Strings
Chapter 8

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To understand how a string constant is stored in an array of characters
- To learn about the placeholder %s and how it is used in printf and scanf operations
- To learn some of the operations that can be performed on strings such as copying strings extracting substrings, and joining strings using functions from the library string

String Basics

- A blank in a string is a valid character.
- null character
  - character '\0' that marks the end of a string in C
- A string in C is implemented as an array.
  - char string_var[30];
  - char str[20] = "Initial value";
- An array of strings is a 2-dimensional array of characters in which each row is a string.

Input/Output

- printf and scanf can handle string arguments
- use %s as the placeholder in the format string

```
char president[20];
scanf("%s\n", president);
printf("%s\n", president);
```

Initializing Strings

- sizeof() gives size in bytes
- strlen() gives length of string

```
char string[16] = “hello world”;

char *str = “hello world”;

char s[] = “hello world”;
```

Initializing Strings

- sizeof() gives size in bytes
- strlen() gives length of string

char string[16] = “hello world”;

char *str = “hello world”;

char s[] = “hello world”;

Initializing Strings

- sizeof() gives size in bytes
- strlen() gives length of string

char string[16] = “hello world”;

sizeof() is 16
strlen() is 11

char *str = “hello world”;

sizeof() is 8
strlen() is 11

char s[] = “hello world”;

sizeof() is 12
strlen() is 11

Buffer Overflow

- more data is stored in an array than its declared size allows
- a very dangerous condition
- unlikely to be flagged as an error by either the compiler or the run-time system

char string[8] = “hello world”;

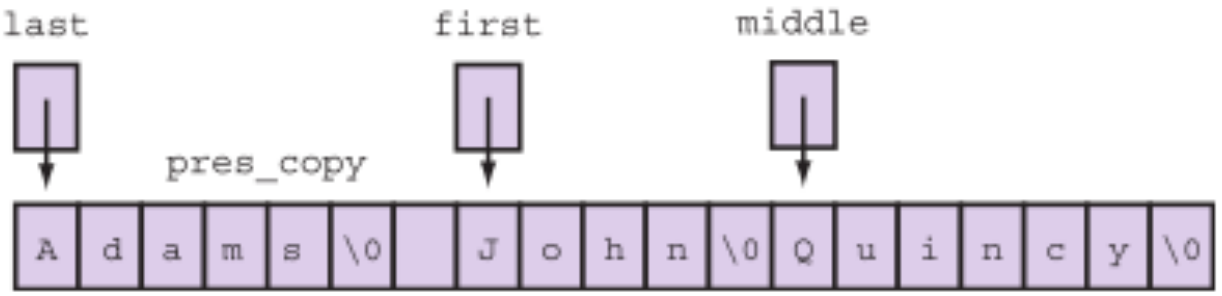
String Assignment

- strcpy
  - copies string in second argument into its first argument
    - strcpy(s1, “hello”);
  - subject to buffer overflow
- strncpy
  - takes an argument specifying the number of chars to copy
  - if the string to be copies is shorter, the remaining characters stored are null
    - strncpy(s2, “inevitable”, 5);

= does not work!

String tokenization

```
char *last, *first, *middle;
char pres[20] = "Adams, John Quincy";
char pres_copy[20];
strcpy(pres_copy, pres);
```



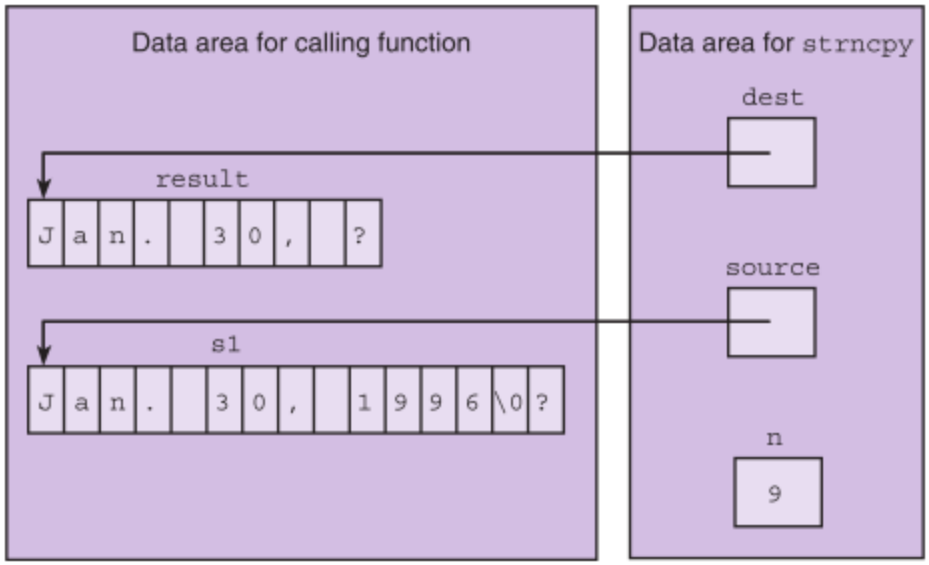

```
last = strtok(pres_copy, ", ");
first = strtok(NULL, ", ");
middle = strtok(NULL, ", ");
```

