Example 9.1

Write a program with multiple functions that do not communicate any data between them.

A program with three user-defined functions is given in Fig.9.4. **main** is the calling function that calls **printline** and **value** functions. Since both the called functions contain no arguments, there are no argument declarations. The **printline** function, when encountered, prints a line with a length of 35 characters as prescribed in the function. The **value** function calculates the value of principal amount after a certain period of years and prints the results. The following equation is evaluated repeatedly:

value = principal(1+interest-rate)

FUNCTIONS WITH THE ARGUMENTS, NO RETURN VALUES

Program

```
/* Function declaration */
void printline (void);
void value (void);

main()
{
    printline();
    value();
    printline();
}

/*       Function1: printline( )       */

void printline(void)   /* contains no arguments */
{
    int i ;

    for(i=1; i <= 35; i++)
       printf("%c",'-');
    printf("\n");
}

/*       Function2: value( )           */

void value(void)       /* contains no arguments */
{
    int    year, period;
    float  inrate, sum, principal;

    printf("Principal amount?");
    scanf("%f", &principal);
    printf("Interest rate?   ");
    scanf("%f", &inrate);
    printf("Period?          ");
    scanf("%d", &period);

    sum = principal;
    year = 1;
    while(year <= period)
    {
        sum = sum *(1+inrate);
        year = year +1;
    }
    printf("\n%8.2f %5.2f %5d %12.2f\n",
            principal,inrate,period,sum);
}
```

Output

```
     ----------------------------------
        Principal amount?  5000
        Interest rate?     0.12
        Period?            5

        5000.00  0.12      5      8811.71
     -----------------------------------
```

Fig.9.4 *Functions with no arguments and no return values*

Example 9.2

Modify the program of Example 9.1 to include the arguments in the function calls.

The modified program with function arguments is presented in Fig.9.7. Most of the program is identical to the program in Fig.9.4. The input prompt and **scanf** assignment statement have been moved from **value** function to **main.** The variables **principal, inrate,** and **period** are declared in **main** because they are used in main to receive data. The function call

value(principal, inrate, period);

passes information it contains to the function **value.**

The function header of **value** has three formal arguments **p,r,** and **n** which correspond to the actual arguments in the function call, namely, **principal, inrate,** and **period.** On execution of the function call, the values of the actual arguments are assigned to the corresponding formal arguments. In fact, the following assignments are accomplished across the function boundaries:

p = principal;
r = inrate;
n = period;

FUNCTIONS WITH ARGUMENTS BUT NO RETURN VALUES

Program

```
   /* prototypes */
   void printline (char c);
   void value (float, float, int);

   main( )
   {
        float principal, inrate;
        int period;

        printf("Enter principal amount, interest");
        printf(" rate, and period \n");
        scanf("%f %f %d",&principal, &inrate, &period);
        printline('Z');
        value(principal,inrate,period);
        printline('C');
   }
```

```
void printline(char ch)
{
     int i ;
     for(i=1; i <= 52; i++)
          printf("%c",ch);
     printf("\n");
}
void value(float p, float r, int n)
{
     int year ;
     float sum ;
     sum = p ;
     year = 1;
     while(year <= n)
     {
         sum = sum * (1+r);
         year = year +1;
     }
     printf("%f\t%f\t%d\t%f\n",p,r,n,sum);
}
```

Output

```
Enter principal amount, interest rate, and period
5000 0.12  5
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
5000.000000     0.120000        5       8811.708984
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

Fig.9.7 *Functions with arguments but no return values*

Example 9.3

In the program presented in Fig. 9.7 modify the function **value,** to return the final amount calculated to the **main**, which will display the required output at the terminal. Also extend the versatility of the function **printline** by having it to take the length of the line as an argument.

The modified program with the proposed changes is presented in Fig. 9.9. One major change is the movement of the **printf** statement from **value** to **main.**

FUNCTIONS WITH ARGUMENTS AND RETURN VALUES

Program

