Sams Teach Yourself C in 24 Hours

Hour 19 - Collecting Data Items of Different Types

The art of programming is the art of organizing complexity.

—W. W. Dijkstra

In Hour 12, "Storing Similar Data Items," you learned how to store data of the same type into arrays. In this hour, you'll learn to use structures to collect data items that have different data types. The following topics are covered in this lesson:

- Declaring and defining structures
- Referencing structure members
- Structures and pointers
- Structures and functions
- Arrays of structures

What Is a Structure?

As you've learned, arrays can be used to collect groups of variables of the same type. The question now is how to aggregate pieces of data that are not identically typed.

The answer is that you can group variables of different types with a data type called a structure. In C, a structure collects different data items in such a way that they can be referenced as a single unit.

There are several major differences between an array and a structure. Besides the fact that data items in a structure can have different types, each data item has its own name instead of a subscript value. In fact, data items in a structure are called fields or members of the structure.

The next two subsections teach you how to declare structures and define structure variables.

Declaring Structures

The general form to declare a structure is

```
struct struct_tag {
     data_type1 variable1;
     data_type2 variable2;
     data_type3 variable3;
     .
     .
     .
     };
```

Here struct is the keyword used in C to start a structure declaration. struct_tag is the tag name of the structure. variable1, variable2, and variable3 are the members of the structure. Their data types are specified respectively by data_type1, data_type2, and data_type3. As you can see, the declarations of the members have to be enclosed within the opening and closing braces ({ and }) in the structure declaration, and a semicolon (;) has to be included at the end of the declaration.

The following is an example of a structure declaration:

```
struct automobile {
     int year;
     char model[8];
     int engine_power;
     float weight;
     };
```

Here struct is used to start a structure declaration. automobile is the tag name of the structure. In this example, there are three types of variables, char, int, and float. The variables have their own names, such as year, model, engine_power, and weight.

Note that a structure tag name, like automobile, is a label to a structure. The compiler uses the tag name to identify the structure labeled by that tag name.

Defining Structure Variables

After declaring a structure, you can define the structure variables. For instance, the following structure variables are defined with the structure data type of automobile from the previous section:

```
struct automobile sedan, pick_up, sport_utility;
```

Here three structure variables, sedan, pick_up, and sport_utility, are defined by the structure of automobile. All three structure variables contain the four members of the structure data type of automobile.

Also, you can combine the structure declaration and definition into one statement like this:

```
struct automobile {
     int year;
     char model[8];
     int engine_power;
     float weight;
     } sedan, pick_up, sport_utility;
```

Here three structure variables, sedan, pick_up, and sport_utility, are defined with the structure data type of automobile in the single statement.

Referencing Structure Members with the Dot Operator

Now, let's see how to reference a structure member. Given the structure data type of automobile and the structure of sedan, for instance, I can access its member, year, and assign an integer to it in the following way:

```
sedan.year = 1997;
```

Here the structure name and its member's name are separated by the dot (.) operator so that the compiler knows that the integer value of 1997 is assigned to the variable called year, which is a member of the structure sedan.

Likewise, the following statement assigns the start address of the character array of model, which is another member of the structure sedan, to a char pointer, ptr:

```
ptr = sedan.model;
```

The program in Listing 19.1 gives another example to reference the members of a structure.

TYPE

Listing 19.1. Referencing the members of a structure.

```
1:  /* 19L01.c Access to structure members */
2:  #include <stdio.h>
3:
4:  main(void)
5:  {
6:     struct computer {
7:        float cost;
8:        int year;
9:        int cpu_speed;
10:       char cpu_type[16];
11:       } model;
12:
13:    printf("The type of the CPU inside your computer?\n");
14:       gets(model.cpu_type);
15:    printf("The speed(MHz) of the CPU?\n");
16:       scanf("%d", &model.cpu_speed);
17:    printf("The year your computer was made?\n");
18:       scanf("%d", &model.year);
19:    printf("How much you paid for the computer?\n");
20:       scanf("%f", &model.cost);
21:
22:    printf("Here are what you entered:\n");
23:    printf("Year: %d\n", model.year);
24:    printf("Cost: $%6.2f\n", model.cost);
25:    printf("CPU type: %s\n", model.cpu_type);
26:    printf("CPU speed: %d MHz\n", model.cpu_speed);
27:
28:    return 0;
29: }
```