Substrings

- a fragment of a longer string

FIGURE 8.5

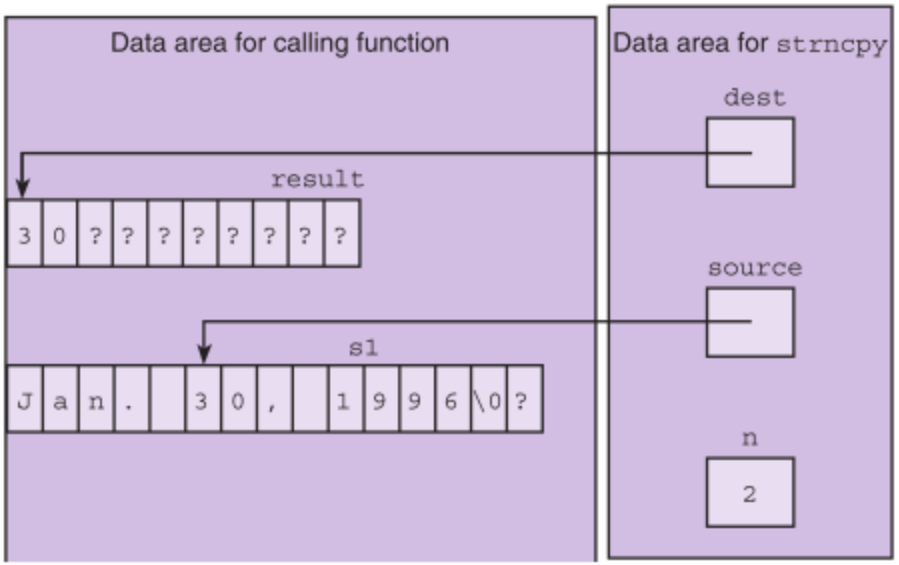
Execution of `strncpy(result, s1, 9);`



Substrings

FIGURE 8.6

Execution of `strncpy(result, &s1[5], 2);`



Substrings

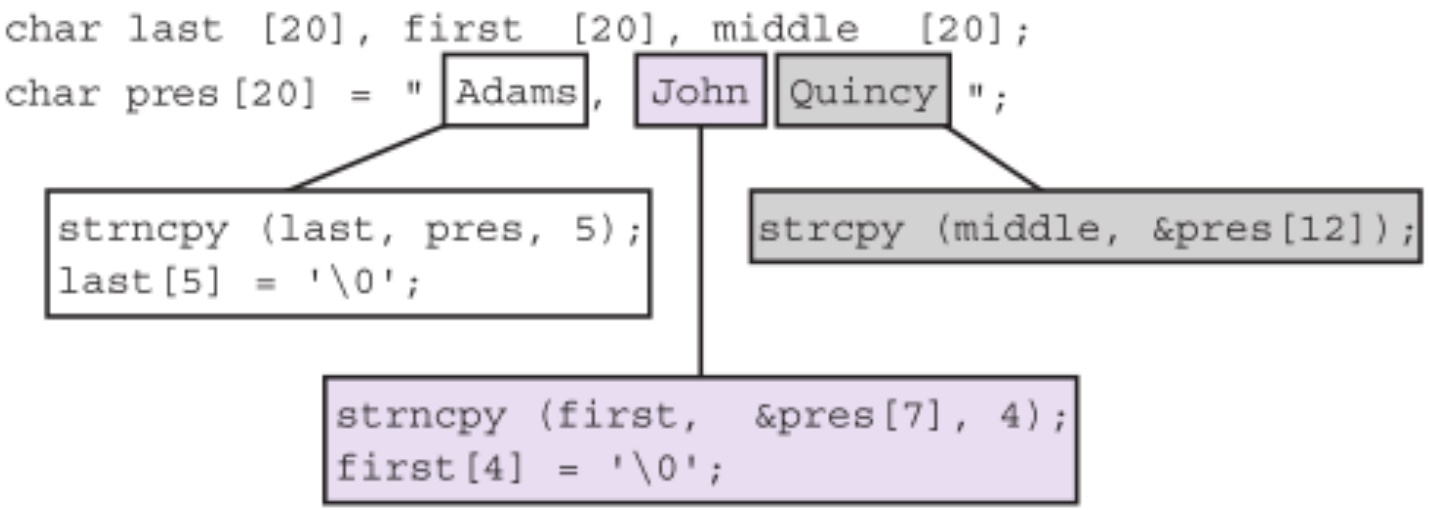


FIGURE 8.7 Program Using strncpy and strcpy Functions to Separate Compounds into Elemental Components

```
/*
 * Displays each elemental component of a compound
 */

#include <stdio.h>
#include <string.h>

#define CMP_LEN 30  /* size of string to hold a compound  */
#define ELEM_LEN 10 /* size of string to hold a component */

int
main(void)
{
     char compound[CMP_LEN];  /* string representing a compound  */
     char elem[ELEM_LEN];     /* one elemental component         */
     int  first, next;

     /* Gets data string representing compound                    */
     printf("Enter a compound> ");
     scanf("%s", compound);

     /* Displays each elemental component. These are identified
        by an initial capital letter.                             */
     first = 0;
     for (next = 1; next < strlen(compound); ++next)
         if (compound[next] >= 'A'  &&  compound[next] <= 'Z') {
                strncpy(elem, &compound[first], next - first);
                elem[next - first] = '\0';
                printf("%s\n", elem);
                first = next;
         }

     /* Displays the last component                               */
     printf("%s\n", strcpy(elem, &compound[first]));

     return (0);
}
```

```
Enter a compound> H2SO4
H2
S
O4
```

String Terminology

- string length
  - in a character array, the number of characters before the first null character

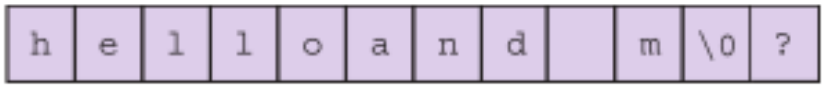
- empty string
  - a string of length zero
  - the first character of the string is the null character

Concatenation

- strcat
  - appends source to the end of dest
  - assumes that sufficient space is allocated for the first argument to allow addition of the extra characters
    - s1 = “hello”;
    - strcat(s1, “and more”);

Concatenation

- strncat
  - appends up to n characters of source to the end of dest, adding the null character if necessary
  - assumes that sufficient space is allocated for the first argument to allow addition of the extra characters
    - s1 = “hello”;
    - strncat(s1, “and more”, 5);



Scanning a Full Line

- For interactive input of one complete line of data, use the gets function.
- The \n character representing the <return> or <enter> key pressed at the end of the line is not stored.

