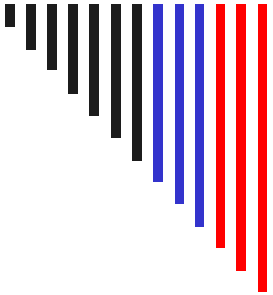

Chapter 11 Strings

Objectives

- To understand design concepts for fixed-length and variable-length strings
- To understand the design implementation for C-language delimited strings
- To write programs that read, write, and manipulate strings
- To write programs that use the string functions
- To write programs that use arrays of strings
- To write programs that parse a string into separate variables
- To understand the software engineering concepts of information hiding and cohesion

11-1 String Concepts

In general, a string is a series of characters treated as a unit. Computer science has long recognized the importance of strings, but it has not adapted a standard for their implementation. We find, therefore, that a string created in Pascal differs from a string created in C.

Topics discussed in this section:

Fixed-Length Strings
Variable-Length Strings

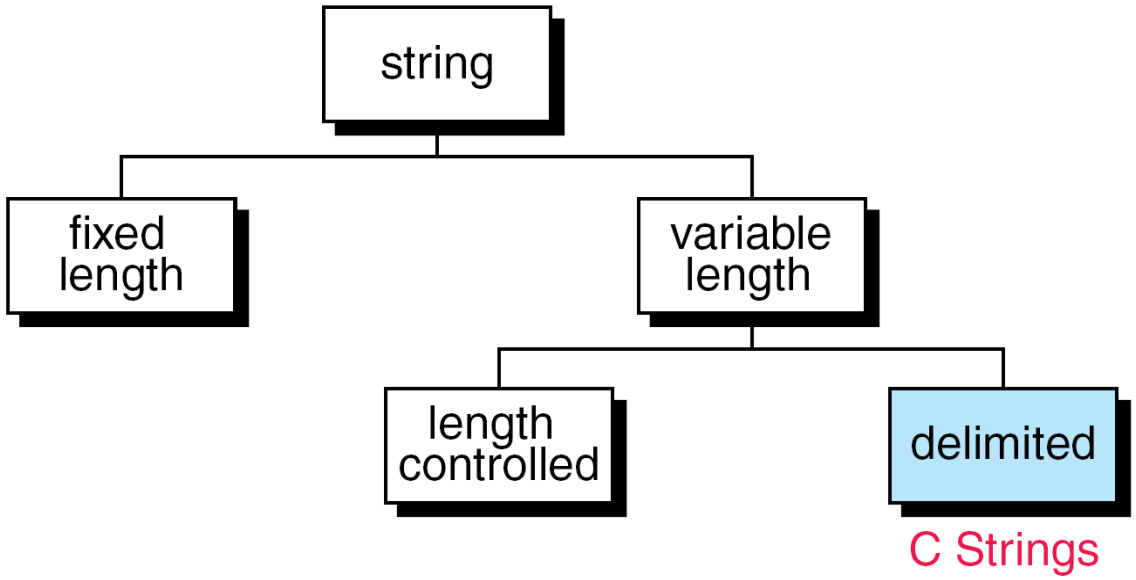

FIGURE 11-1 String Taxonomy

FIGURE 11-2 String Formats

11-2 C Strings

A C string is a variable-length array of characters that is delimited by the null character.

Topics discussed in this section:

Storing Strings
The String Delimiter
String Literals
Strings and Characters
Declaring Strings
Initializing Strings
Strings and the Assignment Operator
Reading and Writing Strings

Note

C uses variable-length, delimited strings.

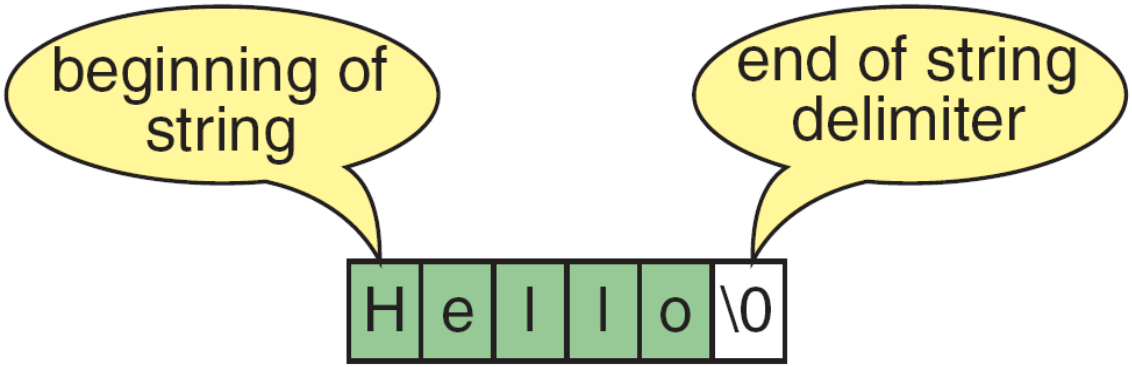

FIGURE 11-3 Storing Strings

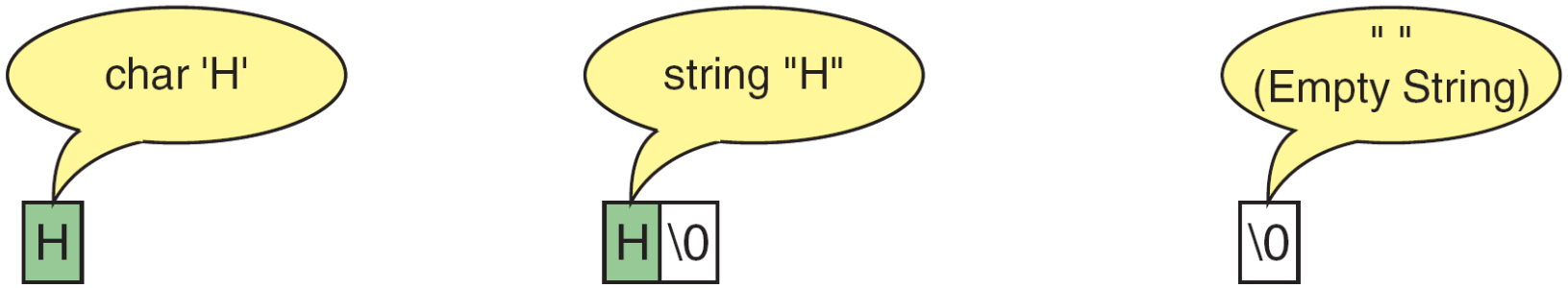

FIGURE 11-4 Storing Strings and Characters

FIGURE 11-5 Differences Between Strings and Character Arrays

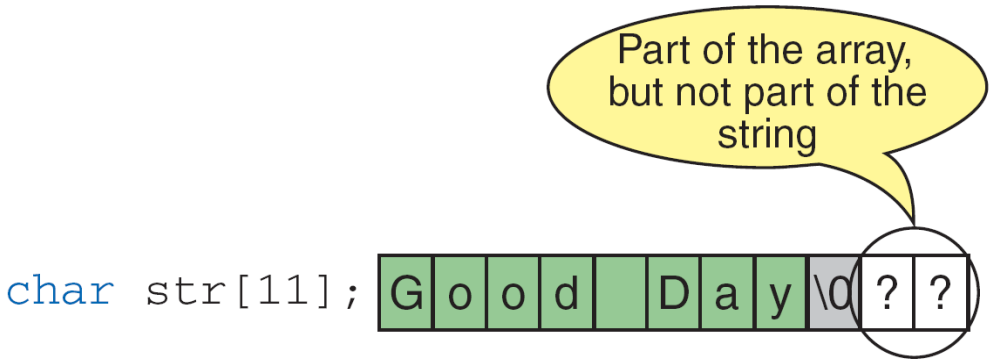

FIGURE 11-6 Strings in Arrays

Note

A string literal is enclosed in double quotes.

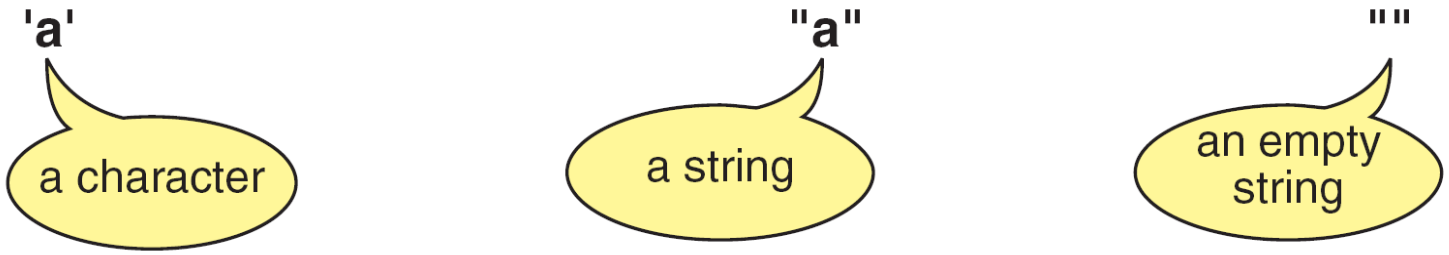

FIGURE 11-7 Character Literals and String Literals

```c
#include <stdio.h>
int main (void)
{
  printf("%c\n", "Hello"[1];
  return 0;
} // main
```

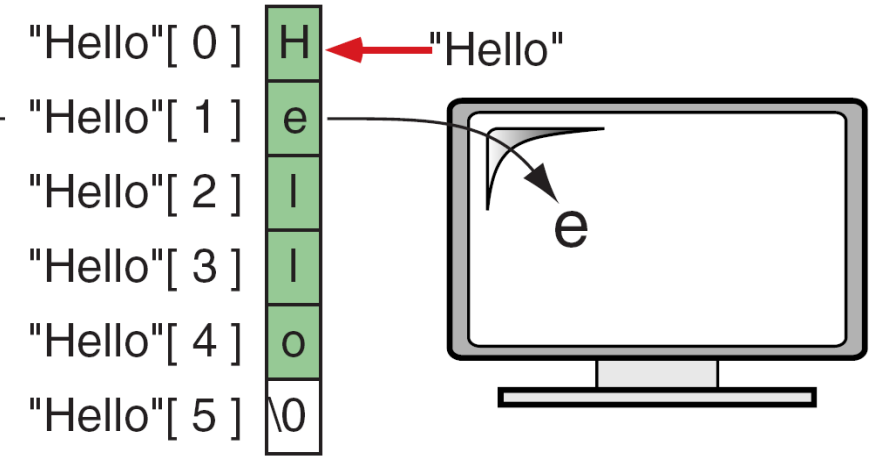

FIGURE 11-8 String Literal References

```
// Local Declarations
   char str[9];
```

(a) String Declaration

```
// Local Declarations
   char* pStr;
```

(b) String Pointer Declaration

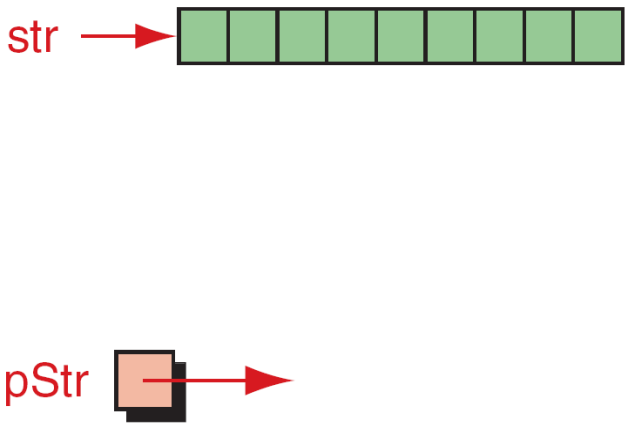

FIGURE 11-9 Defining Strings

Note

Memory for strings must be allocated before the string can be used.

