"Trying to outsmart a compiler defeats much of the purpose of using one."

- Kernighan & Plauger, The Elements of Programming Style

CSE102
Computer Programming with C

2016-2017 Fall Semester

Strings

Largely adapted from J.R. Hanly, E.B. Koffman, F.E. Sevilgen, and others…

Introduction

- String: group of characters
  - Implemented as arrays of char
  - Essential for several applications manipulating textual data
    - Word processing
    - Databases
    - Scientific computing (Ex: DNA sequence, chemical compounds)
  - Already used string constants
    - printf and scanf format strings

String Variables

- Declaration: same as declaring array of chars

```
char string_var[30];
```

- The variable string_var can hold a string of 0 to 29 characters
  - Not 30!..
  - How is varying size handled?
  - Use of null character: '\0'
- String variables can be initialized

```
char string_var[30] = "initial value";
char str[] = "initial value";
```

- What is the size of str?
- The part of array after null character is ignored

Arrays of Strings

- An array of strings: a two-dimensional array of chars
  - Ex: Array of 30 names which is less than 25 characters

```
#define NUM_PEOPLE 30
#define NAME_LEN 25

char names[NUM_PEOPLE][NAME_LEN];
```

  - Ex: Array of 12 month names

```
char months[12][10] = {
               “January”, “February”, “March”, “April”,
               “June”, “July”, “August”, “September”,
               “October”, “November”, “December”};
```

Input/Output of Strings

- Place holder: “%s”
- printf prints characters until null character

```
printf(“The value is: %s \n”, string_var);
```

- What if the array does not contain null character?
- Do not forget to insert null character while building strings
  - This is automatic for constant strings

```
printf(“***%7s**** \n”, “John”);
printf(“***%7s**** \n”, “Marry”);
printf(“***%-7s**** \n”, “Sam”);
```

Input/Output of Strings

- Place holder: “%s”
- scanf can used to input strings

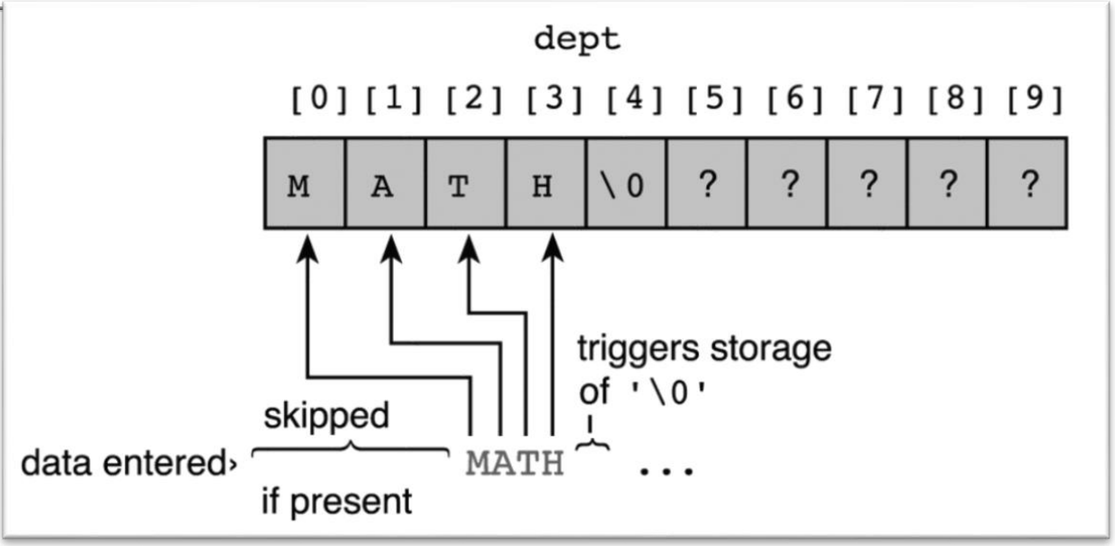
```
scanf(“%s”, string_var);
```

  - Remember string_var is an array
- scanf
  - skips leading whitespace characters
  - copies subsequent characters in memory cells
  - copying stops when a whitespace character is seen
  - places a null character at the end of string

- EX: See following simple example..

Input/Output of Strings

```
#include <stdio.h>

#define STRING_LEN  10

int
main(void)
{
      char dept[STRING_LEN];
      int  course_num;
      char days[STRING_LEN];
      int  time;

      printf("Enter department code, course number, days and ");
      printf("time like this:\n> COSC 2060 MWF 1410\n> ");
      scanf("%s%d%s%d", dept, &course_num, days, &time);
      printf("%s %d meets %s at %d\n", dept, course_num, days, time);

      return (0);
}
```



```
Enter department code, course number, days and time like this:
> COSC 2060 MWF 1410
> MATH 1270 TR 800
MATH 1270 meets TR at 800
```

Input/Output of Strings

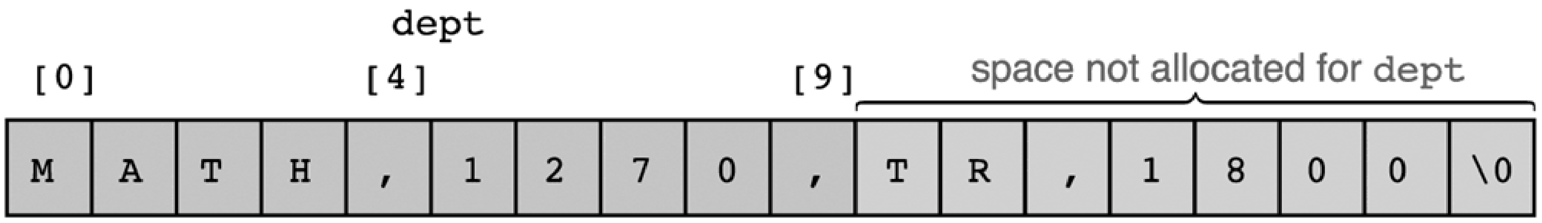
- How to enter the values in previous example?
  - In more than one line or in diferent formats?

```
>         MATH
    1270
                        TR
          1800
```

```
> MATH 1270
TR 1800
```

```
> MATH1270 TR 1800
```

```
> MATH,1270,TR,1800
```



Input/Output of Strings

- EX: Read in 30 names together with their ages

```
#define NUM_PEOPLE 30
#define NAME_LEN 25

char names[NUM_PEOPLE][NAME_LEN];
int ages[NUM_PEOPLE];

for(…..){
  …..
}
```

String Library Functions: Assignment

- Assignment operator: =
  - Used for assigning simple types
  - Can not be used for arrays and strings
    - Other than in declaration with initialization
    - What is array name without subscript?

```
char str[20];
str = “test value”;
```

- C provides library function for assignment
  - Library in string.h
  - Includes several operations
    - Substring functions, concatenation, comparison, length, etc…

String Library Functions

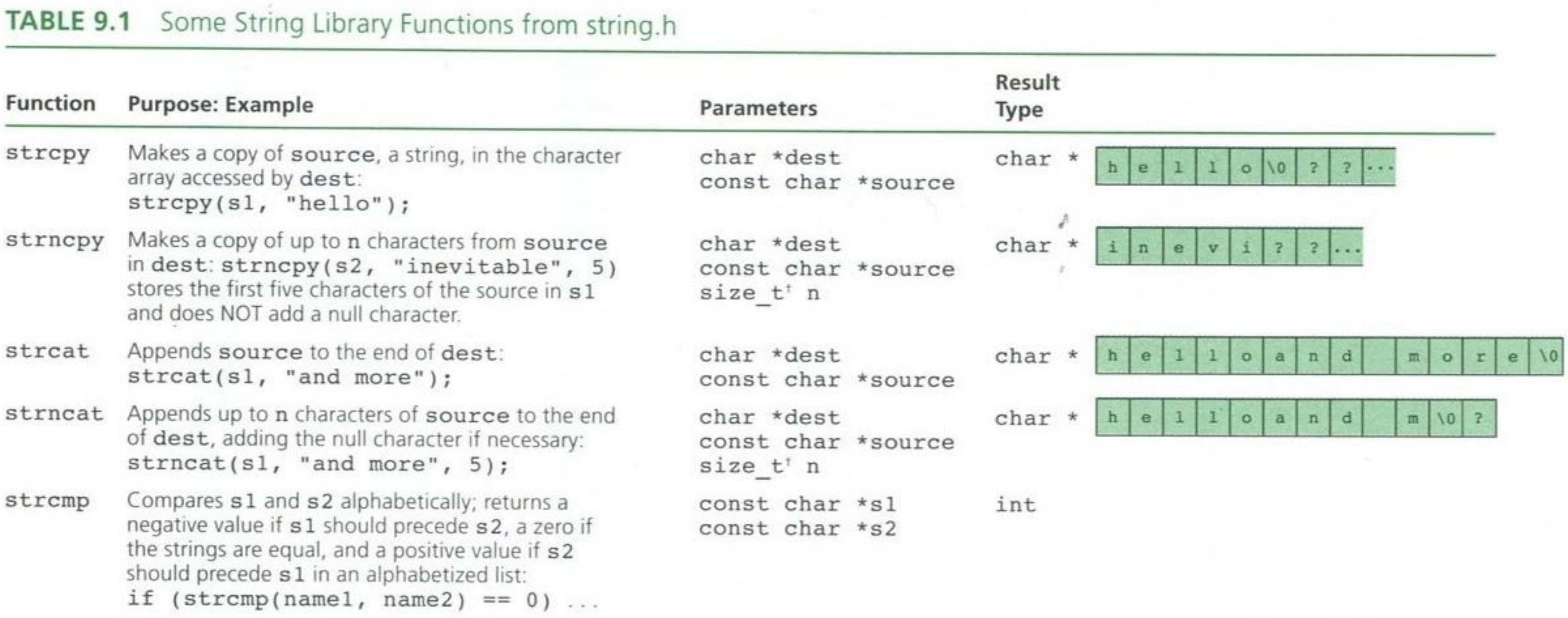
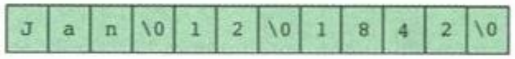


TABLE 9.1 Some String Library Functions from string.h

Function	Purpose: Example	Parameters	Result Type	
`strcpy`	Makes a copy of `source`, a string, in the character array accessed by `dest`: `strcpy(s1, "hello");`	`char *dest` `const char *source`	`char *`	h e l l o \0 ? ? ...
`strncpy`	Makes a copy of up to `n` characters from `source` in `dest`: `strncpy(s2, "inevitable", 5)` stores the first five characters of the source in `s1` and does NOT add a null character.	`char *dest` `const char *source` `size_t† n`	`char *`	i n e v i ? ? ...
`strcat`	Appends `source` to the end of `dest`: `strcat(s1, "and more");`	`char *dest` `const char *source`	`char *`	h e l l o a n d m o r e \0
`strncat`	Appends up to `n` characters of `source` to the end of `dest`, adding the null character if necessary: `strncat(s1, "and more", 5);`	`char *dest` `const char *source` `size_t† n`	`char *`	h e l l o a n d m \0 ?
`strcmp`	Compares `s1` and `s2` alphabetically; returns a negative value if `s1` should precede `s2`, a zero if the strings are equal, and a positive value if `s2` should precede `s1` in an alphabetized list: `if (strcmp(name1, name2) == 0) ...`	`const char *s1` `const char *s2`	`int`	

String Library Functions

`strncmp`	Compares the first `n` characters of `s1` and `s2` returning positive, zero, and negative values as does `strcmp`: `if (strncmp(n1, n2, 12) == 0) ...`	`const char *s1` `const char *s2` `size_t† n`	`int`	
`strlen`	Returns the number of characters in `s`, not counting the terminating null: `strlen("What")` returns 4.	`const char *s`	`size_t`	
`strtok`	Breaks parameter string source into tokens by finding groups of characters separated by any of the delimiter characters in delim. First call must provide both source and delim. Subsequent calls using NULL as the source string find additional tokens in original source. Alters source by replacing first delimiter following a token by '\0'. When no more delimiters remain, returns rest of source. For example, if `s1` is `"Jan.12,.1842"`, `strtok(s1,".,")` returns `"Jan"`, then `strtok (NULL,".,")` returns `"12"` and `strtok (NULL,".,")` returns `"1842"`.The memory in the right column shows the altered `s1` after the three calls to `strtok`. Return values are pointers to substrings of `s1` rather than copies.	`const char *source` `const char *delim`	`char *`	

String Assignment

- strcpy

```
strcpy(str, “test value”);
```

- Be careful about overflow!

```
strcpy(str, “A very long string test value ”);
```

- strncpy : copies first n characters

```
strncpy(str, “test value”, 20);
```

- Be careful to copy a valid string!

```
strncpy(str, “A very long string test value”, 20);
```

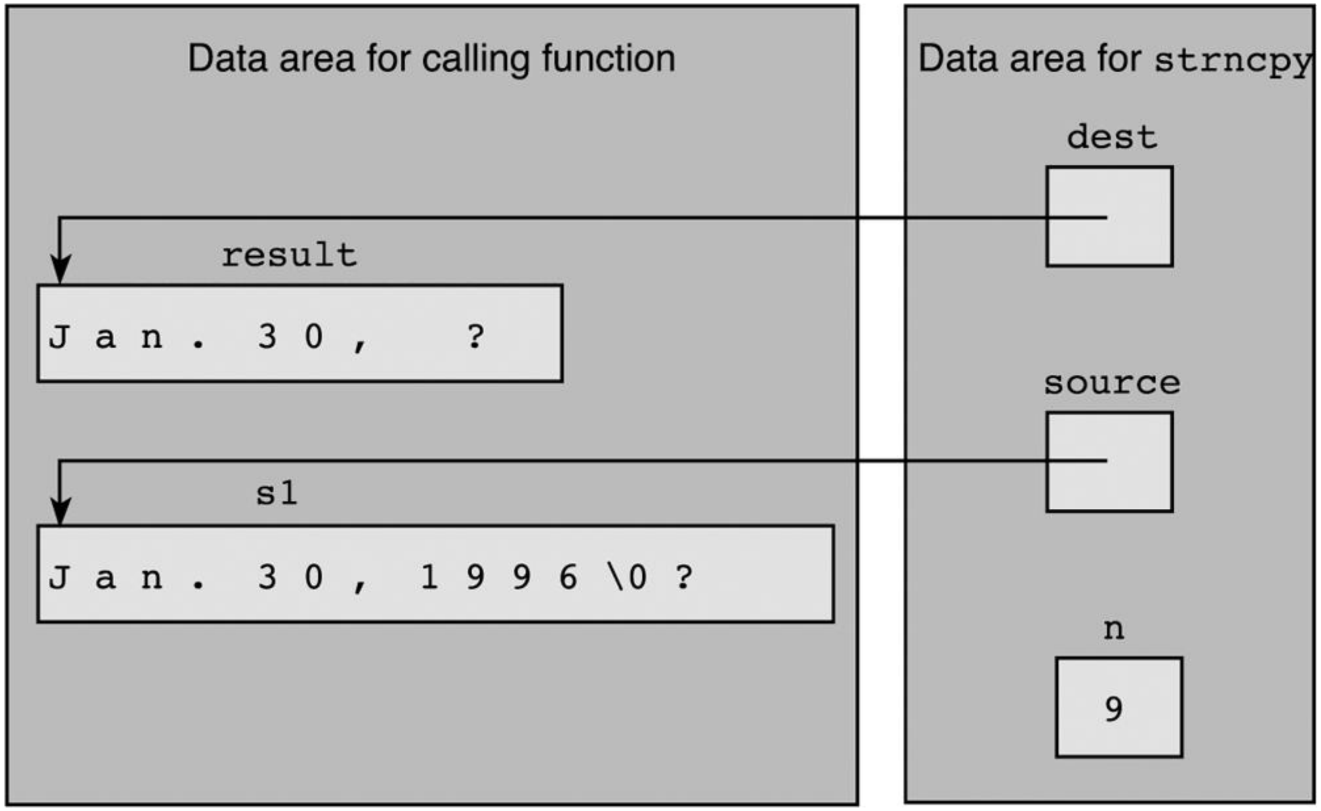
- Safer to use

```
strncpy(str, “A very long string test value”, 19);
str[19] = ‘\0’;
```

Substring copy

char result[10], s1[15] = “Jan. 30, 1996”;

