Example 10.1

Define a structure type, **struct personal** that would contain person name, date of joining and salary. Using this structure, write a program to read this information for one person from the keyboard and print the same on the screen.

Structure definition along with the program is shown in Fig.10.1. The **scanf** and **printf** functions illustrate how the member operator `.' is used to link the structure members to the structure variables. The variable name with a period and the member name is used like an ordinary variable.

DEFINING AND ASSIGNING VALUES TO STRUCTURE MEMBERS

Program

```
struct  personal
{
    char  name[20];
    int   day;
    char  month[10];
    int   year;
    float salary;
};
main()
{
    struct personal person;
    printf("Input Values\n");
    scanf("%s %d %s %d %f",
              person.name,
             &person.day,
              person.month,
             &person.year,
             &person.salary);
    printf("%s %d %s %d %f\n",
              person.name,
              person.day,
              person.month,
              person.year,
              person.salary);
}
```

Output

```
Input Values

M.L.Goel 10 January 1945 4500
M.L.Goel 10 January 1945 4500.00
```

Fig.10.1 *Defining and accessing structure members*

Example 10.2

Write a program to illustrate the comparison of structure variables.

The program shown in Fig.10.2 illustrates how a structure variable can be copied into another of the same type. It also performs member-wise comparison to decide whether two structure variables are identical.

COMPARISON OF STRUCTURE VARIABLES

Program

```
struct class
{
    int  number;
    char name[20];
    float marks;
};

main()
{
    int  x;
    struct class student1 = {111,"Rao",72.50};
    struct class student2 = {222,"Reddy", 67.00};
    struct class student3;

    student3 = student2;

    x = ((student3.number ==  student2.number) &&
         (student3.marks  ==  student2.marks)) ? 1 : 0;

    if(x == 1)
   {
        printf("\nstudent2 and student3 are same\n\n");
        printf("%d %s %f\n", student3.number,
                             student3.name,
                             student3.marks);
    }
```

```
    else
        printf("\nstudent2 and student3 are different\n\n");

}
```

Output

```
student2 and student3 are same

222 Reddy 67.000000
```

***Fig.10.2** Comparing and copying structure variables*

Example 10.3

For the **student** array discussed above, write a program to calculate the subject-wise and student-wise totals and store them as a part of the structure.

The program is shown in Fig.10.4. We have declared a four-member structure, the fourth one for keeping the student-totals. We have also declared an **array** total to keep the subject-totals and the grand-total. The grand-total is given by **total.total.** Note that a member name can be any valid C name and can be the same as an existing structure variable name. The linked name **total.total** represents the **total** member of the structure variable total.

ARRAYS OF STRUCTURES

Program

```
struct marks
{
    int  sub1;
    int  sub2;
    int  sub3;
    int  total;
};
main()
{
    int  i;
    struct marks student[3] =  {{45,67,81,0},
                                {75,53,69,0},
                                {57,36,71,0}};
    struct marks total;

    for(i = 0; i <= 2; i++)
    {
        student[i].total = student[i].sub1 +
                           student[i].sub2 +
                           student[i].sub3;
        total.sub1 = total.sub1 + student[i].sub1;
        total.sub2 = total.sub2 + student[i].sub2;
        total.sub3 = total.sub3 + student[i].sub3;
        total.total = total.total + student[i].total;
```

```
    }
    printf(" STUDENT          TOTAL\n\n");
    for(i = 0; i <= 2; i++)
       printf("Student[%d]      %d\n", i+1,student[i].total);

    printf("\n SUBJECT          TOTAL\n\n");

    printf("%s       %d\n%s       %d\n%s       %d\n",
           "Subject 1   ", total.sub1,
           "Subject 2   ", total.sub2,
           "Subject 3   ", total.sub3);

    printf("\nGrand Total = %d\n", total.total);
}
```

Output

```
 STUDENT           TOTAL
 Student[1]         193
 Student[2]         197
 Student[3]         164

 SUBJECT           TOTAL
 Subject 1          177
 Subject 2          156
 Subject 3          221

 Grand Total  = 554
```

***Fig.10.4** Illustration of subscripted structure variables*

Example 10.4

Rewrite the program of Example 10.3 using an array member to represent the three subjects.

The modified program is shown in Fig.10.5. You may notice that the use of array name for subjects has simplified in code.

ARRAYS WITHIN A STRUCTURE

Program

```
main()
{
    struct  marks
    {
        int  sub[3];
        int  total;
    };
```

```
    struct marks student[3] =
    {45,67,81,0,75,53,69,0,57,36,71,0};

    struct marks total;
    int  i,j;

    for(i = 0; i <= 2; i++)
    {
       for(j = 0; j <= 2; j++)
       {
          student[i].total += student[i].sub[j];
          total.sub[j] += student[i].sub[j];
       }
       total.total += student[i].total;
    }
    printf("STUDENT         TOTAL\n\n");
    for(i = 0; i <= 2; i++)
       printf("Student[%d]      %d\n", i+1, student[i].total);

    printf("\nSUBJECT         TOTAL\n\n");
    for(j = 0; j <= 2; j++)
       printf("Subject-%d        %d\n", j+1, total.sub[j]);

    printf("\nGrand Total  =   %d\n", total.total);

}
```

Output

```
STUDENT         TOTAL
Student[1]       193
Student[2]       197
Student[3]       164

SUBJECT         TOTAL
Subject-1        177
Subject-2        156
Subject-3        221

Grand Total  =   554
```

Fig.10.5 *Use of subscripted members in structures*

Example 10.5

Write a simple program to illustrate the method of sending an entire structure as a parameter to a function.

A program to update an item is shown in Fig.10.6. The function **update** receives a copy of the structure variable **item** as one of its parameters. Note that both the function **update** and the

formal parameter **product** are declared as type **struct stores.** It is done so because the function uses the parameter **product** to receive the structure variable **item** and also to return the updated values of **item**.

The function **mul** is of type **float** because it returns the product of **price** and **quantity.** However, the parameter **stock,** which receives the structure variable **item** is declared as type **struct stores.**

The entire structure returned by **update** can be copied into a structure of identical type. The statement

item = update(item,p_increment,q_increment);

replaces the old values of **item** by the new ones.

STRUCTURES AS FUNCTION PARAMETERS

Program

```
/*        Passing a copy of the entire structure        */

  struct stores
  {
      char  name[20];
      float price;
      int   quantity;
  };
  struct stores update (struct stores product, float p, int q);
  float mul (struct stores stock);

  main()
  {
      float    p_increment, value;
      int      q_increment;

      struct stores item = {"XYZ", 25.75, 12};

      printf("\nInput increment values:");
      printf("   price increment and quantity increment\n");
      scanf("%f %d", &p_increment, &q_increment);

  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
      item  = update(item, p_increment, q_increment);
  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
      printf("Updated values of item\n\n");
      printf("Name      : %s\n",item.name);
      printf("Price     : %f\n",item.price);
      printf("Quantity  : %d\n",item.quantity);

  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
      value  = mul(item);
  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
      printf("\nValue of the item  =  %f\n", value);
  }

  struct stores update(struct stores product, float p, int q)
  {
```

```
    product.price += p;
    product.quantity += q;
    return(product);
}
float mul(struct stores stock)
{
    return(stock.price * stock.quantity);
}
```

Output

```
Input increment values:   price increment and quantity increment
10 12
Updated values of item

Name      : XYZ
Price     : 35.750000
Quantity  : 24

Value of the item  =  858.000000
```

Fig.10.6 *Using structure as a function parameter*