I have the following output shown on the screen after I run the executable (19L01.exe) of the program in Listing 19.1 and enter my answers to the questions (in the output, the bold characters or numbers are the answers entered from my keyboard):

OUTPUT

```
C:\app>19L01
The type of the CPU inside your computer?
Pentium
The speed(MHz) of the CPU?
100
The year your computer was made?
1996
How much you paid for the computer?
1234.56
Here are what you entered:
Year: 1996

Cost: $1234.56

CPU type: Pentium
CPU speed: 100 MHz
C:\app>
```

ANALYSIS

The purpose of the program in Listing 19.1 is to show you how to reference members of a structure. As you can see from the program, there is a structure called model that is defined with a structure data type of computer in lines 6_11. The structure has one float variable, two int variables, and one char array.

The statement in line 13 asks the user to enter the type of CPU (central processing unit)
used inside his or her computer. Then, line 14 receives the string of the CPU type entered by the user and saves the string into the char array called cpu_type. Because cpu_type is a member of the model structure, the model.cpu_type expression is used in line 14 to reference the member of the structure. Note that the dot operator (.) is used to separate the two names in the expression.

Lines 15 and 16 ask for the CPU speed and store the value of an integer entered by the user to another member of the model structure—the

int variable cpu_speed. Note that in line 16, the address-of operator (&)is prefixed to the model.cpu_speed expression inside the scanf() function because the argument should be an int pointer.

Likewise, lines 17 and 18 receive the value of the year in which the user's computer was made, and lines 19 and 20 get the number for the cost of the computer. After the execution, the int variable year and the float variable cost in the model structure contain the corresponding values entered by the user.

Then, lines 23_26 print out all values held by the members of the model structure. From the output, you can tell that each member of the structure has been accessed and assigned a number or string correctly.

Initializing Structures

A structure can be initialized by a list of data called initializers. Commas are used to separate data items in a list of data.

Listing 19.2 contains an example of initializing a structure before it's updated by the user.

TYPE

Listing 19.2. Initializing a structure.

```
1:  /* 19L02.c Initializing a structure */
2:  #include <stdio.h>
3:
4:  main(void)
5:  {
6:     struct employee {
7:        int id;
8:        char name[32];
9:        };
10:    /* structure initialization */
11:    struct employee info = {
12:       1,
13:       "B. Smith"
14:       };
15:
16:    printf("Here is a sample:\n");
17:    printf("Employee Name: %s\n", info.name);
18:    printf("Employee ID #: %04d\n\n", info.id);
19:
20:    printf("What's your name?\n");
21:       gets(info.name);
22:    printf("What's your ID number?\n");
23:       scanf("%d", &info.id);
24:
25:    printf("\nHere are what you entered:\n");
26:    printf("Name: %s\n", info.name);
27:    printf("ID #: %04d\n", info.id);
28:
29:    return 0;
30: }
```

When the executable 19L02.exe is being run, the initial content saved in a structure is displayed. Then, I enter my answers to the questions and get the updated information shown on the screen:

OUTPUT

```
C:\app>19L02
Here is a sample:
Employee Name: B. Smith
Employee ID #: 0001

What's your name?
T. Zhang
What's your ID number?
1234

Here are what you entered:
Name: T. Zhang
ID #: 1234
C:\app>
```

ANALYSIS

The purpose of the program in Listing 19.2 is to initialize a structure and then ask the user to update the content held by the structure.