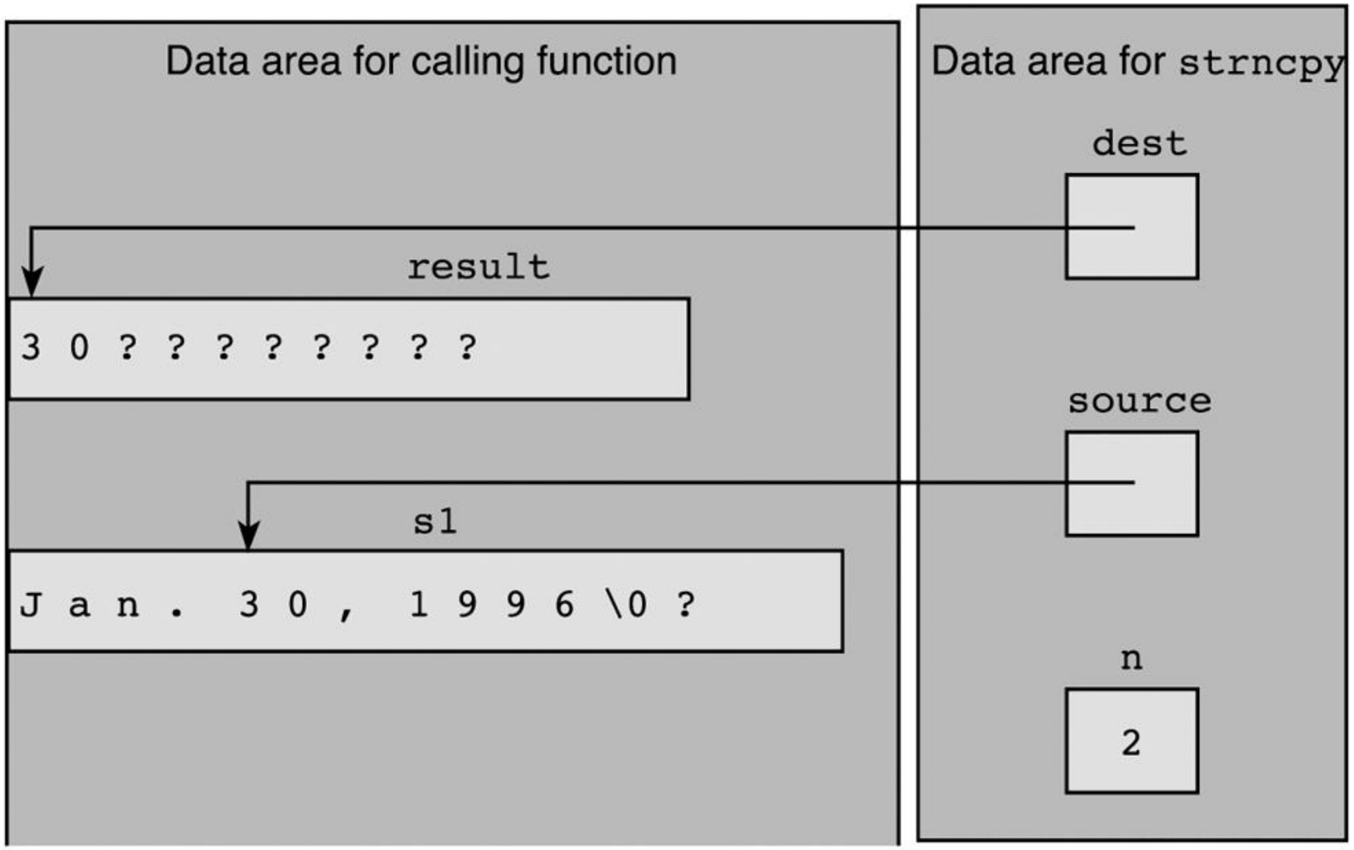
strncpy(result, s1, 9);



Substring copy

char result[10], s1[15] = “Jan. 30, 1996”;

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



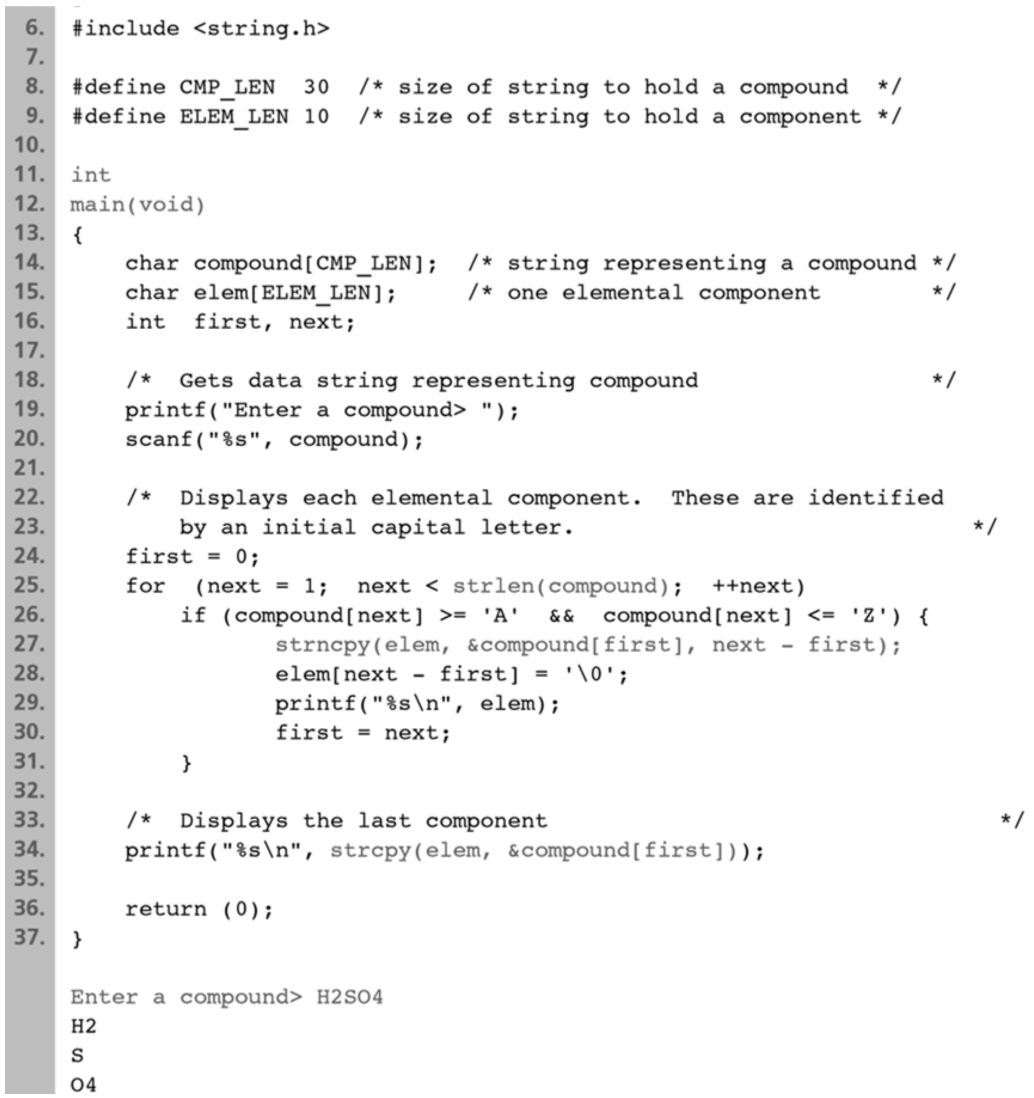
Substring copy

```
char result[10], s1[15] = “Jan. 30, 1996”;
strcpy(result, &s1[9]);
```

Separate Compounds into Elemental Components

- Ex: Break compound names into their elemental components
  - Assume element names start with a capital letter
  - Assume ASCII character set
- Use strncpy
  - to copy parts of compound names into elemental components
- Use strlen
  - To check termination of the loop

Separate Compounds into Elemental Components



```
#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
```

Concatenation

- Add a string at the end of the other string
- strcat and strncat
  - Assumes sufficient space available for the concatenated string

```
char f[15] = “Fatih ”, m[15] = “Erdogan ”, l[15] =
  “Sevilgen”;
strcat(f, l);
strcat(m, l);
printf(“%d  %d \n”, strlen(m), strlen(l))
strncat(m, l, 5);
printf(“%d \n”, strncat(m, l, 15 - strlen(m) - 1));
```

Scanning a Full Line

- Input one complete line of data
  - Do not stop at space or tab characters
  - Do not store end-of-line (new-line, return, enter) character

```
char line[80];
gets(line);
```

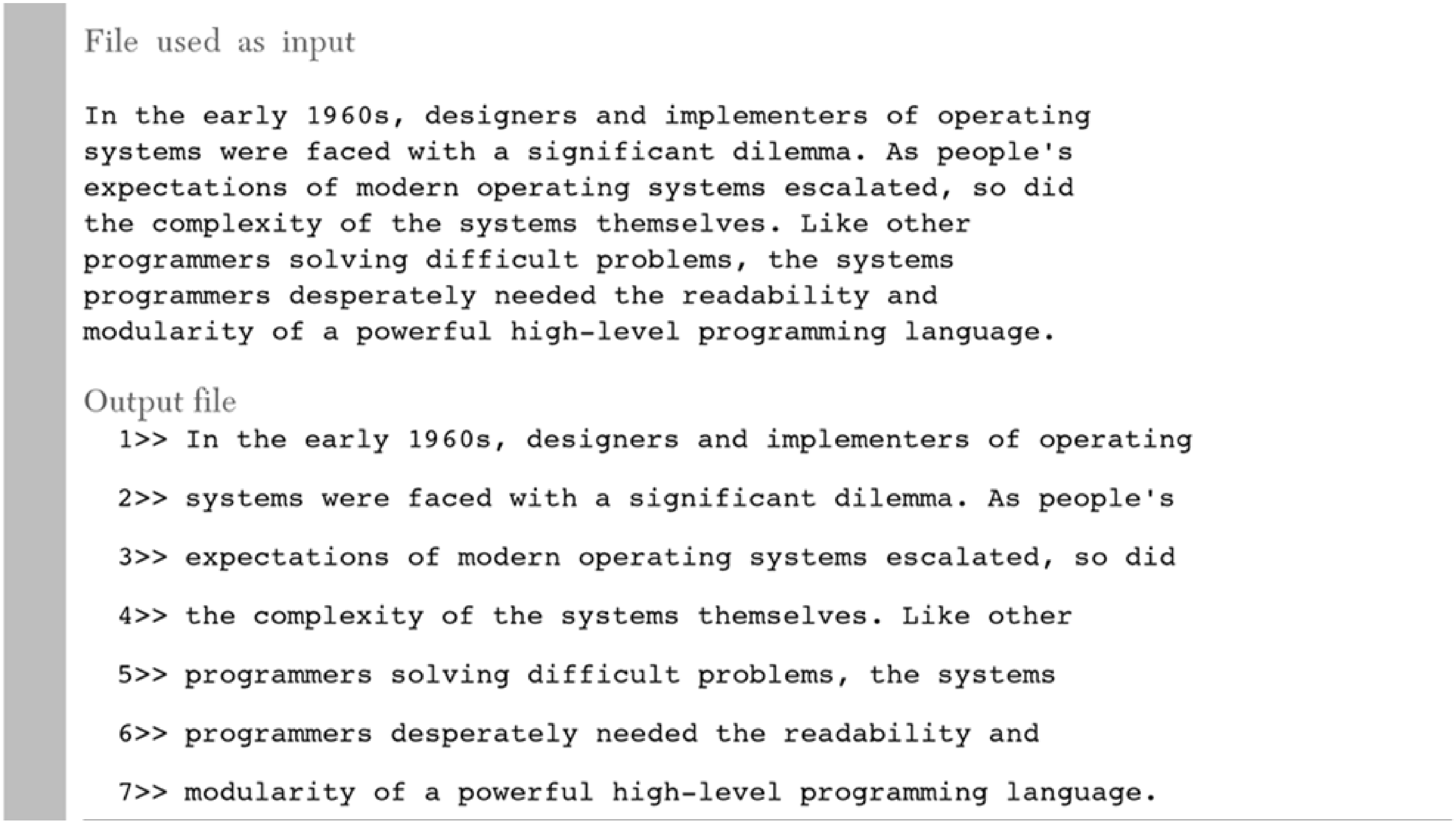
- File input, fgets has different format
  - Final character is always '\0'
  - Stores '\n' character if the line is not truncated

```
fgets(line, 80, inp);
```

char *fgets(char *str, int n, FILE *stream)

Scanning a Full Line

- Ex: Scans a data file and create a new double-spaced version with line numbers

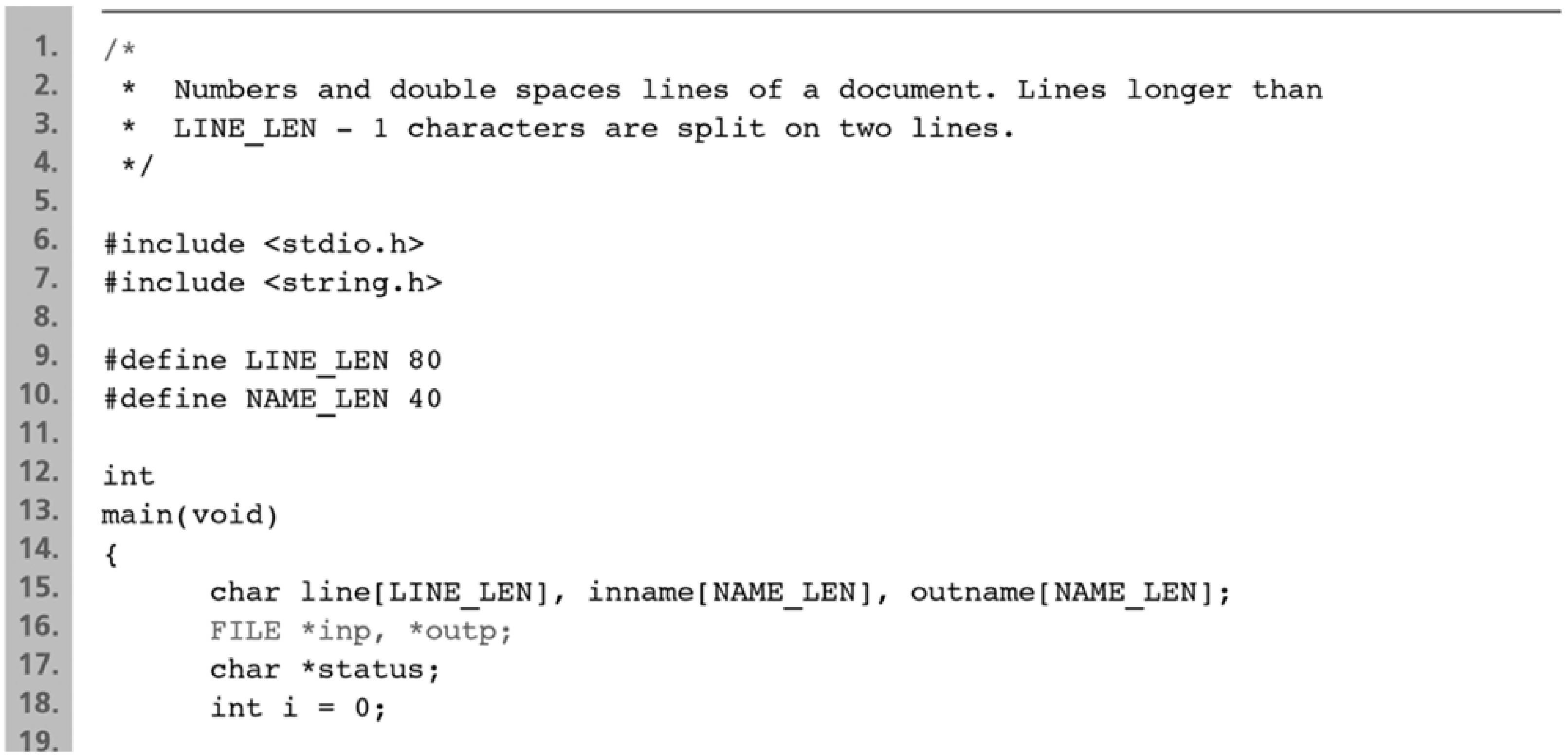


File used as input

```
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.
```



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

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

#define LINE_LEN 80
#define NAME_LEN 40

int
main(void)
{
      char line[LINE_LEN], inname[NAME_LEN], outname[NAME_LEN];
      FILE *inp, *outp;
      char *status;
      int i = 0;

```

