11-7 A Programming Example—Morse Code

Morse code, patented by Samuel F. B. Morse in 1837, is the language that was used to send messages by telegraph from the middle of the nineteenth century until the advent of the modern telephone and today's computer controlled communications systems. In this section, we use a C program to convert English to Morse and Morse to English.

Letter	Code	Letter	Code	Letter	Code	Letter	Code
A	.-	H	....	O	---	V	...-
B	-...	I	..	P	.--.	W	.--
C	-.-.	J	.---	Q	--.-	X	-..-
D	-..	K	-.-	R	.-.	Y	-.--
E	.	L	.-..	S	...	Z	--..
F	..-.	M	--	T	-		
G	--.	N	-.	U	..-		

Table 11-3 Morse Code

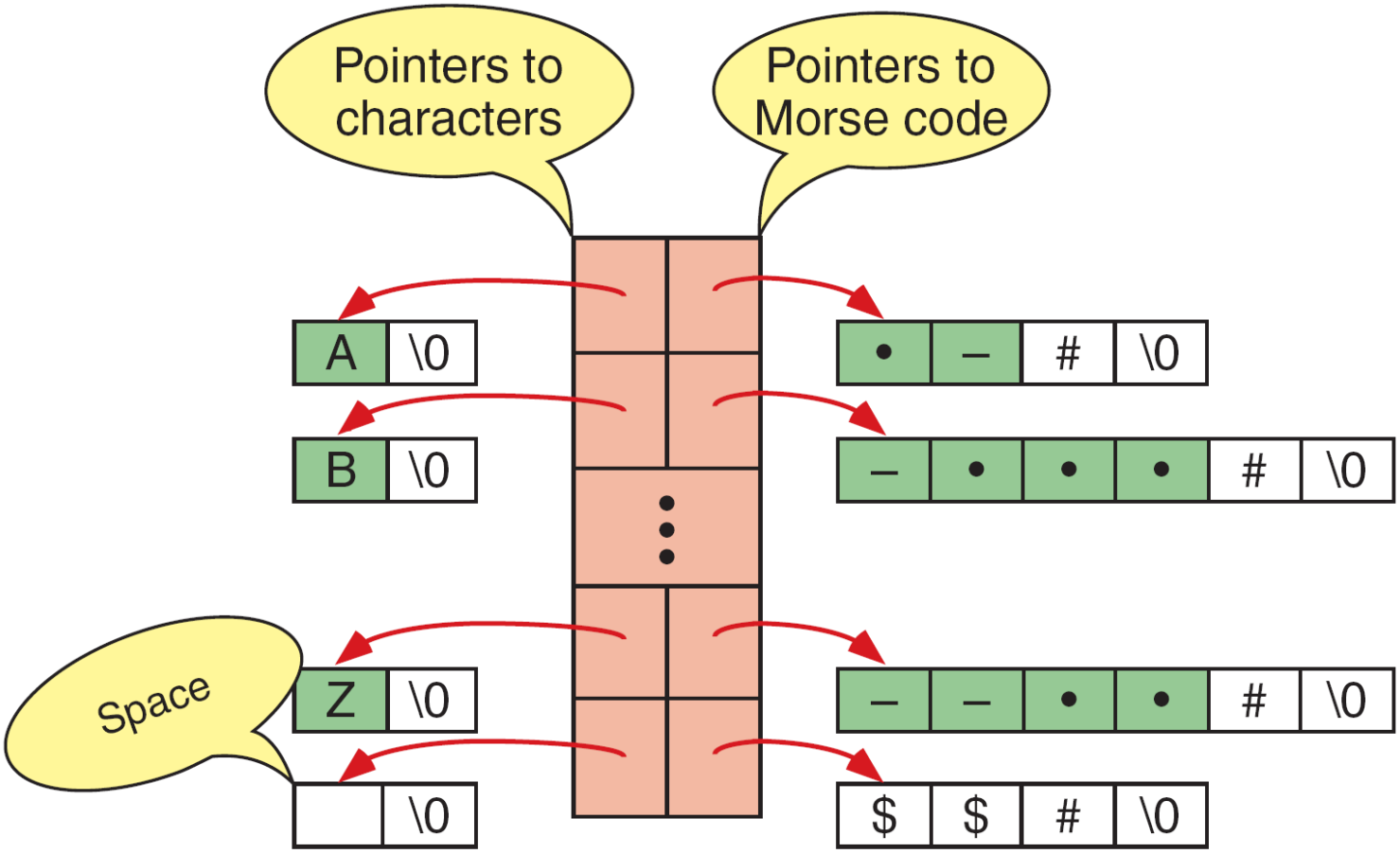

FIGURE 11-26 Character to Morse Code Structure

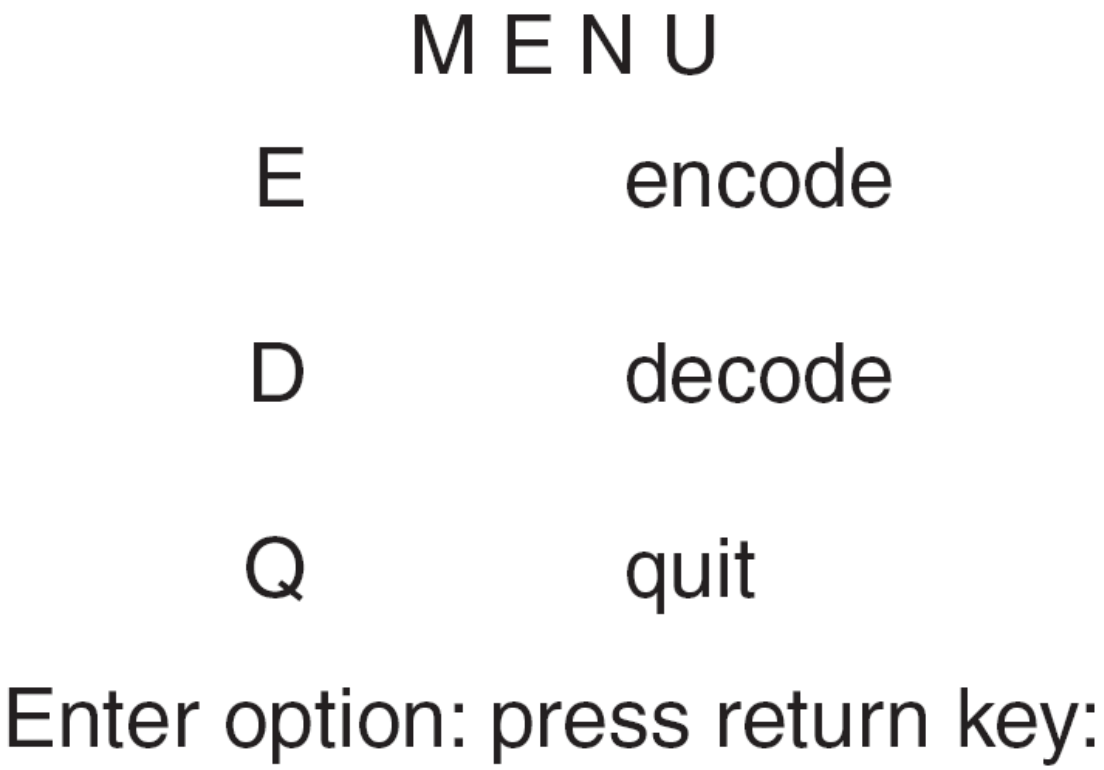

FIGURE 11-27 Morse Code Menu

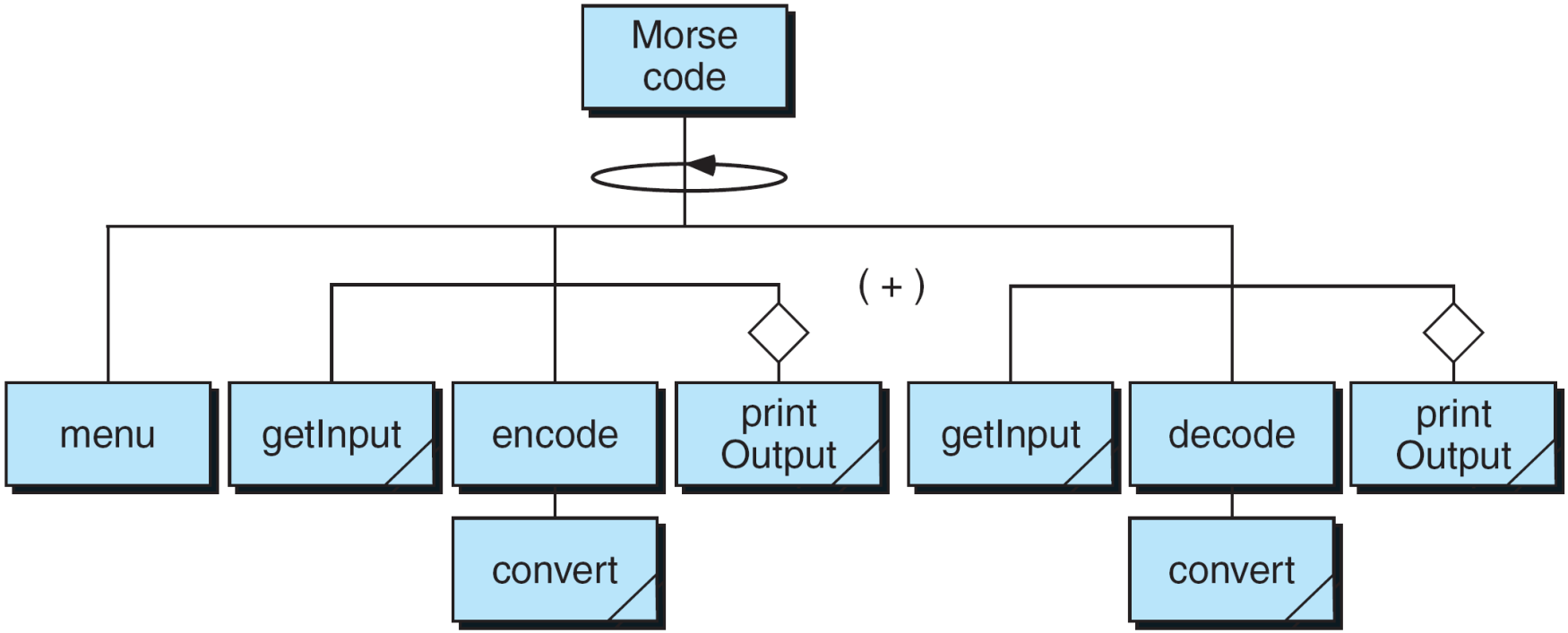

FIGURE 11-28 **Morse Code Program Design**

PROGRAM 11-18 Morse Code: *main*

```