The structure data type, labeled as employee, is declared in lines 6_9. Then, the variable, info, is defined with the structure data type and initialized with the integer 1 and the string "B. Smith" in lines 11_14.

You can also combine the declaration, definition, and initialization of a structure into a single statement. Here's an example:

```
struct employee {
       int id;
       char name[32];
       } info = {
       1,
       "B. Smith"
       };
```

The statements in lines 17 and 18 display the initial contents stored by the two members of the info structure on the screen. Then, lines

20_23 ask the user to enter his or her name and employee ID number and save them into the two structure members, name and id, respectively.

Before the end of the program, the updated contents contained by the two members are printed out by the statements in lines 26 and 27.

Again, the dot operator (.) is used in the program to reference the structure members.

Structures and Function Calls

The C language allows you to pass an entire structure to a function. In addition, a function can return a structure back to its caller.

To show you how to pass a structure to a function, I rewrite the program in Listing 19.1 and create a function called DataReceive() in the program. The upgraded program is shown in Listing 19.3.

TYPE

Listing 19.3. Passing a structure to a function.

```
1:  /* 19L03.c Passing a structure to a function */
2:  #include <stdio.h>
3:
4:  struct computer {
5:     float cost;
6:     int year;
7:     int cpu_speed;
8:     char cpu_type[16];
9:  };
10: /* create synonym */
11: typedef struct computer SC;
12: /* function declaration */
13: SC DataReceive(SC s);
14:
15: main(void)
16: {
17:    SC model;
18:
19:    model = DataReceive(model);
20:    printf("Here are what you entered:\n");
21:    printf("Year: %d\n", model.year);
22:    printf("Cost: $%6.2f\n", model.cost);
23:    printf("CPU type: %s\n", model.cpu_type);
24:    printf("CPU speed: %d MHz\n", model.cpu_speed);
25:
26:    return 0;
27: }
28: /* function definition */
29: SC DataReceive(SC s)
30: {
31:    printf("The type of the CPU inside your computer?\n");
32:       gets(s.cpu_type);
33:    printf("The speed(MHz) of the CPU?\n");
34:       scanf("%d", &s.cpu_speed);
35:    printf("The year your computer was made?\n");
36:       scanf("%d", &s.year);
37:    printf("How much you paid for the computer?\n");
38:       scanf("%f", &s.cost);
39:    return s;
40: }
```

After I run the executable, 19L03.exe, and enter my answers to the questions, I get the following output, which is the same as the output from the executable program of List-
ing 19.1:

OUTPUT

```
C:\app>19L03
The type of the CPU inside your computer?
Pentium
The speed(MHz) of the CPU?
100
The year your computer was made?
1996
How much you paid for the computer?
1234.56
Here are what you entered:
Year: 1996
Cost: $1234.56
CPU type: Pentium
CPU speed: 100 MHz
C:\app>
```

ANALYSIS

The purpose of the program in Listing 19.3 is to show you how to pass a structure to a function. The structure in Listing 19.3, with the tag name of computer, is declared in lines 4_9.

Note that in line 11 the typedef keyword is used to define a synonym, SC, for structure computer. Then SC is used in the sequential declarations.

The DataReceive() function is declared in line 13, with the structure of computer as its argument (that is, the synonym SC and the variable

name s), so that a copy of the structure can be passed to the function.

In addition, the DataReceive() function returns the copy of the structure back to the caller after the content of the structure is updated. To do this, SC is prefixed to the function in line 13 to indicate the data type of the value returned by the function.

The statement in line 17 defines the structure model with SC. The DataReceive() function is passed with the name of the model structure in line 19, and then the value returned by the function is assigned back to model as well. Note that if the DataReceive() function return value is not assigned to model, the changes made to s in the function will not be evident in model.

The definition of the DataReceive() function is shown in lines 29_40, from which you can see that the new data values entered by the user are saved into the corresponding members of the structure that is passed to the function. At the end of the function, the copy of the updated structure is returned in line 39.

