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

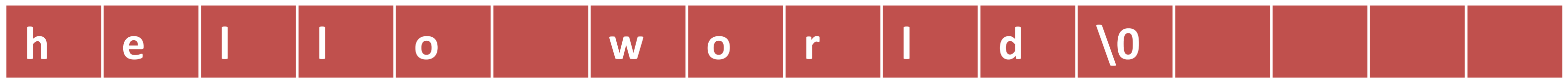
- sizeof() gives size in bytes
- strlen() gives length of string

```
char string[16] = “hello world”;

char *str = “hello world”;

char s[] = “hello world”;
```

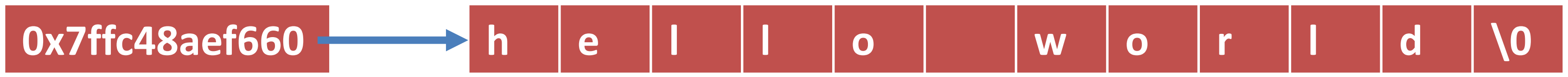
Initializing Strings

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

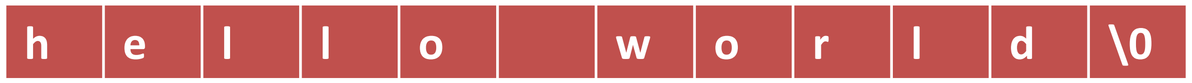
```
char string[16] = “hello world”;
```



```
char *str = “hello world”;
```



```
char s[] = “hello world”;
```

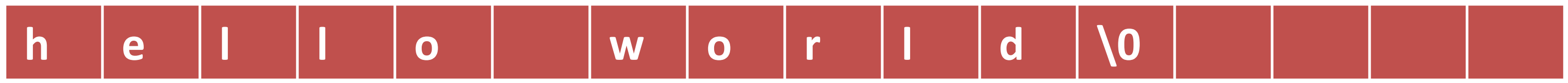


Initializing Strings

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

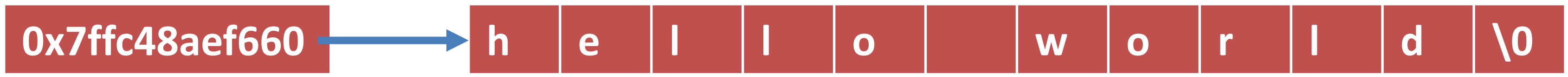
```
char string[16] = “hello world”;
```

sizeof() is 16
strlen() is 11

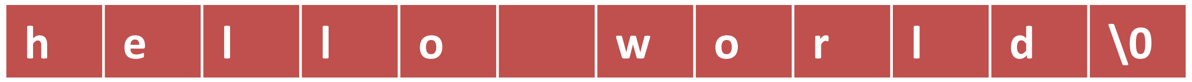


```
char *str = “hello world”;
```

sizeof() is 8
strlen() is 11



```
char s[] = “hello world”;
```

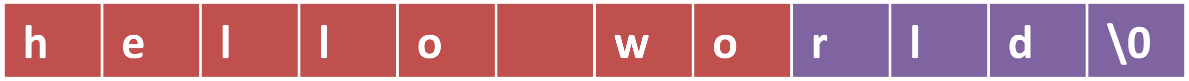


sizeof() is 12
strlen() is 11

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”;