 1  /* Convert English to Morse or Morse to English.
 2        Written by:
 3        Date Written:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <string.h>
 8  #include <ctype.h>
 9  #include <stdbool.h>
10
11  #define FLUSH while(getchar() != '\n')
12  #define STR_LEN 81
13
14  // Function Declarations
15  char menu          (void);
16  void getInput      (char* inStr);
17  void printOutput   (char* inStr, char* outSt);
18  bool encode        (char* (*encDec)[2],
19                      char* inStr,
20                      char* outStr);
```

PROGRAM 11-18 Morse Code: *main*

```
21  bool decode        (char* (*encDec)[2],
22                      char* inStr,
23                      char* outStr);
24  int  convert       (char* (*encDec)[2],
25                      char* s1,
26                      int   col,
27                      char* s2);
28
29  int main (void)
30  {
31  // Local Declarations
32     char* encDec [27][2] =
33         {
34          { "A", ".-#" },
35          { "B", "-...#" },
36          { "C", "-.-.#" },
37          { "D", "-..#" },
38          { "E", ".#" },
39          { "F", "..-.#" },
40          { "G", "--.#" },
```

PROGRAM 11-18 Morse Code: *main*

```
41          { "H", "....#" },
42          { "I", "..#" },
43          { "J", ".---#" },
44          { "K", "-.-#" },
45          { "L", ".-..#" },
46          { "M", "--#" },
47          { "N", "-.#" },
48          { "O", "---#" },
49          { "P", ".--.#" },
50          { "Q", "--.-#" },
51          { "R", ".-.#" },
52          { "S", "...#" },
53          { "T", "-#" },
54          { "U", "..-#" },
55          { "V", "...-#" },
56          { "W", ".--#" },
57          { "X", "-..-#" },
58          { "Y", "-.--#" },
59          { "Z", "--..#" },
60          { " ", "$$#" },
```

