USER-DEFINED FUNCTIONS

In C language, functions are declared to compute and return the value of a specific data type to the calling program.

Functions can also be written to perform a task. It may return many values indirectly to the calling program and these are referred to as **`void`** functions.

FUNCTION DECLARATION

The general form of a function declaration is given below.

```
type name(type argu1, type argu2,..., type argu n)
{
    <local declaration>
    -   -   -   -
    <statement block>
    -   -   -   -
    return(variable or expression);
}
```

where `type` is the data type of the value return by the function and arguments expected.

`argu1, argu2,...,argu n` are the arguments which are variables that will receive the values from the calling program.

`name` is the name of the function by which the function is called by the calling program.

CALLING A FUNCTION

A function is called by the calling program using the function name with the required number of arguments in parentheses. The function call appears in an assignment statement or in an output statement. Let us consider the following examples to call a function.

(i) `ncr = fact(n)/(fact(n-r) * fact(r));`
(ii) `printf("\n %d factorial is %d", m, fact(m));`

Note that the function `fact()` is called three times to find the value of `ncr`.

- A function is called using its name with required number of arguments in parentheses.

ACTUAL AND FORMAL ARGUMENTS

Passing of values between the main program and the function takes place through arguments.

The arguments listed in the function calling statement are referred to as *actual arguments*. They are the actual values passed to a function to compute a value or to perform a task. Consider a function call statement with actual arguments as given below.

```
ncr = fact(n)/(fact(n-r)*fact(r));
```

Actual arguments

The arguments used in the function declaration are referred as *formal arguments*. The arguments are formal variables that accept or receive the values supplied by the calling program. Consider a function declaration with formal arguments as given below.

```
int fact(int k)
{
    int i, p = 1;
    for(i = 1; i <= k; i++)
        p = p * i;
    return(p);
}
```

Formal argument

Note that the number of actual and formal arguments and their data type should match.

Rules to Call a Function

The following rules are used to call a function in a C program.

(i) A function has a statement block which is called by the **main()** or any other function.

(ii) When the data type in a function declaration is omitted, the function will return a value of type integer.

(iii) The data type of the formal argument may be declared in the next line which follows the function declaration statement. Consider the following example

```
big(a,b)
int a,b;
{
    if (a > b)
        return (a);
    else
        return (b);
}
```

Note that the data type of the function is not mentioned. So it returns an integer type result to the calling program.

(iv) A function can return a value at any stage of its execution by using more than one return statements.

(v) A function can also be called by another function.

(vi) The **return** statement is omitted if it does not return a value directly to the calling program.

(vii) Arguments (formal and actual) listed can be omitted, but not the parentheses () following the function name. For example,

```
add();
```

es do not accep

Example 1

Write a function to find the factorial of a given integer and use it to find nCr.

$$nCr = \frac{n!}{r!(n-r)!}$$

Solution

A function is written to find the factorial of an integer and is called three times to find `n!`, `r!` and `(n-r)!`. The values of `n` and `r` are read in the main program. Note that the function prototype declaration is also written in the main program.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
main()
{ int fact(int k);
```

```
        /* function prototype declaration */
        int n,r,ncr;
        clrscr();
        printf("\n Enter value to n and r : ");
        scanf("%d %d",&n,&r);
        ncr = fact(n)/(fact(r)*fact(n-r));
        printf("\n Value of nCr = %d",ncr);
        getch();
    }
    /* function subprogram to find factorial */
    int fact(int k)
    {   int i,p = 1;
        for(i = 1; i <= k; i++)
            p = p * i;
        return(p);
    }
```

Example 4

Write a function in C to find the GCD of two integers. Using the function, write a program to find the GCD of 3 integers.

Solution

Greatest Common Divisor (GCD) for any two given number is obtained with a function. The main program will read the three integers and call the function to find the GCD of various combinations of two integers out of the three given integers.