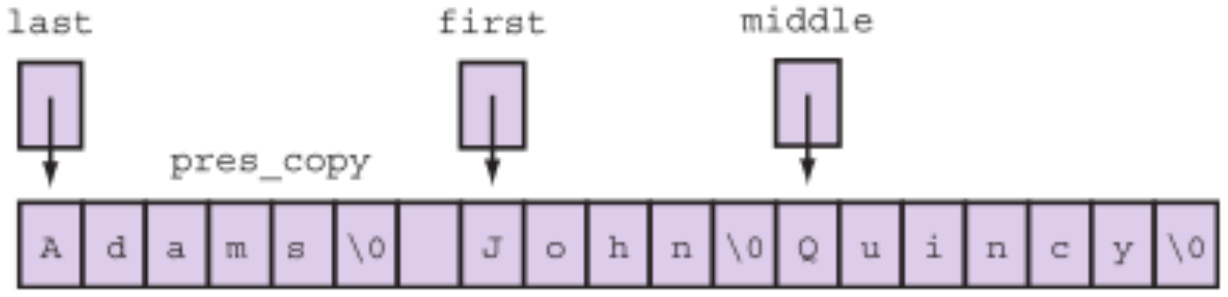
String Assignment

- 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);
```

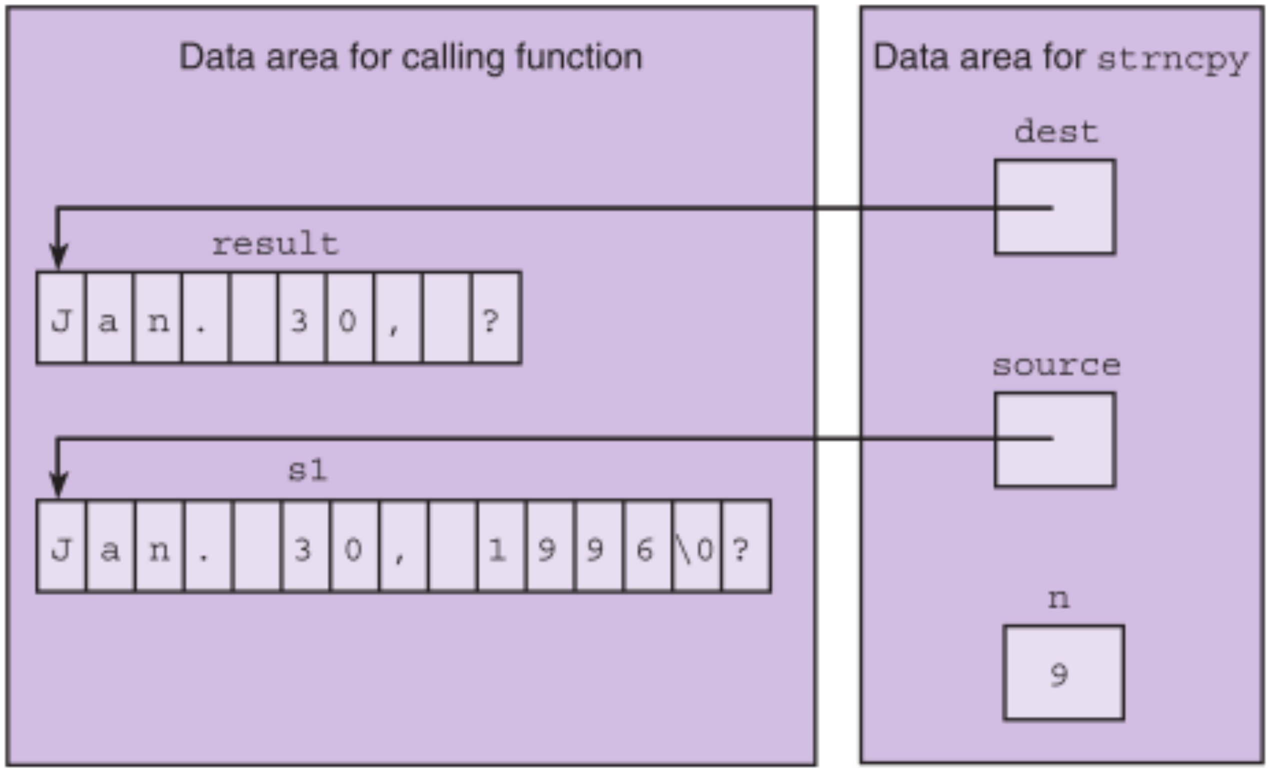


