CHAPTER EIGHT

Functions

[A] What will be the output of the following programs:

(a)
```
# include <stdio.h>
void display( ) ;
int main( )
{
    printf ( "Learn C\n" ) ;
    display( ) ;
    return 0 ;
}
void display( )
{
    printf ( "Followed by C++, C# and Java!\n" ) ;
    main( ) ;
}
```

Output:

Both the messages will get printed indefinitely

(b)
```
# include <stdio.h>
int check ( int ) ;
int main( )
{
    int i = 45, c ;
    c = check ( i ) ;
```

```
    printf ( "%d\n", c ) ;
    return 0 ;
}
int check ( int  ch )
{
    if ( ch >= 45 )
        return ( 100 ) ;
    else
        return ( 10 * 10 ) ;
}
```

Output:

```
100
```

(c)
```
# include <stdio.h>
float circle ( int ) ;
int main( )
{
    float  area ;
    int  radius = 1 ;
    area = circle ( radius ) ;
    printf ( "%f\n", area ) ;
    return 0 ;
}
float circle ( int r )
{
    float  a ;
    a = 3.14 * r * r ;
    return ( a ) ;
}
```

Output:

```
3.000000
```

(d)
```
# include <stdio.h>
int main( )
{
```

```
    void slogan( ) ;
    int c = 5 ;
    c = slogan( ) ;
    printf ( "%d\n", c ) ;
    return 0 ;
}
void slogan( )
{
    printf ( "Only He men use C!\n" ) ;
}
```

Output:

Error message by compiler

[B] Point out the errors, if any, in the following programs:

(a)
```
# include <stdio.h>
int addmult ( int, int )
int main( )
{
    int i = 3, j = 4, k, l ;
    k = addmult ( i, j ) ;
    l = addmult ( i, j ) ;
    printf ( "%d %d\n", k, l ) ;
    return 0 ;
}
int addmult ( int ii, int jj )
{
    int kk, ll ;
    kk = ii + jj ;
    ll = ii * jj ;
    return ( kk, ll ) ;
}
```

Error. Missing ; in prototype declaration of **addmult( )**. Also, a function cannot return 2 values at a time.

(b)
```
# include <stdio.h>
void message( ) ;
int main( )
{
    int  a ;
    a = message( ) ;
    return 0 ;
}
void message( )
{
    printf ( "Viruses are written in C\n" ) ;
    return ;
}
```

No Error. But since no value is being returned there is no need to collect it in variable **a**.

(c)
```
# include <stdio.h>
int main( )
{
    float  a = 15.5 ;
    char  ch = 'C' ;
    printit ( a, ch ) ;
    return 0 ;
}
printit ( a, ch )
{
    printf ( "%f %c\n", a, ch ) ;
}
```

Error. **a** and **ch** should be declared as **float** and char respectively in the function **printit( )**.

(d)
```
# include <stdio.h>
void message( ) ;
int main( )
{
    message( ) ;
```

```
    message( ) ;
    return 0 ;
}
void message( ) ;
{
    printf ( "Praise worthy and C worthy are synonyms\n" ) ;
}
```

Error. Semicolon shouldn't be present immediately after **message( )** in the function definition.

(e)
```
# include <stdio.h>
int main( )
{
    let_us_c( )
    {
        printf ( "C is a Cimple minded language !\n" ) ;
        printf ( "Others are of course no match !\n" ) ;
    }
    return 0 ;
}
```

Error. One function cannot be defined within another function.

(f)
```
# include <stdio.h>
void message( ) ;
int main( )
{
    message ( message( ) ) ;
    return 0 ;
}
void message( )
{
    printf ( "It's a small world after all...\n" ) ;
}
```

Error. **void** returned by inner call to **message( )** cannot be passed to the outer call.

[C] Answer the following:

(a) Is this a correctly written function:

```
int sqr ( int a ) ;
{
   return ( a * a ) ;
}
```

Answer:

No. Semicolon shouldn't be present immediately after **sqr( )**.

(b) State whether the following statements are True or False:

(1) The variables commonly used in C functions are available to all the functions in a program.

Answer: False

(2) To return the control back to the calling function we must use the keyword **return**.

Answer: False

(3) The same variable names can be used in different functions without any conflict.

Answer: True

(4) Every called function must contain a **return** statement.

Anwer: False

(5) A function may contain more than one **return** statements.

Answer: True

(6) Each **return** statement in a function may return a different value.

Answer: True

(7) A function can still be useful even if you don't pass any arguments to it and the function doesn't return any value back.

Answer: True

(8) Same names can be used for different functions without any conflict.