Initializing Strings

- char str[9] = “Good Day”;
- char month[] = “January”;
- char *pStr = “Good Day”;
- char str[9] = {‘G’, ‘o’, ‘o’, ‘d’, ‘ ‘, ‘D’, ‘a’, ‘y’, ‘\ 0’};

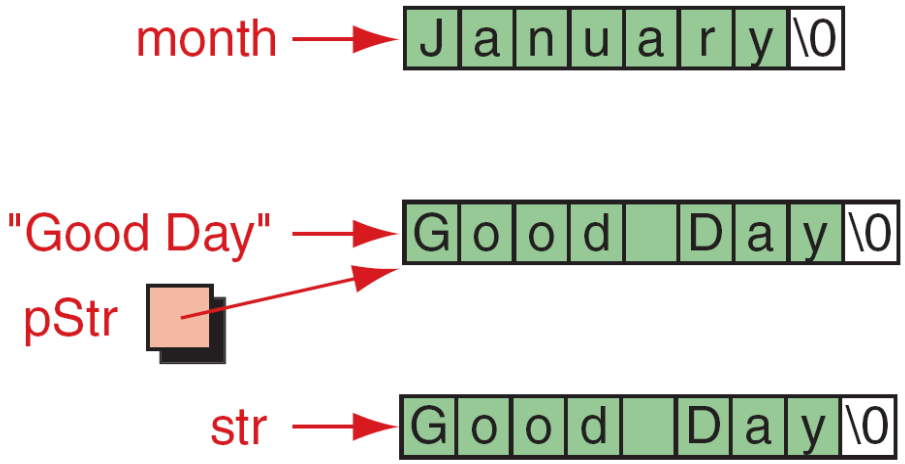

FIGURE 11-10 Initializing Strings

Assignments of Strings

- char str1[6] = “Hello”;
  char str2[6];
  str2 = str1; // ?

11-3 String Input/Output Functions

C provides two basic ways to read and write strings. First, we can read and write strings with the formatted input/output functions, *scanf/fscanf* ***and*** *printf/fprintf.* ***Second, we can use a special set of string-only functions, get string (****gets/fgets****) and put string (*** *puts/fputs* ***).***

Topics discussed in this section:

Formatted String Input/Output

String Input/Output

Note

The string conversion code(s) skips whitespace.

PROGRAM 11-1 Reading Strings

```
1  {  // Read Month
2     #define FLUSH while (getchar() != '\n')
3     char month[10];
4
5     printf("Please enter a month. ");
6     scanf("%9s", month);
7     FLUSH;
8  }  // Read Month
```

Note

Always use a width in the field specification when reading strings.

```
char month[10];
scanf("%9s", month);
```

Note

The maximum number of characters to be printed is specified by the precision in the format string of the field specification.

```
printf("|%-15.14s|", "123456789012345678901");
```

Output

```
|12345678901234|
```

Note

The edit set does not skip whitespace.

scanf(“%10[0123456789.,-$]”, month);

PROGRAM 11-2 Demonstrate String Scan Set

```
 1  /* Read only second integer.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  int main (void)
 9  {
10  // Local Declarations
11     int   amount;
12     FILE* spData;
13
14  // Statements
15     if (!(spData = fopen ("P11-03.TXT", "r")))
16        {
17         printf("\aCould not open input file.\n");
18         exit (100);
19        } // if
```

PROGRAM 11-2 Demonstrate String Scan Set

```
20      // Read and print only second integer
21      while (fscanf(spData,
22             " %*d%d%*[^\n]", &amount) != EOF)
23          printf("Second integer: %4d\n", amount);
24
25      printf("End of program\n");
26      fclose (spData);
27      return 0;
28  }   // main
```

```
Input:
123 456 7.89
987 654 3.21

Results:
Second integer:  456
Second integer:  654
End of program
```

PROGRAM 11-3 Delete Leading Whitespace

```
1  /* Delete leading spaces at beginning of line.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <ctype.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    char line[80];
12
13 // Statements
14    printf("Enter data:  ");
15    while ((fscanf(stdin, "%*[\t\v\f ]%79[^\n]", line))
16                          != EOF)
17       {
18        printf("You entered: %s\n", line);
19
```

PROGRAM 11-3 Delete Leading Whitespace

```
20          // Discard newline and set line to null string
21          fgetc (stdin);
22          *(line) = '\0';
23          printf("Enter data:  ");
24        } // while
25
26     printf("\nThank you\n");
27     return 0;
28  }  // main
```

```
Results:
Enter data:  No whitespace here.
You entered: No whitespace here.
Enter data:   Only one whitespace character.
You entered: Only one whitespace character.
Enter data:         Tabs and spaces here.
You entered: Tabs and spaces here.
Enter data:  Next line is only one space.
You entered: Next line is only one space.
Enter data:
You entered:
Enter data:  ^d
Thank you
```

PROGRAM 11-4 Read Student Names and Scores

```
 1  /* Demonstrate reading names from a file.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <string.h>
 8
 9  int main (void)
10  {
11  // Local Declarations
12     char  first[80];
13     char  last[80];
14     int   score;
15     FILE* spStuScores;
16
17  // Statements
18     if (!(spStuScores = fopen ("P11-04.TXT", "r")))
19        {
```

PROGRAM 11-4 Read Student Names and Scores

```
20          printf("\aCould not open student file.\n");
21          exit (100);
22        } // if
23
24     // Read and print first name, last name, and score
25     while (fscanf(spStuScores, " %s %s %d",
26              first, last, &score) == 3)
27        printf("%s %s %3d\n", first, last, score);
28
29     printf("End of Student List\n");
30     fclose (spStuScores);
31     return 0;
32  } // main
```

```
Results:
George Washington  95
Benedict Arnold  53
Mary Todd-Lincoln  91
End of Student List
```

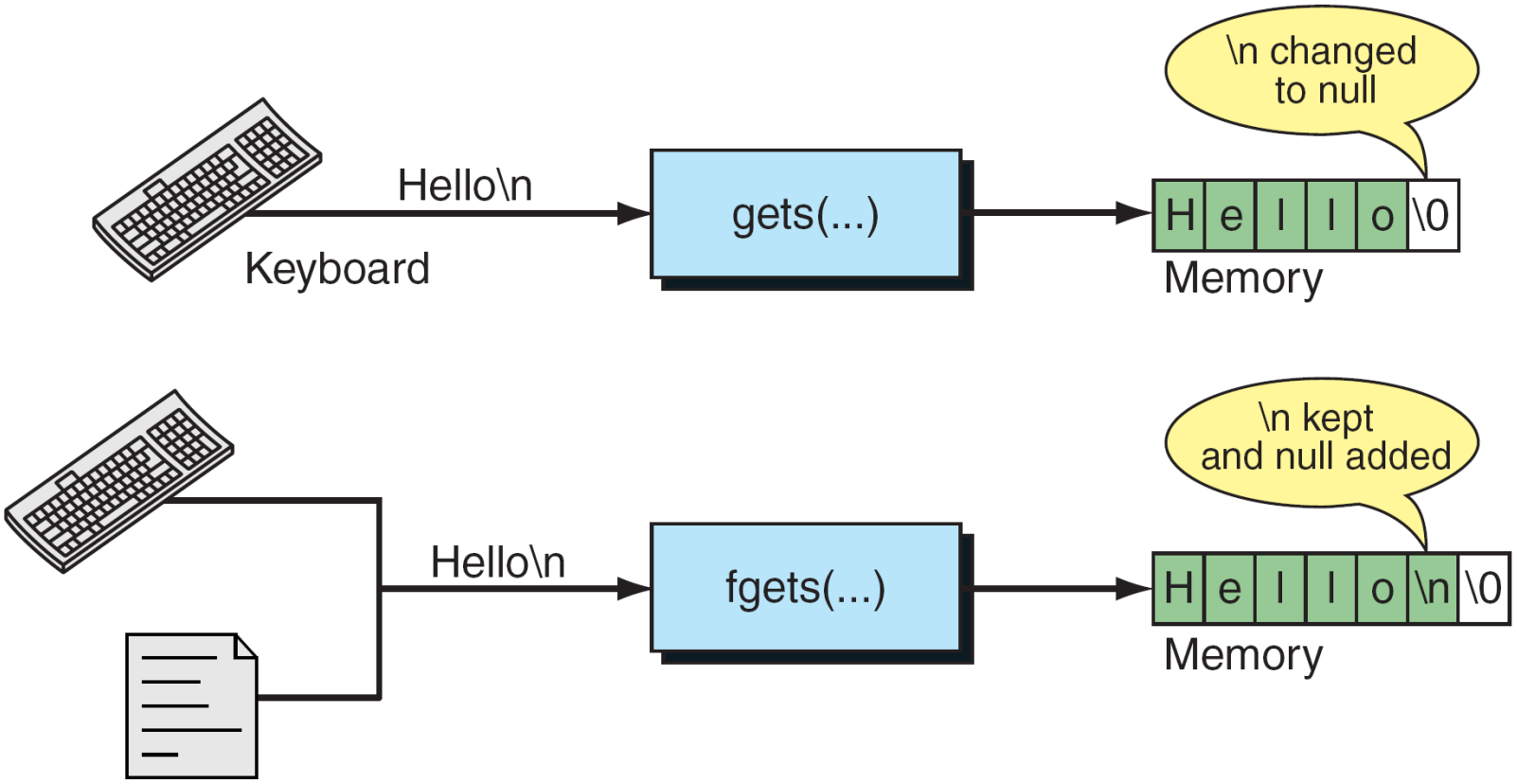

FIGURE 11-11 ***gets*** **and** ***fgets*** **Functions**

PROGRAM 11-5 Demonstrate *fgets* Operation

```
 1  /* Demonstrate the use of fgets in a program
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     char str[81];
11
12  // Statements
13     printf("Please enter a string: ");
14     fgets (str, sizeof (str), stdin);
15     printf("Here is your string: \n\t%s", str);
16     return 0;
17  }  // main
```

PROGRAM 11-5 Demonstrate *fgets* Operation

```
Results:
Please enter a string: Now is the time for all students
Here is your string:
    Now is the time for all students
```

