-Functions and Recursions-

Programming Fundamentals

Introduction

- Reading, processing, and writing of data are the three essential functions of a computer program.

- Most programs take some data as input and display the processed data, often known as information or results, on a suitable medium.

- When we say **Input**, it means to feed some data into a program. An input can be given in the form of a file or from the command line.

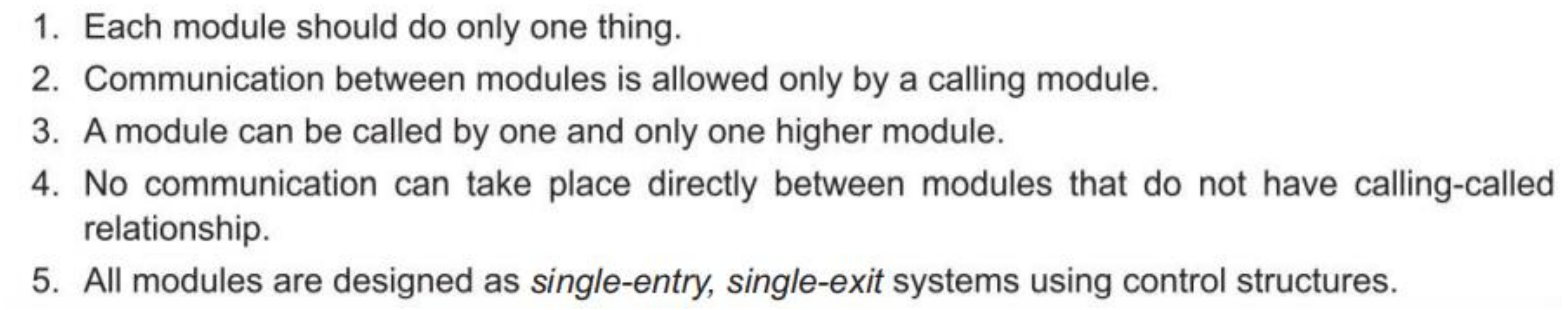
- When we say **Output**, it means to display some data on screen, printer, or in any file.

- C programming provides a set of built-in functions to read the given input and output the data on the computer screen as well as to save it in text or binary files.

Modular Programming

- It is the strategy applied to the design and development of software systems.
- A large program can be well-organized using smaller program segments- modules/program-units.
- The modules are integrated carefully to form a software-system satisfying the system requirements.

Characteristics of Modular Programming:



1. Each module should do only one thing.
2. Communication between modules is allowed only by a calling module.
3. A module can be called by one and only one higher module.
4. No communication can take place directly between modules that do not have calling-called relationship.
5. All modules are designed as *single-entry, single-exit* systems using control structures.

ELEMENTS OF USER-DEFINED FUNCTIONS

- Functions are defined as one of the derived datatypes in C.
- There are 3 elements of user-defined functions:

1. Function Declaration
2. Function Definition
3. Function Call

DEFINITION OF FUNCTIONS

- Function definition/implementation includes :

a. Function name
b. Function type
c. List of parameters

Function Header

d. Local variable declarations
e. Function statements

Function Body

f. Return statements

- General format of function definition:

```
function_type  function_name(parameter list)
   {
      local variable declaration;
      executable statement1;
      executable statement2;
      . . . . .
      . . . . .
      return statement;
   }
```

- Formal Parameter list: It declares the variables that receives the data sent by calling function.
- They serve as input data to a the given function. They represent actual arguments received from function-call.

- When a function reaches its return statement, control is transferred back to the calling function.
- In absence of return statement, closing brace acts as a void return.
- Local variable- a variable declared inside a function and is limited to use within that function.