Then, back to the main() function of the program, lines 21_24 print out the updated contents held by the members of the structure. Because the program in Listing 19.3 is basically the same as the one in Listing 19.1, I see the same output on my screen after running the executable file, 19L03.exe.

Pointing to Structures

As you can pass a function with a pointer that refers to an array, you can also pass a function with a pointer that points to a structure.

However, unlike passing a structure to a function, which sends an entire copy of the structure to the function, passing a pointer of a structure to a function is simply passing the address that associates the structure to the function. The function can then use the address to access the structure members without duplicating the structure. Therefore, it's more efficient to pass a pointer of a structure, rather than the structure itself, to a function.

Accordingly, the program in Listing 19.3 can be rewritten to pass the DataReceive() function with a pointer that points to the structure. The rewritten program is shown in Listing 19.4.

TYPE

Listing 19.4. Passing a function with a pointer that points to a structure.

```
1:  /* 19L04.c Pointing to a structure */
2:  #include <stdio.h>
3:
4: struct computer {

5:    float cost;
6:    int year;
7:    int cpu_speed;
8:    char cpu_type[16];
9: };
10:
11: typedef struct computer SC;
12:
13: void DataReceive(SC *ptr_s);
14:
15: main(void)
16: {
17:    SC model;
18:
19:    DataReceive(&model);
20:    printf("Here are what you entered:\n");
21:    printf("Year: %d\n", model.year);
22:    printf("Cost: $%6.2f\n", model.cost);
23:    printf("CPU type: %s\n", model.cpu_type);
24:    printf("CPU speed: %d MHz\n", model.cpu_speed);
25:
26:    return 0;
27: }
28: /* function definition */
29: void DataReceive(SC *ptr_s)
30: {
31:    printf("The type of the CPU inside your computer?\n");
32:       gets((*ptr_s).cpu_type);
33:    printf("The speed(MHz) of the CPU?\n");
34:       scanf("%d", &(*ptr_s).cpu_speed);
35:    printf("The year your computer was made?\n");
36:       scanf("%d", &(*ptr_s).year);
37:    printf("How much you paid for the computer?\n");
38:       scanf("%f", &(*ptr_s).cost);
39: }
```

Similarly, I obtain output that is the same as the one from the program in Listing 19.3 after I run the executable (19L04.exe) of the program in Listing 19.4:

OUTPUT

```
C:\app>19L04
The type of the CPU inside your computer?
Pentium
The speed(MHz) of the CPU?
100
The year your computer was made?
1996
How much you paid for the computer?
1234.56
Here are what you entered:
```

```
Year: 1996
Cost: $1234.56
CPU type: Pentium
CPU speed: 100 MHz

C:\app>
```

ANALYSIS

The program in Listing 19.4 is almost identical to the one in Listing 19.3, except that the argument passed to the DataReceive() function is a pointer defined with SC—that is, structure computer. (Refer to lines 11 and 13.) Also, the DataReceive() function does not need to return a copy of the structure because the function can access all members of the original structure, not the copy, via the pointer passed to it. That's why the void keyword is prefixed to the function name in line 13.

The statement in line 17 defines the structure model. And in line 19, the address of the model structure is passed to the DataReceive function by applying the address-of operator (&).

When you look at the definition of the DataReceive() function in lines 29_39, you see that the dereferenced pointer *ptr_s is used to reference the members of the model structure. For instance, to access the char array of cpu_type, (*ptr_s) is used in the (*ptr_s).cpu_type expression to indicate to the compiler that cpu_type is a member in the structure pointed to by the pointer ptr_s. Note that the dereferenced pointer *ptr_s has to be enclosed within the parentheses ((and)).

Another example is the &(*ptr_s).cpu_speed expression in line 34, which leads to the address of the cpu_speed variable that is a member of the structure pointed to by the pointer ptr_s. Again, the dereferenced pointer *ptr_s is surrounded by the parentheses ((and)).