```
        printf("Name of input file> ");
        scanf("%s", inname);
        printf("Name of output file> ");
        scanf("%s", outname);

        inp = fopen(inname, "r");
        outp = fopen(outname, "w");

        for  (status = fgets(line, LINE_LEN, inp);
              status != 0;
              status = fgets(line, LINE_LEN, inp)) {
            if (line[strlen(line) - 1] == '\n')
                  line[strlen(line) - 1] = '\0';
            fprintf(outp, "%3d>> %s\n\n", ++i, line);
        }
```

(continued)

String Comparison

- Comparison operators can not be used
  - Strings are implemented as arrays
  - What is the meaning of

    string1 < string2
  - strcmp: compares two strings and returns an integer

    strcmp(str1,str2)
    - Has negative value if str1 is less than str2
    - Has value 0 if str1 is equal to str2
    - Has positive value if str1 is greater than str2
  - strncmp: compares first n characters

Sentinel-Controlled Loop for String Input

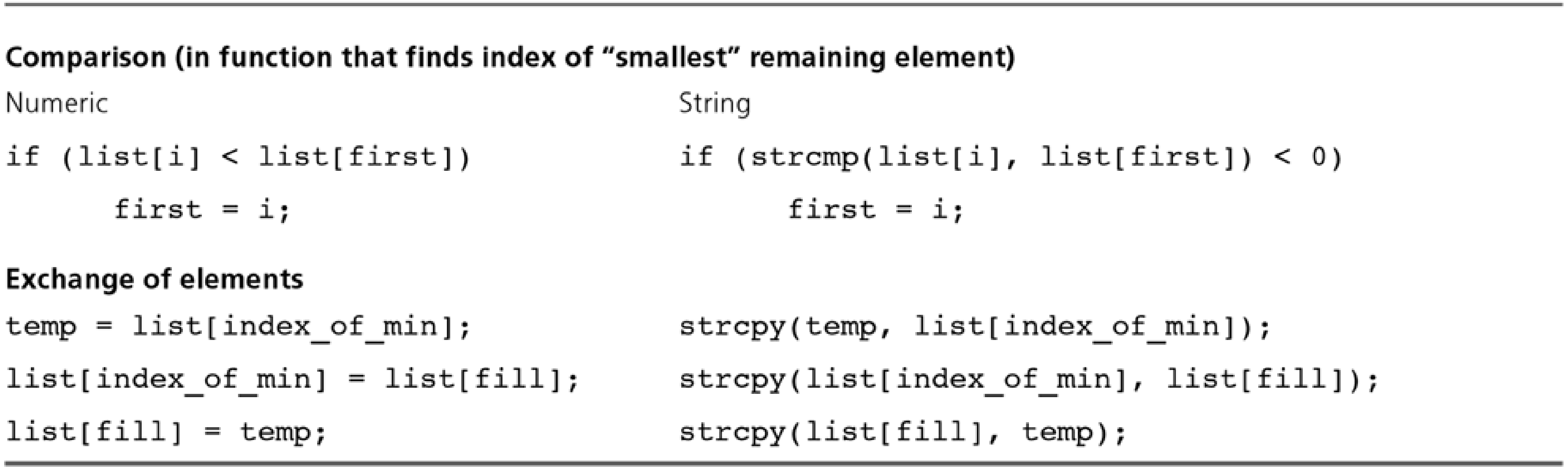
```
printf("Enter list of words on as many lines as you like.\n");
printf("Separate words by at least one blank.\n");
printf("When done, enter %s to quit.\n", SENT);

for  (scanf("%s", word);
      strcmp(word, SENT) != 0;
      scanf("%s", word)) {
    /* process word */
    ...
}
```

Sorting and Searching

- Sorting a list of words (array of strings)

  char list[30][20];

  – Comparison
  – Swap



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

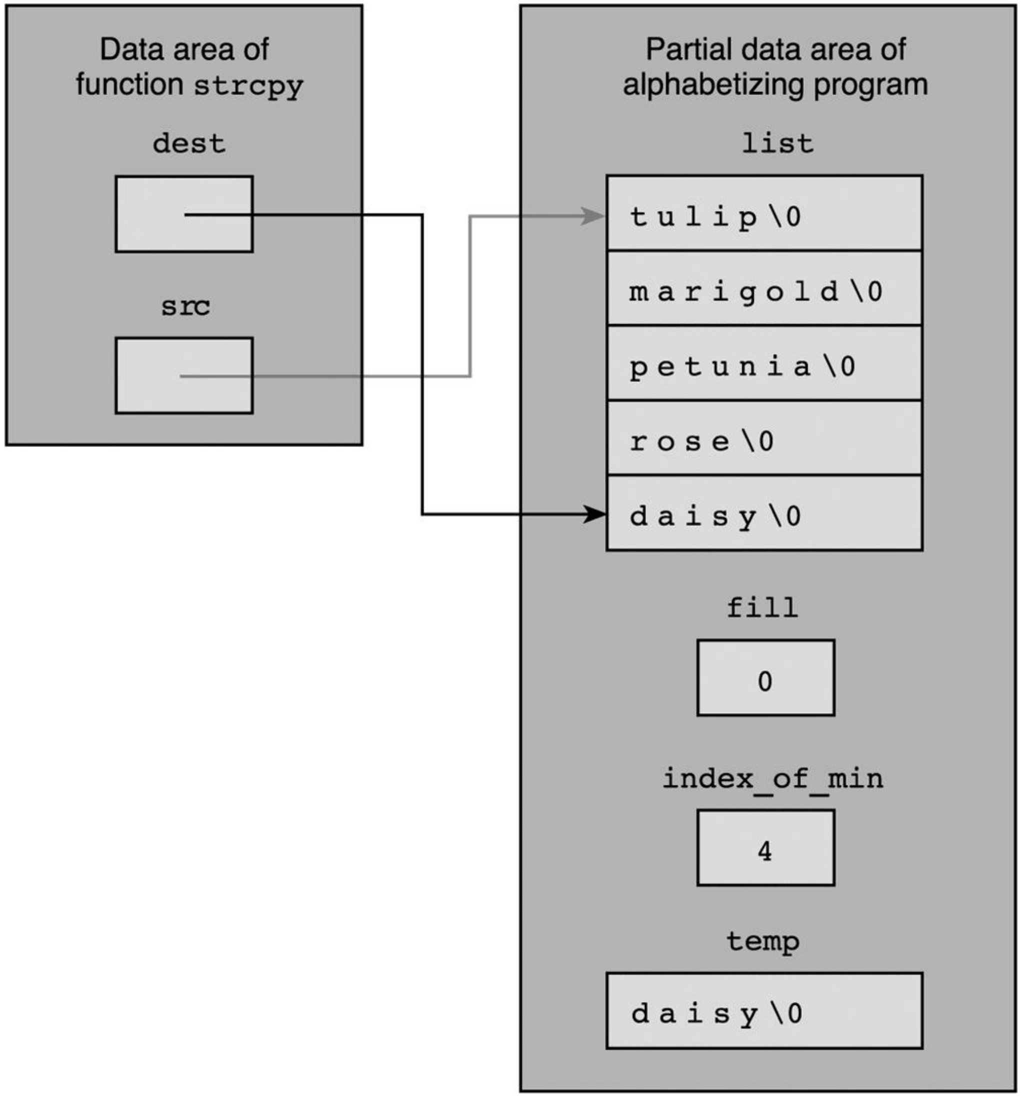
Numeric	String
`if (list[i] < list[first])`	`if (strcmp(list[i], list[first]) < 0)`
`first = i;`	`first = i;`

Exchange of elements

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

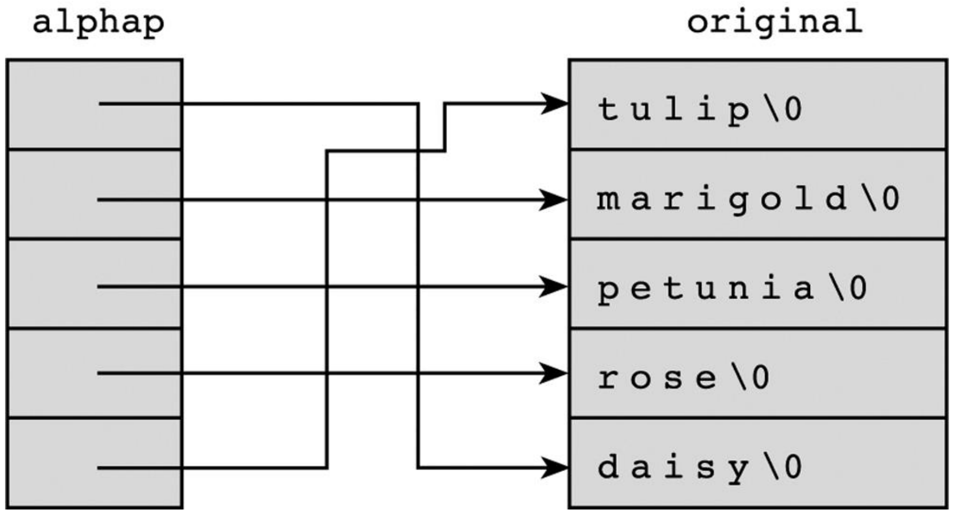
  – What do we mean by list[i]?

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



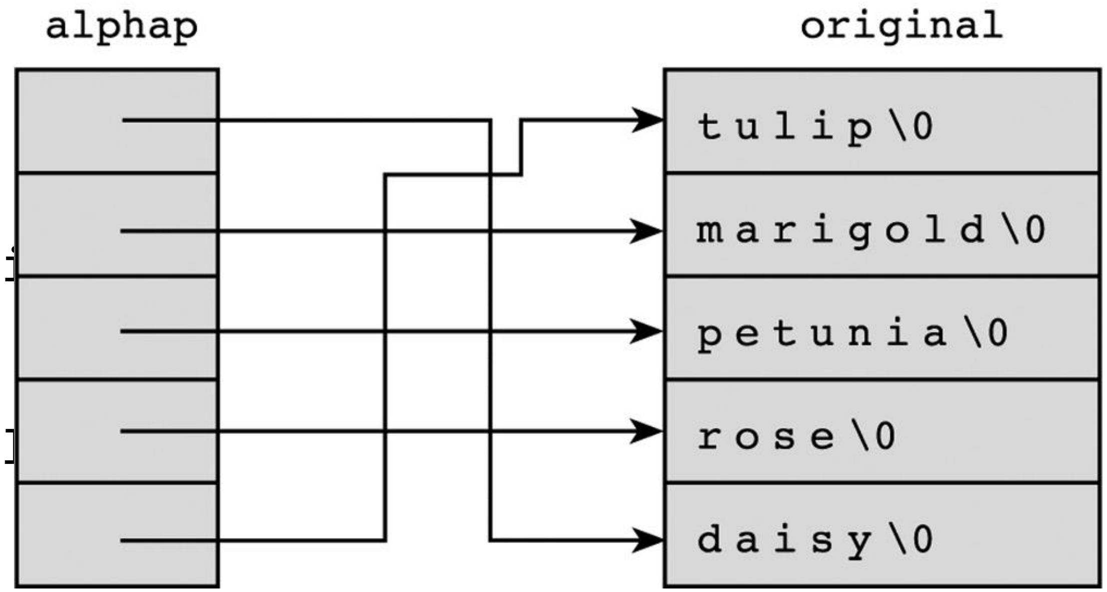
Arrays of Pointers

- Previous example requires a lot of copying of characters to sort a list of strings
  - Three copy operations per exchange
- Alternative approach: use arrays of pointers
  - Pointers to strings (arrays)
  - Sort the pointers not the strings
  - Saves the original order as well.



Arrays of Pointers

```
char orignal[5][10];

for(i = 0; i < 5; ++i)
   printf(“%s\n”, alphap[i

for(i = 0; i < 5; ++i)
   printf(“%s\n”, original
```



- How to define alphap array?

```
     char *alphap[5];
```

- How to initalize alphap array?

```
     for(i = 0; i < 5; ++i)
         alphap[i] = original[i];
```

Arrays of Pointers

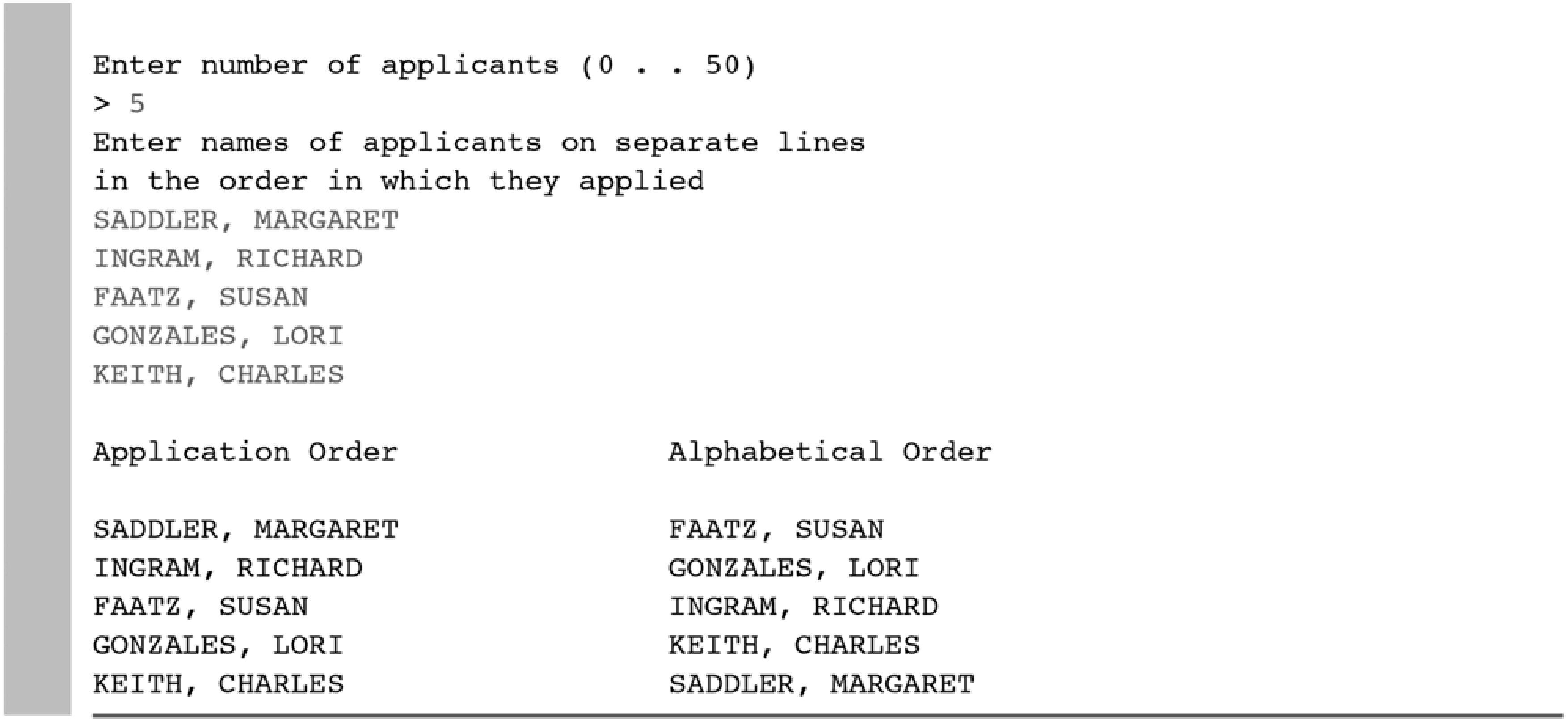
- Arrays of pointers has several advantages
  - Can represents many orderings
    - All refers to the same string
    - One corrected all corrected
  - Requires less space
    - Pointer vs string
  - Can sort faster

- Array of String constants

```
char months[12][10] = {“January”, “February”, “March”, “April”, “June”,
                        “July”, “August”, “September”, “October”,
                        “November”, “December”};
char *months[] = {“January”, “February”, “March”, “April”, “June”,
                        “July”, “August”, “September”, “October”,
                        “November”, “December”};
```

Arrays of Pointers

EX: Input a list of names and access it in sorted order and original order.