PROGRAM 11-18 Morse Code: *main*

```
61          }; // Encode / Decode array
62      char inStr  [STR_LEN];
63      char outStr [STR_LEN];
64      char option;
65      bool done = false;
66
67  // Statements
68      while (!done)
69          {
70           option = menu ();
71           switch (option)
72              {
73               case 'E' :
74                       getInput (inStr);
75                       if (!encode (encDec, inStr, outStr))
76                          {
77                           printf("Error! Try again");
78                           break;
79                          } // if
80                       printOutput (inStr, outStr);
```

PROGRAM 11-18 Morse Code: *main*

```
61          }; // Encode / Decode array
62      char inStr  [STR_LEN];
63      char outStr [STR_LEN];
64      char option;
65      bool done = false;
66
67  // Statements
68      while (!done)
69          {
70           option = menu ();
71           switch (option)
72              {
73               case 'E' :
74                       getInput (inStr);
75                       if (!encode (encDec, inStr, outStr))
76                          {
77                           printf("Error! Try again");
78                           break;
79                          } // if
80                       printOutput (inStr, outStr);
```

PROGRAM 11-18 Morse Code: *main*

```
81                      break;
82              case 'D' : getInput (inStr);
83                      if (!decode (encDec, inStr, outStr))
84                         {
85                          printf("Error! Try again");
86                          break;
87                         } // if
88                      printOutput (inStr, outStr);
89                      break;
90              default :
91                      done = true;
92                      printf("\nEnd of Morse Code.\n");
93                      break;
94            } // switch
95        } // while
96    return 0;
97  }  // main
```

PROGRAM 11-19 Morse Code: Menu

```
1  /* ===================== menu =====================
2     Display menu of choices; return selected character.
3        Pre   nothing
4        Post  returns validated option code
5  */
6  char menu (void)
7  {
8  // Local Declarations
9     char  option;
10    bool  validData;
11
12 // Statements
13    printf("\t\t\tM E N U \n");
14    printf("\t\tE)  encode \n");
15    printf("\t\tD)  decode \n");
16    printf("\t\tQ)  quit \n");
17
```

PROGRAM 11-19 Morse Code: Menu

```
18      do
19          {
20           printf ("\nEnter option: press return key: ");
21           option = toupper (getchar());
22           FLUSH;
23           if (option == 'E' || option == 'D' ||
24               option == 'Q')
25              validData = true;
26           else
27              {
28               validData = false;
29               printf("\aEnter only one option\n");
30               printf(" \tE, D, or Q\n ");
31              } // else
32          } while (!validData);
33      return option;
34  }   // menu
```

PROGRAM 11-20 Morse Code: Get Input

```
1  /* ===================== getInput ====================
2     Reads input string to be encoded or decoded.
3        Pre   inStr is a pointer to the input area
4        Post  string read into input area
5  */
6  void getInput (char* inStr)
7  {
8  // Statements
9     printf ("\nEnter line of text to be coded: \n");
10    fgets  (inStr, STR_LEN, stdin);
11
12    // Eliminate newline in input string
13    *(inStr-1 + strlen(inStr)) = '\0';
14
15    if (isalpha(*inStr) && strlen(inStr) > 16)
16       {
17        // Exceeds English input length
18        printf("\n***WARNING: Input length exceeded: ");
```

PROGRAM 11-20 Morse Code: Get Input

```
19          printf("Only 16 chars will be encoded.\a\a\n");
20          *(inStr + 16) = '\0';
21        } // if
22    return;
23  } // getInput
```

PROGRAM 11-21 Morse Code: Print Output

```
1  /* ================== printOutput ==================
2     Print the input and the transformed output
3        Pre   inStr contains the input data
4              outStr contains the transformed string
5        Post  output printed
6  */
7  void printOutput (char* inStr, char* outStr)
8  {
9  // Statements
10    printf("\nThe information entered was: \n");
11    puts(inStr);
12    printf("\nThe transformed information is: \n");
13    puts(outStr);
14    return;
15 }  // printOutput
```

PROGRAM 11-22 Morse Code: Encode to Morse

```
1  /* ====================== encode ======================
2     Transforms character data to Morse code
3        Pre     encDec is the conversion table
4                inStr contains data to be put into Morse
5        Post    data have been encoded in outStr
6        Return  true  if all valid characters;
7                false if invalid character found
8  */
9  bool encode (char* (*encDec)[2],
10              char* inStr, char* outStr)
11 {
12 // Local Declarations
13    char s1[2];
14    char s2[6];
15    int  error = 0;
16
17 // Statements
18    outStr[0] = '\0';
```