Answer: False

(9) A function may be called more than once from any other function.

Answer: True

(10) It is necessary for a function to return some value.

Answer: False

[D] Answer the following:

(a) Write a function to calculate the factorial value of any integer entered through the keyboard.

Program:

```
/* Calculate factorial value of an integer using a function */
# include <stdio.h>

long fact ( int ) ;
int main( )
{
    int num ;
    long factorial ;
```

```
    printf ( "\nEnter a number: " ) ;
    scanf ( "%d", &num ) ;

    factorial = fact ( num ) ;
    printf ( "Factorial of %d = %ld\n", num, factorial ) ;

    return 0 ;
}

long fact ( int num )
{
    int i ;
    long factorial = 1 ;

    for ( i = 1 ; i <= num ; i++ )
        factorial = factorial * i ;
    return ( factorial ) ;
}
```

(b) Write a function **power (a, b)**, to calculate the value of **a** raised to **b**.

Program:

```
/* Program to calculate power of a value */
# include <stdio.h>

long power ( int, int ) ;
int main( )
{
    int x, y ;
    long pow ;

    printf ( "\nEnter two numbers: " ) ;
    scanf ( "%d %d", &x, &y ) ;
```

```
    pow = power ( x , y ) ;  /* Function call */
    printf ( "%d to the power %d = %d\n", x, y, pow ) ;

    return 0 ;
}

long power ( int x, int y )
{
    int i ;
    long p = 1 ;

    for ( i = 1 ; i <= y ; i++ )
        p = p * x ;
    return ( p ) ;
}
```

(c) Write a general-purpose function to convert any given year into its roman equivalent. Use these roman equivalents for decimal numbers: 1 – I, 5 – V, 10 – X, 50 – L, 100 – C, 500 – D, 1000 – M.

Example:
Roman equivalent of 1988 is mdcccclxxxviii
Roman equivalent of 1525 is mdxxv

Program:

```
/* Convert given year into its roman equivalent */
# include <stdio.h>

int romanise ( int, int, char ) ;
int main( )
{
    int yr ;

    printf ( "\nEnter year: " ) ;
    scanf ( "%d", &yr ) ;
```

```
    yr = romanise ( yr, 1000, 'm' ) ;  /* Series of function calls */
    yr = romanise ( yr, 500, 'd' ) ;
    yr = romanise ( yr, 100, 'c' ) ;
    yr = romanise ( yr, 50, 'l' ) ;
    yr = romanise ( yr, 10, 'x' ) ;
    yr = romanise ( yr, 5, 'v' ) ;
    yr = romanise ( yr, 1, 'i' ) ;

    return 0 ;
}

int romanise ( int y, int k, char ch )
{
    int i, j ;

    if ( y == 9 )
    {
        printf ( "ix" ) ;
        return ( y % 9 ) ;
    }

    if ( y == 4 )
    {
        printf ( "iv" ) ;
        return ( y % 4 ) ;
    }

    j = y / k ;

    for ( i = 1 ; i <= j ; i++ )
        printf ( "%c", ch ) ;

    return ( y - k * j ) ;
}
```

(d) Any year is entered through the keyboard. Write a function to determine whether the year is a leap year or not.

Program:

```
/* Using a function, determine whether a year is leap year or not */
# include <stdio.h>

void leapyear ( int ) ;
int main( )
{
    int  year ;

    printf ( "\nEnter year: " ) ;
    scanf ( "%d", &year ) ;

    leapyear ( year ) ;  /* Function call */

    return 0 ;
}

void leapyear ( int year )
{
    if ( year % 4 == 0 && year % 100 != 0 || year % 400 == 0 )
        printf ( "%d is leap year\n", year ) ;
    else
        printf ( "%d is not a leap year\n", year ) ;
}
```

(e) A positive integer is entered through the keyboard. Write a function to obtain the prime factors of this number.

For example, prime factors of 24 are 2, 2, 2 and 3, whereas prime factors of 35 are 5 and 7.

Program:

```
/* Obtain prime factors of a number */
# include <stdio.h>

void prime ( int ) ;
```

```
int main( )
{
    int  num ;

    printf ( "Enter number:" ) ;
    scanf ( "%d", &num ) ;

    prime ( num ) ;  /* Function call */

    return 0 ;
}

void prime ( int num )
{
    int  i = 2 ;
    printf ( "Prime factors of %d are ", num ) ;

    while ( num != 1 )
    {
        if ( num % i == 0 )
            printf ( "%d ", i ) ;
        else
        {
            i++ ;
            continue ;
        }
        num = num / i ;
    }
}
```