```
last = strtok(pres_copy, ", ");
first = strtok(NULL, ", ");
middle = strtok(NULL, ", ");
```

Substrings

FIGURE 8.5

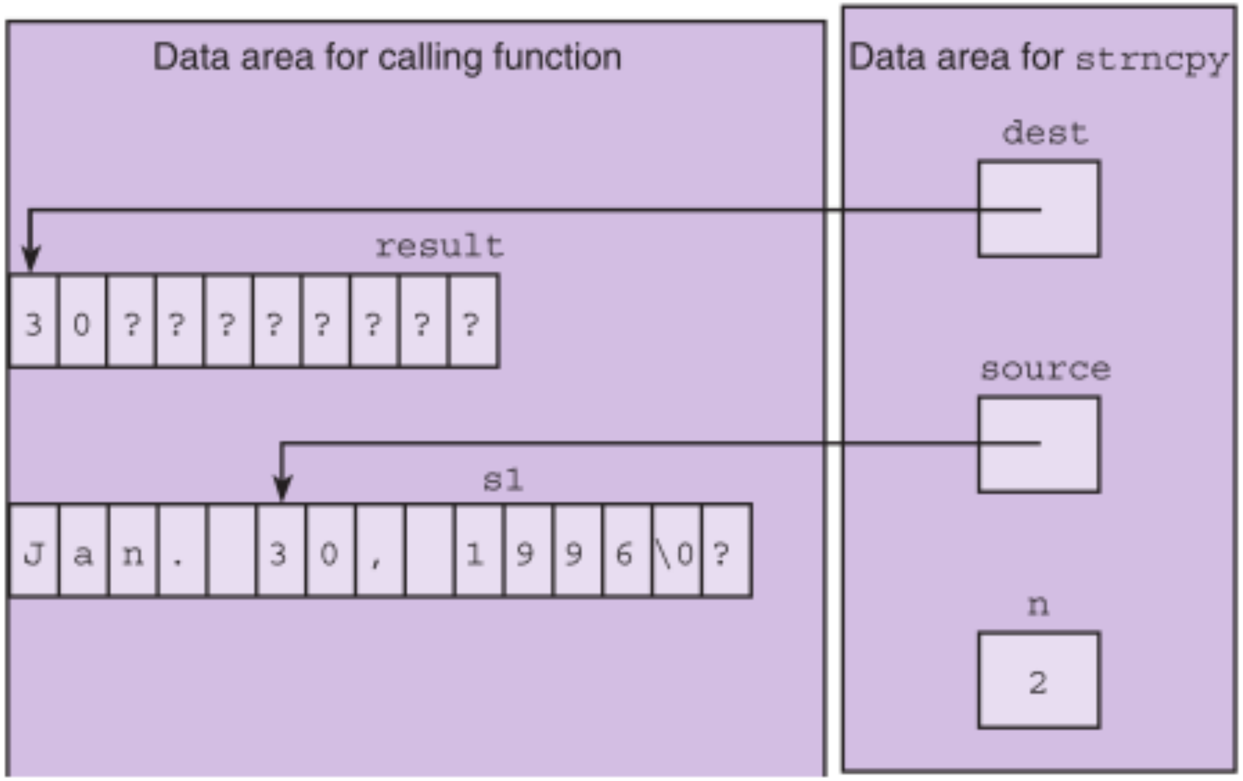
Execution of `strncpy(result, s1, 9);`