```
Enter number of applicants (0 . . 50)
> 5
Enter names of applicants on separate lines
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
```

Two Orderings of One List

```
/*
 *  Maintains two orderings of a list of applicants:  the original
 *  ordering of the data, and an alphabetical ordering accessed through an
 *  array of pointers.
 */

#include <stdio.h>
#define STRSIZ 30  /*  maximum string length */
#define MAXAPP 50  /*  maximum number of applications accepted */

int alpha_first(char *list[], int min_sub, int max_sub);
void select_sort_str(char *list[], int n);

int
main(void)
{
      char  applicants[MAXAPP][STRSIZ]; /* list of applicants in the
                                           order in which they applied       */
      char *alpha[MAXAPP];              /* list of pointers to
                                           applicants                        */
      int   num_app,                    /* actual number of applicants       */
            i;
      char  one_char;
```

```c
#define LINE_LEN 80
#define NAME_LEN 40

int
main(void)
{
      char line[LINE_LEN], inname[NAME_LEN], outname[NAME_LEN];
      FILE *inp, *outp;
      char *status;
      int i = 0;

      printf("Name of input file> ");
      scanf("%s", inname);
      printf("Name of output file> ");
      scanf("%s", outname);

      inp = fopen(inname, "r");
      outp = fopen(outname, "w");

      for  (status = fgets(line, LINE_LEN, inp);
            status != 0;
            status = fgets(line, LINE_LEN, inp)) {
          if (line[strlen(line) - 1] == '\n')
                line[strlen(line) - 1] = '\0';
          fprintf(outp, "%3d>> %s\n\n", ++i, line);
      }
```

(continued)

```
int
main(void)
{
        char  applicants[MAXAPP][STRSIZ]; /* list of applicants in the
                                             order in which they applied       */
        char *alpha[MAXAPP];              /* list of pointers to
                                             applicants                        */
        int   num_app,                    /* actual number of applicants       */
              i;
        char  one_char;

        /*  Gets applicant list                                                */
        printf("Enter number of applicants (0 . . %d)\n> ", MAXAPP);
        scanf("%d", &num_app);
        do    /* skips rest of line after number */
            scanf("%c", &one_char);
        while (one_char != '\n');

        printf("Enter names of applicants on separate lines\n");
        printf("in the order in which they applied\n");
        for  (i = 0;  i < num_app; ++i)
            gets(applicants[i]);

        /*  Fills array of pointers and sorts                                  */
        for  (i = 0;  i < num_app;  ++i)
            alpha[i] = applicants[i];  /* copies ONLY address */
        select_sort_str(alpha, num_app);
        /*  Displays both lists                                                */
        printf("\n\n%-30s%5c%-30s\n\n", "Application Order", ' ',
                  "Alphabetical Order");
        for  (i = 0;  i < num_app;  ++i)
            printf("%-30s%5c%-30s\n", applicants[i], ' ', alpha[i]);

        return(0);
}
```

```
/*
 *  Finds the index of the string that comes first alphabetically in
 *  elements min_sub..max_sub of list
 *  Pre:  list[min_sub] through list[max_sub] are of uniform case;
 *          max_sub >= min_sub
 */
int
alpha_first(char *list[],          /* input - array of pointers to strings     */
            int    min_sub,        /* input - minimum and maximum subscripts   */
            int    max_sub)        /*   of portion of list to consider         */
{
      int first, i;

      first = min_sub;
      for  (i = min_sub + 1;  i <= max_sub;  ++i)
          if (strcmp(list[i], list[first]) < 0)
                first = i;

      return (first);
}
```

```c
/*
 *  Orders the pointers in array list so they access strings
 *  in alphabetical order
 *  Pre:  first n elements of list reference strings of uniform case;
 *        n >= 0
 */
void
select_sort_str(char *list[], /* input/output - array of pointers being
                                 ordered to access strings alphabetically */
                int   n)      /* input - number of elements to sort       */
{

      int   fill,          /* index of element to contain next string in order */
            index_of_min; /* index of next string in order */
      char *temp;

      for  (fill = 0;  fill < n - 1;  ++fill) {
          index_of_min = alpha_first(list, fill, n - 1);

          if (index_of_min != fill) {
                temp = list[index_of_min];
                list[index_of_min] = list[fill];
                list[fill] = temp;
          }
      }
}
```

Character Operations

- Strings processing usually requires character manipulation
- Character library provides several functions
  - Include ctype.h

Character I/O

- getchar
  - returns the next character from standard input
  - Return value of getchar is an integer.
    - Return EOF if getchar end-of-file is reached.
    - The value of EOF is -1 which is not of type char

```
ch = getchar();        scanf("%c",&ch);
```

- getc: get a single character from a file
- putchar and putc are used to display a character

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

scanline Function Using getchar

```c
/*
 *  Gets one line of data from standard input.  Returns an empty string on
 *  end of file.  If data line will not fit in allotted space, stores
 *  portion that does fit and discards rest of input line.
 */
char *
scanline(char *dest,      /* output - destination string                      */
         int   dest_len) /* input  - space available in dest                  */
{
      int i, ch;

      /*  Gets next line one character at a time.                             */
      i = 0;
      for  (ch = getchar();
            ch != '\n'  &&  ch != EOF   &&  i < dest_len - 1;
            ch = getchar())
          dest[i++] = ch;
      dest[i] = '\0';

      /*  Discards any characters that remain on input line                   */
      while (ch != '\n'  &&  ch != EOF)
          ch = getchar();

      return (dest);
}
```

Character Analysis and Conversion

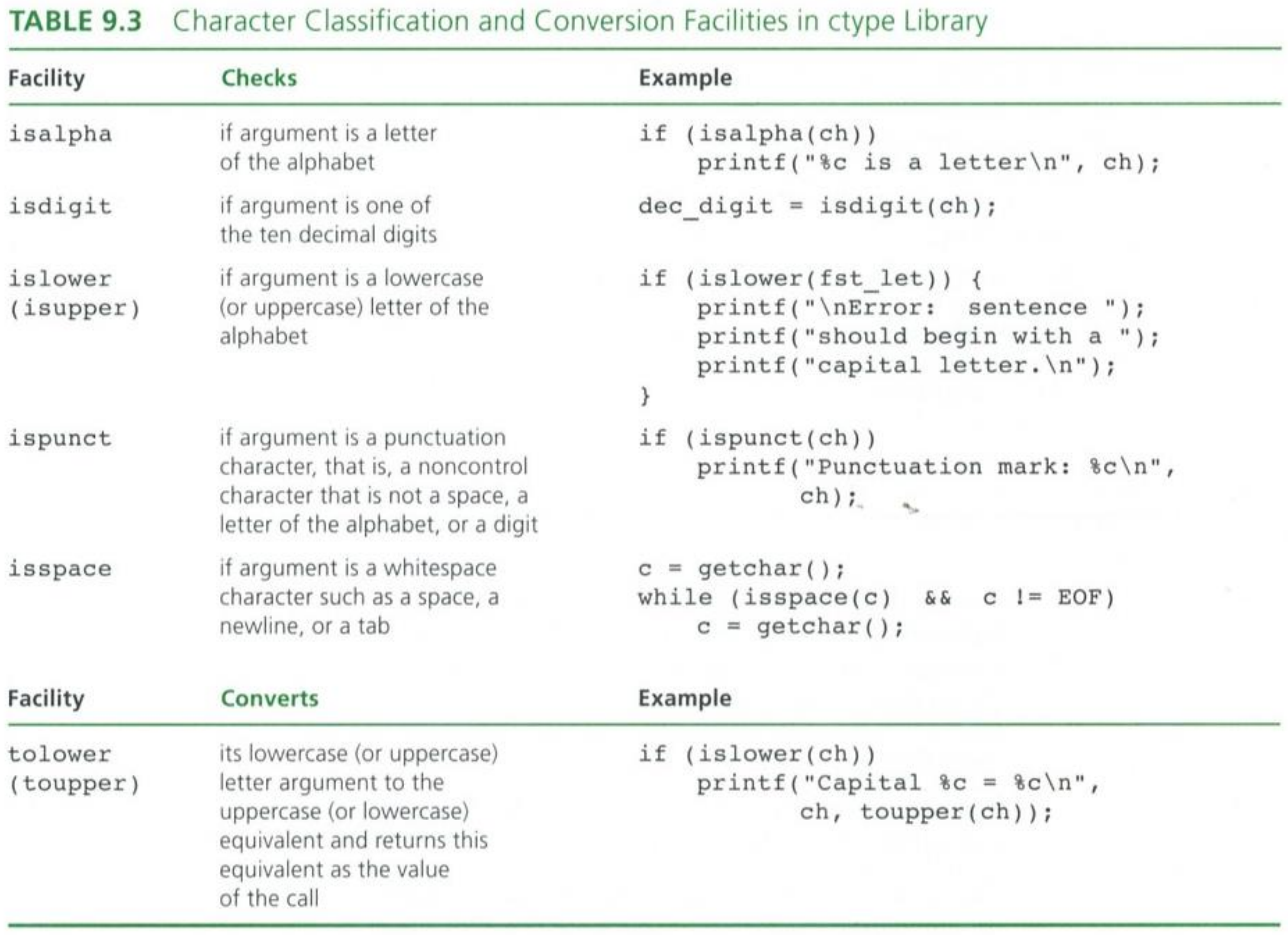


TABLE 9.3 Character Classification and Conversion Facilities in ctype Library

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");`
		`}`

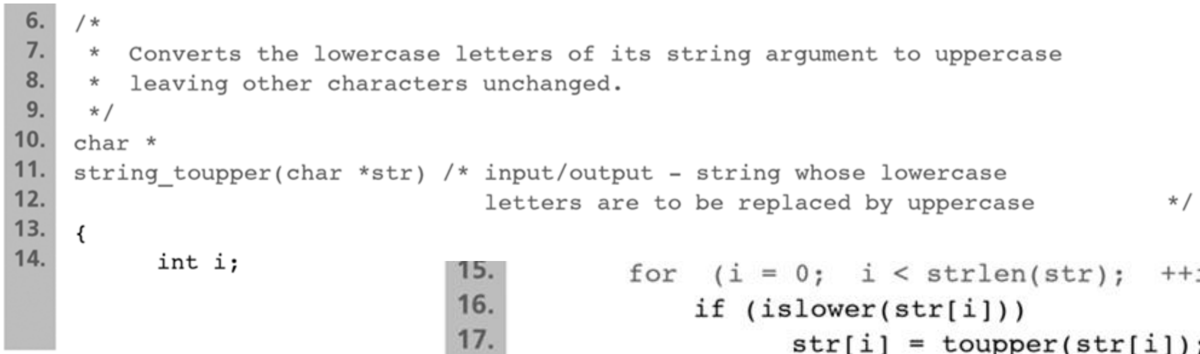
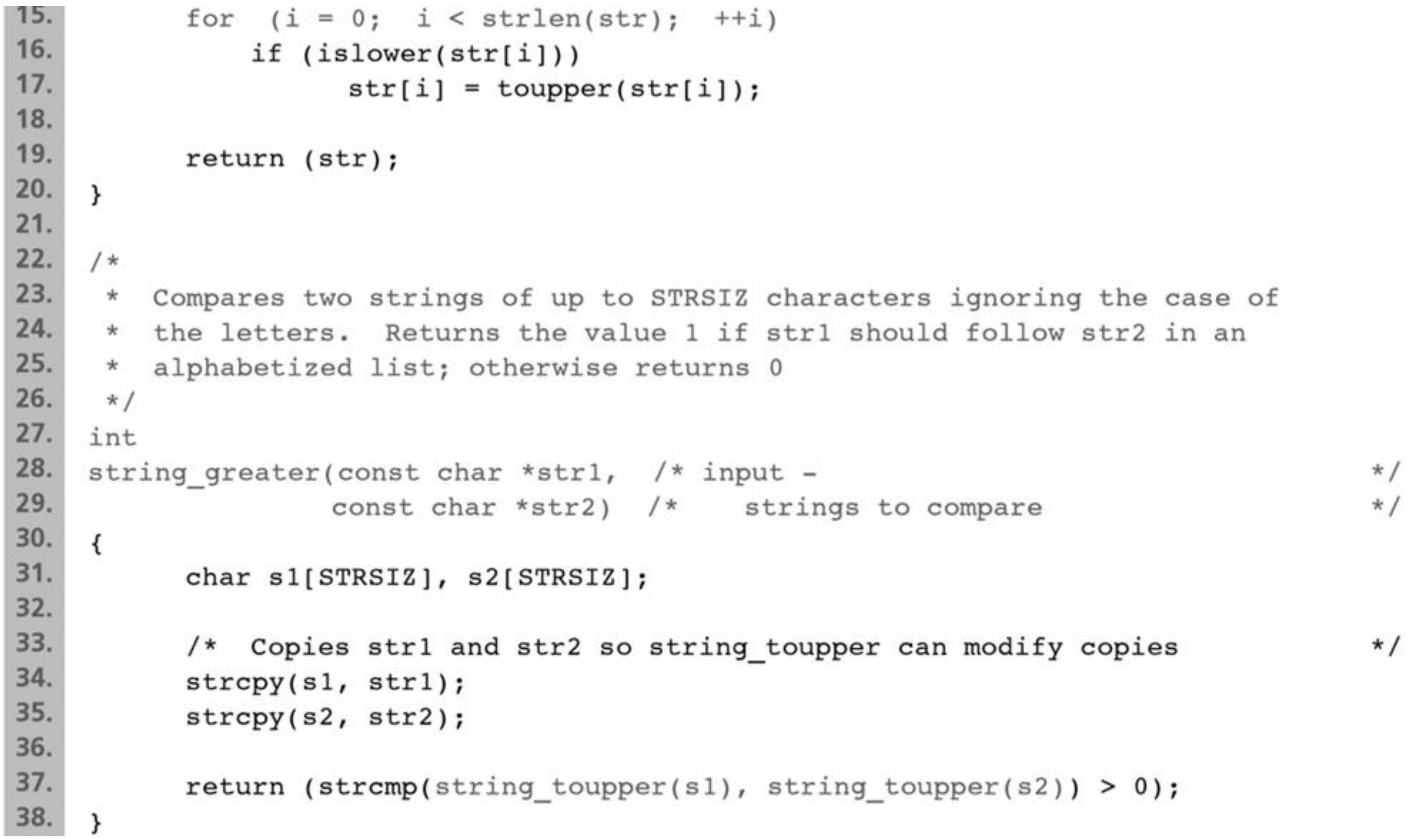
`ispunct`	if argument is a punctuation character, that is, a noncontrol character that is not a space, a letter of the alphabet, or a digit	`if (ispunct(ch))`
		`    printf("Punctuation mark: %c\n",`
		`           ch);`

`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));`

Greater-Than Operator Ignoring Case

```c
/*
 *  Converts the lowercase letters of its string argument to uppercase
 *  leaving other characters unchanged.
 */
char *
string_toupper(char *str) /* input/output - string whose lowercase
                             letters are to be replaced by uppercase         */
{
      int i;
      for  (i = 0;  i < strlen(str);  ++i)
          if (islower(str[i]))
                str[i] = toupper(str[i]);

      return (str);
}

/*
 *  Compares two strings of up to STRSIZ characters ignoring the case of
 *  the letters.  Returns the value 1 if str1 should follow str2 in an
 *  alphabetized list; otherwise returns 0
 */
int
string_greater(const char *str1,  /* input -                                */
               const char *str2)  /*    strings to compare                  */
{
      char s1[STRSIZ], s2[STRSIZ];