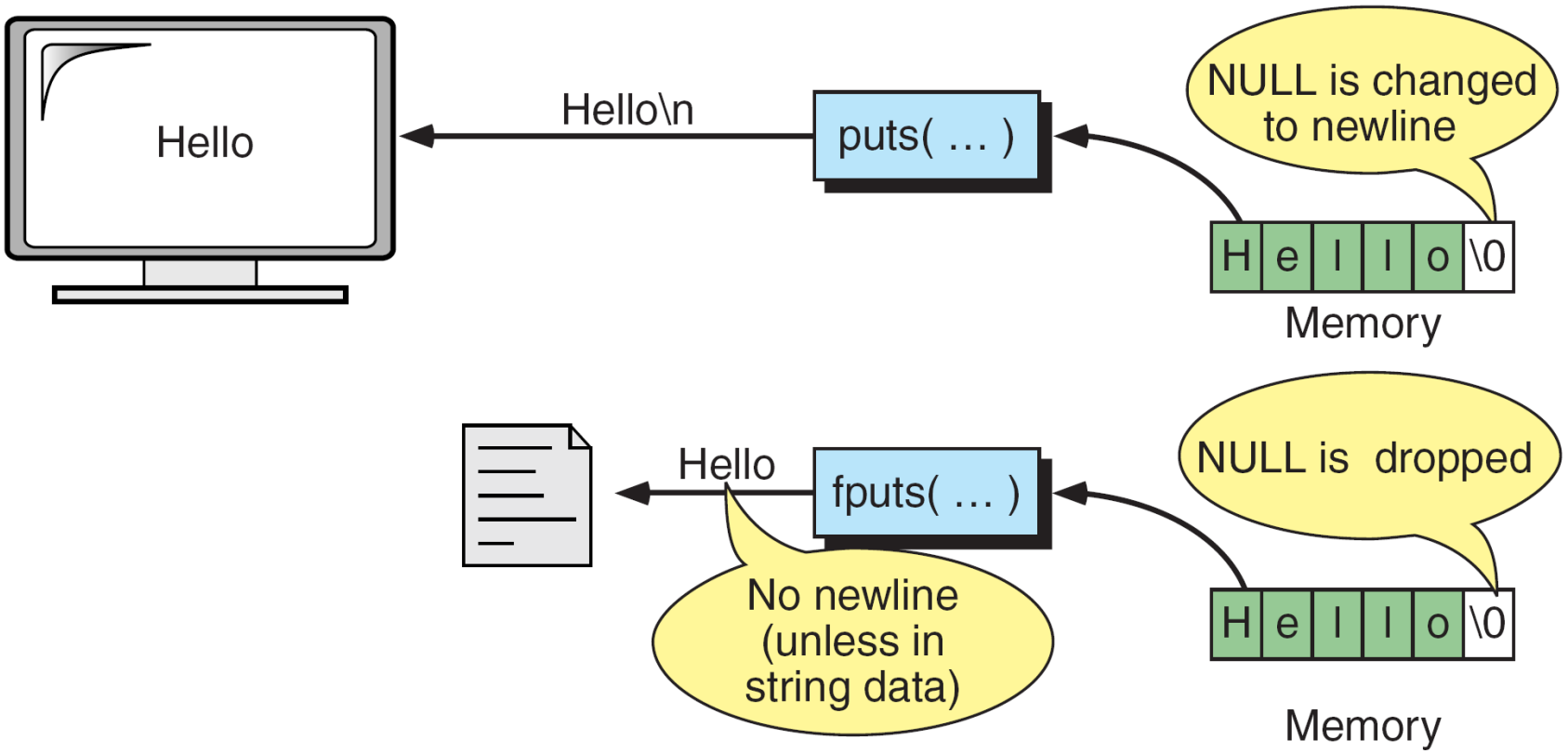

FIGURE 11-12 ***puts*** **and** ***fputs*** **Operations**

PROGRAM 11-6 Demonstration of Put String

```
1  /* Demonstrate fput string
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9  // Local Definitions
10    char str[] = "Necessity Is the Mother of Invention.";
11    char* pStr = str;
12
13 // Statements
14    fputs(pStr, stdout);
15    fputs("\n", stdout);
16    fputs(pStr + 13, stdout);
17    return 0;
18 }  // main
```

PROGRAM 11-6 Demonstration of Put String

```
Results:
Necessity Is the Mother of Invention
the Mother of Invention.
```

PROGRAM 11-7 Typewriter Program

```
1  /* This program creates a text file from the keyboard.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    char  str[100];
12    FILE* spOut;
13
14 // Statements
15    if (!(spOut = fopen ("P11-07.TXT", "w")))
16       {
17        printf("\aCould not open output file.\n");
18        exit (100);
19       } // if
```

PROGRAM 11-7 Typewriter Program

```
20      while (fgets(str, sizeof (str), stdin))
21          fputs(str,  spOut);
22      fclose (spOut);
23      return 0;
24  }   // main
```

PROGRAM 11-8 Print Selected Sentences

```
1  /* Echo keyboard input that begins with capital letter.
2        Written by:
3        Date written:
4  */
5  #include <ctype.h>
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    char strng[81];
12
13 // Statements
14    while (fgets (strng, sizeof(strng), stdin))
15        if (isupper(*strng))
16            fputs(strng, stdout);
17    return 0;
18 }  // main
```

PROGRAM 11-8 Print Selected Sentences

```
Results:
Now is the time
Now is the time
for all good students
to come to the aid
of their school.
Amen
Amen
```

PROGRAM 11-9 Print File Double spaced

```
 1  /* Write file double spaced.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7
 8  int main (void)
 9  {
10  // Local Declarations
11     char  strng[81];
12     FILE* textIn;
13
14  // Statements
15     if (!(textIn = fopen("P11-07.TXT", "r")))
16        {
17         printf("\aCan't open textdata\n");
18         exit (100);
19        } // if
```

PROGRAM 11-9 Print File Double spaced

```
20      while (fgets(strng, sizeof(strng), textIn))
21          {
22           fputs(strng, stdout);
23           putchar ('\n');
24          } // while
25      return 0;
26  }   // main
```

11-4 Arrays of Strings

When we discussed arrays of pointers in Chapter 10, we introduced the concept of a ragged array. Ragged arrays are very common with strings. Consider, for example, the need to store the days of the week in their textual format. We could create a two-dimensional array of seven days by ten characters, but this wastes space.

PROGRAM 11-10 Print Days of the Week

```
1  /* Demonstrates an array of pointers to strings.
2        Written by:
3        Date written:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9  // Local Declarations
10    char*  pDays[7];
11    char** pLast;
12
13 // Statements
14    pDays[0] = "Sunday";
15    pDays[1] = "Monday";
16    pDays[2] = "Tuesday";
17    pDays[3] = "Wednesday";
18    pDays[4] = "Thursday";
```

PROGRAM 11-10 Print Days of the Week

```
19      pDays[5] = "Friday";
20      pDays[6] = "Saturday";
21
22      printf("The days of the week\n");
23      pLast = pDays + 6;
24      for (char** pWalker = pDays;
25                  pWalker <= pLast;
26                  pWalker++)
27           printf("%s\n", *pWalker);
28      return 0;
29  } // main
```

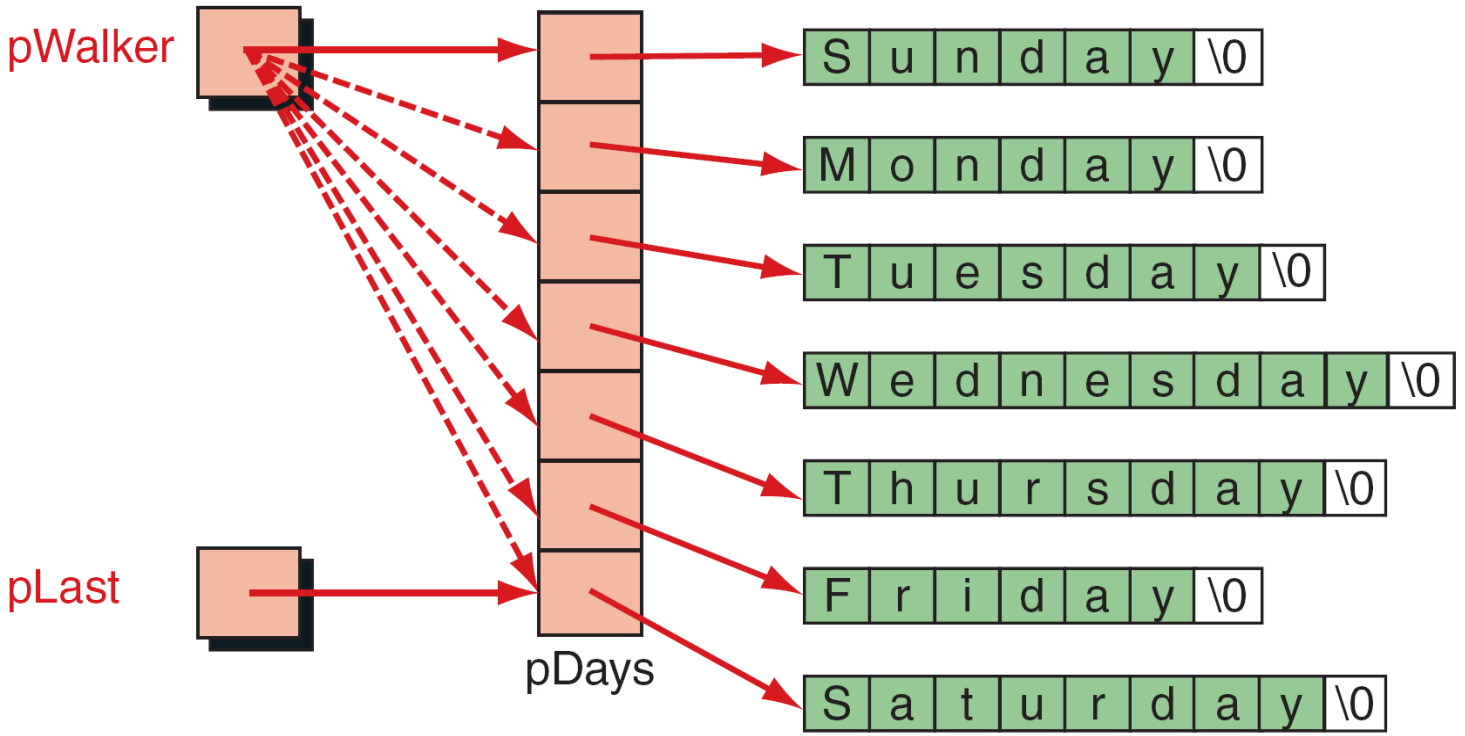

FIGURE 11-13 Pointers to Strings

11-5 String Manipulation Functions

Because a string is not a standard type, we cannot use it directly with most C operators. Fortunately, C provides a set of functions to manipulates strings.

Topics discussed in this section:

String Length and String Copy
String Compare and String Concatenate
Character in String
Search for a Substring and Search for Character in Set
String Span and String Token
String to Number

PROGRAM 11-11 Add Left Margin

```
1  /* Typewriter program: adds two spaces to the left
2      margin and writes line to file
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9
10 int main (void)
11 {
12 // Local Declarations
13    FILE* spOutFile;
14    char  strng[81];
15
16 // Statements
17    if (!(spOutFile = fopen("P11-11.TXT", "w")))
```

PROGRAM 11-11 Add Left Margin

```
18          {
19           printf("\aCould not open output file.\n");
20           exit (100);
21          } // if
22
23      while (fgets(strng, sizeof(strng), stdin))
24          {
25           fputc(' ', spOutFile);
26           fputc(' ', spOutFile);
27           fputs(strng, spOutFile);
28           if (strng[strlen(strng) - 1] != '\n')
29               fputs("\n", spOutFile);
30          } // while
31      fclose (spOutFile);
32      return 0;
33  }  // main
```