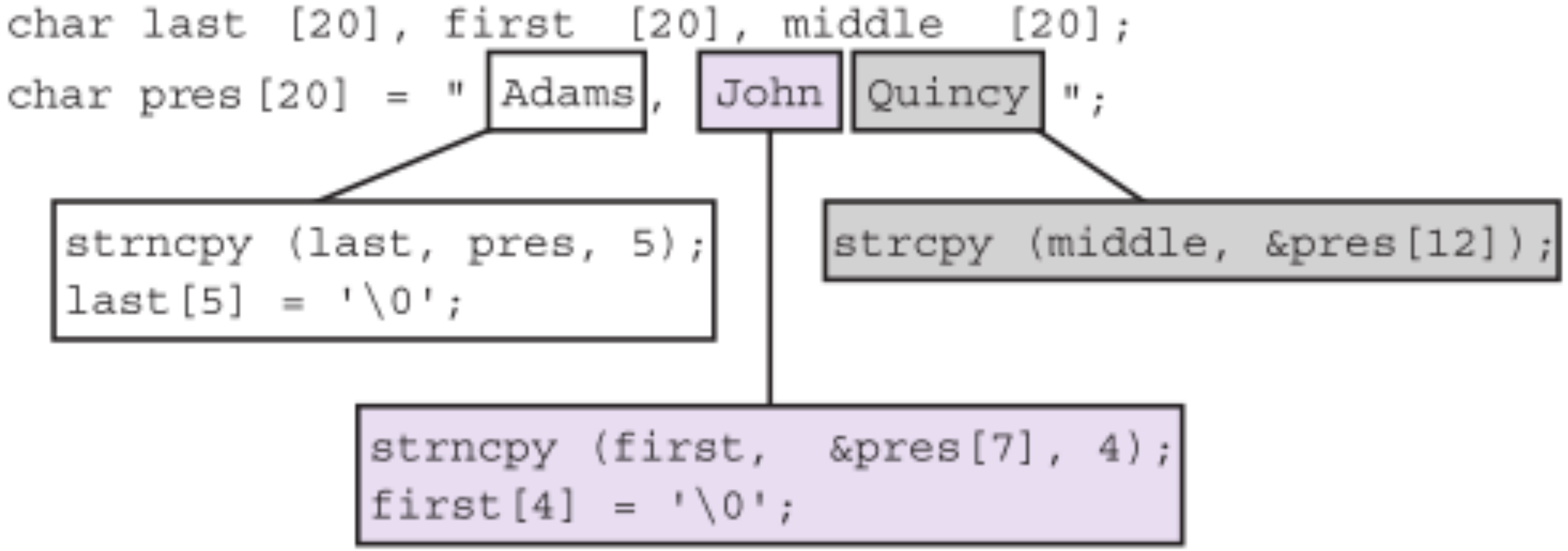
Substrings

FIGURE 8.6

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

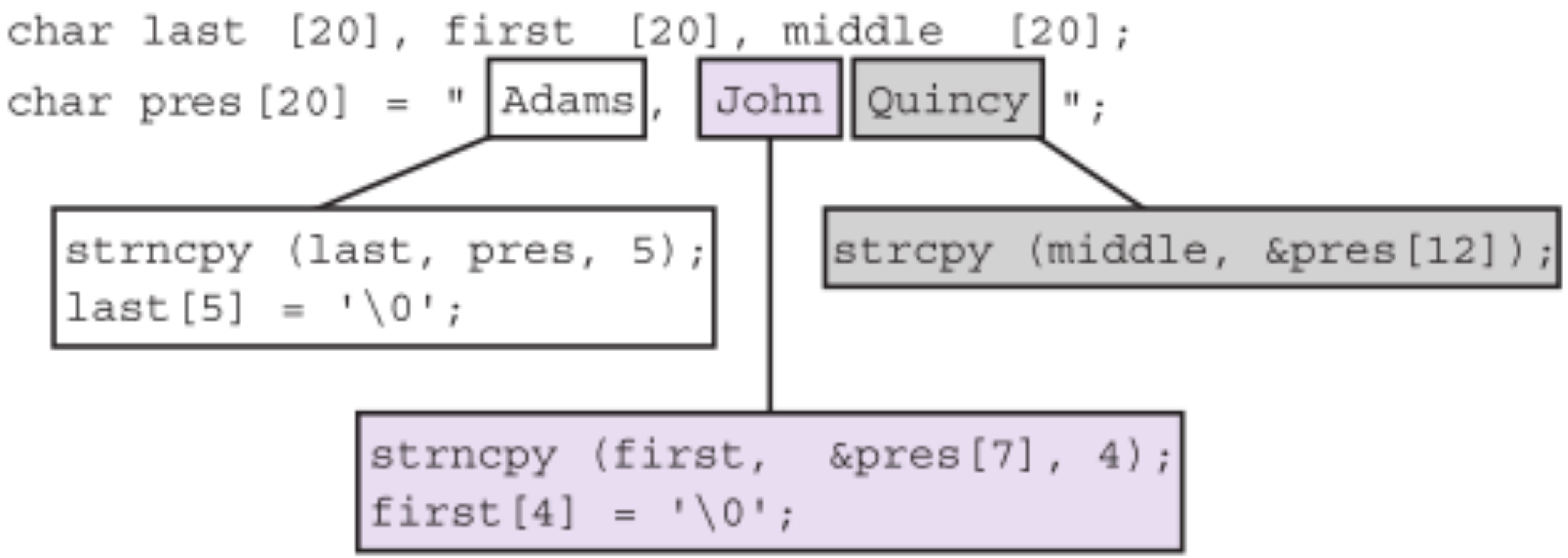


Substrings