      /*  Copies str1 and str2 so string_toupper can modify copies          */
      strcpy(s1, str1);
      strcpy(s2, str2);

      return (strcmp(string_toupper(s1), string_toupper(s2)) > 0);
}
```

String-Number Conversion

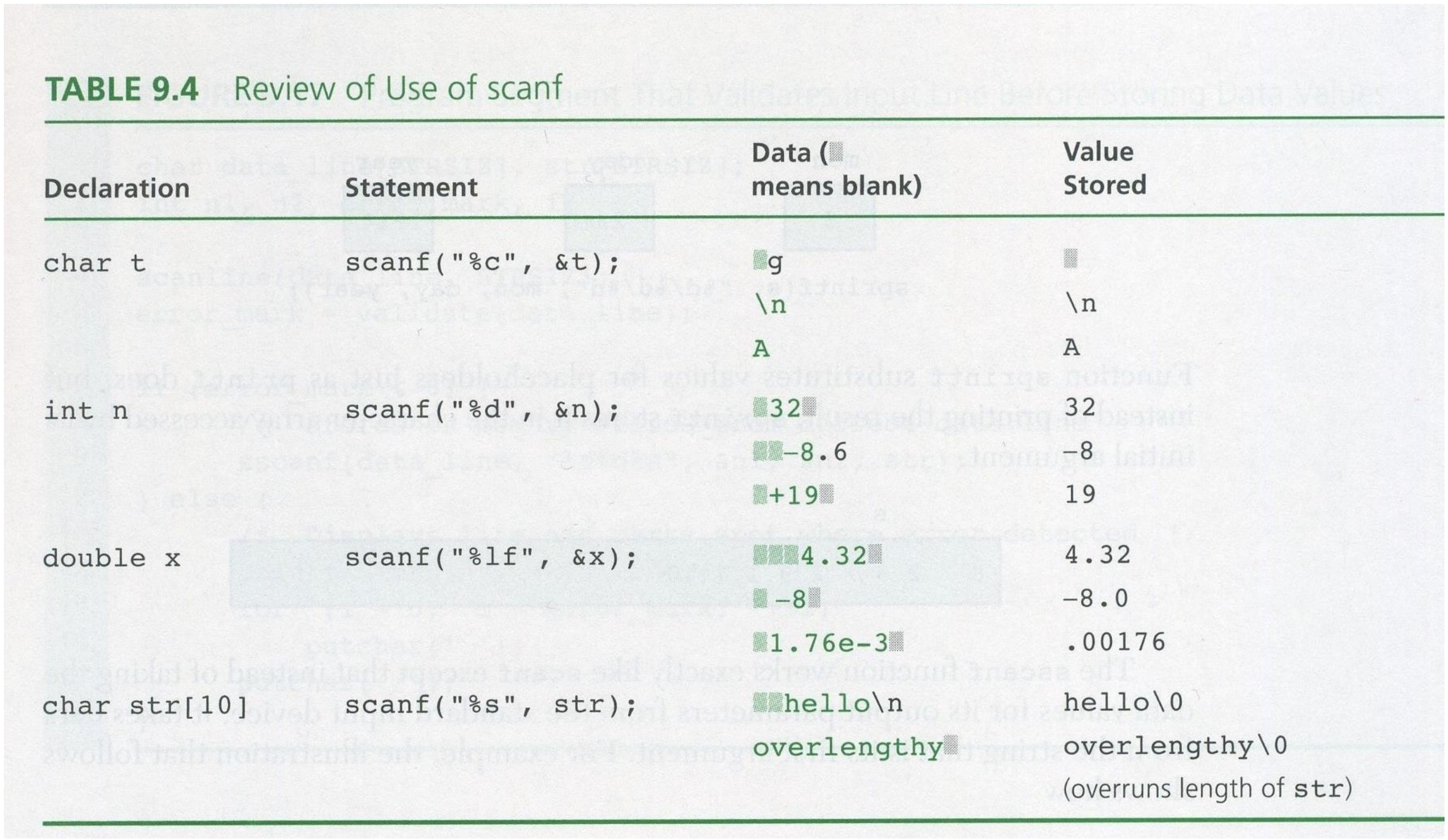


TABLE 9.4 Review of Use of scanf

Declaration	Statement	Data (▨ means blank)	Value Stored
`char t`	`scanf("%c", &t);`	`▨g`	`▨`
		`\n`	`\n`
		`A`	`A`
`int n`	`scanf("%d", &n);`	`▨32▨`	`32`
		`▨▨-8.6`	`-8`
		`▨+19▨`	`19`
`double x`	`scanf("%lf", &x);`	`▨▨▨4.32▨`	`4.32`
		`▨-8▨`	`-8.0`
		`▨1.76e-3▨`	`.00176`
`char str[10]`	`scanf("%s", str);`	`▨▨hello\n`	`hello\0`
		`overlengthy▨`	`overlengthy\0` (overruns length of `str`)

String-Number Conversion

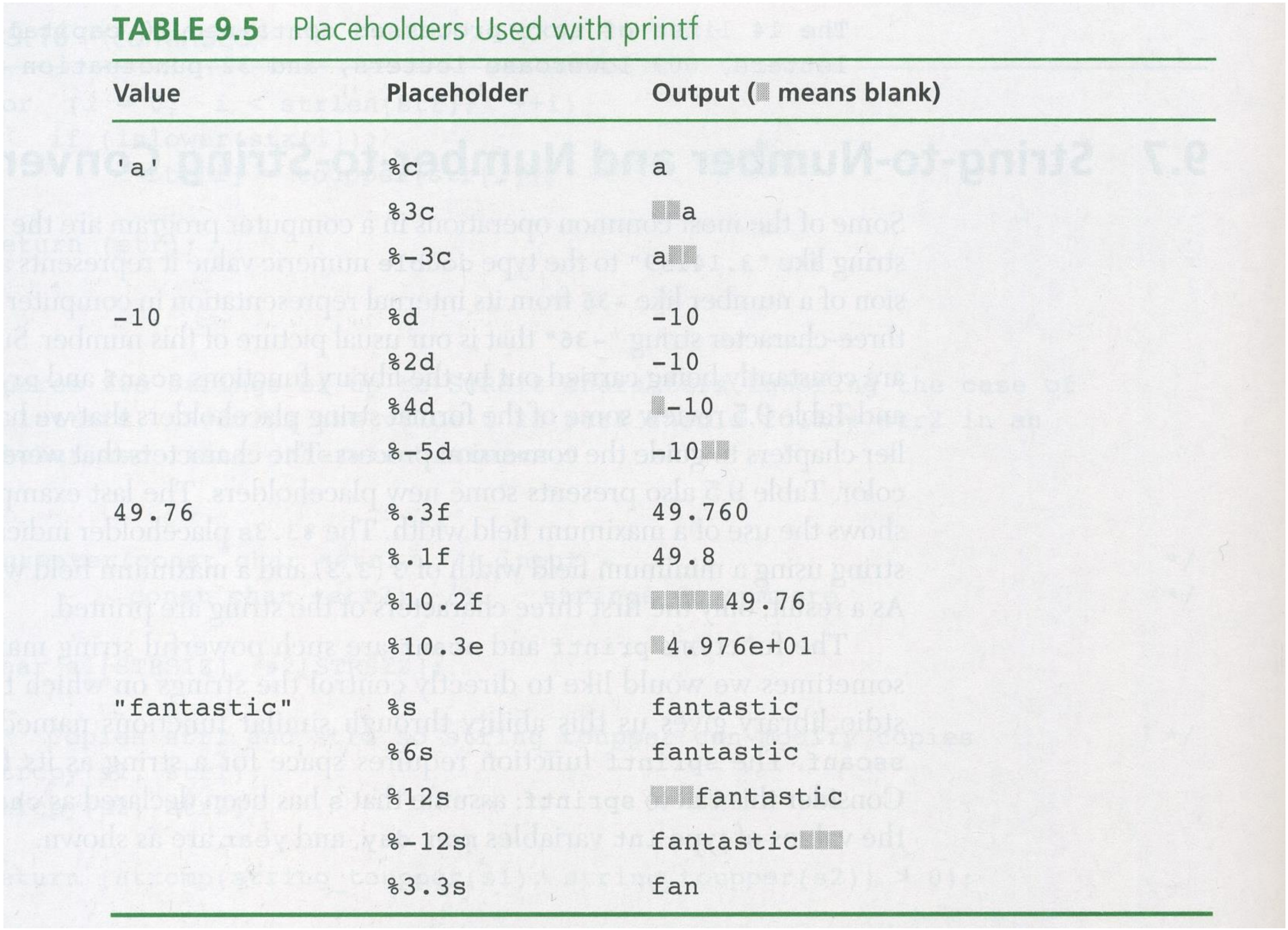


TABLE 9.5 Placeholders Used with printf

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`

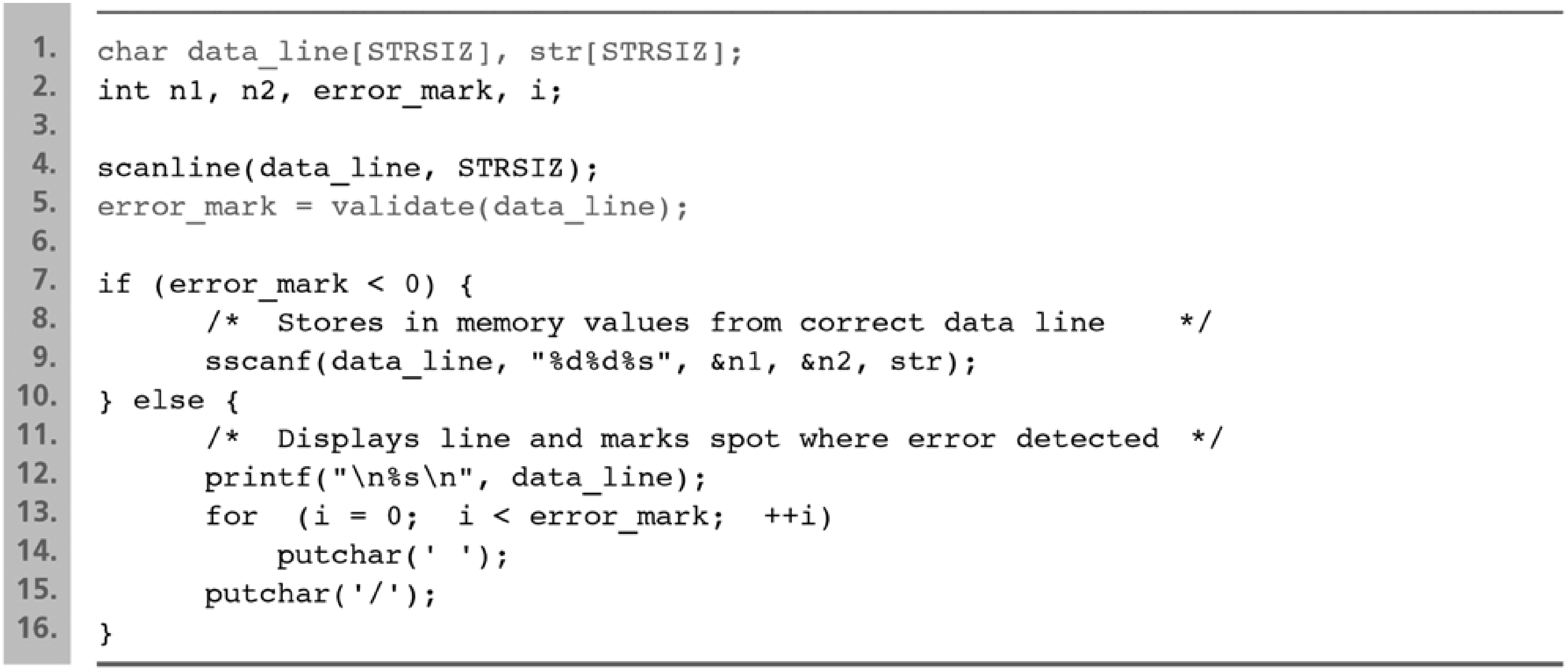
String-Number Conversion

- sscanf and sprintf similar to scanf and printf
  - They perform the operation on a string
  - sscanf: reads input from the parameter string
  - sprintf: outputs into the parameter string

```
char s[100];
sprintf(s, “%d/%d/%d”, mon, day, year);
sscanf(“  85   96.5   hello”, “%d %lf %s”, &n, &f, w);
```

  - You can read the entire data as a line of input, verify its format and convert to correct values using sscanf

Validate Input Line Before Storing Values



```
char data_line[STRSIZ], str[STRSIZ];
int n1, n2, error_mark, i;

scanline(data_line, STRSIZ);
error_mark = validate(data_line);

if (error_mark < 0) {
      /*  Stores in memory values from correct data line    */
      sscanf(data_line, "%d%d%s", &n1, &n2, str);
} else {
      /*  Displays line and marks spot where error detected  */
      printf("\n%s\n", data_line);
      for  (i = 0;  i < error_mark;  ++i)
          putchar(' ');
      putchar('/');
}
```

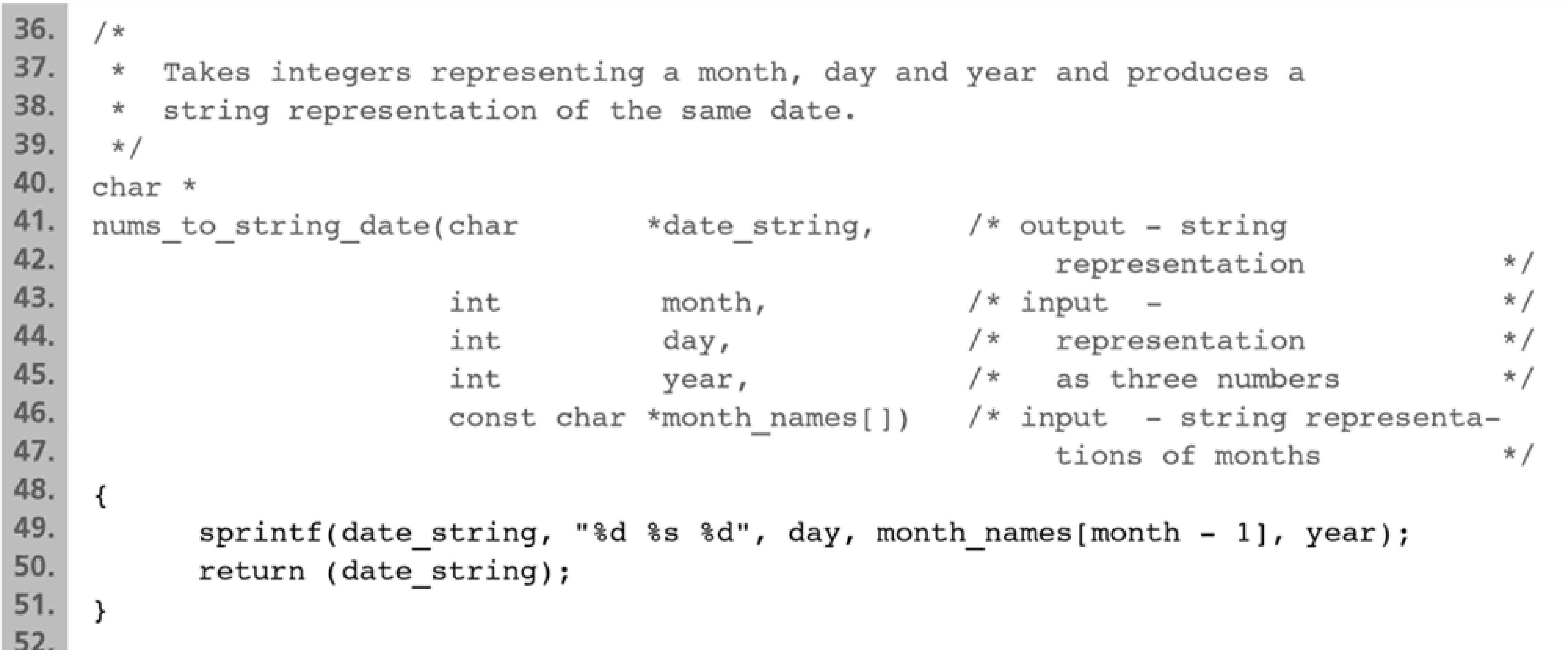
Ex: Date Conversion

- Date representations
  - string containing day month name and year
    - (12 June 1968)
  - three integers (day month year)
    - (12 6 1968)
- Convert a string representation of date to three integer representation and vice versa

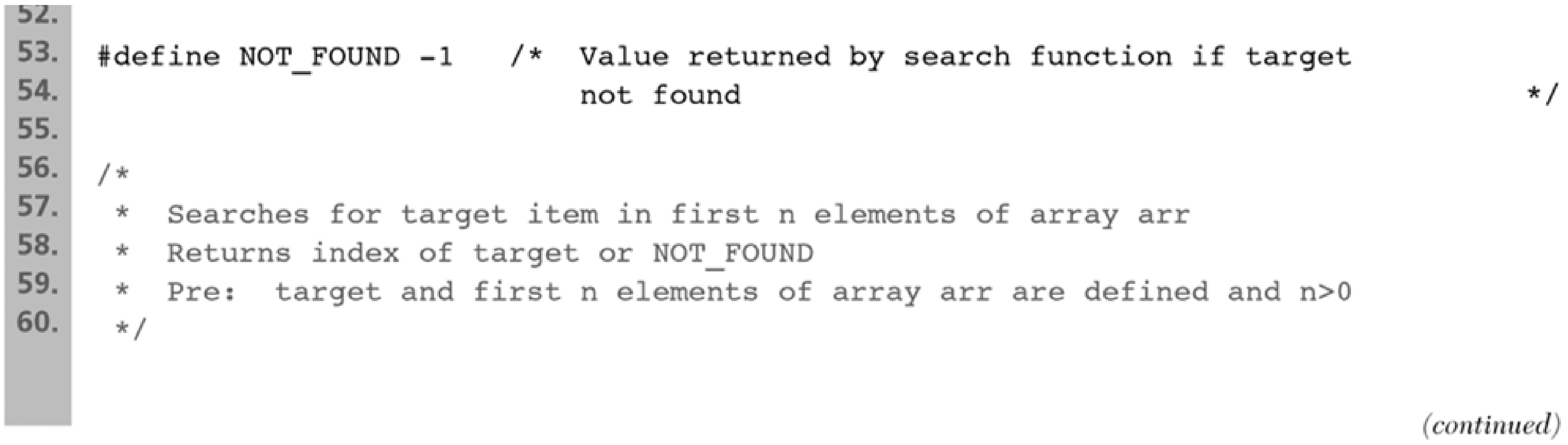
```
#include <stdio.h>
#include <string.h>