```
#include <stdio.h>
#include <conio.h>
main()
{ int a,b,c,d1,d2,d3;
  int gcd(int,int);
  clrscr();
  printf("\n Enter three integers : ");
  scanf("%d %d %d",&a,&b,&c);
  d1 = gcd(a,b);
  d2 = gcd(a,c);
  d3 = gcd(b,c);
  if(d1 == d2)
   if(d1 == d3)
    printf("\n Greatest common divisor is %d",d1);
   else
    printf("\n Greatest common divisor is %d",gcd(d1,d3));
  else
```

```
    printf("\n Greatest common divisor is %d",gcd(d1,d2));
    printf("\n\n Press any key to continue ...");
    getch();
  }
int gcd(int x,int y)
{ int nr,dr,r;
  if(x >= y)
   { nr = x;
     dr = y;
   }
  else
   { nr = y;
     dr = x;
   }
  r = nr%dr;
  while( r != 0)
   {  nr = dr;
      dr = r;
      r = nr % dr;
   }
  return(dr);
}
```

Example 5

Write a function to reverse the given integer (e.g. 1234 should be printed as 4321). Also write the main program.

Solution

The main program is written to read an integer and the function is called to print the reverse of it.

```
#include <stdio.h>
#include <conio.h>
main()
{ int n;
  int reverse(int n);
  clrscr();
  printf("\n Enter the integer : ");
  scanf("%d",&n);
/* call the function to print the reverse integer */
  printf("\n %d is reversed as %d ",n,reverse(n));
  printf("\n\n Press any key to continue...");
  getch();
}
/* function to reverse an integer */
reverse(int n)
{ int rn,r;
  rn = 0;
  while( n > 0)
   { r = n % 10;
     rn = rn*10 + r;
     n = n / 10;
   }
  return(rn);
}
```

When this program is executed the

Example 7

Write a C program to find the arithmetic mean of n values using a function.

Solution

The list of n values can be read in the `main()` program. The function is called with the array as an argument to find the arithmetic mean.

```
#include <stdio.h>
#include <conio.h>
main()
{ float x[10];
  int n,i;
  float amean(float x[],int n);
  clrscr();
  printf("\n How many values? ");
  scanf("%d",&n);
 /* loop to read all values */
  printf("\n Enter all values \n");
  for(i = 0; i < n; i++)
   scanf("%f",&x[i]);
 /* call function to to print arithmetic mean */
  printf("\n Arithmetic mean = %6.2f ",amean(x,n));
  getch();
 }
 /* function to find arithmetic mean */
float amean(float x[],int n)
{ int i;
  float s = 0;
  for(i = 0; i < n; i++)
    s += x[i];
  return(s/n);
 }
```

When this program is executed, the user has to enter the value of n and all its values. The function finds the sum of all values and returns the arithmetic mean to the main program and is printed as shown below.

```
How many values? 6
Enter all values
3.1  3.8  3.6  4.0  3.4  3.8
Arithmetic mean = 3.62
```

Example 8

Write a program in C to read a matrix of order m × n and print the sum of all elements using functions.

Solution

The matrix of order m × n and its values are read in the **main()** program. The function is called with the matrix as an argument along with m and n to find the sum of all elements.

```
#include <stdio.h>
#include <conio.h>
main()
{
  int a[10][10],m,n,i,j;
  int elem_sum(int a[][10],int m,int n);
  clrscr();
  printf("\n How many rows and columns : ");
  scanf("%d %d",&m,&n);
 /* loop to read all values */
  printf("\n Enter the matrix values \n");
  for(i = 0; i < m; i++)
   for(j = 0; j < n; j++)
    scanf("%d",&a[i][j]);
 /* call function  to print the result */
  printf("\n Sum of all elements in the matrix = %d",elem_sum(a,m,n));
  getch();
}
 /* function to add all elements in the matrix */
int elem_sum(int a[][10],int m,int n)
{ int i,j,s = 0;
  for(i = 0; i < m; i++)
   for(j = 0; j < n; j++)
     s = s + a[i][j];
  return(s);
}
```

When this program is executed, the user has to enter the order of the matrix (i.e. m and n) and their values. The function finds the sum of all values and returns it to the main program. The result will be printed as shown below.