```
char last  [20], first  [20], middle  [20];
char pres[20] = " Adams, John Quincy ";

strncpy (last, pres, 5);
last[5] = '\0';

strcpy (middle, &pres[12]);

strncpy (first,  &pres[7], 4);
first[4] = '\0';
```

Substrings



```
char last [20], first [20], middle [20];
char pres[20] = " Adams, John Quincy ";
```

```
strncpy (last, pres, 5);
last[5] = '\0';
```

```
strcpy (middle, &pres[12]);
```

```
strncpy (first, &pres[7], 4);
first[4] = '\0';
```

J	o	h	n	\0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

String Terminology

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

J	o	h	n	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

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

Scanning a Full Line

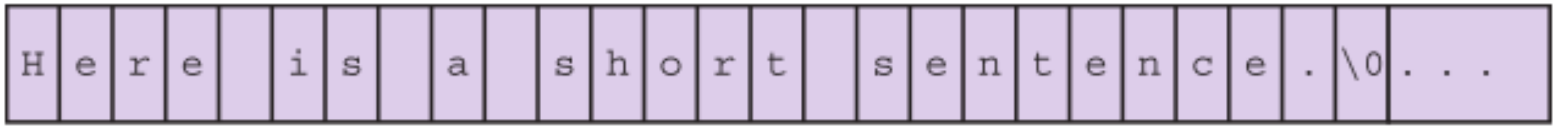
- For interactive input of one complete line of data, use the fgets function from stdio.
- Arguments: destination string, max characters to read, input
- Output: destination string or NULL if nothing read
- The \n character is stored if space.

```
fgets(<dest_string>, <num_chars>, <input>)
```

Scanning a Full Line with gets

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

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



subject to buffer overflow – we use fgets

String Comparison

a

b	l	u	e	\0

b

b	l	a	c	k	\0

c

b	l	u	e	\0	?	?	?	?	?

d

b	l	u	e	\0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

String Comparison

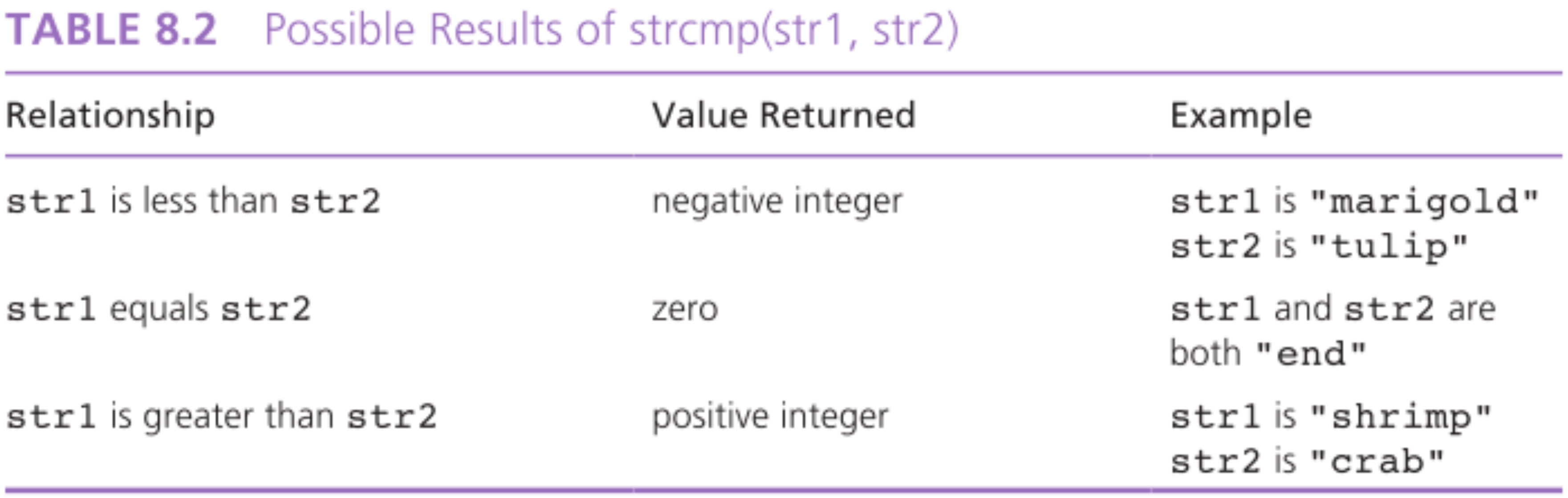


TABLE 8.2 Possible Results of strcmp(str1, str2)

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"`

FIGURE 8.10 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 */
     . . .
}
```