#define STRSIZ 40
char *nums_to_string_date(char *date_string, int month, int day,
                          int year, const char *month_names[]);
int search(const char *arr[], const char *target, int n);
void string_date_to_nums(const char *date_string, int *monthp,
                         int *dayp, int *yearp, const char *month_names[]);

/*  Tests date conversion functions                                    */
int
main(void)
{
```

(continued)

```
        char *month_names[12] = {"January", "February", "March", "April", "May",
                                 "June", "July", "August", "September", "October",
                                 "November", "December"};
        int m, y, mon, day, year;
        char date_string[STRSIZ];
        for  (y = 1993;  y < 2010;  y += 10)
            for  (m = 1;  m <= 12;  ++m) {
                printf("%s", nums_to_string_date(date_string,
                                                 m, 15, y, month_names));
                string_date_to_nums(date_string, &mon, &day, &year, month_names);
                printf(" = %d/%d/%d\n", mon, day, year);
            }

        return (0);
}
```



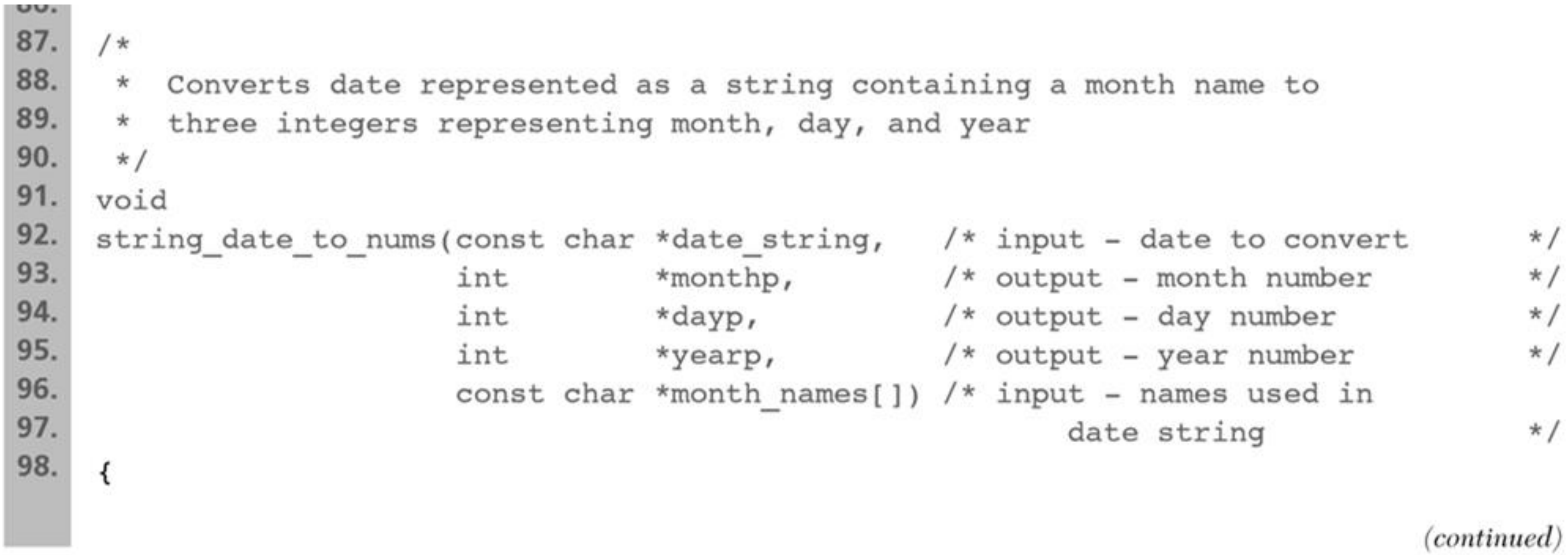
```
/*
 *  Takes integers representing a month, day and year and produces a
 *  string representation of the same date.
 */
char *
nums_to_string_date(char       *date_string,     /* output - string
                                                      representation           */
                    int         month,           /* input  -                   */
                    int         day,             /*   representation           */
                    int         year,            /*   as three numbers         */
                    const char *month_names[])   /* input  - string representa-
                                                      tions of months          */
{
      sprintf(date_string, "%d %s %d", day, month_names[month - 1], year);
      return (date_string);
}
```



```
#define NOT_FOUND -1   /*  Value returned by search function if target
                           not found                                          */

/*
 *  Searches for target item in first n elements of array arr
 *  Returns index of target or NOT_FOUND
 *  Pre:  target and first n elements of array arr are defined and n>0
 */
```

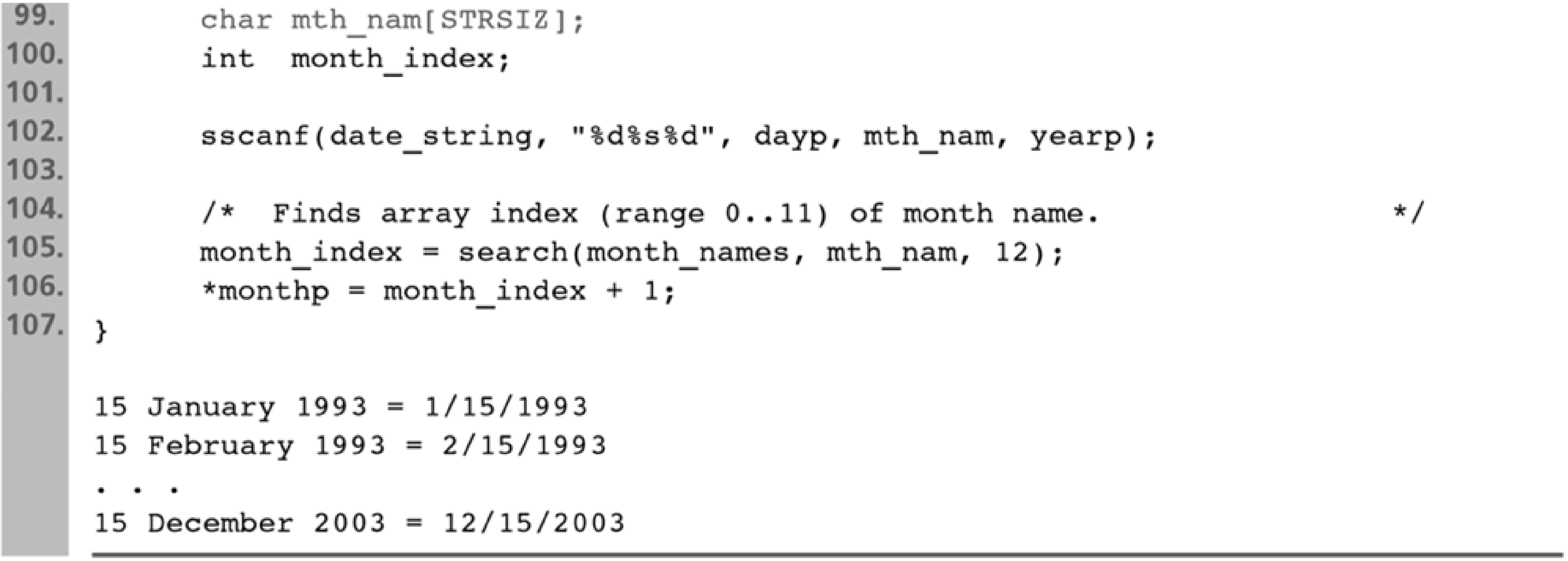
(continued)

```c
int
search(const char *arr[],          /*  array to search                       */
       const char *target,         /*  value searched for                    */
       int         n)              /*  number of array elements to search    */
{
      int i,
          found = 0,     /*  whether or not target has been found */
          where;         /*  index where target found or NOT_FOUND*/

      /*  Compares each element to target                     */
      i = 0;
      while (!found && i < n) {
          if (strcmp(arr[i], target) == 0)
                found = 1;
          else
                ++i;
      }

      /* Returns index of element matching target or NOT_FOUND    */
      if (found)
            where = i;
      else
            where = NOT_FOUND;
      return (where);
}
```



```
/*
 *  Converts date represented as a string containing a month name to
 *  three integers representing month, day, and year
 */
void
string_date_to_nums(const char *date_string,   /* input - date to convert       */
                    int        *monthp,        /* output - month number         */
                    int        *dayp,          /* output - day number           */
                    int        *yearp,         /* output - year number          */
                    const char *month_names[]) /* input - names used in
                                                           date string          */
{
```

(continued)



```c
      char mth_nam[STRSIZ];
      int  month_index;

      sscanf(date_string, "%d%s%d", dayp, mth_nam, yearp);

      /*  Finds array index (range 0..11) of month name.                   */
      month_index = search(month_names, mth_nam, 12);
      *monthp = month_index + 1;
}
```

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

Case Study: Text Editor

Problem: Editing operations on a line of text:

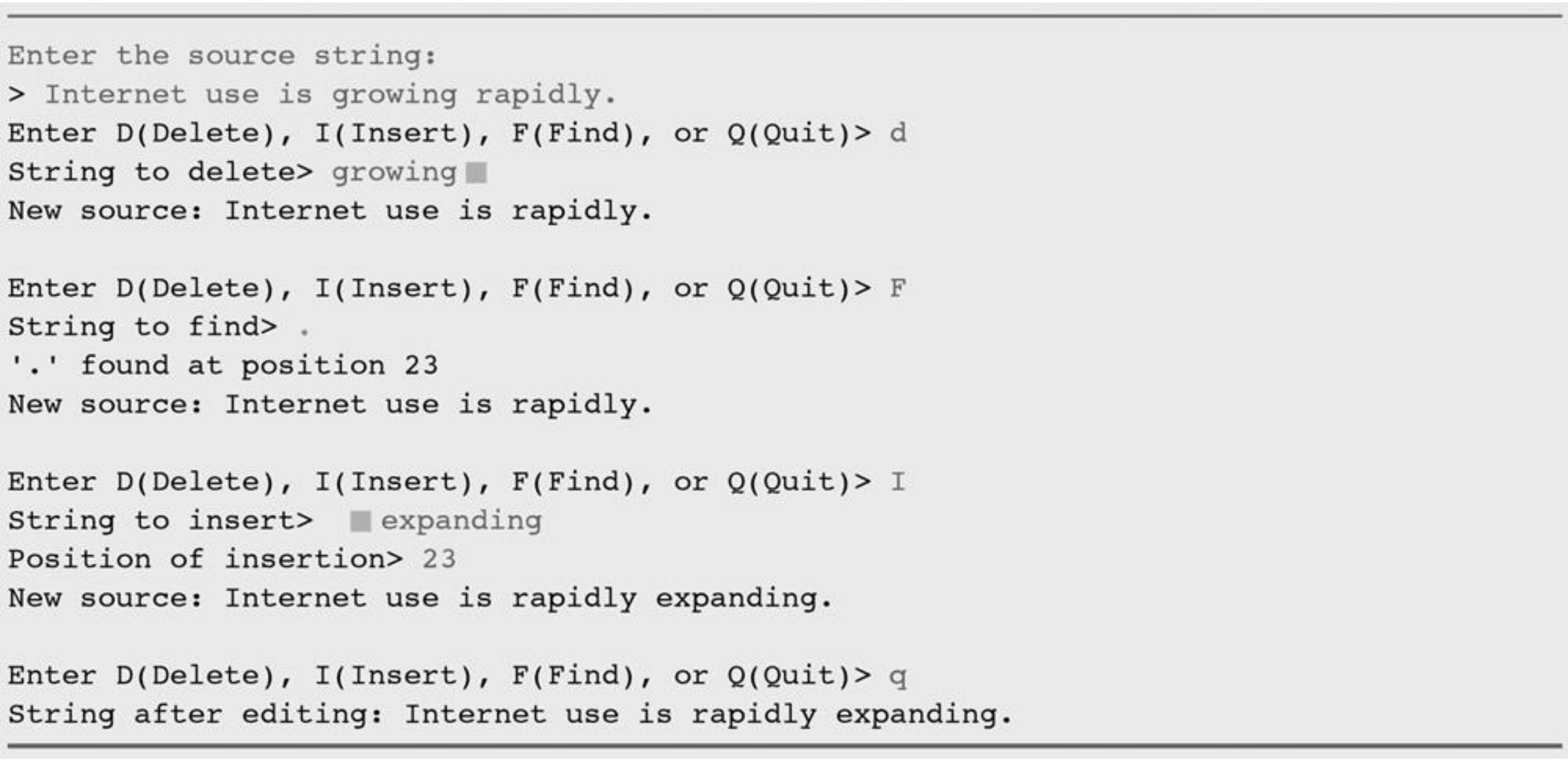
- Locate a target string
- Delete a substring
- Insert a substring at a location

Analysis:

- Keep the source line to edit
- Get the operation until it is Q

- Data Requirements
  - source array
  - command

Sample Run of Text Editor



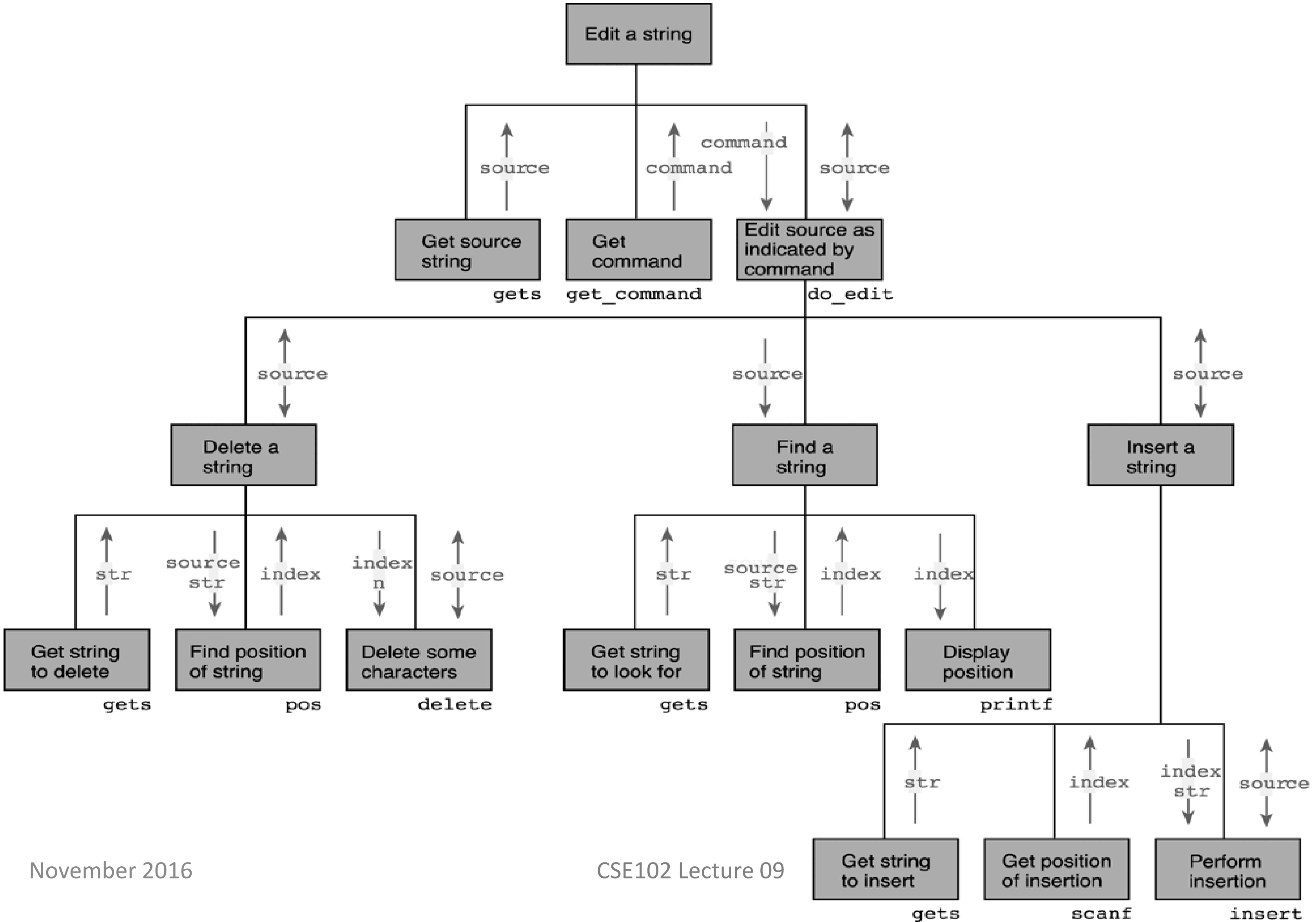
```
Enter the source string:
> Internet use is growing rapidly.
Enter D(Delete), I(Insert), F(Find), or Q(Quit)> d
String to delete> growing■
New source: Internet use is rapidly.