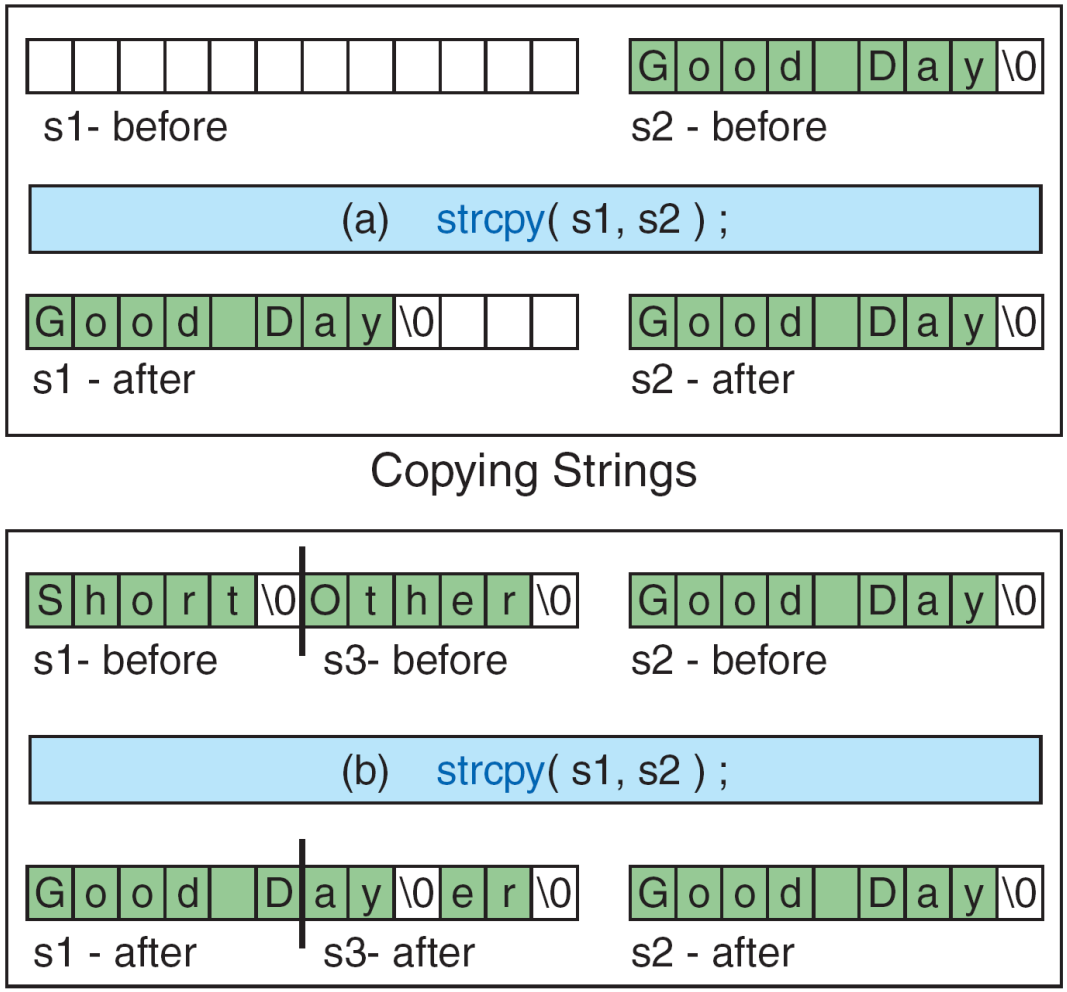

FIGURE 11-14 String Copy

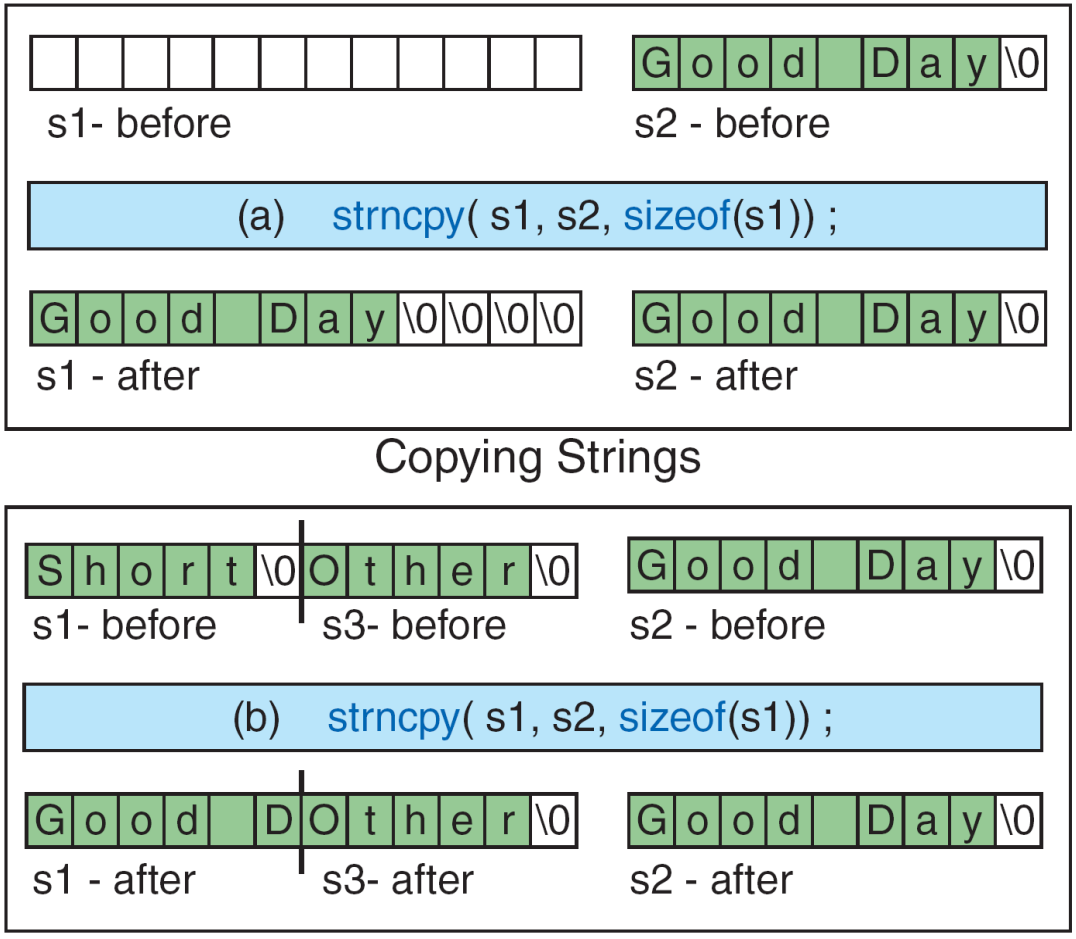

FIGURE 11-15 String-number Copy

Note

Always use strncpy to copy one string to another.

PROGRAM 11-12 Build Name Array in Heap

```
1  /* Build a dynamic array of names.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  #define FLUSH while (getchar() != '\n')
10
11 int main (void)
12 {
13 // Local Declarations
14    char   input[81];
15    char** pNames;          // array of pointers to char
16
17    int size;
18    int namesIndex;
19
```

PROGRAM 11-12 Build Name Array in Heap

```
20  // Statements
21     printf("How many names do you plan to input? ");
22     scanf ("%d", &size);
23     FLUSH;
24
25     // Allocate array in heap.
26     // One extra element added for loop control
27     pNames = calloc (size + 1, sizeof (char*));
28     printf("Enter names:\n");
29
30     namesIndex = 0;
31     while (namesIndex < size
32         && fgets(input, sizeof(input), stdin))
33         {
34          *(pNames + namesIndex)  = (char*)
35               calloc (strlen(input) + 1, sizeof(char));
36          strcpy (*(pNames + namesIndex), input);
37          namesIndex++;
38        } // while
```

PROGRAM 11-12 Build Name Array in Heap

```
39
40      *(pNames + namesIndex) = NULL;
41      printf("\nYour names are: \n");
42      namesIndex = 0;
43      while (*(pNames + namesIndex))
44          {
45           printf("%3d: %s",
46                   namesIndex, *(pNames + namesIndex));
47           namesIndex++;
48          } // while
49      return 0;
50  }   // main
```

```
Results:
How many names do you plan to input? 3
Enter names:
Tom
Rico
Huang

Your names are:
  0: Tom
  1: Rico
  2: Huang
```