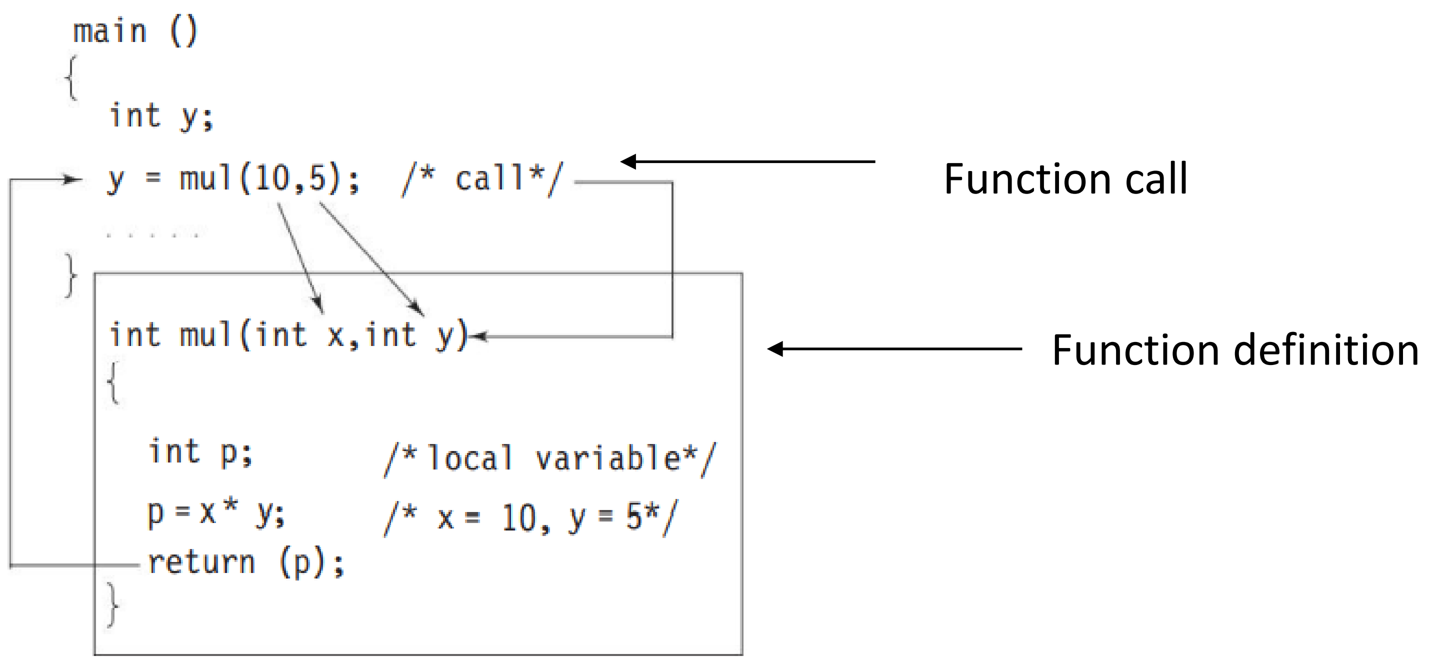
RETURN VALUES AND THEIR TYPES:

```
return;
or
return(expression);
```

- Former return statement will simply return the control whereas former will return a value at the time of returning the control back to the calling function