Scanning a Full Line

```
char line[80];
printf("Type in a line of data.\n> ");
gets(line);
```

```
Type in a line of data.
> Here is a short sentence.
```

FIGURE 8.8 Demonstration of Whole-Line Input

```
1.  /*
2.   *  Numbers and double spaces lines of a document. Lines longer than
3.   *  LINE_LEN - 1 characters are split on two lines.
4.   */
5.
```

FIGURE 8.8 (continued)

```
6.  #include <stdio.h>
7.  #include <string.h>
8.
9.  #define LINE_LEN 80
10. #define NAME_LEN 40
11.
12. int
13. main(void)
14. {
15.       char line[LINE_LEN], inname[NAME_LEN], outname[NAME_LEN];
16.       FILE *inp, *outp;
17.       char *status;
18.       int i = 0;
```

```
19.
20.         printf("Name of input file> ");
21.         scanf("%s", inname);
22.         printf("Name of output file> ");
23.         scanf("%s", outname);
24.
25.         inp = fopen(inname, "r");
26.         outp = fopen(outname, "w");
27.
28.         for (status = fgets(line, LINE_LEN, inp);
29.              status != 0;
30.              status = fgets(line, LINE_LEN, inp)) {
31.            if (line[strlen(line) - 1] == '\n')
32.                 line[strlen(line) - 1] = '\0';
33.            fprintf(outp, "%3d>> %s\n\n", ++i, line);
34.         }
35.       return (0);
36. }
```

File used as input

```
In the early 1960s, designers and implementers of operating
systems were faced with a significant dilemma. As people's
expectations of modern operating systems escalated, so did
the complexity of the systems themselves. Like other
programmers solving difficult problems, the systems
programmers desperately needed the readability and
modularity of a powerful high-level programming language.
```

Output file

```
1>> In the early 1960s, designers and implementers of operating
2>> systems were faced with a significant dilemma. As people's
3>> expectations of modern operating systems escalated, so did
4>> the complexity of the systems themselves. Like other
5>> programmers solving difficult problems, the systems
6>> programmers desperately needed the readability and
7>> modularity of a powerful high-level programming language.
```

String Comparison

TABLE 8.2 Possible Results of strcmp(str1, str2)

Relationship	Value Returned	Example
`str1` is less than `str2`	negative integer	`str1` is `"marigold"` `str2` is `"tulip"`
`str1` equals `str2`	zero	`str1` and `str2` are both `"end"`
`str1` is greater than `str2`	positive integer	`str1` is `"shrimp"` `str2` is `"crab"`

String Comparison

FIGURE 8.9 Numeric and String Versions of Portions of Selection Sort That Compare and Exchange Elements

Comparison (in function that finds index of "smallest" remaining element)

Numeric

```
if (list[i] < list[first])
     first = i;
```

String

```
if (strcmp(list[i], list[first]) < 0)
      first = i;
```

Exchange of elements

```
temp = list[index_of_min];
list[index_of_min] = list[fill];
list[fill] = temp;
```

```
strcpy(temp, list[index_of_min]);
strcpy(list[index_of_min], list[fill]);
strcpy(list[fill], temp);
```

FIGURE 8.10 Sentinel-Controlled Loop for String Input

```
1.  printf("Enter list of words on as many lines as you like.\n");
2.  printf("Separate words by at least one blank.\n");
3.  printf("When done, enter %s to quit.\n", SENT);
4.
5.  for (scanf("%s", word);
6.       strcmp(word, SENT) != 0;
7.       scanf("%s", word)) {
8.       /* process word */
9.       . . .
10. }
```

Arrays of Pointers

- When sorting a list of strings, there is a lot of copying of characters from one memory cell to another.
  - 3 operations for every exchange
- C represents every array by its starting address.
- Consider an array of pointers, each element the address of a character string.

FIGURE 8.11 Exchanging String Elements of an Array

```
1. strcpy(temp, list[index_of_min]);
2. strcpy(list[index_of_min], list[fill]);
3. strcpy(list[fill], temp);
```