The next subsection shows you how to use the arrow operator (->) to refer to a structure member with a pointer.

Referencing a Structure Member with ->

You can use the arrow operator -> to refer to a structure member with a pointer that points to the structure.

For instance, you can rewrite the (*ptr_s).cpu_type expression in Listing 19.4 with this:

```
ptr_s -> cpu_type
```

or you could replace the &(*ptr_s).cpu_speed expression with this:

```
&(ptr_s->cpu_speed)
```

Because of its clearness, the -> operator is more frequently used in programs than the dot operator. Exercise 3, later in this hour, gives you a chance to rewrite the entire program in Listing 19.4 using the -> operator.

Arrays of Structures

In C, you can declare an array of a structure by preceding the array name with the structure name. For instance, given a structure with the tag name of x, the following statement

```
struct x array_of_structure[8];
```

declares an array, called array_of_structure, with the structure data type of x. The array has eight elements.

The program shown in Listing 19.5 demonstrates how to use an array of a structure by printing out two pieces of Japanese haiku and their authors' names.

TYPE

Listing 19.5. Using arrays of structures.

```
1:  /* 19L05.c Arrays of structures */
2:  #include <stdio.h>
3:
4:  struct haiku {
5:     int start_year;
6:     int end_year;
7:     char author[16];
8:     char str1[32];
9:     char str2[32];
10:    char str3[32];
11: };
12:
13: typedef struct haiku HK;
14:
15: void DataDisplay(HK *ptr_s);
16:
17: main(void)
18: {
19:    HK poem[2] = {
20:      { 1641,
21:        1716,
22:        "Sodo",
23:        "Leading me along",
24:        "my shadow goes back home",
25:        "from looking at the moon."
26:      },
27:      { 1729,
28:        1781,
29:        "Chora",
```

```
30:        "A storm wind blows",
31:        "out from among the grasses",
32:        "the full moon grows."
33:      }
34:    };
35:    int i;
36:
37:    for (i=0; i<2; i++)
38:       DataDisplay(&poem[i]);
39:
40:    return 0;
41: }
42: /* function definition */
43: void DataDisplay(HK *ptr_s)
44: {
45:    printf("%s\n", ptr_s->str1);

46:    printf("%s\n", ptr_s->str2);
47:    printf("%s\n", ptr_s->str3);
48:    printf("--- %s\n", ptr_s->author);
49:    printf("   (%d-%d)\n\n", ptr_s->start_year, ptr_s->end_year);
50: }
```

After running the executable (19L05.exe) of the program in Listing 19.5, I see the two pieces of Japanese haiku printed on the screen:

OUTPUT

```
C:\app>19L05
Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
   (1641-1716)

A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
   (1729-1781)
C:\app>
```

ANALYSIS

In Listing 19.5, a structure data type, with the tag name of haiku, is declared in lines 4_11. The structure data type contains two int variables and four char arrays as its members. The statement in line 13 creates a synonym, HK, for the struct haiku data type.

Then, in lines 19_34, an array of two elements, poem, is declared and initialized with two pieces of haiku written by Sodo and Chora, respectively. The following is a copy of the two pieces of haiku from poem:

```
"Leading me along",
"my shadow goes back home",
"from looking at the moon."
```

and

```
"A storm wind blows",
"out from among the grasses",
"the full moon grows."
```

The initializer also includes the authors' names and the years of their births and deaths (refers to lines 20_22 and lines 27_29). Note that the poem array, declared with HK, is indeed an array of the haiku structure.

The DataDisplay() function is called twice in a for loop in lines 37 and 38. Each time, the address of an element of poem is passed to the DataDisplay() function. According to the definition of the function in lines 43_50, DataDisplay() prints out three strings of a haiku, the author's name, and the period of time in which he lived.

From the output, you can see that the contents stored in the poem array of the haiku structure are displayed on the screen properly.