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.
```

Case Study: Text Editor

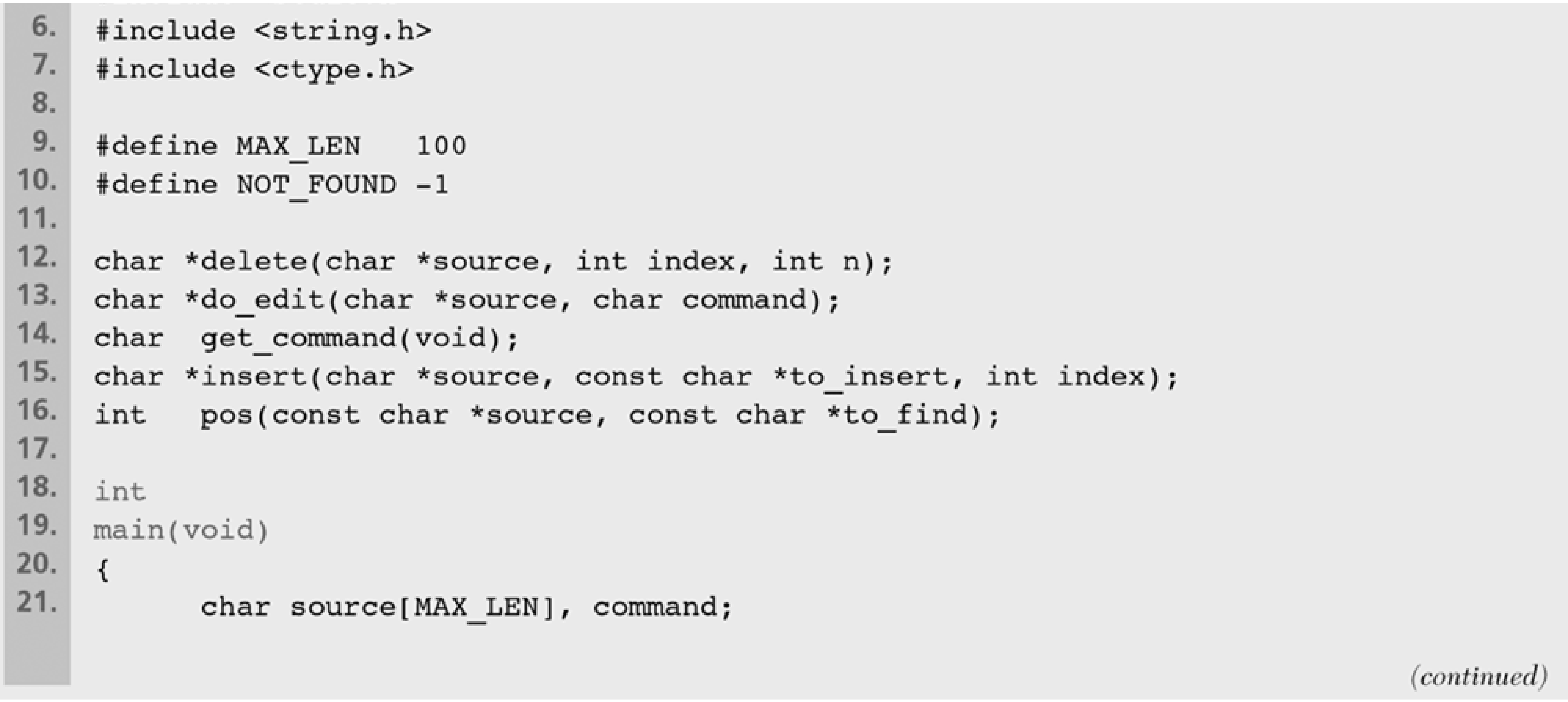
Algorithm

```
1. Scan the string
2. Get an edit command
3. While command is not Q
    4. Perform operation
            4.1 switch command
                    'D':        4.2 Get the substring
                                4.3 Find the position
                                4.4 If found delete it
                    'I'         4.5 Get the substring
                                4.6 Get the position index
                                4.7 Perform insertion
                    'F'         4.8 Get the substring
                                4.9 Find the position
                                4.10 Report position
                    o/w         4.11 Display error message
    5. Get an edit command
```

Structure Chart for Text Editor Program



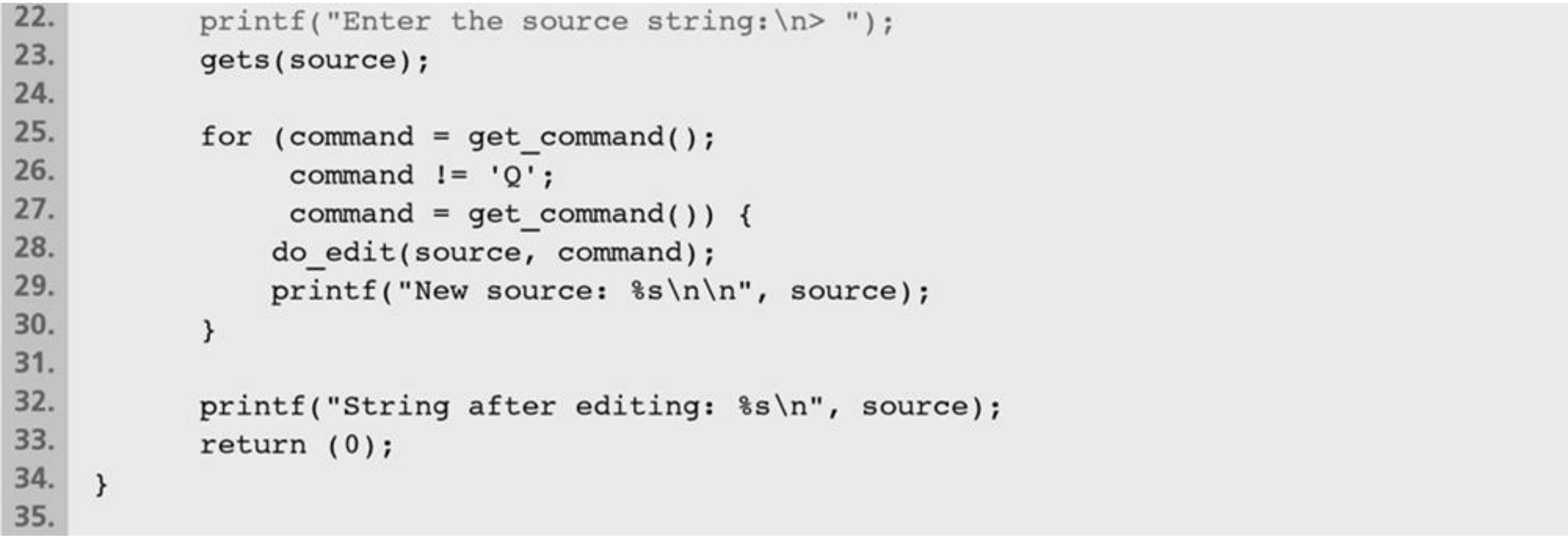
Text Editor Program



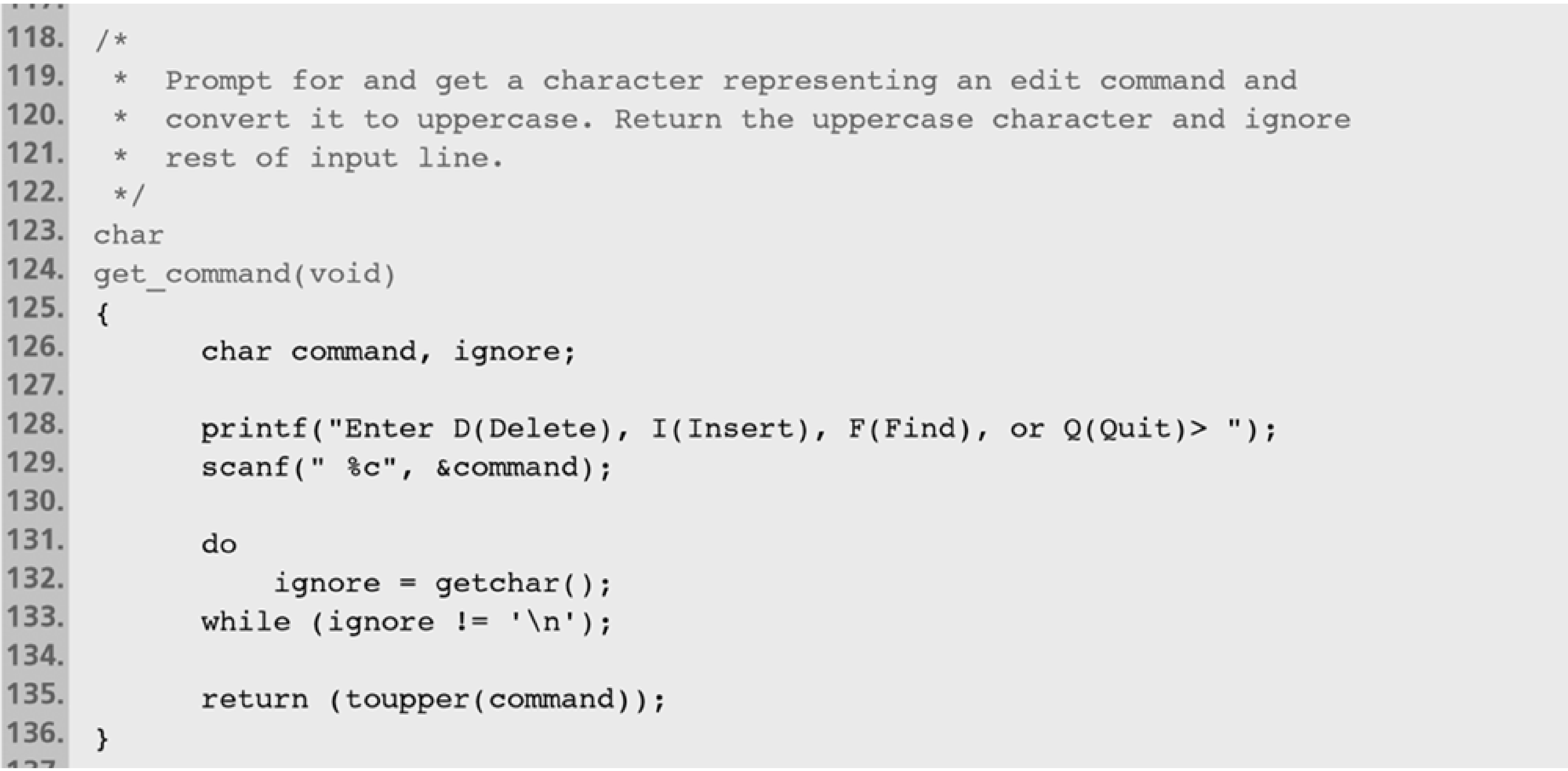
```
#include <string.h>
#include <ctype.h>

#define MAX_LEN   100
#define NOT_FOUND -1

char *delete(char *source, int index, int n);
char *do_edit(char *source, char command);
char  get_command(void);
char *insert(char *source, const char *to_insert, int index);
int   pos(const char *source, const char *to_find);

int
main(void)
{
      char source[MAX_LEN], command;
```

(continued)

Text Editor Program

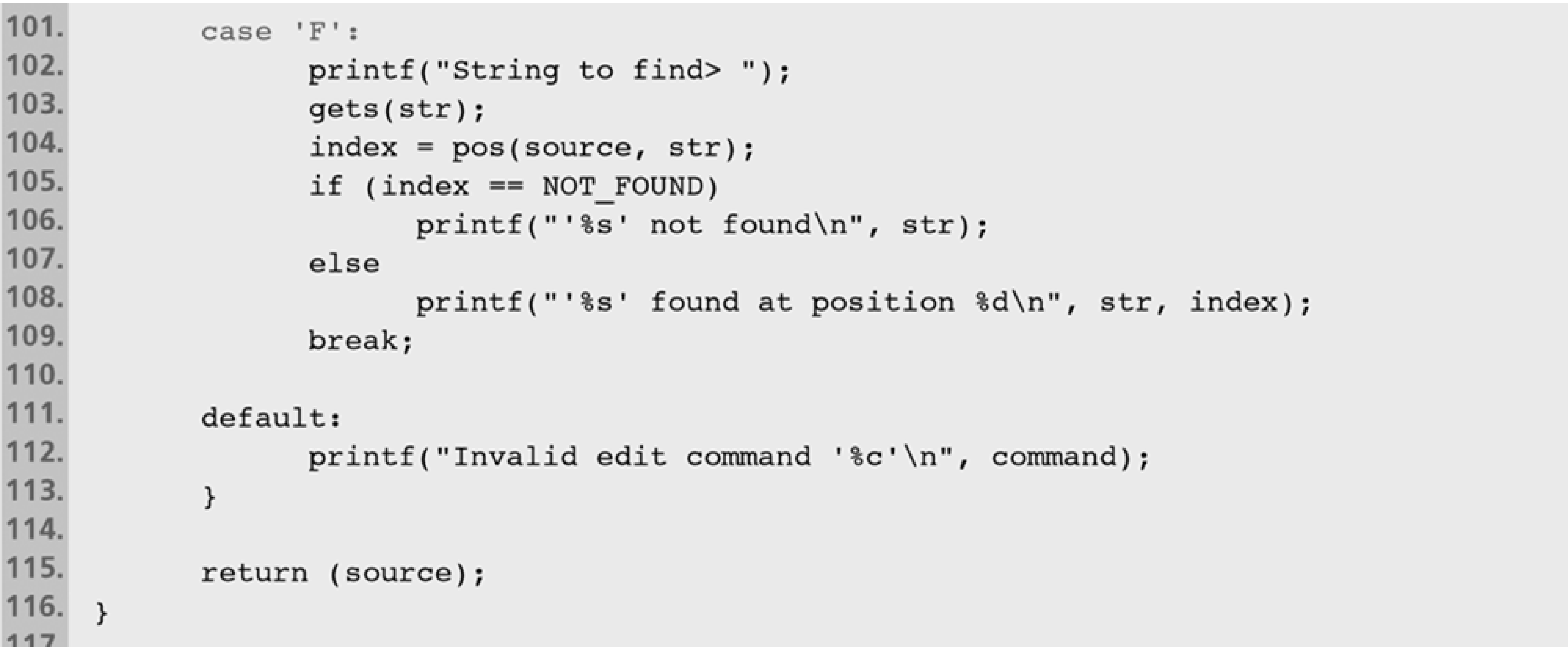