FIGURE 8.12

Executing `strcpy (list [index_of_min], list[fill]);`

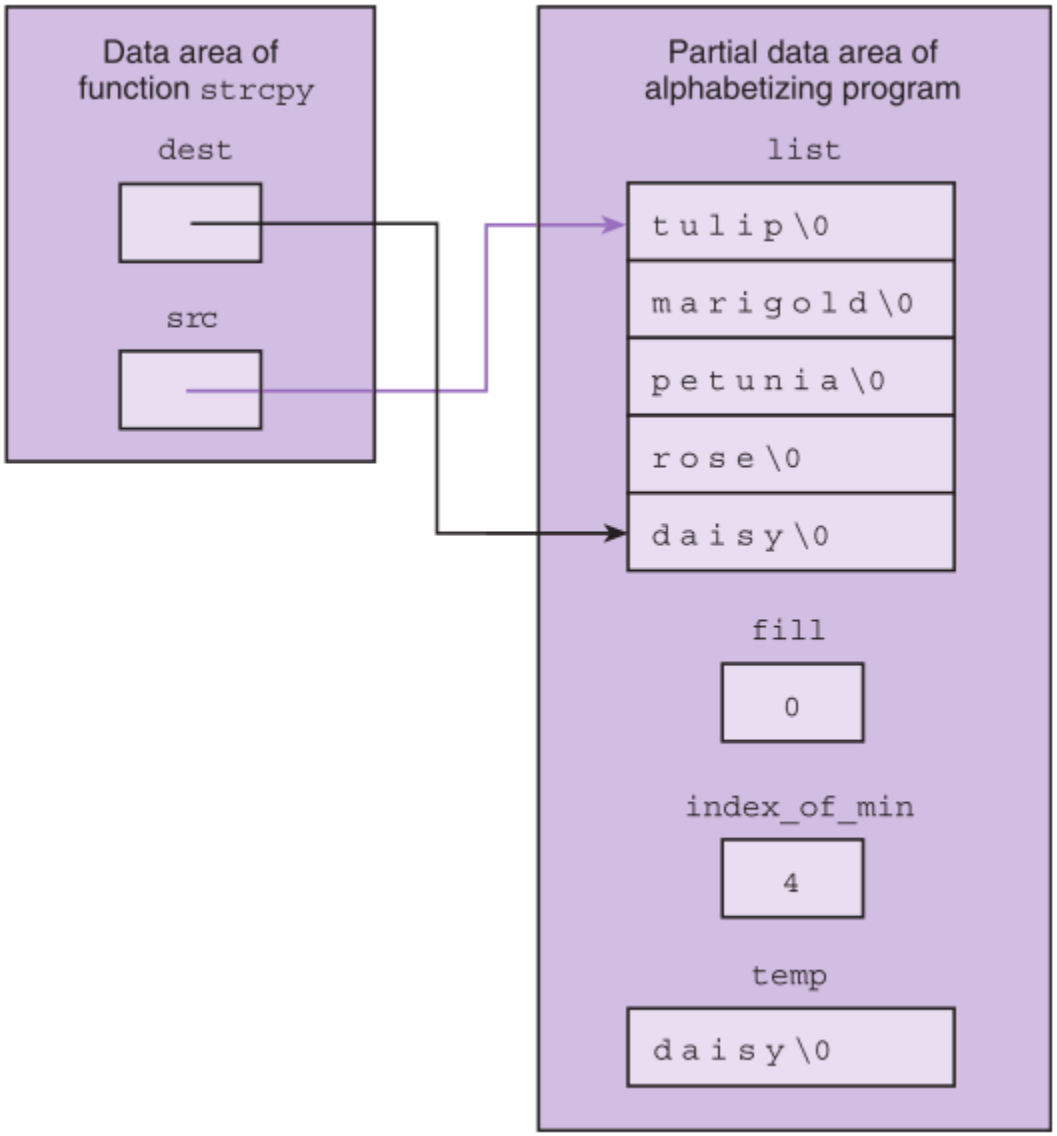


FIGURE 8.13

An Array of Pointers

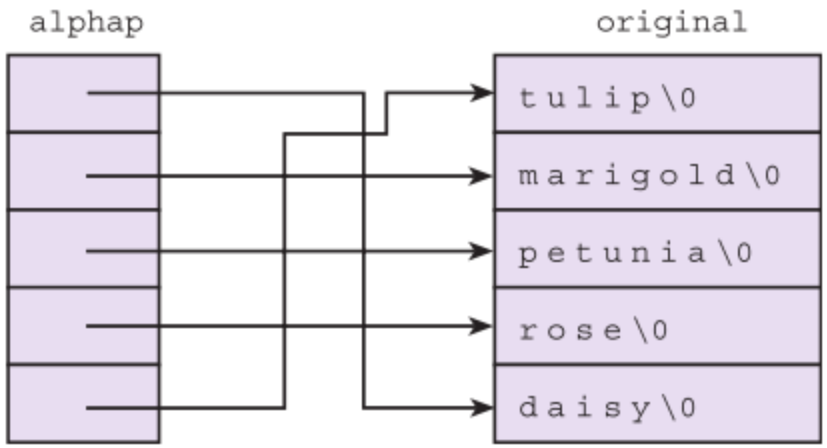


FIGURE 8.14 Two Orderings of One List Using an Array of Pointers

```
1.  /*
2.   *  Maintains two orderings of a list of applicants: the original
3.   *  ordering of the data, and an alphabetical ordering accessed through an
4.   *  array of pointers.
5.   */
6.
7.  #include <stdio.h>
8.  #define STRSIZ 30     /*     maximum string length */
9.  #define MAXAPP 50     /*     maximum number of applications accepted */
10.
11. int alpha_first(char *list[], int min_sub, int max_sub);
12. void select_sort_str(char *list[], int n);
13.
14. int
15. main(void)
16. {
17.       char applicants[MAXAPP][STRSIZ]; /* list of applicants in the
18.                                           order in which they applied        */
19.       char *alpha[MAXAPP];            /* list of pointers to
20.                                           applicants                         */
21.       int   num_app,                   /* actual number of applicants        */
22.             i;
23.       char  one_char;
24.
```

```
25.       /* Gets applicant list                                                    */
26.       printf("Enter number of applicants (0 . . %d)\n> ", MAXAPP);
27.       scanf("%d", &num_app);
28.       do    /* skips rest of line after number */
29.           scanf("%c", &one_char);
30.       while (one_char != '\n');
31.
32.       printf("Enter names of applicants on separate lines of less than\n");
33.       printf(" 30 characters in the order in which they applied\n");
34.       for (i = 0; i < num_app; ++i)
35.          gets(applicants[i]);
36.
```

(continued)

FIGURE 8.14 (continued)

```
37.       /* Fills array of pointers and sorts                                      */
38.       for (i = 0; i < num_app; ++i)
39.          alpha[i] = applicants[i]; /* copies ONLY address */
40.       select_sort_str(alpha, num_app);
41.
42.      /* Displays both lists                                                     */
43.       printf("\n\n%-30s%5c%-30s\n\n", "Application Order", ' ',
44.                 "Alphabetical Order");
45.       for (i = 0;  i < num_app;  ++i)
46.           printf("%-30s%5c%-30s\n", applicants[i], ' ', alpha[i]);
47.
48.       return(0);
49. }
```

```
50. /*
51.  *  Finds the index of the string that comes first alphabetically in
52.  *  elements min_sub..max_sub of list
53.  *  Pre: list[min_sub] through list[max_sub] are of uniform case;
54.  *       max_sub >= min_sub
55. */
56. int
57. alpha_first(char *list[],         /* input - array of pointers to strings      */
58.            int   min_sub,        /* input - minimum and maximum subscripts    */
59.            int   max_sub)        /*   of portion of list to consider          */
60. {
61.       int first, i;
62.
63.       first = min_sub;
64.       for  (i = min_sub + 1;  i <= max_sub;  ++i)
65.          if (strcmp(list[i], list[first]) < 0)
66.                first = i;
67.
68.       return (first);
69. }
70.
71. /*
72.  * Orders the pointers in array list so they access strings
73.  * in alphabetical order
74.  * Pre:  first n elements of list reference strings of uniform case;
75.  *       n >= 0
76.  */
77. void
```