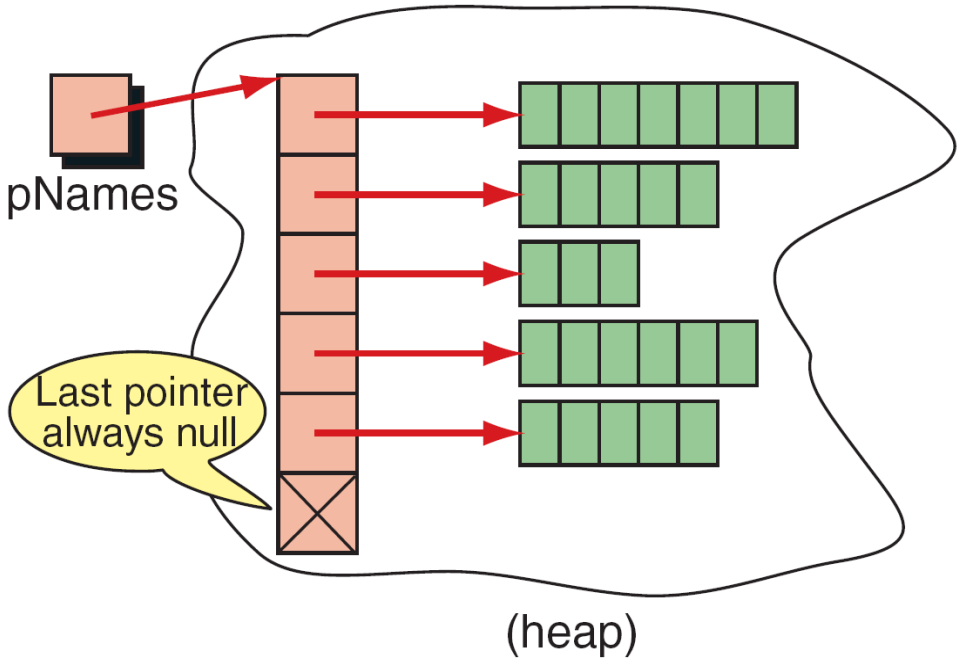

FIGURE 11-16 Structure for Names Array

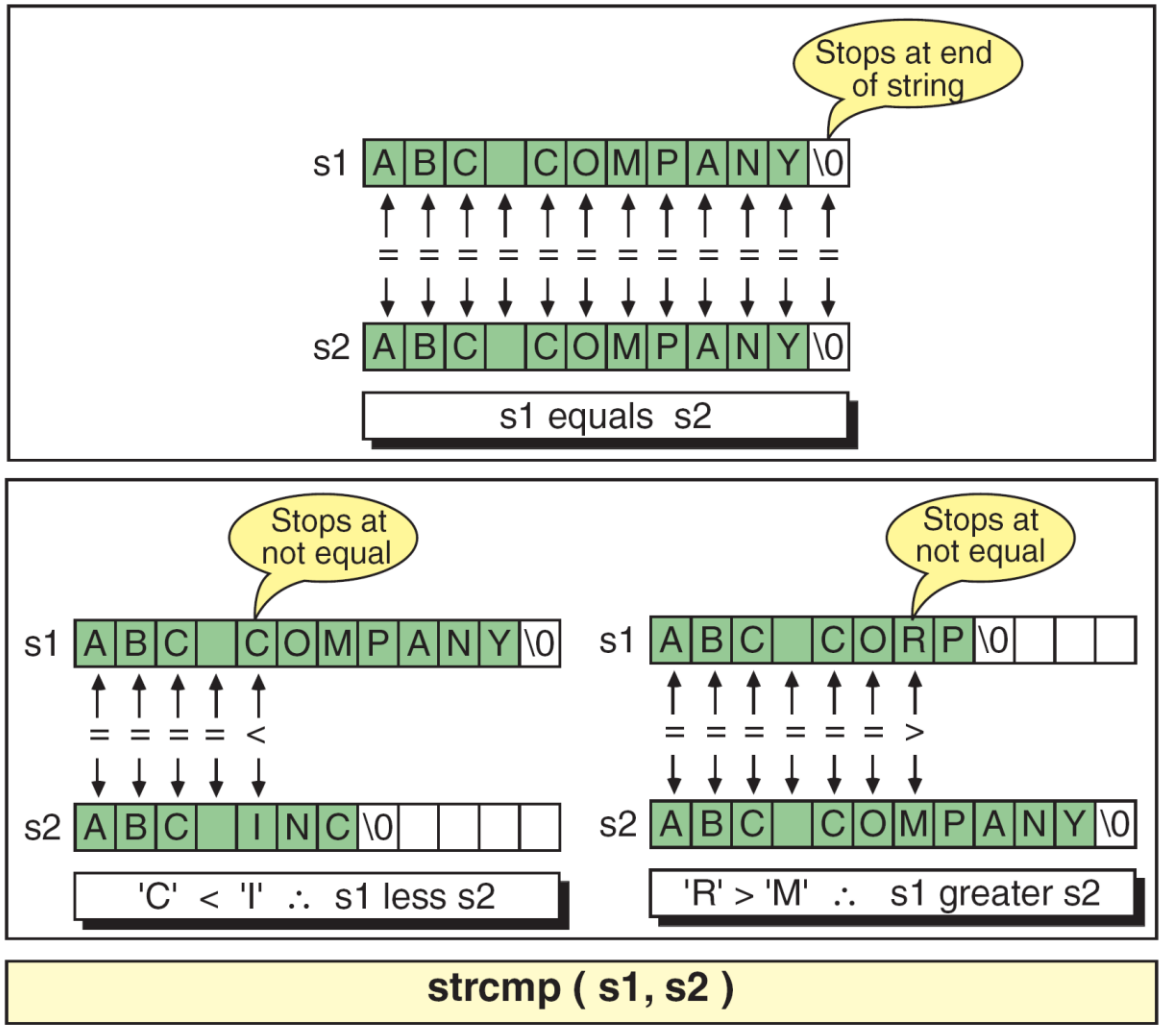

FIGURE 11-17 String Compares

string1	string2	Size	Results	Returns
"ABC123"	"ABC123"	8	equal	0
"ABC123"	"ABC456"	3	equal	0
"ABC123"	"ABC456"	4	string1 < string2	< 0
"ABC123"	"ABC"	3	equal	0
"ABC123"	"ABC"	4	string1 > string2	> 0
"ABC"	"ABC123"	3	equal	0
"ABC123"	"123ABC"	–1	equal	0

Table 11-1 Results for String Compare

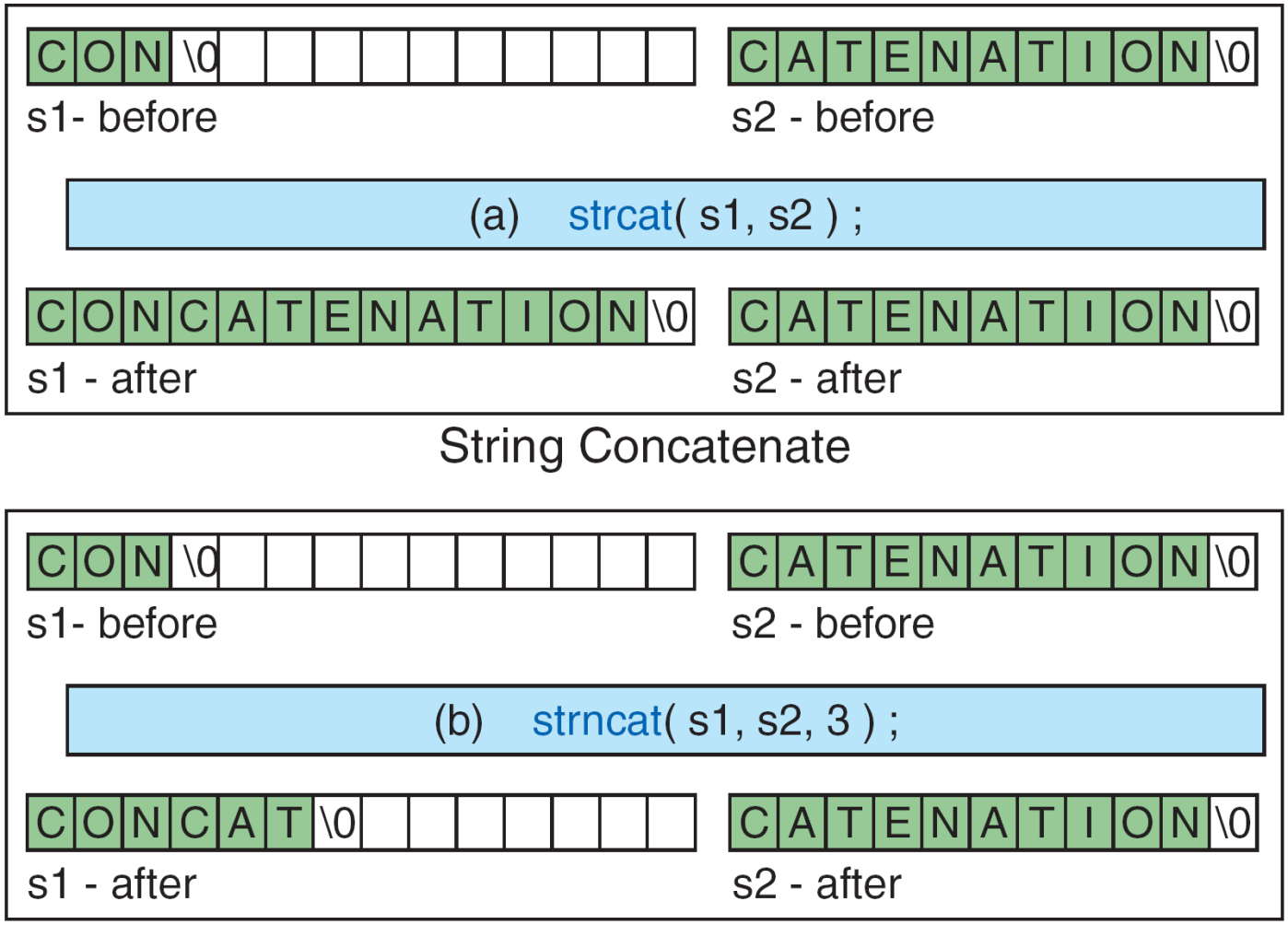

FIGURE 11-18 String Concatenation

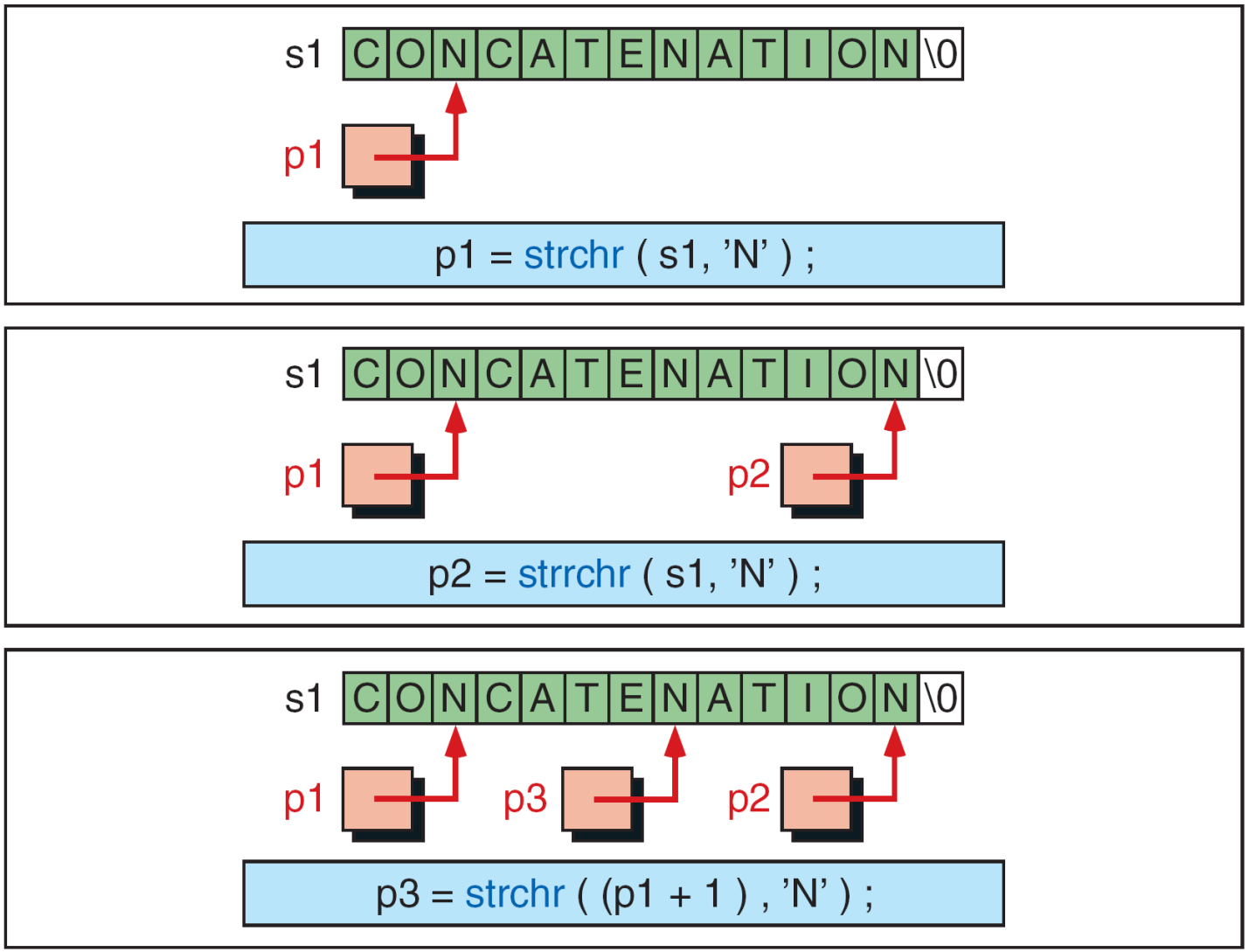

FIGURE 11-19 Character in String (*strchr*)

