fputc(c, fp2);
```

Formatted and Un-formatted I/O

- Formatted I/O
  - Using fprintf/fscanf
  - Can specify format strings to directly read as integers, float etc.
- Unformatted I/O
  - Using fgets/fputs/fgetc/fputc
  - No format string to read different data types
  - Need to read as characters and convert explicitly

Closing a file

- Should close a file when no more read/write to a file is needed in the rest of the program
- File is closed using fclose() and the file pointer

```
FILE *fptr;
char filename[]= "myfile.dat";
fptr = fopen (filename,"w");
fprintf (fptr,"Hello World of filing!\n");
…. Any more read/write to myfile.dat….
fclose (fptr);
```