(continued)

FIGURE 8.14 (continued)

```
78. select_sort_str(char *list[],  /* input/output - array of pointers being
79.                                   ordered to access strings alphabetically */
80.                 int n)         /* input - number of elements to sort       */
81. {
82.
83.     int fill,              /* index of element to contain next string in order */
84.         index_of_min;      /* index of next string in order */
85.     char *temp;
86.
87.     for  (fill = 0;  fill < n - 1;  ++fill) {
88.         index_of_min = alpha_first(list, fill, n - 1);
89.
90.         if (index_of_min != fill) {
91.               temp = list[index_of_min];
92.               list[index_of_min] = list[fill];
93.               list[fill] = temp;
94.         }
95.     }
96. }
```

```
Enter number of applicants (0 . . 50)
> 5
Enter names of applicants on separate lines of less than
30 characters in the order in which they applied
SADDLER, MARGARET
INGRAM, RICHARD
FAATZ, SUSAN
GONZALES, LORI
KEITH, CHARLES

Application Order              Alphabetical Order

SADDLER, MARGARET              FAATZ, SUSAN
INGRAM, RICHARD                GONZALES, LORI
FAATZ, SUSAN                   INGRAM, RICHARD
GONZALES, LORI                 KEITH, CHARLES
KEITH, CHARLES                 SADDLER, MARGARET
```

Character Input/Output

- getchar
  - get the next character from the standard input source (that scanf uses)
  - does not expect the calling module to pass the address of a variable to store the input character
  - takes no arguments, returns the character as its result

```
ch = getchar()
```

Character Input/Output

- getc
  - used to get a single character from a file
  - comparable to getchar except that the character returned is obtained from the file accessed by a file pointer (ex., inp)

```
getc(inp)
```

Character Input/Output

- putchar
  - single-character output
  - first argument is a type int character code
  - recall that type char can always be converted to type in with no loss of information

```
putchar('a');
```

Character Input/Output

- putc
  - identical to putchar except it sends the single character/int to a file, ex., outp

```
putc('a', outp);
```

FIGURE 8.15 Implementation of scanline Function Using getchar

```
1.  /*
2.   *  Gets one line of data from standard input. Returns an empty string on
3.   *  end of file. If data line will not fit in allotted space, stores
4.   *  portion that does fit and discards rest of input line.
5.   */
6.  char *
7.  scanline(char *dest,     /* output  - destination string                  */
8.           int   dest_len) /* input   - space available in dest              */
9.  {
10.       int i, ch;
11.
12.       /*  Gets next line one character at a time.                          */
13.       i = 0;
14.       for (ch = getchar();
15.            ch != '\n'  &&  ch != EOF  &&  i < dest_len - 1;
16.            ch = getchar())
17.          dest[i++] = ch;
18.      dest[i] = '\0';
19.
20.     /* Discards any characters that remain on input line                  */
21.     while (ch != '\n'  &&  ch != EOF)
22.         ch = getchar();
23.
24.     return (dest);
25. }
```

TABLE 8.3 Character Classification and Conversion Facilities in ctype Library

Facility	Checks	Example
`isalpha`	if argument is a letter of the alphabet	`if (isalpha(ch))` `  printf("%c is a letter\n", ch);`
`isdigit`	if argument is one of the ten decimal digits	`dec_digit = isdigit(ch);`

`islower` `(isupper)`	if argument is a lowercase (or uppercase) letter of the alphabet	`if (islower(fst_let)) {`
		`  printf("\nError: sentence ");`
		`  printf("should begin with a ");`
		`  printf("capital letter.\n");`
		`}`

`ispunct`	if argument is a punctuation character, that is, a noncontrol character that is not a space, a letter of the alphabet, or a digit	`if (ispunct(ch))`
		`  printf("Punctuation mark: %c\n",`
		`  ch);`

`isspace`	if argument is a whitespace character such as a space, a newline, or a tab	`c = getchar();`
		`while (isspace(c) && c != EOF)`
		`  c = getchar();`

Facility	**Converts**	**Example**

`tolower` `(toupper)`	its lowercase (or uppercase) letter argument to the uppercase (or lowercase) equivalent and returns this equivalent as the value of the call	`if (islower(ch))`
		`  printf("Capital %c = %c\n",`
		`  ch, toupper(ch));`

FIGURE 8.16 String Function for a Greater-Than Operator That Ignores Case

```
1.  #include <string.h>
2.  #include <ctype.h>
3.
4.  #define STRSIZ 80
5.
6.  /*
7.   * Converts the lowercase letters of its string argument to uppercase
8.   * leaving other characters unchanged.
9.   */
10. char *
11. string_toupper(char *str) /* input/output - string whose lowercase
12.                              letters are to be replaced by uppercase         */
13. {
```

(continued)

FIGURE 8.16 (continued)

```
14.         int i;
15.         for  (i = 0;  i < strlen(str);  ++i)
16.            if (islower(str[i]))
17.                  str[i] = toupper(str[i]);
18.
19.         return (str);
20. }
21.
22. /*
23.  * Compares two strings of up to STRSIZ characters ignoring the case of
24.  * the letters. Returns the value 1 if str1 should follow str2 in an
25.  * alphabetized list; otherwise returns 0
26.  */
27. int
28. string_greater(const char *str1, /* input -                                  */
29. const char *str2) /* strings to compare                                     */
30. {
31.          char s1[STRSIZ], s2[STRSIZ];
32.
33.          /* Copies str1 and str2 so string_toupper can modify copies    */
34.          strcpy(s1, str1);
35.          strcpy(s2, str2);
36.
37.          return (strcmp(string_toupper(s1), string_toupper(s2)) > 0);
38. }
```