```
void printline (char ch, int len);
```

```
value (float, float, int);

main( )
{
      float principal, inrate, amount;
      int period;
      printf("Enter principal amount, interest");
      printf("rate, and period\n");
      scanf(%f %f %d", &principal, &inrate, &period);
      printline ('*' , 52);
      amount = value (principal, inrate, period);
      printf("\n%f\t%f\t%d\t%f\n\n",principal,
           inrate,period,amount);
      printline('=',52);
}
void printline(char ch, int len)
{
      int i;
      for (i=1;i<=len;i++) printf("%c",ch);
      printf("\n");
}
value(float p, float r, int n) /* default return type */
{
      int year;
      float sum;
      sum = p; year = 1;
      while(year <=n)
      {
           sum = sum * (l+r);
           year = year +1;
      }
      return(sum);        /* returns int part of sum */
}
```

Output

```
Enter principal amount, interest rate, and period
5000  0.12    5
****************************************************

5000.000000          0.1200000  5    8811.000000

= = = = = = = = = = = = = = = = = = = = = = = = = = =
```

Fig.9.9 *Functions with arguments and return values*

Example 9.4

Write a function **power** that computes x raised to the power y for integers x and y and returns double-type value.

Fig 9.10 shows a **power** function that returns a **double.** The prototype declaration

double power(int, int);

appears in **main**, before **power** is called.

```
                              POWER FUNCTIONS
```

Program

```
     main( )
     {    int x,y;                      /*input data */

          double power(int, int);     /* prototype declaration*/

          printf("Enter x,y:");

          scanf("%d %d" , &x,&y);
          printf("%d to power %d is %f\n", x,y,power (x,y));
     }

     double power (int x, int y);

     {
          double p;
          p = 1.0 ;         /* x to power zero */

          if(y >=o)
               while(y--)    /* computes positive powers */
                p *= x;
          else
               while (y++)   /* computes negative powers */
                p /= x;
          return(p);

     }
```

Output

```
     Enter x,y:16 2
     16 to power 2 is 256.000000

     Enter x,y:16 -2
     16 to power -2 is 0.003906
```

Fig 9.10 *Illustration of return of float values*

Example 9.5

Write a program to calculate the standard deviation of an array of values. The array elements are read from the terminal. Use functions to calculate standard deviation and mean.

Standard deviation of a set of n values is given by

$$S.D = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\bar{x}-x_i)^2}$$

Where $\bar{x}$ is the mean of the values.

FUNCTIONS WITH ARRAYS

Program

```
#include     <math.h>
#define SIZE    5
float std_dev(float a[], int n);
float mean (float a[], int n);

main( )
{
     float value[SIZE];
     int i;

     printf("Enter %d float values\n", SIZE);
     for (i=0 ;i < SIZE ; i++)
         scanf("%f", &value[i]);
     printf("Std.deviation is %f\n", std_dev(value,SIZE));
}

float std_dev(float a[], int n)

{    int i;
     float x, sum = 0.0;
     x = mean (a,n);
     for(i=0; i < n; i++)
        sum += (x-a[i])*(x-a[i]);
     return(sqrt(sum/(float)n));
}

float mean(float a[],int n)

{
     int i ;
     float sum = 0.0;
     for(i=0 ; i < n ; i++)
        sum = sum + a[i];
     return(sum/(float)n);
}
```

Output

```
          Enter 5 float values
          35.0 67.0 79.5 14.20 55.75

          Std.deviation is 23.231582
```

Fig.9.11 *Passing of arrays to a function*

Example 9.6

Write a program that uses a function to sort an array of integers.

A program to sort an array of integers using the function **sort()** is given in Fig.9.12. Its output clearly shows that a function can change the values in an array passed as an argument.

SORTING OF ARRAY ELEMENTS

Program

```
   void sort(int m, int x[ ]);
   main()
   {
       int i;
       int marks[5] = {40, 90, 73, 81, 35};

       printf("Marks before sorting\n");
       for(i = 0; i < 5; i++)
          printf("%d ", marks[i]);
       printf("\n\n");

       sort (5, marks);

       printf("Marks after sorting\n");
       for(i = 0; i < 5; i++)
          printf("%4d", marks[i]);
       printf("\n");
   }

   void sort(int m, int x[ ])