Nested Structures

A structure is called a nested structure when one of the members of the structure is itself a structure. For instance, given a structure data type of x, the following statement

```
struct y {
   int i;
   char ch[8];
   struct x nested;
};
```

declares a nested structure with a tag name of y, because one of the members of the y structure is a structure with the variable name of nested that is defined by the structure data type
of x.

Listing 19.6 contains an example of using a nested structure to receive and print out information about an employee.

TYPE

Listing 19.6. Using nested structures.

```
1:  /* 19L06.c Using nested structures */
2:  #include <stdio.h>
3:
4:  struct department {
5:     int  code;
6:     char name[32];
7:     char position[16];
8:  };
9:
10: typedef struct department DPT;
11:
12: struct employee {
13:    DPT d;
14:    int id;
15:    char name[32];
16: };
17:
18: typedef struct employee EMPLY;
19:
20: void InfoDisplay(EMPLY *ptr);
21: void InfoEnter(EMPLY *ptr);
22:
23: main(void)
24: {
25:    EMPLY info = {
26:       { 1,
27:         "Marketing",
28:         "Manager"
29:       },
30:       1,
31:       "B. Smith"
32:    };
33:
34:    printf("Here is a sample:\n");
35:    InfoDisplay(&info);
36:
37:    InfoEnter(&info);
38:
39:    printf("\nHere are what you entered:\n");
40:    InfoDisplay(&info);
41:
42:    return 0;
43: }
44: /* function definition */
45: void InfoDisplay(EMPLY *ptr)
46: {
47:    printf("Name: %s\n", ptr->name);
48:    printf("ID #: %04d\n", ptr->id);
49:    printf("Dept. name: %s\n", ptr->d.name);
50:    printf("Dept. code: %02d\n", ptr->d.code);
51:    printf("Your position: %s\n", ptr->d.position);
52: }
53: /* function definition */
54: void InfoEnter(EMPLY *ptr)
55: {
56:    printf("\nPlease enter your information:\n");
57:    printf("Your name:\n");
58:       gets(ptr->name);
59:    printf("Your position:\n");
60:       gets(ptr->d.position);
61:    printf("Dept. name:\n");
62:       gets(ptr->d.name);
63:    printf("Dept. code:\n");
64:       scanf("%d", &(ptr->d.code));
65:    printf("Your employee ID #:\n");
66:       scanf("%d", &(ptr->id));
67: }
```

When the executable, 19L06.exe, is running, the initial content of the nested structure is printed out first. Then, I enter my employment information, which is in bold in the following output and displayed back on the screen, too:

OUTPUT

```
C:\app>19L06
Here is a sample:
Name: B. Smith
ID #: 0001
Dept. name: Marketing
Dept. code: 01
Your position: Manager

Please enter your information:\n");
Your name:
T. Zhang
Your position:\n");
Engineer
Dept. name:
R&D
Dept. code:
3

Your employee ID #:
1234

Here are what you entered:
Name: T. Zhang
```

```
ID #: 1234
Dept. name: R&D
Dept. code: 03
Your position: Engineer
C:\app>
```

ANALYSIS

There are two structure data types in Listing 19.6. The first one, called department, is declared in lines 4_8. The second one, employee, declared in lines 12_16, contains a member of the department structure data type. Therefore, the employee structure data type is a nested structure data type.

Two synonyms, DPT for the struct department data type, and EMPLY for the struct employee data type, are created in two typedef statements, respectively, in lines 10 and 18. In the program, there are two functions, InfoDisplay() and InfoEnter(), whose prototypes are declared with a pointer of EMPLY as the argument (see lines 20 and 21).

The statement in 25_32 initializes a nested structure, which is called info and defined with EMPLY. Note that the nested braces ({ and }) in lines 26 and 29 enclose the initializers for the d structure of DPT that is nested inside the info structure.

Then, the statement in line 35 displays the initial contents held by the nested info structure by calling the InfoDisplay() function. Line 37 calls the InfoEnter() function to ask the user to enter his or her employment information and then save it into the info structure. The InfoDisplay() function is called again in line 40 to display the information that the user entered and is now stored in the nested structure.