String-to-Number and Number-to-String Conversions

TABLE 8.4 Review of Use of scanf

Declaration	Statement	Data (␣ means blank)	Value Stored

`char t`	`scanf("%c", &t);`	`␣g`
		`\n`
		`A`
			
			`\n`
			`A`

`int n`	`scanf("%d", &n);`	`␣32␣`
		`␣␣-8.6`
		`␣+19␣`
			`32`
			`-8`
			`19`

`double x`	`scanf("%lf", &x);`	`␣␣␣4.32␣`
		`␣-8␣`
		`␣1.76e-3␣`
			`4.32`
			`-8.0`
			`.00176`

`char str[10]`	`scanf("%s", str);`	`␣␣hello\n`
		`overlengthy␣`
			`hello\0`
			`overlengthy\0`
			(overruns length of `str`)

String-to-Number and Number-to-String Conversions

TABLE 8.5 Placeholders Used with printf

Value	Placeholder	Output (␣ means blank)
'a'	%c	a
	%3c	␣␣a
	%-3c	a␣␣
-10	%d	-10
	%2d	-10
	%4d	␣-10
	%-5d	-10␣␣
49.76	%.3f	49.760
	%.1f	49.8
	%10.2f	␣␣␣␣49.76
	%10.3e	␣4.976e+01
"fantastic"	%s	fantastic
	%6s	fantastic
	%12s	␣␣␣fantastic
	%-12s	fantastic␣␣␣
	%3.3s	fan

FIGURE 8.17 Program Segment That Validates Input Line Before Storing Data Values

```
1.  char data_line[STRSIZ], str[STRSIZ];
2.  int n1, n2, error_mark, i;
3.
4.  scanline(data_line, STRSIZ);
5.  error_mark = validate(data_line);
6.
7.  if (error_mark < 0) {
8.        /* Stores in memory values from correct data line    */
9.        sscanf(data_line, "%d%d%s", &n1, &n2, str);
10. } else {
11.       /* Displays line and marks spot where error detected */
12.       printf("\n%s\n", data_line);
13.       for (i = 0;  i < error_mark;  ++i)
14.          putchar(' ');
15.       putchar('/');
16. }
```

FIGURE 8.18 Functions That Convert Representations of Dates

```
1.  /*
2.   * Functions to change the representation of a date from a string containing
3.   * day, month name and year to three integers (month day year) and vice versa
4.   */
5.
6.  #include <stdio.h>
7.  #include <string.h>
8.
9.  #define STRSIZ 40
10. char *nums_to_string_date(char *date_string, int month, int day,
11.                           int year, const char *month_names[]);
```

(continued)

FIGURE 8.18 (continued)

```
12. int search(const char *arr[], const char *target, int n);
13. void string_date_to_nums(const char *date_string, int *monthp,
14.                          int *dayp, int *yearp, const char *month_names[]);
15.
16. /* Tests date conversion functions                                        */
17. int
18. main(void)
19. {
20.     char *month_names[12] = {"January", "February", "March", "April", "May",
21.                              "June", "July", "August", "September", "October",
22.                              "November", "December"};
23.     int m, y, mon, day, year;
24.     char date_string[STRSIZ];
25.     for  (y = 1993;  y < 2010;  y += 10)
26.         for (m = 1; m <= 12; ++m) {
27.             printf("%s", nums_to_string_date(date_string,
28.                                                 m, 15, y, month_names));
29.             string_date_to_nums(date_string, &mon, &day, &year, month_names);
30.             printf(" = %d/%d/%d\n", mon, day, year);
31.         }
32.
33.     return (0);
34. }
```

```
35.
36. /*
37.  * Takes integers representing a month, day and year and produces a
38.  * string representation of the same date.
39.  */
40. char *
41. nums_to_string_date(char        *date_string,       /* output - string
42.                                                          representation     */
43.                    int          month,              /* input -                */
44.                    int          day,                /* representation         */
45.                    int          year,               /* as three numbers       */
46.                    const char  *month_names[])      /* input - string representa-
47.                                                         tions of months       */
48. {
49.       sprintf(date_string, "%d %s %d", day, month_names[month - 1], year);
50.       return (date_string);
51. }
52.
```

(continued)

FIGURE 8.18 (continued)

```
53. #define NOT_FOUND -1    /* Value returned by search function if target
54.                            not found                                    */
55.
56. /*
57.  * Searches for target item in first n elements of array arr
58.  * Returns index of target or NOT_FOUND
59.  * Pre: target and first n elements of array arr are defined and n>0
60.  */
61. int
62. search(const char *arr[],        /* array to search                      */
63.        const char *target,      /* value searched for                   */
64.        int n)                   /* number of array elements to search   */
65. {
66.        int i,
67.            found = 0,     /* whether or not target has been found      */
68.            where;         /* index where target found or NOT_FOUND     */
69.
70.        /* Compares each element to target                               */
71.        i = 0;
72.        while (!found && i < n) {
73.            if (strcmp(arr[i], target) == 0)
74.                  found = 1;
75.             else
76.                  ++i;
77.        }
78.
79.        /* Returns index of element matching target or NOT_FOUND */
80.        if (found)
81.              where = i;
82.        else
83.              where = NOT_FOUND;
84.        return (where);
85. }
```

```
86.
87. /*
88.  * Converts date represented as a string containing a month name to
89.  * three integers representing month, day, and year
90.  */
91. void
92. string_date_to_nums(const char *date_string,   /* input - date to convert     */
93.                     int        *monthp,        /* output - month number       */
94.                     int        *dayp,          /* output - day number         */
95.                     int        *yearp,         /* output - year number        */
96.                     const char *month_names[]) /* input - names used in
97.                                                   date string                 */
98. {
99.       char  mth_nam[STRSIZ];
100.      int   month_index;
101.
102.      sscanf(date_string, "%d%s%d", dayp, mth_nam, yearp);
103.
104.      /* Finds array index (range 0..11) of month name.                        */
105.      month_index = search(month_names, mth_nam, 12);
106.      *monthp = month_index + 1;
107. }

15 January 1993 = 1/15/1993
15 February 1993 = 2/15/1993
. . .
15 December 2003 = 12/15/2003
```

Text Editor

Case Study

Problem

- Design and implement a program to perform editing operations on a line of text.
- Your editor should be able to
  - locate a specified target substring
  - delete a substring
  - insert a substring at a specified location.
- The editor should expect source strings of less than 80 characters.