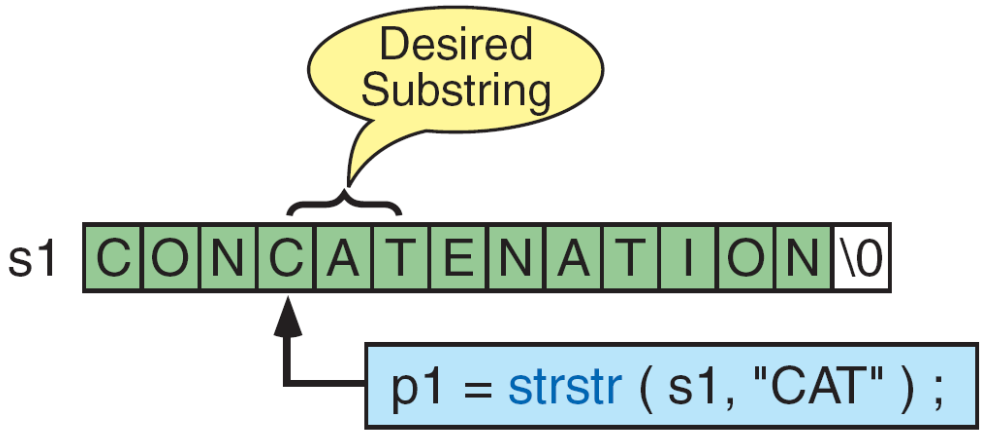

FIGURE 11-20 String in String

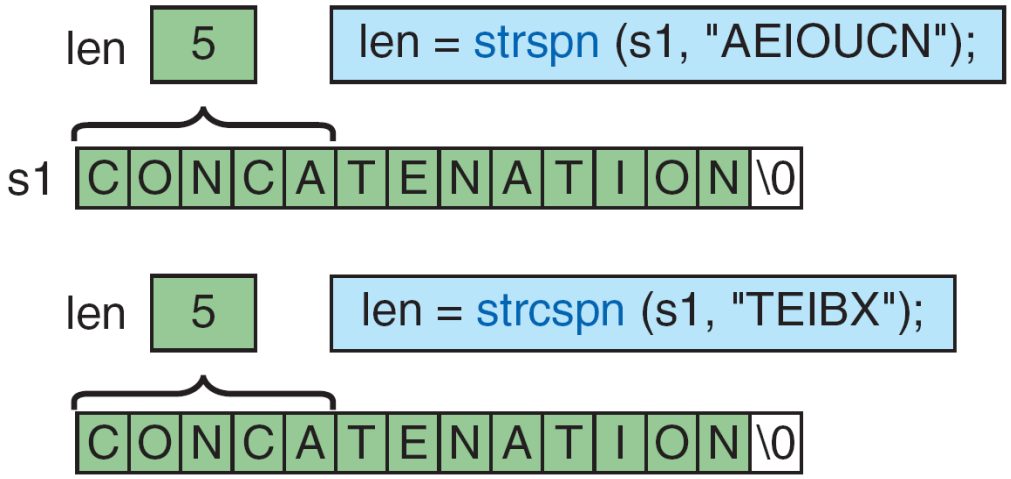

FIGURE 11-21 String Span

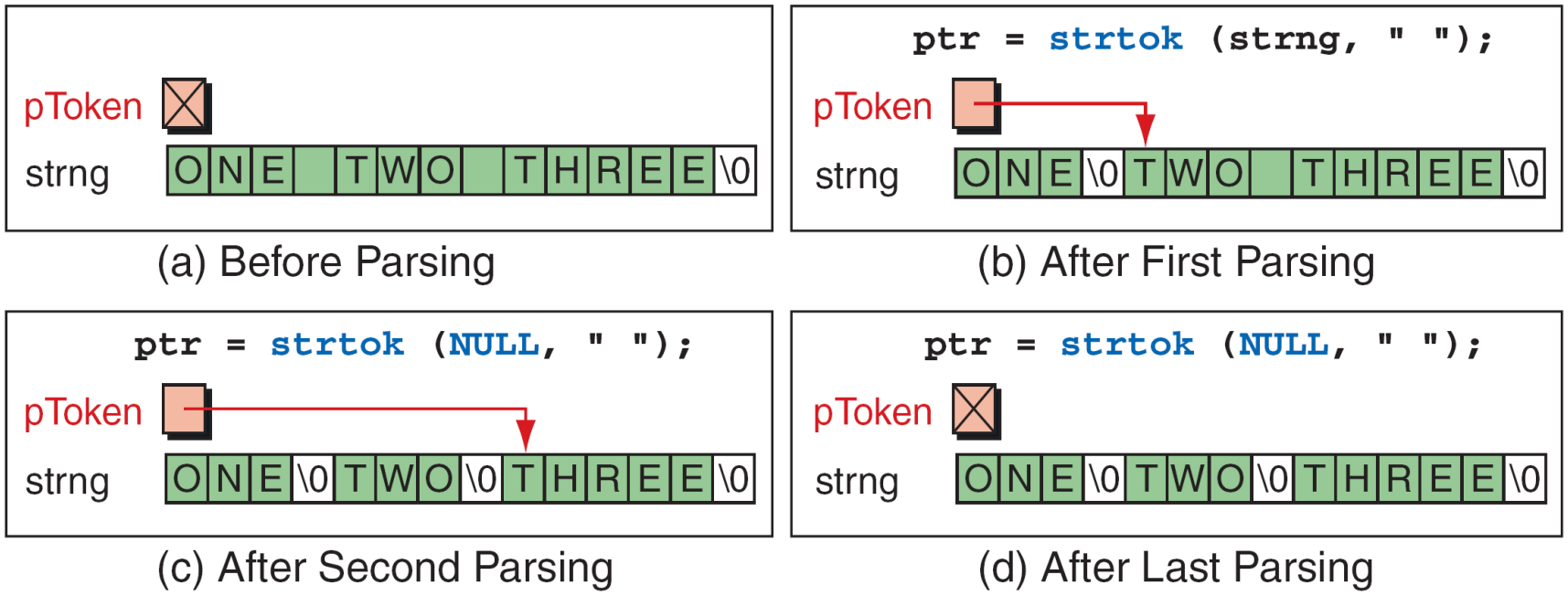

FIGURE 11-22 Parse a Simple String

PROGRAM 11-13 Demonstrate String to Long

```
1  /* Demonstrate string to long function.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    long  num;
12    char* ptr;
13
14 // Statements
15    num = strtol ("12345 Decimal constant: ", &ptr, 0);
16    printf("%s %ld\n", ptr, num);
17
18    num = strtol ("11001 Binary constant : ", &ptr, 2);
19    printf("%s %ld\n", ptr, num);
```

PROGRAM 11-13 Demonstrate String to Long

```
20
21      num = strtol ("13572 Octal constant  : ", &ptr, 8);
22      printf("%s %ld\n", ptr, num);
23
24      num = strtol (" 7AbC Hex constant    : ", &ptr, 16);
25      printf("%s %ld\n", ptr, num);
26
27      num = strtol ("11001 Base 0-Decimal  : ", &ptr, 0);
28      printf("%s %ld\n", ptr, num);
29
30      num = strtol ("01101 Base 0-Octal    : ", &ptr, 0);
31      printf("%s %ld\n", ptr, num);
32
33      num = strtol ("0x7AbC Base 0-Hex      : ", &ptr, 0);
34      printf("%s %ld\n", ptr, num);
35
36      num = strtol ("Invalid input    : ", &ptr, 0);
37      printf("%s %ld\n", ptr, num);
```

PROGRAM 11-13 Demonstrate String to Long

```
38
39      return 0;
40  }   // main
```

```
Results:
    Decimal constant:  12345
    Binary constant :  25
    Octal constant  :  6010
    Hex constant    :  31420
    Base 0-Decimal  :  11001
    Base 0-Octal    :  577
    Base 0-Hex      :  31420
    Invalid input   :  0
```

Numeric Format	ASCII Function	Wide-character Function
double	strtod	wcstod
float	strtof	wcstof
long double	strtold	wcstold
long int	strtol	wcstol
long long int	strtoll	wcstoll
unsigned long int	strtoul	wcstoul
unsigned long long int	strtoull	wcstoull

Table 11-2 String-to-Number Functions

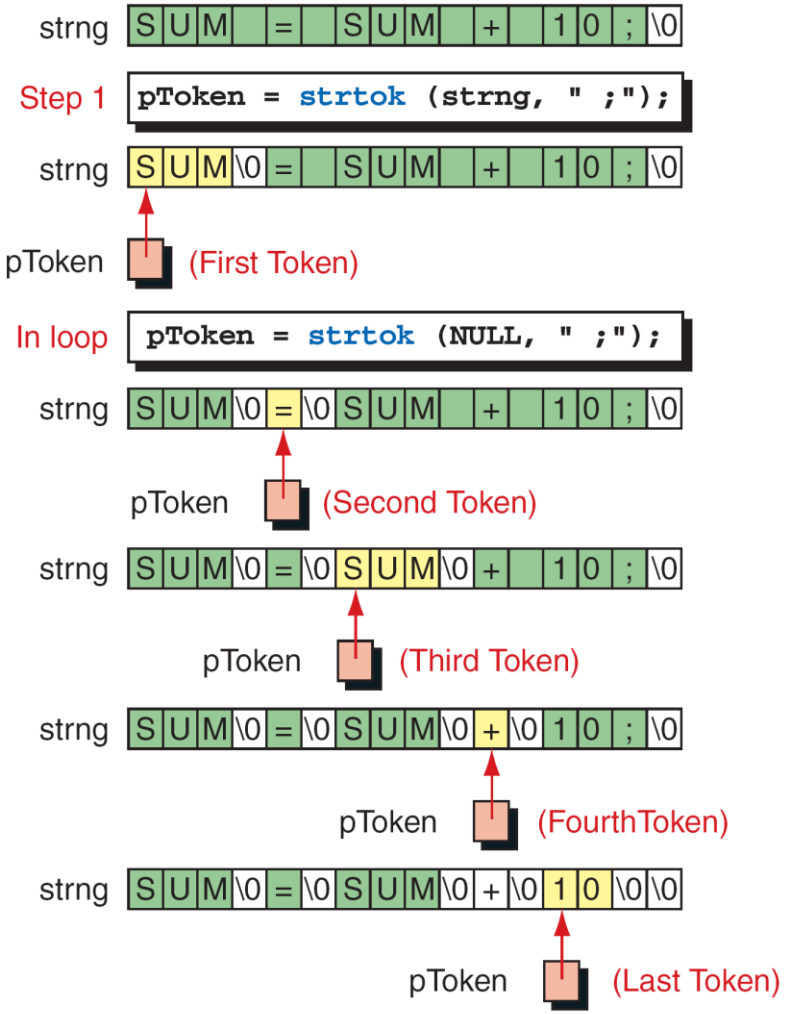

FIGURE 11-23 Parsing with String Token

PROGRAM 11-14 Parsing a String with String Token

```
1  /* Parse a simple algebraic expression.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <string.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    char  strng [16] = "sum = sum + 10;";
12    char* pToken;
13    int   tokenCount;
14
15 // Statements
16    tokenCount = 0;
17    pToken = strtok (strng, " ;");
18
```

PROGRAM 11-14 Parsing a String with String Token

```
19      while (pToken)
20          {
21           tokenCount++;
22           printf("Token %2d contains %s\n",
23                    tokenCount, pToken);
24           pToken = strtok (NULL, " ;");
25          } // while
26
27      printf("\nEnd of tokens\n");
28      return 0;
29  }   // main
```

```
Results:
Token   1 contains sum
Token   2 contains =
Token   3 contains sum
Token   4 contains +
Token   5 contains 10

End of tokens
```

PROGRAM 11-15 Compare Packed String Function

```
1  /* This program packs and compares a string.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <string.h>
7
8  #define ALPHA \
9   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
10
11 // Function Declarations
12 int  strCmpPk (char* S1, char* S2);
13 void strPk    (char* s1, char* s2);
14
15 int main (void)
16 {
17 // Local Declarations
18    int  cmpResult;
19    char s1[80];
20    char s2[80];
```

PROGRAM 11-15 Compare Packed String Function

```
21
22  // Statements
23      printf("Please enter first string:\n");
24      fgets (s1, 80, stdin);
25      s1[strlen(s1) - 1] = '\0';
26
27      printf("Please enter second string:\n");
28      fgets (s2, 80, stdin);
29      s2[strlen(s2) - 1] = '\0';
30
31      cmpResult = strCmpPk (s1, s2);
32      if (cmpResult < 0)
33          printf("string1 < string2\n");
34      else if (cmpResult > 0)
35          printf("string1 > string2\n");
36      else
37          printf("string1 == string2\n");
38
39      return 0;
40  }  // main
```

PROGRAM 11-15 Compare Packed String Function

```
41
42  /* ==================== strCmpPk ====================
43     Packs two strings and then compares them.
44        Pre   s1 and s2 contain strings
45        Post  returns result of strcmp of packed strings
46  */
47  int strCmpPk (char* s1, char* s2)
48  {
49  // Local Declarations
50     char s1In [80];
51     char s1Out[81];
52     char s2In [80];
53     char s2Out[81];
54
55  // Statements
56     strncpy (s1In, s1, sizeof(s1In) - 1);
57     strncpy (s2In, s2, sizeof(s2In) - 1);
58     strPk (s1In, s1Out);
59     strPk (s2In, s2Out);
60     return (strcmp (s1Out, s2Out));
61  }  // strCmpPk
62
```

PROGRAM 11-15 Compare Packed String Function

```
63  /* ==================== strPk ====================
64     Deletes all non-alpha characters from s1 and
65     copies to s2.
66        Pre   s1 is a string
67        Post  packed string in s2
68              s1 destroyed
69  */
70  void strPk (char* s1, char* s2)
71  {
72  // Local Declarations
73     int strSize;
74
75  // Statements
76     *s2 = '\0';
77     while (*s1 != '\0')
78        {
79         // Find non-alpha character & replace
80         strSize = strspn(s1, ALPHA);
81         s1[strSize] = '\0';
82         strncat (s2, s1, 79 - strlen(s2));
83         s1 += strSize + 1;
```

PROGRAM 11-15 Compare Packed String Function

```
84          } // while
85       return;
86    }  // strPk
```

```
Results:
Please enter first string:
a b!c 234d
Please enter second string:
abcd
string1 == string2

Please enter first string:
abcd
Please enter second string:
aabb
string1 > string2
```

11-6 String/Data Conversion

A common set of applications format data by either converting a sequence of characters into corresponding data types or vice versa. Two such applications are parsing and telecommunications.