The definitions for the two functions, InfoDisplay() and InfoEnter(), are listed in lines 45_52 and lines 54_67, respectively.

Forward-Referencing Structures

If one of the members of a structure is a pointer pointing to another structure that has not been declared yet, the structure is called a forward-referencing structure.

For example, the following statement declares a forward-referencing structure:

```
struct x {
   int i;
   char ch[8];
   struct y *ptr;
};
```

It is presumed that the structure y has not been declared yet.

If the pointer in a structure points to the structure itself, the structure is called a self-referencing structure. The following declaration is an example of a self-referencing structure:

```
struct x {
   int i;
   char ch[8];
   struct x *ptr;
};
```

Now, let's move to the program in Listing 19.7, which provides you with another example of declaring a forward-referencing structure.

TYPE

Listing 19.7. Using forward-referencing structures.

```
1:  /* 19L07.c Forward-referencing structures */
2:  #include <stdio.h>
3:  /* forward-referencing structure */
4:  struct resume {
5:     char name[16];
6:     struct date *u;
7:     struct date *g;
8:  };
9:  /* referenced structure */
10: struct date {
11:    int year;
12:    char school[32];
13:    char degree[8];
14: };
15:
16: typedef struct resume RSM;
17: typedef struct date DATE;
18:
19: void InfoDisplay(RSM *ptr);
20:
21: main(void)
22: {
23:    DATE under = {
24:        1985,
25:        "Rice University",
26:        "B.S."
27:        };
28:    DATE graduate = {
29:        1987,
30:        "UT Austin",
31:        "M.S."
32:        };
33:    RSM new_employee = {
34:        "Tony",
```

```
35:        &under,
36:        &graduate
37:        };
38:
39:    printf("Here is the new employee's resume:\n");
40:    InfoDisplay(&new_employee);
41:
42:    return 0;
43: }
44: /* function definition */
45: void InfoDisplay(RSM *ptr)
46: {
47:    printf("Name: %s\n", ptr->name);
48:    /* undergraduate */
49:    printf("School name: %s\n", ptr->u->school);
50:    printf("Graduation year: %d\n", ptr->u->year);
51:    printf("Degree: %s\n", ptr->u->degree);
52:    /* graduate */
53:    printf("School name: %s\n", ptr->g->school);
54:    printf("Graduation year: %d\n", ptr->g->year);
55:    printf("Degree: %s\n", ptr->g->degree);
56: }
```

I have the following output displayed on the screen after I run the executable, 19L07.exe, of the program in Listing 19.7:

OUTPUT

```
C:\app>19L07
Here is the new employee's resume:
Name: Tony
School name: Rice University
Graduation year: 1985
Degree: B.S.
School name: UT Austin
Graduation year: 1987
Degree: M.S.
C:\app>
```

ANALYSIS

The purpose of the program in Listing 19.7 is to show you how to declare and initialize a forward-referencing structure. As you can see, the forward-referencing structure data type, labeled with resume, is declared in lines 4_8. This structure data type has two members, u and g, which are pointers defined with the date structure data type. However, the date structure data type has not yet been declared at this moment. The structure data type of resume is thus called a forward-referencing structure data type.

The referenced date structure data type is declared in lines 10_14. Then, I make two synonyms, RSM and DATE, to represent struct resume and struct date, respectively, in lines 16 and 17.

Because there are two pointers inside the resume structure data type that reference the date structure data type, I define and initialize two structures with DATE in lines 23_32. under and graduate are the names of the two structures.

Then, in lines 33_37, a structure, called new_employee, is defined with RSM and initialized with the addresses of the two structures, under and graduate (see lines 35 and 36). In this way, the forward-referencing structure new_employee is assigned with access to the contents of the two referenced structures, under and graduate.

The statement in line 40, which calls the InfoDisplay() function, prints out all the contents held by the new_employee structure, as well as that of the under and graduate structures.