   {
       int i, j, t;
```

```
        for(i = 1; i <= m-1; i++)
           for(j = 1; j <= m-i; j++)
              if(x[j-1] >= x[j])
              {
                 t = x[j-1];
                 x[j-1] = x[j];
                 x[j] = t;
              }
    }
```

Output

```
           Marks before sorting
           40 90 73 81 35

           Marks after sorting
           35  40  73  81  90
```

Fig.9.12 *Sorting of array elements using a function*

Example 9.7

Write a multifunction to illustrate how automatic variables work.

A program with two subprograms **function1** and **function2** is shown in Fig.9.13. **m** is an automatic variable and it is declared at the beginning of each function. **m** is initialized to 10, 100, and 1000 in function1, function2, and **main** respectively.

When executed, **main** calls **function2** which in turn calls **function1**. When **main** is active, m = 1000; but when **function2** is called, the **main**'s **m** is temporarily put on the shelf and the new local **m** = 100 becomes active. Similarly, when **function1** is called, both the previous values of **m** are put on the shelf and the latest value of **m** (=10) becomes active. As soon as **function1** (m=10) is finished, **function2** (m=100) takes over again. As soon it is done, **main** (m=1000) takes over. The output clearly shows that the value assigned to **m** in one function does not affect its value in the other functions; and the local value of **m** is destroyed when it leaves a function.

ILLUSTRATION OF WORKING OF **auto** VARIABLES

Program

```
   void function1(void);
   void function2(void);
   main( )
   {
        int m = 1000;
        function2();

        printf("%d\n",m);   /* Third output */
   }
   void function1(void)
   {
        int m = 10;

        printf("%d\n",m);   /* First output */
   }


   void function2(void)
   {
        int m = 100;

        function1();
        printf("%d\n",m);   /* Second output */
   }
```

Output

```
        10
        100
        1000
```

Fig.9.13 *Working of automatic variables*

Example 9.8

Write a multifunction program to illustrate the properties of global variables.

A program to illustrate the properties of global variables is presented in Fig.9.14. Note that variable **x** is used in all functions but none except **fun2,** has a definition for **x.** Because **x** has been declared 'above' all the functions, it is available to each function without having to pass x as a function argument. Further, since the value of **x** is directly available, we need not use **return(x)** statements in **fun1** and **fun3.** However, since **fun2** has a definition of **x,** it returns its local value of **x** and therefore uses a **return** statement. In **fun2**, the global **x** is not visible. The local **x** hides its visibility here.

ILLUSTRATION OF PROPERTIES OF GLOBAL VARIABLES

Program

```
   int fun1(void);
   int fun2(void);
   int fun3(void);
   int x ;            /* global */
   main( )
```

```
{
     x = 10 ;       /* global x */
     printf("x = %d\n", x);
     printf("x = %d\n", fun1());
     printf("x = %d\n", fun2());
     printf("x = %d\n", fun3());
}
fun1(void)
{
     x = x + 10 ;
}
int fun2(void)
{
     int x ;        /* local */
     x = 1 ;
     return (x);
}
fun3(void)
{
     x = x + 10 ;  /* global x */
}
```

Output

```
x = 10
x = 20
x = 1
x = 30
```

Fig.9.14 *Illustration of global variables*

Example 9.9

Write a program to illustrate the properties of a static variable.

The program in Fig.9.15 explains the behaviour of a static variable.

```
                    ILLUSTRATION OF STATIC VARIABLE
```

Program

```
        void stat(void);
        main ( )
        {
           int i;
           for(i=1; i<=3; i++)
            stat( );
         }
        void stat(void)
        {
```

```
     static int x = 0;

     x = x+1;
     printf("x = %d\n", x);
   }
```

Output

```
   x = 1
   x = 2
   x = 3
```

Fig.9.15 *Illustration of static variable*