INITIAL ALGORITHM

1. Scan the string to be edited into source.
2. Get an edit command.
3. while `command` isn't `Q`
    4. Perform edit operation.
    5. Get an edit command.

ALGORITHM FOR DO_EDIT

1. switch `command`

`'D'`:
2. Get the substring to be deleted (`str`).
3. Find the position of `str` in `source`.
4. if `str` is found, delete it.

`'I'`:
5. Get the substring to insert (`str`).
6. Get position of insertion (`index`).
7. Perform insertion of `str` at `index` position of `source`.

`'F'`:
8. Get the substring to search for (`str`).
9. Find the position of `str` in `source`.
10. Report position.

Otherwise:
11. Display error message.

FIGURE 8.19 Structure Chart for Text Editor Program

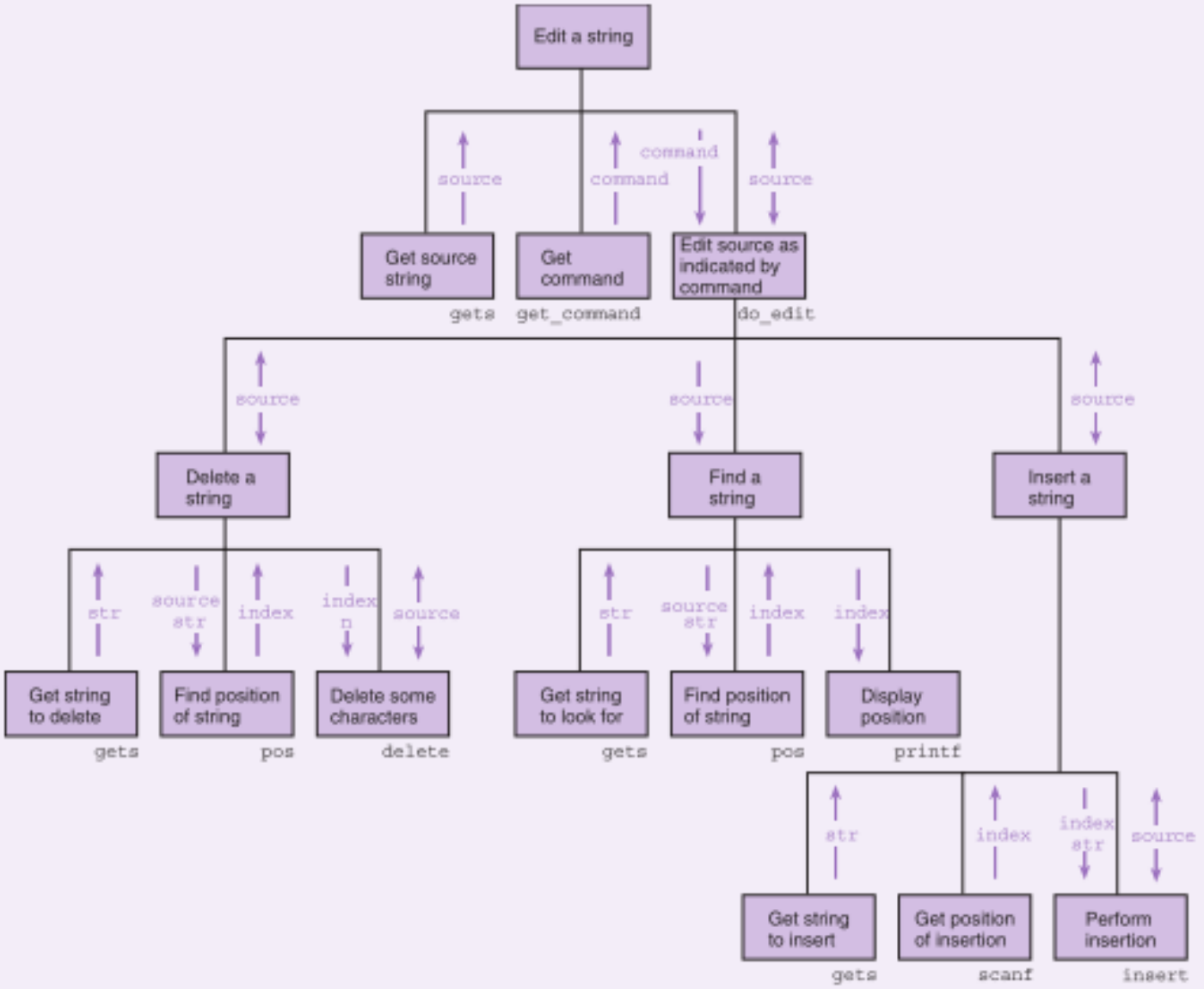


FIGURE 8.20 Text Editor Program

```
1.  /*
2.   * Performs text editing operations on a source string
3.   */
4.
5.  #include <stdio.h>
6.  #include <string.h>
7.  #include <ctype.h>
8.
9.  #define MAX_LEN 100
10. #define NOT_FOUND -1
11.
12. char *delete(char *source, int index, int n);
13. char *do_edit(char *source, char command);
14. char get_command(void);
15. char *insert(char *source, const char *to_insert, int index);
16. int  pos(const char *source, const char *to_find);
17.
18. int
19. main(void)
20. {
21.       char source[MAX_LEN], command;
22.       printf("Enter the source string:\n> ");
23.       gets(source);
24.
25.       for (command = get_command();
26.            command != 'Q';
27.            command = get_command()) {
28.           do_edit(source, command);
29.           printf("New source: %s\n\n", source);
```

(continued)