```
      printf("Enter the source string:\n> ");
      gets(source);

      for (command = get_command();
           command != 'Q';
           command = get_command()) {
          do_edit(source, command);
          printf("New source: %s\n\n", source);
      }

      printf("String after editing: %s\n", source);
      return (0);
}
```

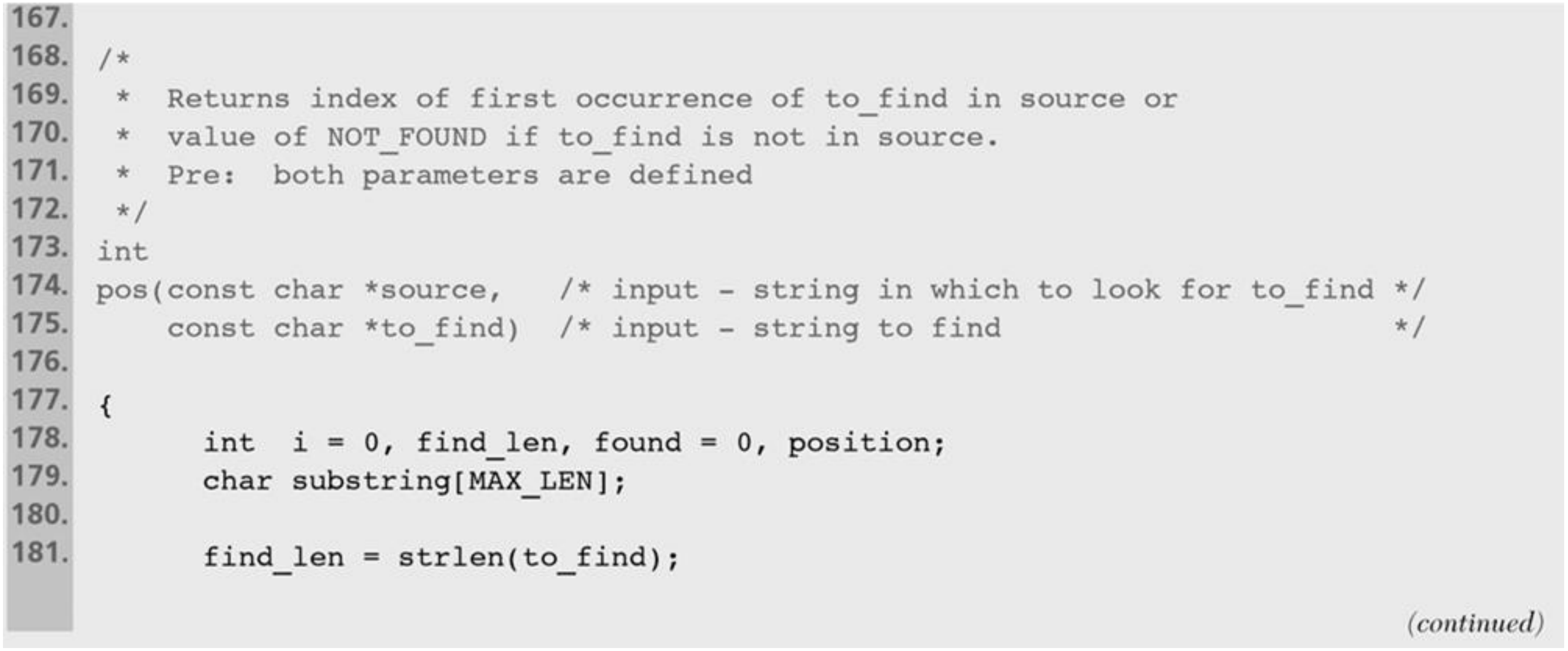


```
/*
 *  Prompt for and get a character representing an edit command and
 *  convert it to uppercase. Return the uppercase character and ignore
 *  rest of input line.
 */
char
get_command(void)
{
      char command, ignore;

      printf("Enter D(Delete), I(Insert), F(Find), or Q(Quit)> ");
      scanf(" %c", &command);

      do
          ignore = getchar();
      while (ignore != '\n');

      return (toupper(command));
}
```

```
/*
 *  Performs the edit operation specified by command
 *  Pre:  command and source are defined.
 *  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;

```

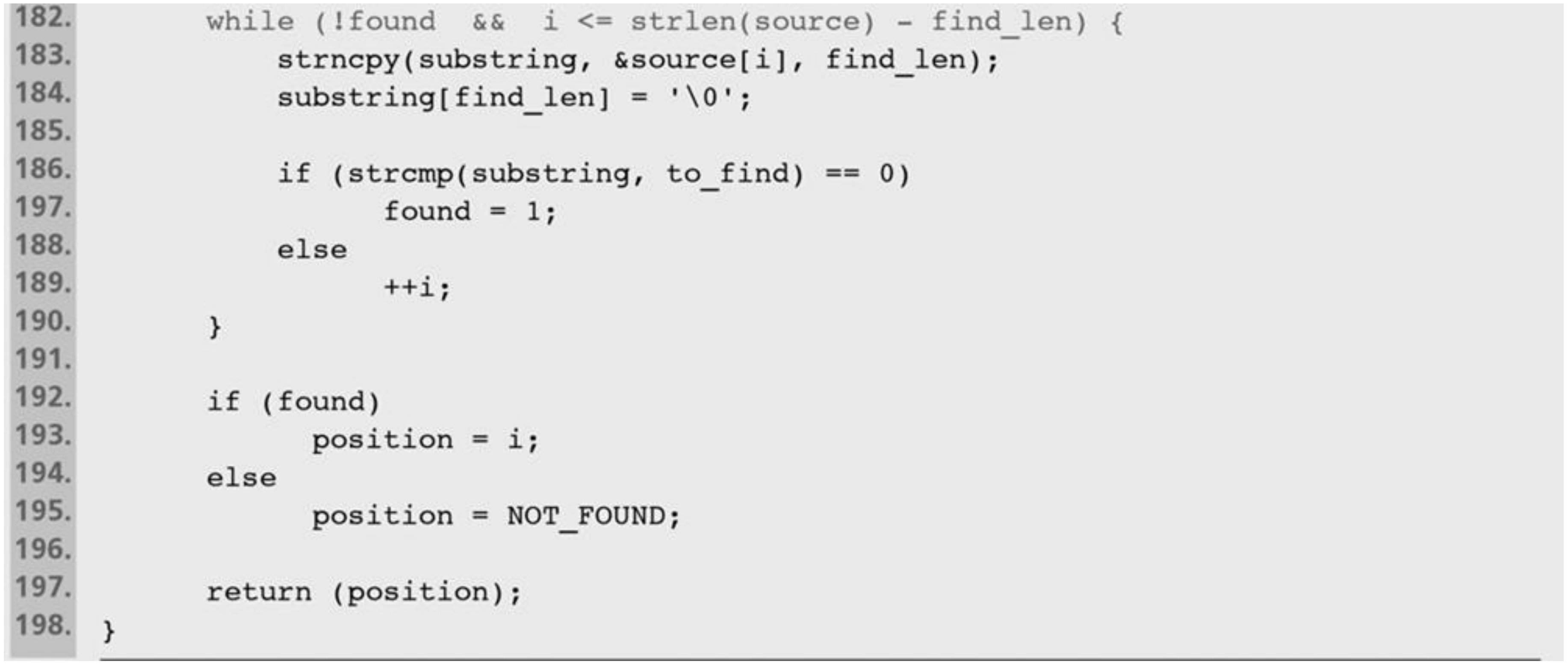


```
        case 'F':
                printf("String to find> ");
                gets(str);
                index = pos(source, str);
                if (index == NOT_FOUND)
                        printf("'%s' not found\n", str);
                else
                        printf("'%s' found at position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

        default:
                printf("Invalid edit command '%c'\n", command);
        }

        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);
```

(continued)



```
   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;
   }

   if (found)
      position = i;
   else
      position = NOT_FOUND;

   return (position);
}
```

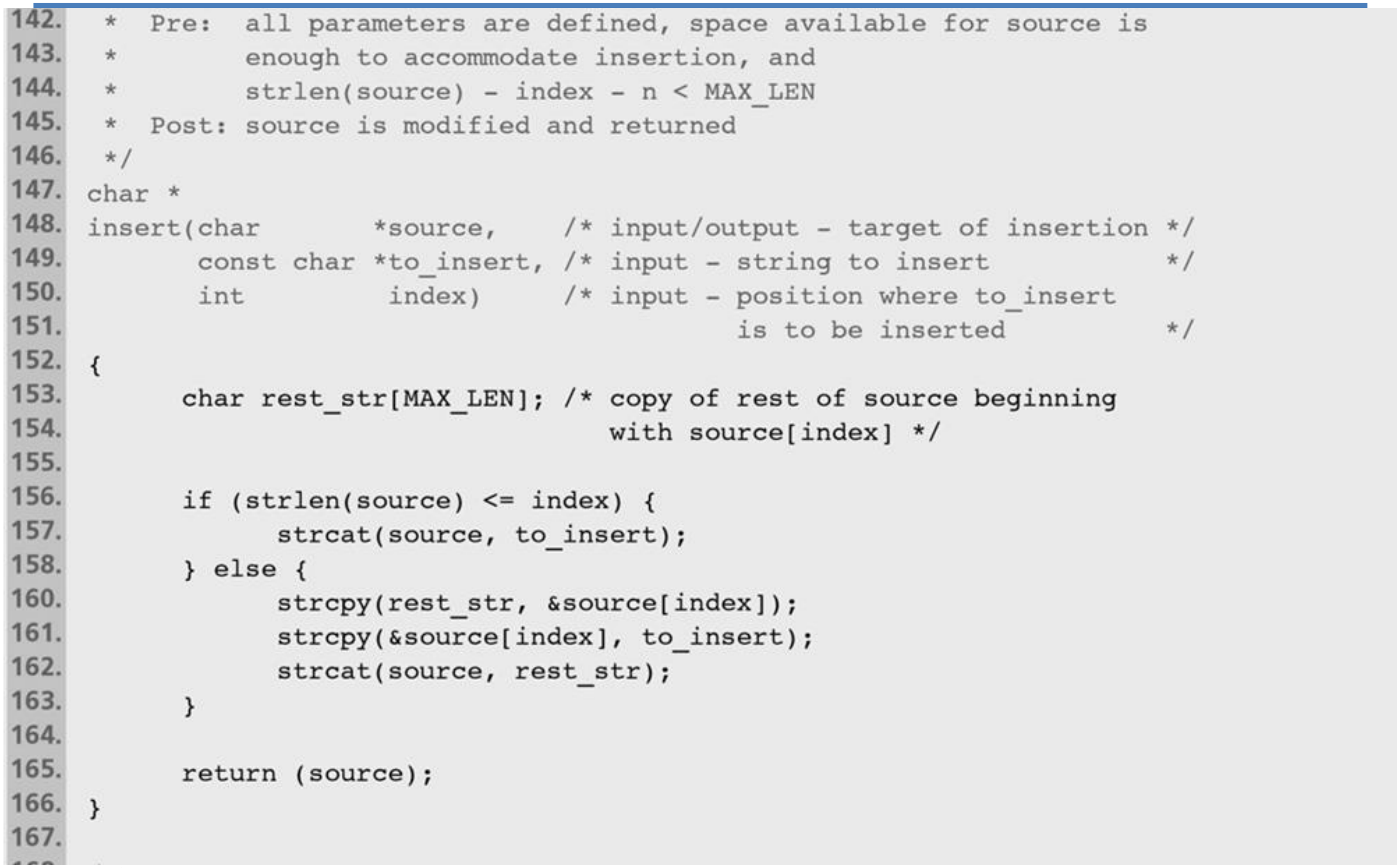
```
/*
 *  Returns source after deleting n characters beginning with source[index].
 *  If source is too short for full deletion, as many characters are
 *  deleted as possible.
 *  Pre:  All parameters are defined and
 *        strlen(source) - index - n < MAX_LEN
 *  Post:  source is modified and returned
 */
char *
delete(char *source,  /* input/output - string from which to delete part */
       int   index,   /* input - index of first char to delete           */
       int   n)       /* input - number of chars to delete               */
{
      char rest_str[MAX_LEN];  /* copy of source substring following
                                  characters to delete */

      /*  If there are no characters in source following portion to
          delete, delete rest of string */
      if (strlen(source) <= index + n) {
            source[index] = '\0';

      /*  Otherwise, copy the portion following the portion to delete
          and place it in source beginning at the index position       */
      } else {
            strcpy(rest_str, &source[index + n]);
```

(continued)

```
                strcpy(&source[index], rest_str);
        }

        return (source);
}
```

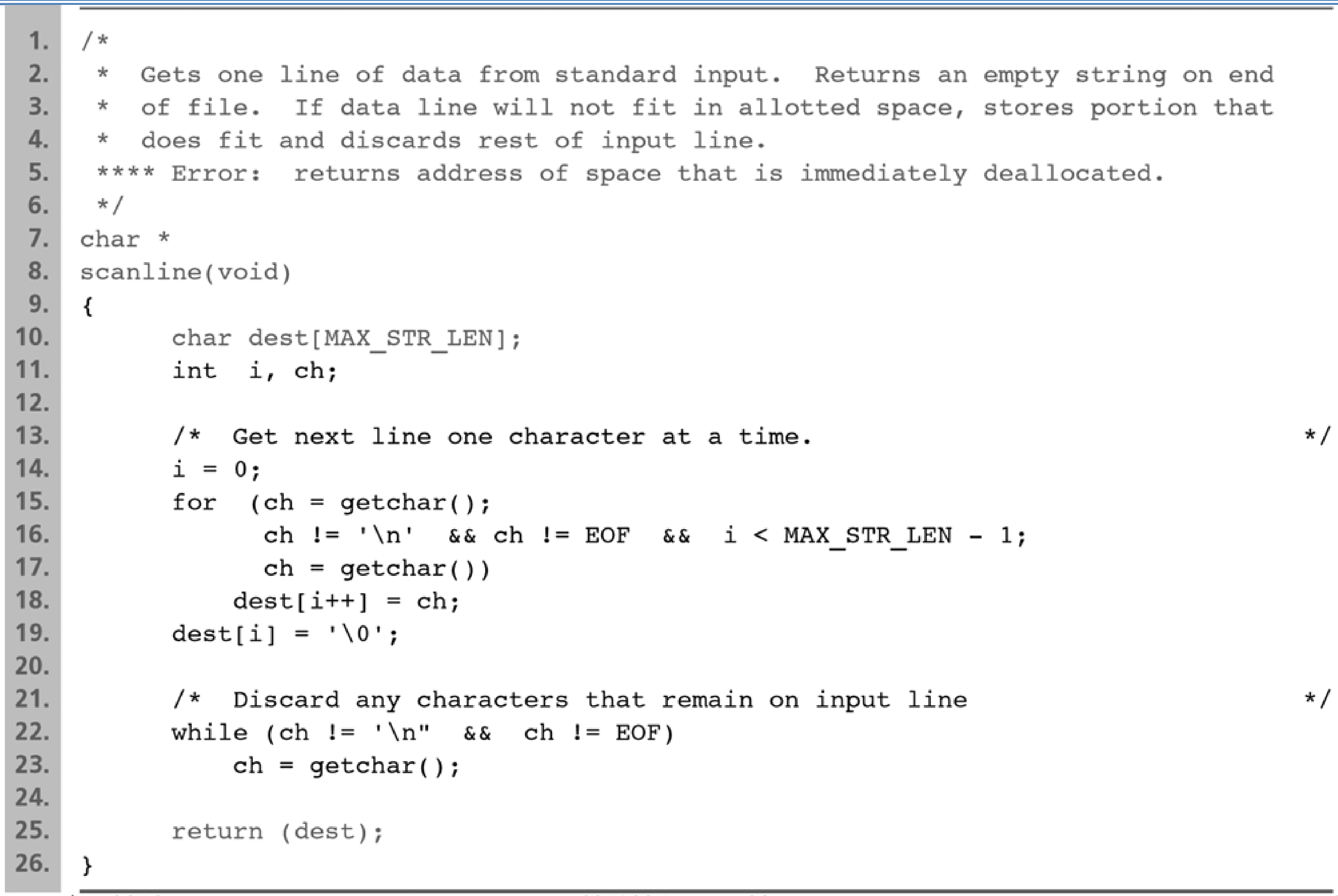
```

/*
 *  Returns source after inserting to_insert at position index of
 *  source. If source[index] doesn't exist, adds to_insert at end of
 *  source.
```

(continued)



```c
 *  Pre:  all parameters are defined, space available for source is
 *        enough to accommodate insertion, and
 *        strlen(source) - index - n < MAX_LEN
 *  Post: source is modified and returned
 */
char *
insert(char       *source,    /* input/output - target of insertion */
       const char *to_insert, /* input - string to insert           */
       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);
}
```

scanline Returns Address of Deallocated Space



```
/*
 *  Gets one line of data from standard input.  Returns an empty string on end
 *  of file.  If data line will not fit in allotted space, stores portion that
 *  does fit and discards rest of input line.
 **** Error:  returns address of space that is immediately deallocated.
 */
char *
scanline(void)
{
      char dest[MAX_STR_LEN];
      int  i, ch;

      /*  Get next line one character at a time.                              */
      i = 0;
      for  (ch = getchar();
            ch != '\n'  && ch != EOF  &&  i < MAX_STR_LEN - 1;
            ch = getchar())
          dest[i++] = ch;
      dest[i] = '\0';

      /*  Discard any characters that remain on input line                    */
      while (ch != '\n"  &&  ch != EOF)
          ch = getchar();

      return (dest);
}
```