Lines 45_56 give the definition of the InfoDisplay() function, from which you can see that the expressions, such as ptr->u->school and ptr->g->school, are used to reference the members in the under and graduate structures. Since ptr, u, and g are all pointers, two arrows (->) are used in the expressions.

In exercise 5, later in this hour, you'll be asked to rewrite the program in Listing 19.7 with an array of pointers to replace the two pointers defined inside the forward-referencing structure of resume.

WARNING

Don't include forward references in a typedef statement. It's not legal in C.

Summary

In this lesson you've learned the following:

- You can group variables of different types with a data type called a structure.
- The data items in a structure are called fields or members of the structure.
- The struct keyword is used to start a structure declaration or a structure variable definition.
- The dot operator (.) is used to separate a structure name and a member name in referencing the structure member.
- The arrow operator (->) is commonly used to reference a structure member with a pointer.
- A structure can be passed to a function, and a function can return a structure back to the caller.
- Passing a function with a pointer that points to a structure is more efficient than passing the function with the entire structure.

- Arrays of structures are permitted in C.
- You can enclose a structure within another structure. The latter is called a nested structure.
- It's legal to put a pointer into a structure even though the pointer may point to another structure that has not been declared yet.

In the next lesson you'll learn to use unions to collect dissimilar data items in C.

Q&A

Q Why do you need structures?

A In practice, you need to collect and group data items that are relevant but of different types. The structure data type provides a convenient way to aggregate those differently typed data items.

Q Can you declare a structure and define a structure variable in a single statement?

A Yes. You can put the struct keyword, a tag name, a list of declarations of structure members, and a variable name into a single statement to declare a structure and define a structure variable. Then, the structure can be identified by the tag name; the variable is of the struct data type of the tag name.

Q How do you reference a structure member?

A You can reference a structure member by prefixing the structure member's name with the structure variable name and a dot operator (.). If the structure is pointed to by a pointer, you can use the arrow operator (->), followed by the pointer name, to reference the structure member.

Q Why is it more efficient to pass a pointer that refers to a structure to a function?

A When an entire structure is passed to a function, a copy of the structure is made and saved in a temporary block of memory called the stack. After the copy is modified by the function, it has to be returned and written back to the storage that holds the original content of the structure. Passing a function with a pointer that points to a structure, on the other hand, simply passes the address of the structure to the function, not the entire copy of the structure. The function can then access the original memory location of the structure and modify the content held by the structure without duplicating the structure on the stack. Therefore, it's more efficient to pass a pointer of a structure than to

pass the structure itself to a function.

Workshop

To help solidify your understanding of this hour's lesson, you are encouraged to answer the quiz questions and finish the exercises provided in the Workshop before you move to the next lesson. The answers and hints to the questions and exercises are given in Appendix E, "Answers to Quiz Questions and Exercises."

Quiz

1. What's wrong with the following structure declaration?

```
struct automobile {
     int year;
     char model[8];
     int engine_power;
     float weight;
     }
```

2. How many structure variables are defined in the following statement?

```
struct x {int y; char z} u, v, w;
```

3. Given a structure declaration

```
struct automobile {
     int year;
     char model[8]};
```

and two car models, Taurus and Accord, which are made in 1997, initialize an array of two elements, car, that is defined with the automobile structure data type.

4. In the following structure declarations, which one is a forward-referencing structure, and which one is a self-referencing structure? (Assume that the structures, employment and education, have not been declared yet.)

```
struct member {
     char name[32];
     struct employment *emp;
     struct education *edu;};

struct list {
     int x, y;
     float z;
     struct list *ptr_list;};
```

Exercises

1. Given the following declaration and definition of a structure

```
struct automobile {
     int year;
     char model[10];
     int engine_power;
     double weight;
     } sedan = {
     1997,
     "New Model",
```

```
200,
2345.67};
```

Previous | Table of Contents | Next