Chapter 9

Strings

Figure 9.1 Right and Left Justification of Strings

Right-Justified	Left-Justified
George Washington	George Washington
John Adams	John Adams
Thomas Jefferson	Thomas Jefferson
James Madison	James Madison

Figure 9.2 String Input/Output with scanf and printf

```
#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
```


Figure 9.3 Execution of scanf ("%s", dept);

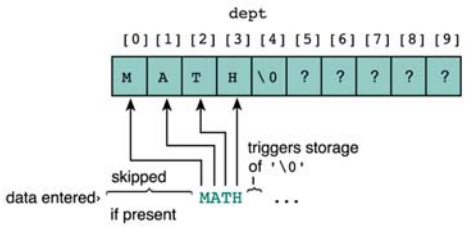

Figure 9.4 Execution of scanf("%s%d%s%d", dept, &course_num, days, &time); on Entry of Invalid Data

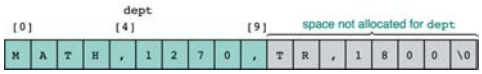

Figure 9.5 Execution of strncpy(result, s1, 9);

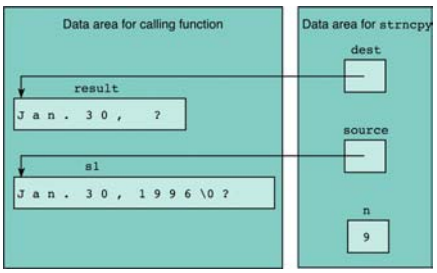

Figure 9.6 Execution of strncpy(result, &s1[5], 2);

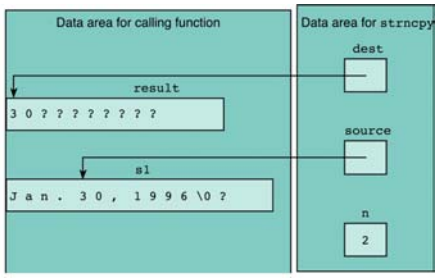

Figure 9.7 Program Using strncpy and strcpy Functions to Separate Compounds into Elemental Components

```
/*
 *  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
```


Figure 9.8 Demonstration of Whole-Line Input

```
/*
 *  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)

Figure 9.8 Demonstration of Whole-Line Input

```
    return (0);
}
```

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.
```


Figure 9.9 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	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);`

Figure 9.10 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 */
    ...
}
```

Figure 9.11 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 9.12 Executing strcpy (list [index_of_min], list[fill]);

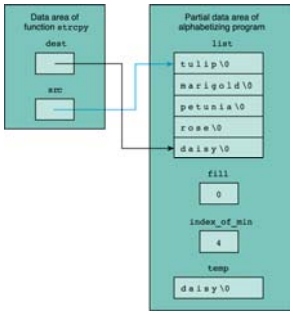

Figure 9.13 An Array of Pointers