Topics discussed in this section:

String to Data Conversion

Data to String Conversion

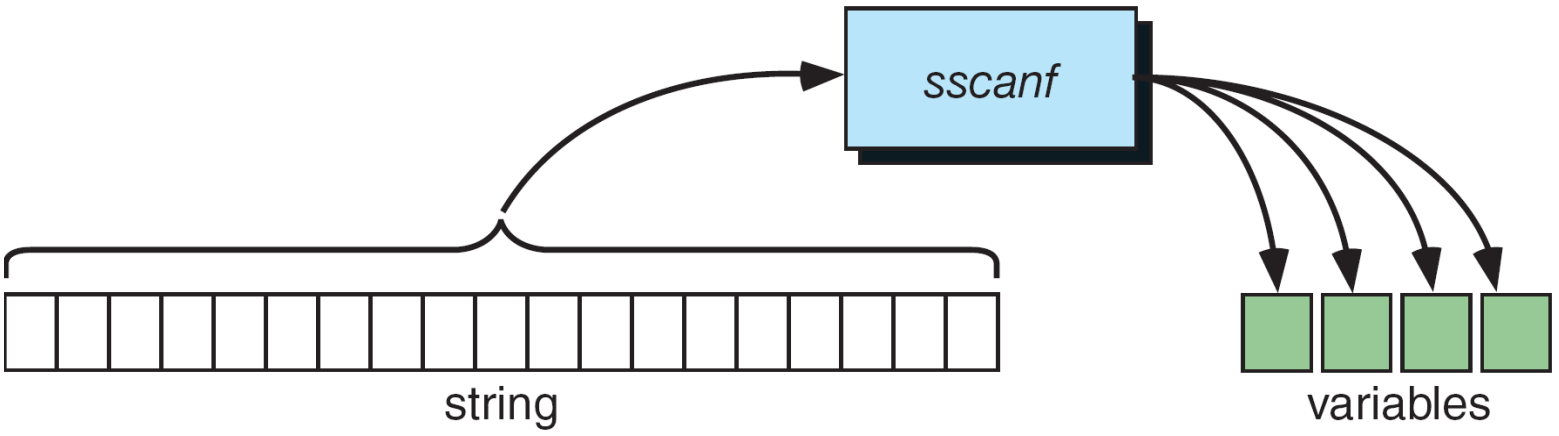

FIGURE 11-24 ***sscanf*** **Operation**

Note

sscanf is a one-to-many function. It splits one string into many variables.

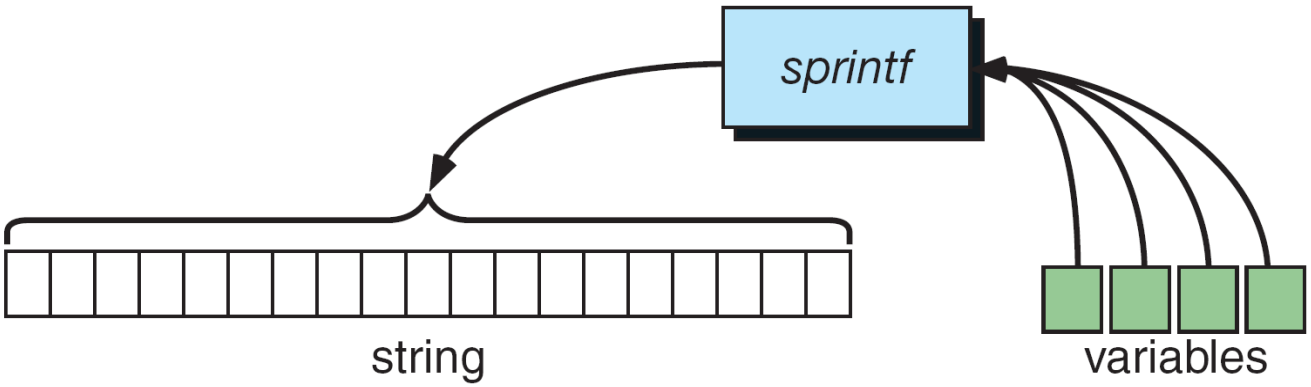

FIGURE 11-25 ***sprintf*** **Operation**

Note

sprintf is a many-to-one function. It joins many pieces of data into one string.

PROGRAM 11-16 Demonstrate Memory Formatting

```
 1  /* Demonstrate memory formatting.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     char  strng[80] = "Einstein, Albert; 1234 97 A";
11     char  strngOut[80];
12     char  name[50];
13     char  id[5];
14     int   score;
15     char  grade;
16
17  // Statements
18     printf("String contains:  \"%s\"\n", strng);
```

PROGRAM 11-16 Demonstrate Memory Formatting

```
19
20      sscanf(strng, "%49[^;] %*c %4s %d %c",
21             name, id, &score, &grade);
22
23      printf("Reformatted data: \n");
24      printf("   Name:           \"%s\"\n", name);
25      printf("   id:             \"%s\"\n", id);
26      printf("   score:           %d\n", score);
27      printf("   grade:           %c\n", grade);
28
29      sprintf(strngOut, "%s %4s %3d %c",
30                name, id, score, grade);
31      printf("New string:        \"%s\"\n", strngOut);
32      return 0;
33  }   // main
```

```
Results:
String contains:  "Einstein, Albert; 1234 97 A"
Reformatted data:
   Name:          "Einstein, Albert"
   id:            "1234"
   score:          97
   grade:          A
New string:       "Einstein, Albert 1234  97 A"
```

PROGRAM 11-17 Determine Internet Addresses in a Range

```
 1  /* Given two Internet addresses, determine the number
 2     of unique addresses in their range.
 3        Written by:
 4        Date:
 5  */
 6  #include <stdio.h>
 7  #include <stdlib.h>
 8
 9  int main (void)
10  {
11  // Local Declarations
12     unsigned int   strt[4];
13     unsigned int   end[4];
14     unsigned long  add1 = 0;
15     unsigned long  add2 = 0;
16     unsigned long  range;
17
18     char addr1[15];
19     char addr2[15];
20
```

PROGRAM 11-17 Determine Internet Addresses in a Range

```
21  // Statements
22     printf ("Enter first address:  ");
23     fgets (addr1 , sizeof (addr1) , stdin);
24
25     printf ("Enter second address: ");
26     fgets (addr2 , sizeof (addr2) , stdin);
27
28     sscanf (addr1 , "%d %*c %d %*c %d %*c %d\n",
29             &strt[3], &strt[2], &strt[1], &strt[0]);
30     sscanf (addr2 , "%d %*c %d %*c %d %*c %d\n",
31             &end[3], &end[2], &end[1], &end[0]);
32
33     for (int i = 3 ; i >= 0  ; i--)
34         {
35           add1 = add1 * 256 + strt[i];
36           add2 = add2 * 256 + end[i];
37         } // for
38     range = abs (add1 - add2) + 1;
39
```

PROGRAM 11-17 Determine Internet Addresses in a Range

```
40      printf ("\nFirst Address: %s", addr1);
41      printf ("Second Address %s", addr2);
42      printf ("\nThe range: %ld\n", range);
43
44      return 0;
45  }   // main
```

```
Results:
Enter first address:  23.56.34.0
Enter second address: 23.56.32.255

First Address: 23.56.34.0
Second Address 23.56.32.255

The range: 258
```