Arrays of Pointers

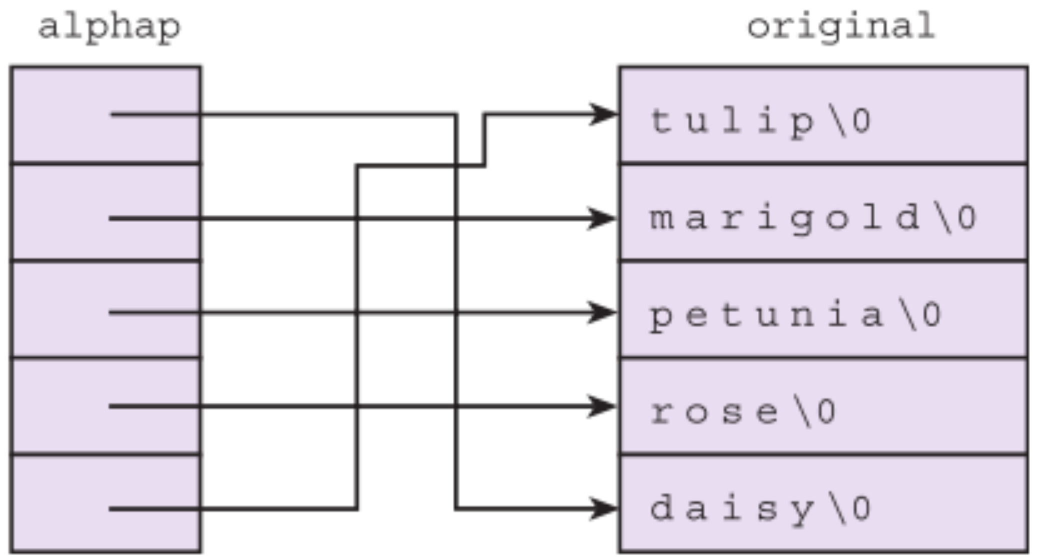
- When sorting a list of strings, there is a lot of copying of characters from one memory cell to another.
  - 3 operations for every exchange
- C represents every array by its starting address.
- Consider an array of pointers, each element the address of a character string.

FIGURE 8.11 Exchanging String Elements of an Array

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

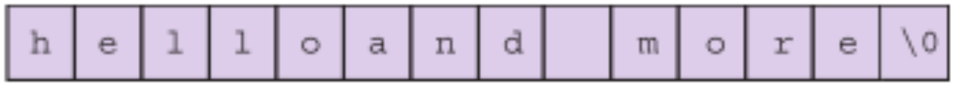
FIGURE 8.13

An Array of Pointers



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);

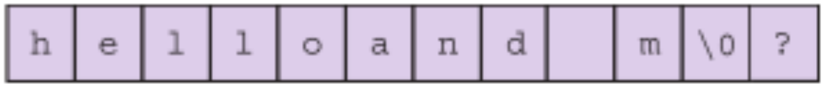


FIGURE 8.15 Implementation of scanline Function Using getchar

```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);
}
```

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 lower-case) 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