PROGRAM 11-22 Morse Code: Encode to Morse

```
19      while (*inStr != '\0' && !error)
20          {
21           s1[0] = toupper(*inStr);
22           s1[1] = '\0';
23           error = !convert (encDec, s1, 0, s2);
24           strcat (outStr, s2);
25           inStr++;
26          } // while
27      return (!error);
28  }   // encode
```

PROGRAM 11-23 Morse code: Decode to English

```
1  /* ====================== decode ======================
2     Transforms Morse code data to character string
3        Pre    encDec is the conversion table
4               inStr contains data to transform to string
5        Post   data encoded and placed in outStr
6        Return true  if all valid characters;
7               false if invalid character found
8  */
9  bool decode (char* (*encDec)[2],
10              char* inStr, char* outStr)
11 {
12 // Local Declarations
13    char  s1[6];
14    char  s2[2];
15    bool  error = false;
16    int   i;
17
```

PROGRAM 11-23 Morse code: Decode to English

```
18  // Statements
19     outStr[0] ='\0';
20     while (*inStr != '\0' && !error)
21        {
22         for (i = 0; i < 5 && *inStr != '#'; i++, inStr++)
23              s1[i] = *inStr;
24
25         s1[i]   = *inStr;
26         s1[++i] = '\0';
27
28         error = !convert (encDec, s1, 1, s2);
29         strcat (outStr, s2);
30         inStr++;
31        } // while
32     return (!error);
33  } // decode
```

PROGRAM 11-24 Morse Code: Convert Codes

```
1  /* ==================== convert ====================
2     Looks up code and converts to opposite format
3        Pre   encDec is a pointer decoding table
4              s1 is string being converted
5              s2 is output string
6               col is code: 0 for character to Morse
7                            1 for Morse to character
8        Post  converted output s2
9  */
10 int convert (char* (*encDec)[2],
11              char* s1, int col, char* s2)
12 {
13 // Local Declarations
14    bool found = false;
15    int  i;
16
```

PROGRAM 11-24 Morse Code: Convert Codes

```
17  // Statements
18     for (i = 0; i < 27 && !found; i++)
19        found = !strcmp(s1, encDec[i][col]);
20
21     if (found)
22        strcpy (s2, encDec [i - 1][(col + 1) % 2]);
23     else
24        *s2 = '\0';
25
26     return found;
27  }  // convert
28  // ================ End of Program ================
```

11-8 Software Engineering

In this section, we've formalized some of the principles of good programming that we've discussed throughout the text. Although you will find little in this discussion of software engineering that relates directly to the subject of strings, all of the string functions have been written using the principles discussed on the following pages.

Topics discussed in this section:

Program Design Concepts
Information Hiding
Cohesion

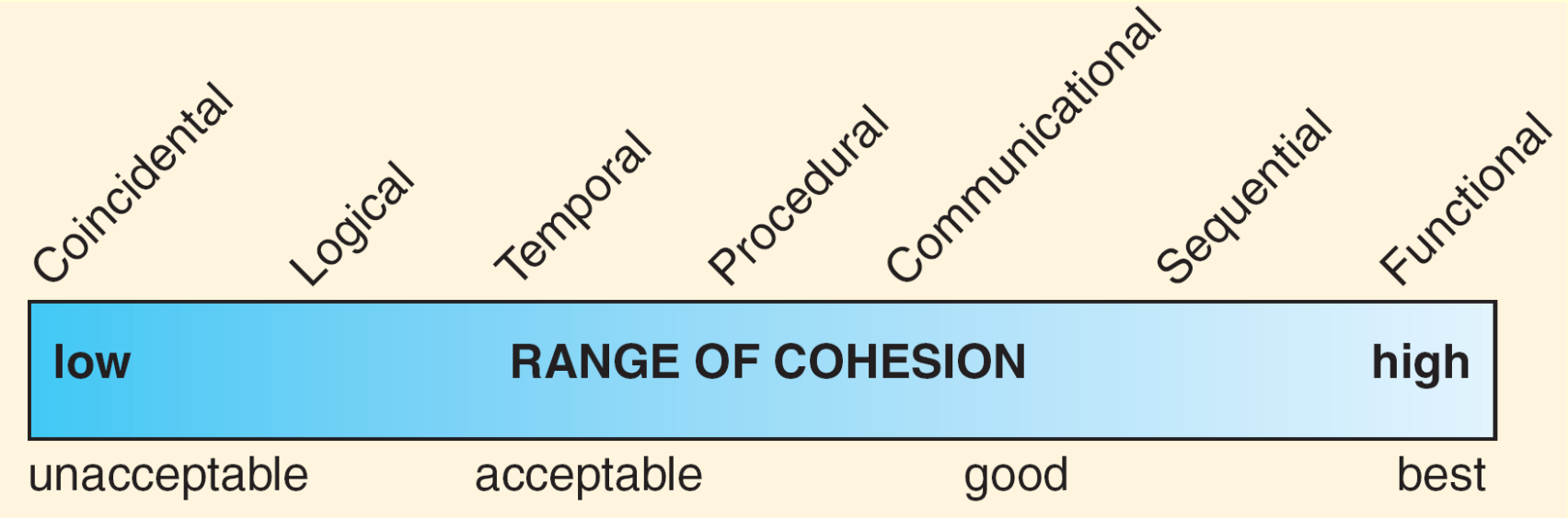

FIGURE 11-29 **Types of Cohesion**

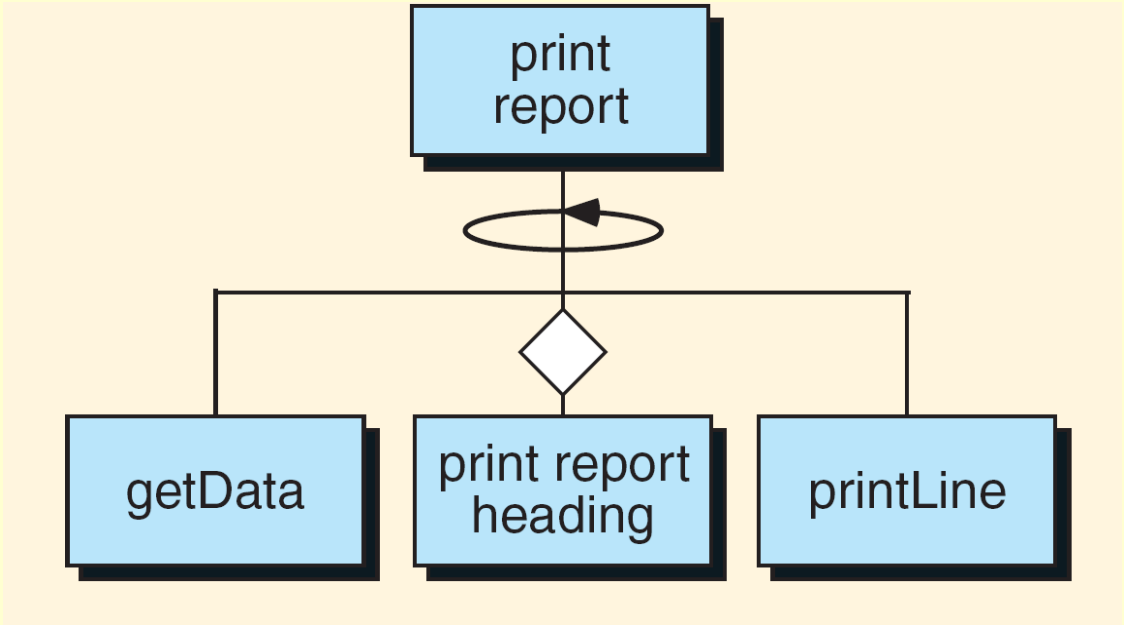

FIGURE 11-30 Example of Functional Cohesion

ALGORITHM 11-1 Process Inventory Pseudocode

```
Algorithm Process Inventory
 1 while not end of file
    1  read a record
    2  print report
    3  check reorder point
 2 end while
end Process Inventory
```

Note

Well-structured programs are highly cohesive and loosely coupled.

ALGORITHM 11-2 Process List Pseudocode

```
Algorithm Process List
 1 open files
 2 initialize work areas
 3 create list
 4 print menu
 5 while not stop
    1   get users response
    2   if locate …
    3   if insert …
    4   if delete …
    5   print menu
 6 end while
 7 clean up
 8 close files
end Process List
```