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

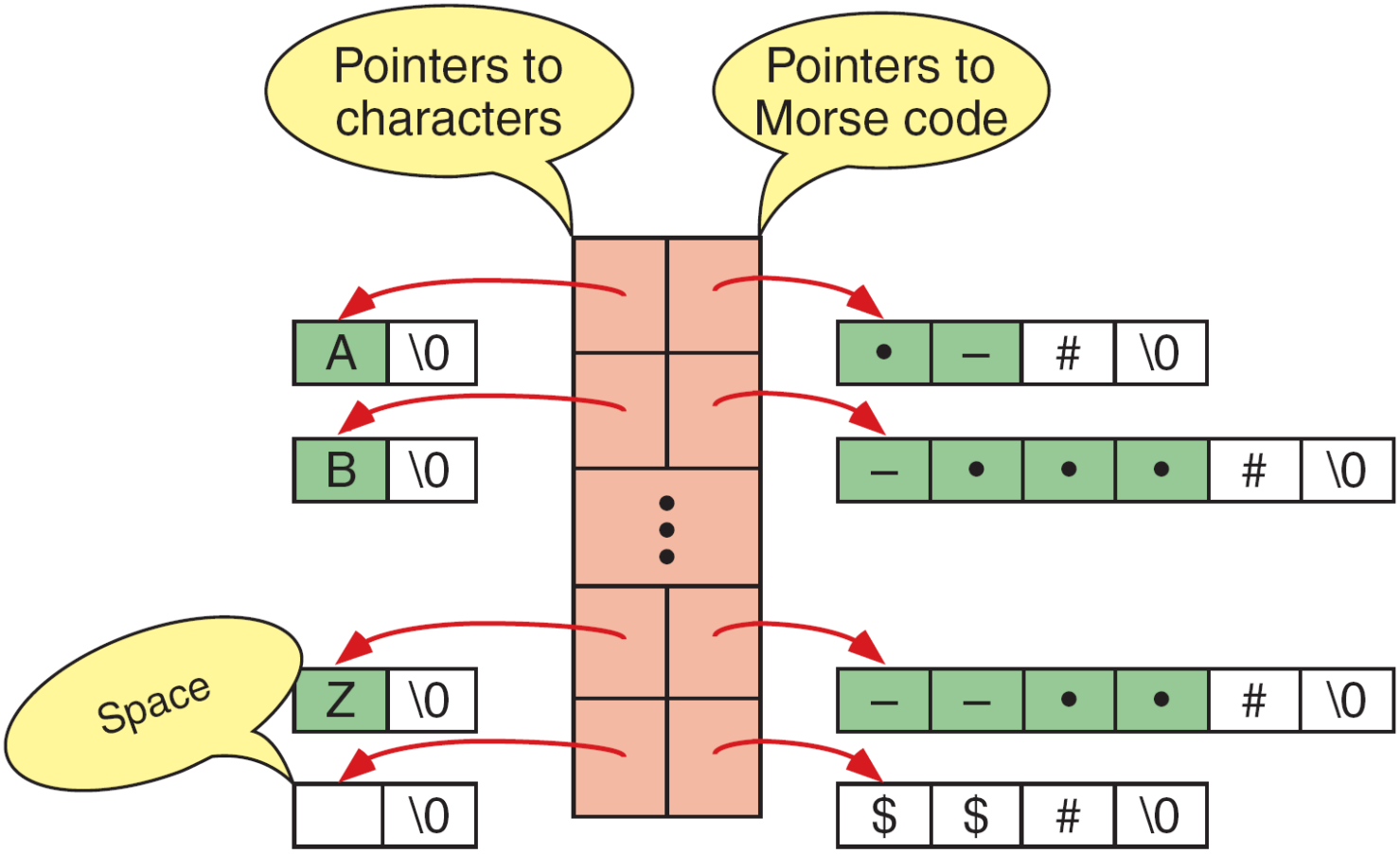

FIGURE 11-26 Character to Morse Code Structure

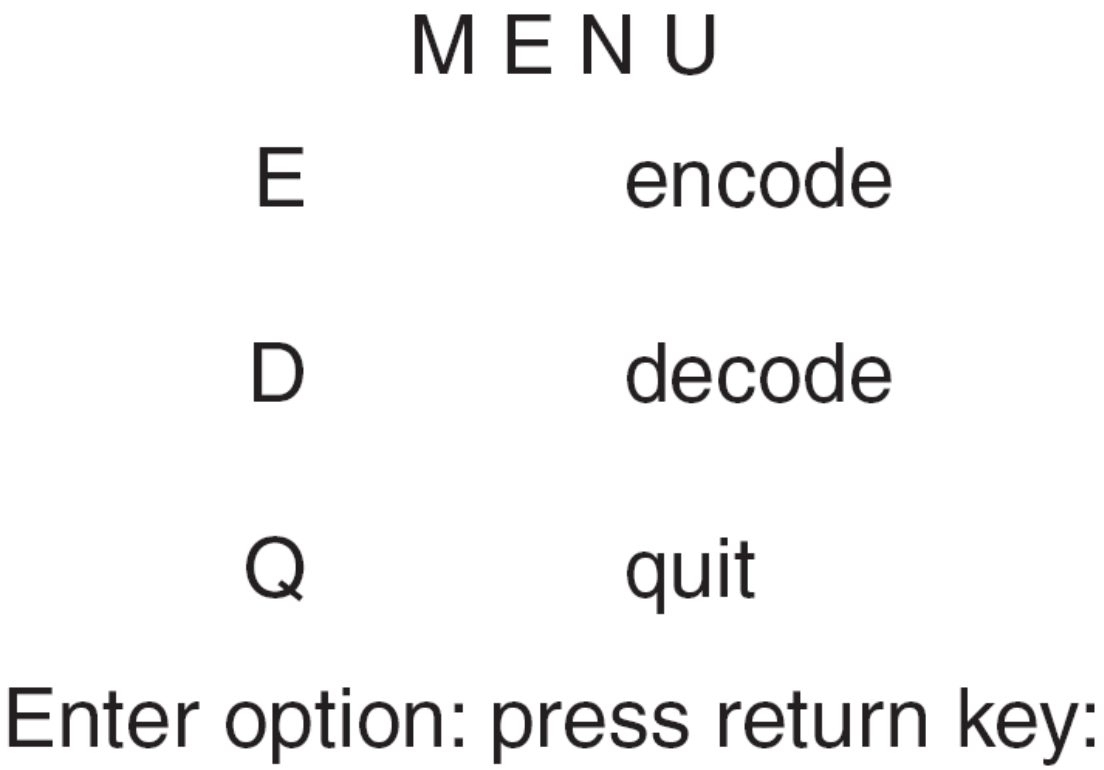

FIGURE 11-27 Morse Code Menu

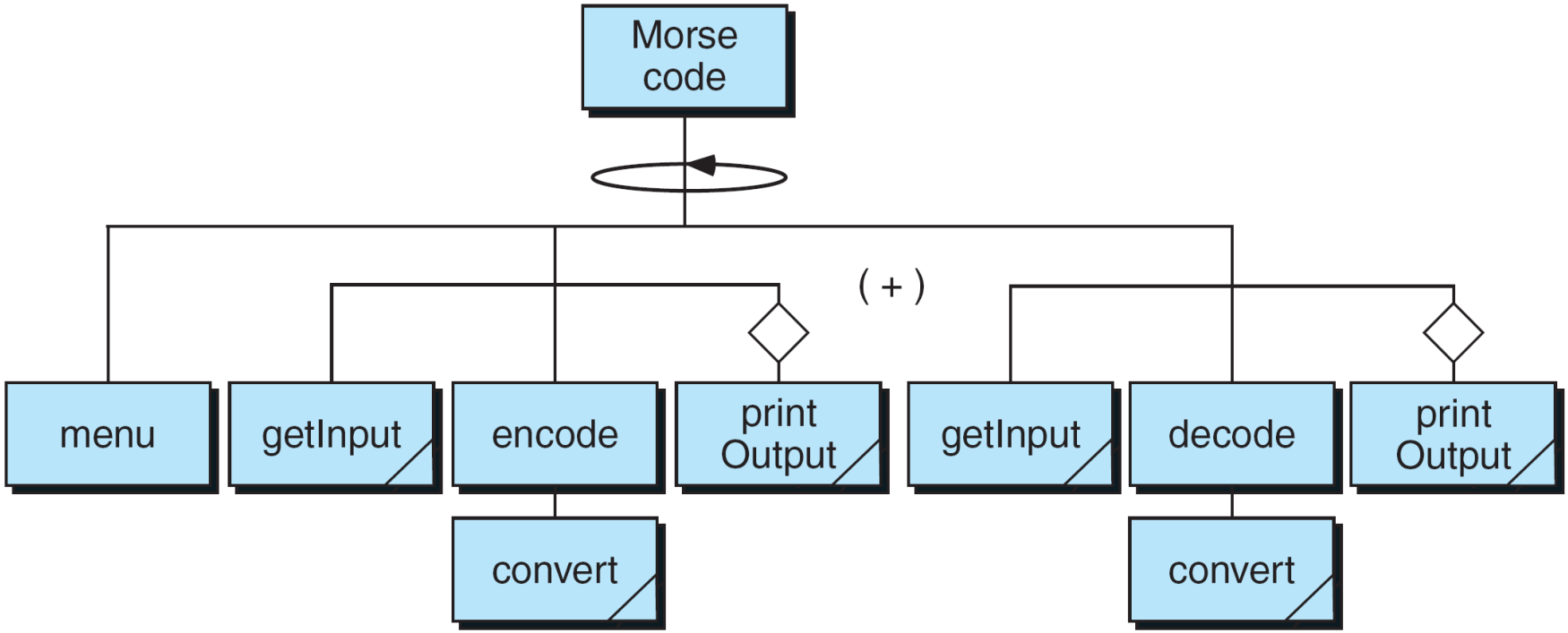

FIGURE 11-28 Morse Code Program Design

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

Program Design Concept

- Modular program
  - Well structured
  - Modules are independent and have single purposes
- Information hiding
  - The data structure and functional implementation are screened from the user

Program Design Concept

- Cohesion
  - A measure of how closely the processes in a function are related
- Primary reasons for cohesion
  - Accuracy
  - Maintainability
  - Resuable

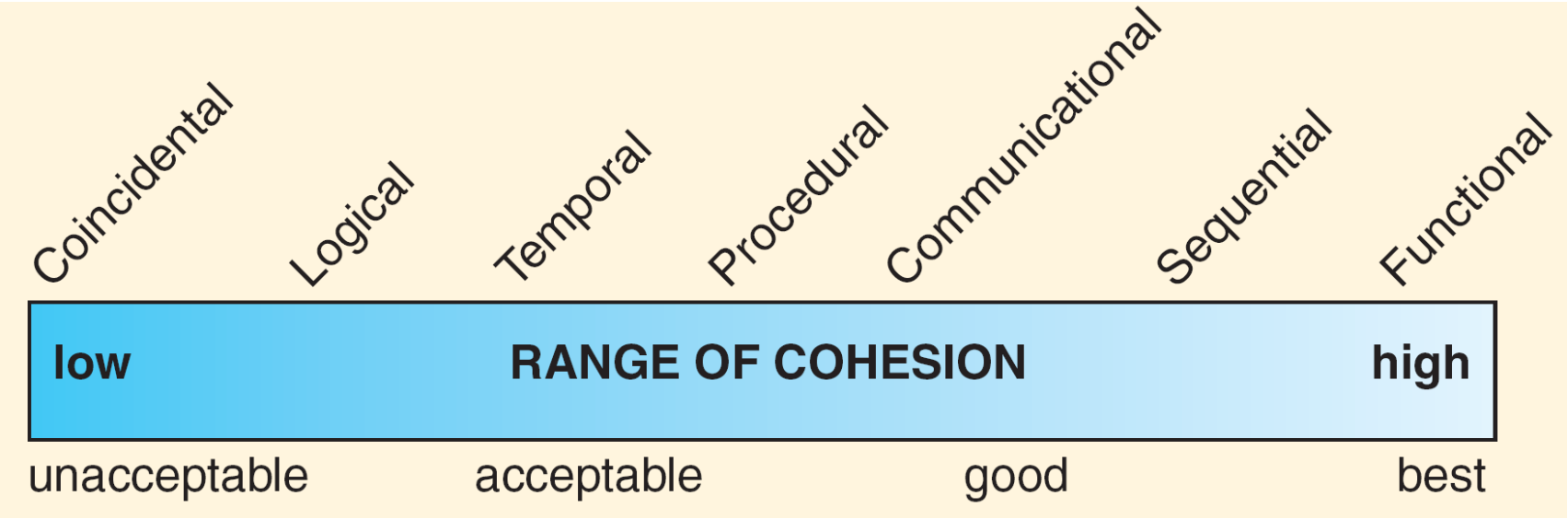

FIGURE 11-29 **Types of Cohesion**

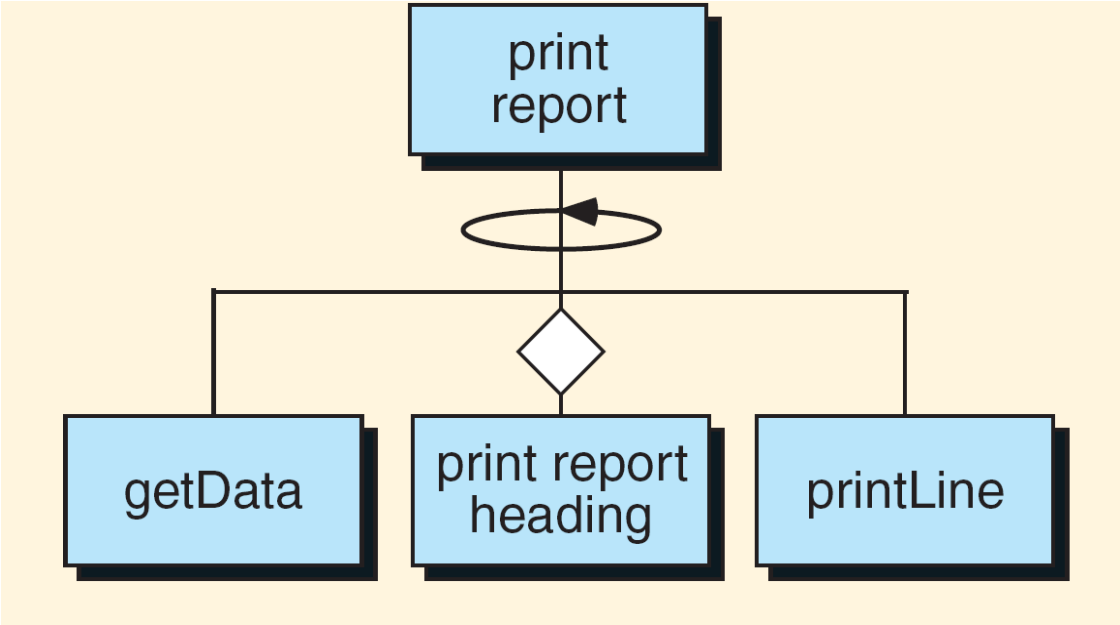

FIGURE 11-30 Example of Functional Cohesion

Note

Well-structured programs are highly cohesive and loosely coupled.