FIGURE 8.20 (continued)

```
30.       }
31.
32.       printf("String after editing: %s\n", source);
33.       return (0);
34. }
35.
36. /*
37.  * Returns source after deleting n characters beginning with source[index].
38.  * If source is too short for full deletion, as many characters are
39.  * deleted as possible.
40.  * Pre: All parameters are defined and
41.  *      strlen(source) - index - n < MAX_LEN
42.  * Post: source is modified and returned
43.  */
44. char *
45. delete(char *source,  /* input/output - string from which to delete part   */
46.        int   index,     /* input - index of first char to delete           */
47.        int   n)         /* input - number of chars to delete               */
48. {
49.        char rest_str[MAX_LEN]; /* copy of source substring following
50.                                   characters to delete */
51.
52.        /* If there are no characters in source following portion to
53.           delete, delete rest of string */
54.        if (strlen(source) <= index + n) {
55.              source[index] = '\0';
56.
57.        /* Otherwise, copy the portion following the portion to delete
58.           and place it in source beginning at the index position          */
59.        } else {
60.              strcpy(rest_str, &source[index + n]);
61.              strcpy(&source[index], rest_str);
62.        }
63.
64.        return (source);
65. }
66.
67. /*
68.  * Performs the edit operation specified by command
69.  * Pre: command and source are defined.
```

(continued)

FIGURE 8.20 (continued)

```
 * Post: After scanning additional information needed, performs a
 *       deletion (command = 'D') or insertion (command = 'I') or
 *       finds a substring ('F') and displays result; returns
 *       (possibly modified) source.
 */
char *
do_edit(char *source, /* input/output - string to modify or search */
        char command) /* input - character indicating operation    */
{
        char str[MAX_LEN]; /* work string */
        int index;

        switch (command) {
        case 'D':
              printf("String to delete> ");
              gets(str);
              index = pos(source, str);
              if (index == NOT_FOUND)
                    printf("'%s' not found\n", str);
              else
                    delete(source, index, strlen(str));
              break;

       case 'I':
            printf("String to insert> ");
            gets(str);
            printf("Position of insertion> ");
            scanf("%d", &index);
            insert(source, str, index);
            break;

       case 'F':
             printf("String to find> ");
             gets(str);
             index = pos(source, str);
             if (index == NOT_FOUND)
                  printf("'%s' not found\n", str);
             else
                  printf("'%s' found at position %d\n", str, index);
             break;
```

(continued)

FIGURE 8.20 (continued)

```
110.
111.       default:
112.             printf("Invalid edit command '%c'\n", command);
113.       }
114.
115.       return (source);
116. }
117.
118. /*
119.  * Prompt for and get a character representing an edit command and
120.  * convert it to uppercase. Return the uppercase character and ignore
121.  * rest of input line.
122.  */
123. char
124. get_command(void)
125. {
126.       char command, ignore;
127.
128.       printf("Enter D(Delete), I(Insert), F(Find), or Q(Quit)> ");
129.       scanf(" %c", &command);
130.
131.       do
132.            ignore = getchar();
133.       while (ignore != '\n');
134.
135.       return (toupper(command));
136. }
137.
138. /*
139.  * Returns source after inserting to_insert at position index of
140.  * source. If source[index] doesn't exist, adds to_insert at end of
141.  * source.
142.  * Pre:  all parameters are defined, space available for source is
143.  *       enough to accommodate insertion, and
144.  *       strlen(source) - index = n < MAX_LEN
145.  * Post: source is modified and returned
146.  */
147. char *
148. insert(char       *source,     /* input/output - target of insertion */
149.       const char *to_insert,  /* input - string to insert           */
```

(continued)

FIGURE 8.20 (continued)

```
       int         index)      /* input - position where to_insert
                                  is to be inserted         */
{
      char rest_str[MAX_LEN]; /* copy of rest of source beginning
                                 with source[index] */

      if (strlen(source) <= index) {
            strcat(source, to_insert);
      } else {
            strcpy(rest_str, &source[index]);
            strcpy(&source[index], to_insert);
            strcat(source, rest_str);
      }

      return (source);
}

/*
 * Returns index of first occurrence of to_find in source or
 * value of NOT_FOUND if to_find is not in source.
 * Pre: both parameters are defined
 */
int
pos(const char *source,  /* input - string in which to look for to_find */
    const char *to_find) /* input - string to find                      */

{
        int i = 0, find_len, found = 0, position;
        char substring[MAX_LEN];

        find_len = strlen(to_find);
        while (!found  &&  i <= strlen(source) - find_len) {
            strncpy(substring, &source[i], find_len);
            substring[find_len] = '\0';

            if (strcmp(substring, to_find) == 0)
                  found = 1;
            else
                  ++i;
        }

```

(continued)

FIGURE 8.20 (continued)

```
192.        if (found)
193.              position = i;
194.        else
195.              position = NOT_FOUND;
196.
197.        return (position);
198. }
```

FIGURE 8.21 Sample Run of Text Editor Program

```
Enter the source string:
> Internet use is growing rapidly.
Enter D(Delete), I(Insert), F(Find), or Q(Quit)> d
String to delete> growing
New source: Internet use is rapidly.

Enter D(Delete), I(Insert), F(Find), or Q(Quit)> F
String to find> .
'.' found at position 23
New source: Internet use is rapidly.

Enter D(Delete), I(Insert), F(Find), or Q(Quit)> I
String to insert> expanding
Position of insertion> 23

New source: Internet use is rapidly expanding.

Enter D(Delete), I(Insert), F(Find), or Q(Quit)> q
String after editing: Internet use is rapidly expanding.
```

Wrap Up

- Strings in C are arrays of characters terminated by the null character '\0'.
- String input is done using
  - scanf and fscanf for strings separated by whitespace
  - gets and fgets for input of while lines
  - getchar and getc for single character input

Wrap Up

- The string library provides functions for
  - assignment and extraction
  - string length
  - concatenation
  - alphabetic comparison
- The standard I/O library includes functions for
  - string-to-number conversion
  - number-to-string conversion