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

#define STRSIZ 80

/*
 * 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        */
{
```

(continued)

FIGURE 8.16 (continued)

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

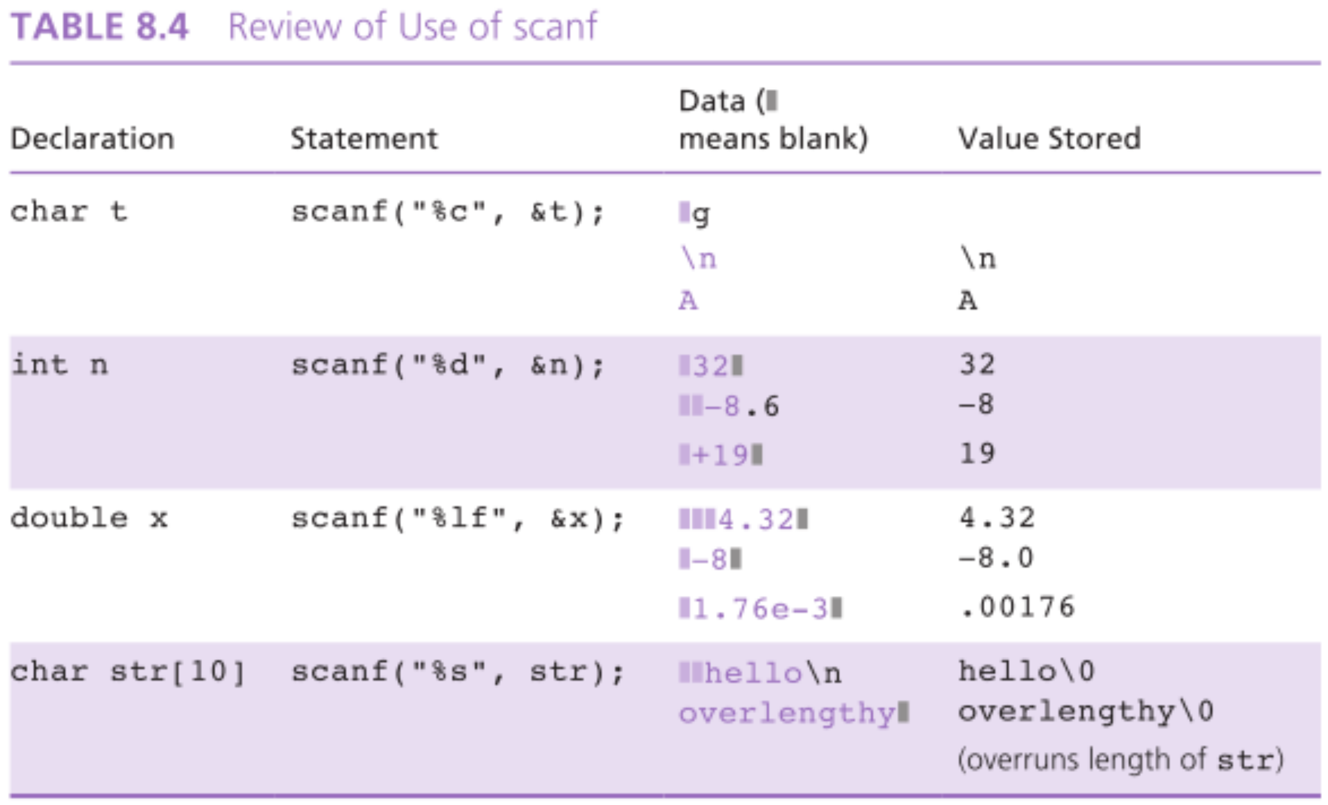


TABLE 8.4 Review of Use of scanf

Declaration	Statement	Data (▮ means blank)	Value Stored
`char t`	`scanf("%c", &t);`	`▮g`	
		`\n`	`\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-to-Number and Number-to-String Conversions

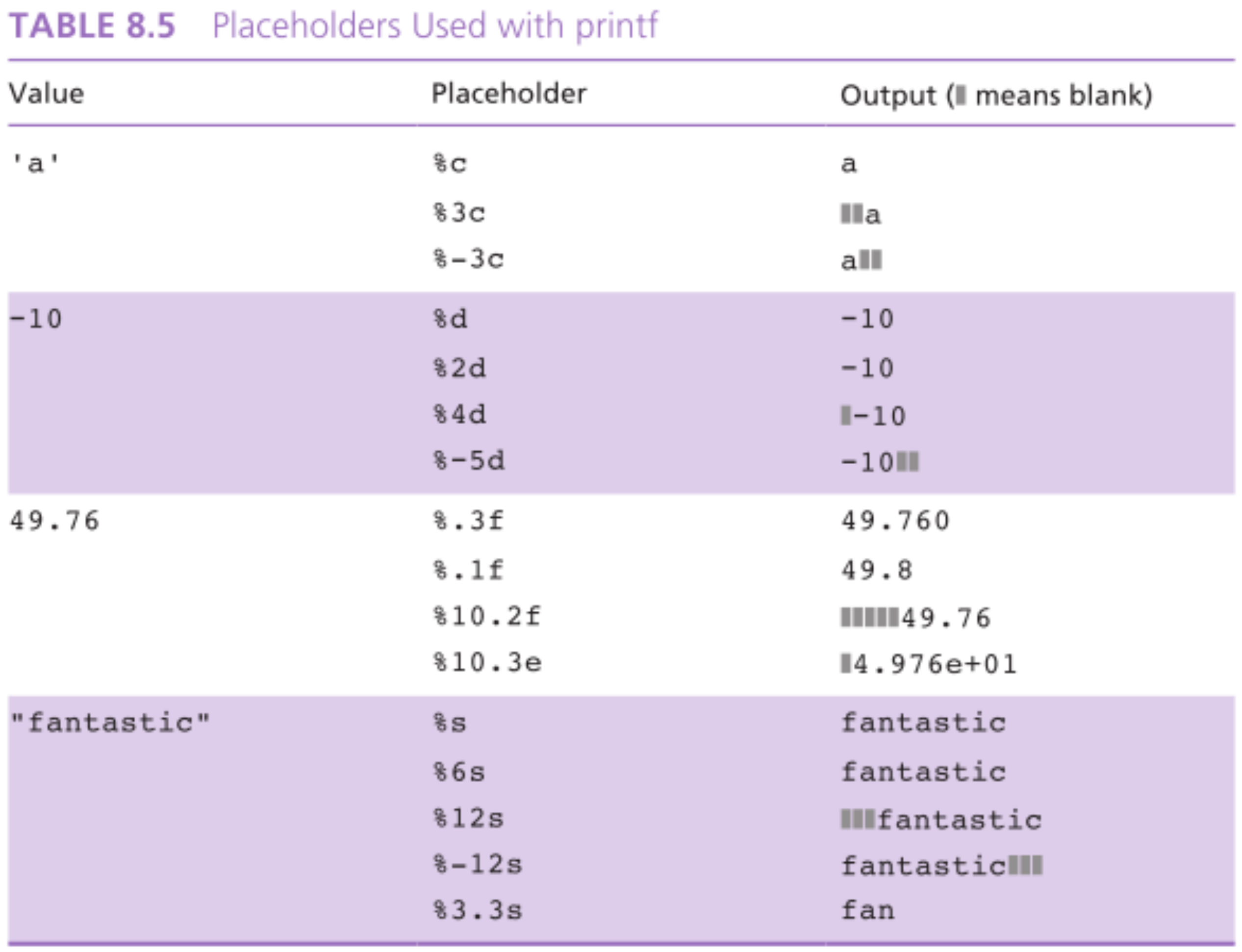


TABLE 8.5 Placeholders Used with printf

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

```
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('/');
}
```

FIGURE 8.18 Functions That Convert Representations of Dates

```
/*
 * Functions to change the representation of a date from a string containing
 * day, month name and year to three integers (month day year) 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[]);
```

(continued)

FIGURE 8.18 (continued)

```
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)
{
    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);
}
```

(continued)

FIGURE 8.18 (continued)

```
#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
 */
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         */
{
      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
```