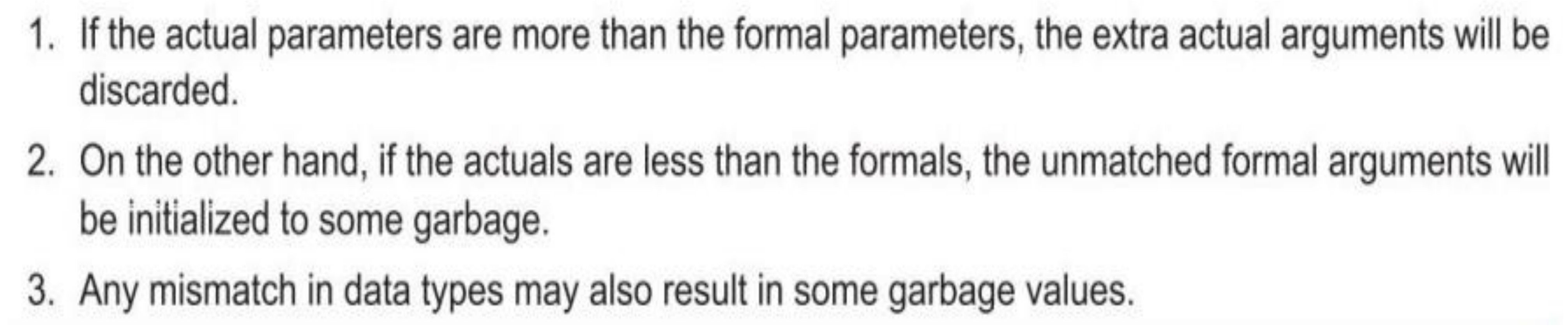
FUNCTION CALLS

- A function is called by writing the function name followed by list of actual arguments, if any, in parentheses.
- E.g.



```
main ()
{
  int y;
  y = mul(10,5);  /* call*/
  . . . . .
}
  int mul(int x,int y)
  {
    int p;        /*local variable*/
    p = x* y;     /* x = 10, y = 5*/
    return (p);
  }
```

- A function call is a postfix expression (operands then operators).
- The operator () has very high precedence. If a function call is used in an expression, it is evaluated first.
- E.g. *int a*= ***func(2,3)*** *+5*; //expression containing function call
- Another points to remember:



1. If the actual parameters are more than the formal parameters, the extra actual arguments will be discarded.
2. On the other hand, if the actuals are less than the formals, the unmatched formal arguments will be initialized to some garbage.
3. Any mismatch in data types may also result in some garbage values.

FUNCTION DECLARATION

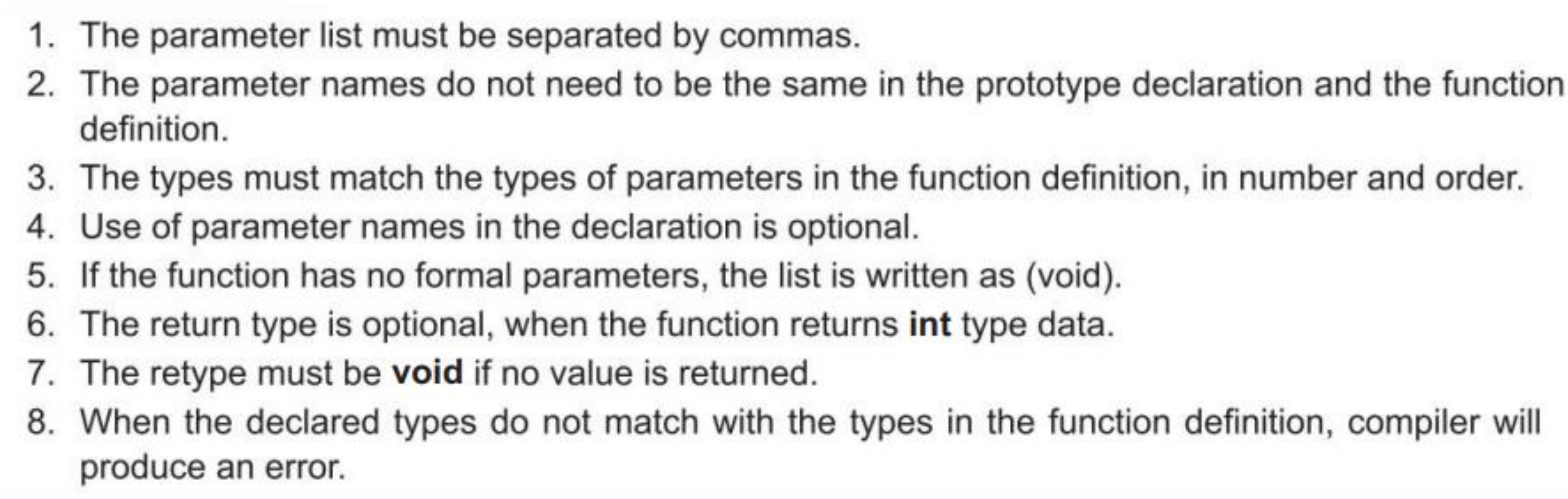
- Before invoking (calling) a function, any function, just like variables must be declared.
- Function declaration/prototype has four main parts:

1. Function/Return Type
2. Function Name/Identifier
3. List of arguments/Parameter lists
4. Terminating Semicolon

- Syntax:

```
Function-type function-name (parameter list);
```

- Points to remember:



1. The parameter list must be separated by commas.
2. The parameter names do not need to be the same in the prototype declaration and the function definition.
3. The types must match the types of parameters in the function definition, in number and order.
4. Use of parameter names in the declaration is optional.
5. If the function has no formal parameters, the list is written as (void).
6. The return type is optional, when the function returns **int** type data.
7. The retype must be **void** if no value is returned.
8. When the declared types do not match with the types in the function definition, compiler will produce an error.

- Prototype declaration can be placed at two places:

1. Above all the functions

- global declaration
- can be used by any other function

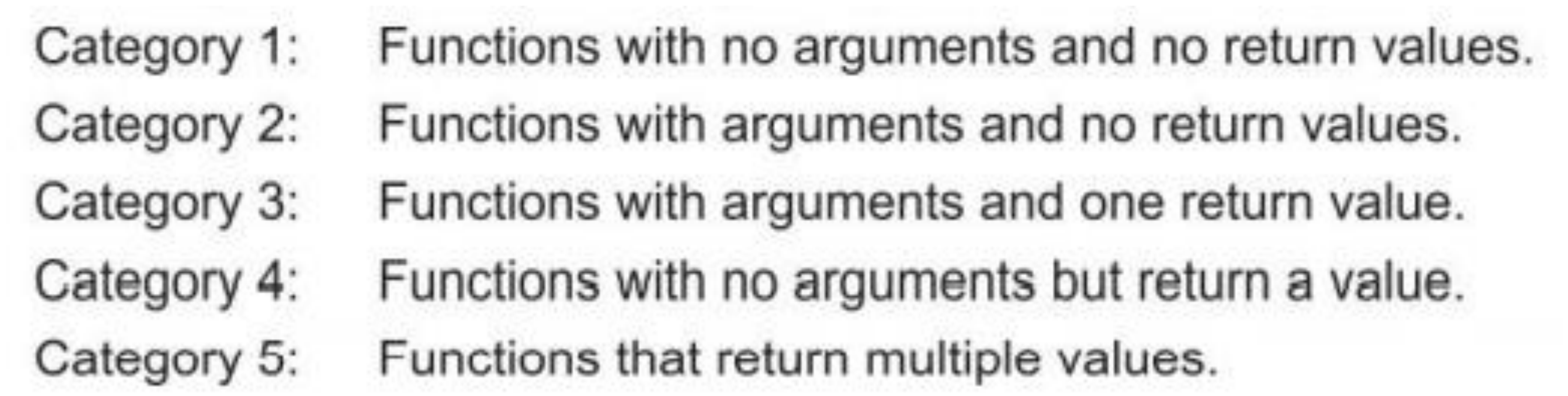
2. Inside a function definition

- Local to the function in which it is declared
- Only the function declaring the function can use it

Function Declaration- compulsory?

- NO
- If no function declaration is provided, linker will assume that the return type is int and the types of parameter match the formal arguments (present inside function definition).
- If it is not the case, the linker will fail and it is required to change the program.

CATEGORY OF FUNCTIONS



Category 1:	Functions with no arguments and no return values.
Category 2:	Functions with arguments and no return values.
Category 3:	Functions with arguments and one return value.
Category 4:	Functions with no arguments but return a value.
Category 5:	Functions that return multiple values.

FUNCTIONS THAT RETURN MULTIPLE VALUES

- ‘return’ can return only single value.
- Arguments used to send out information are called output parameters.
- It can be achieved by address operator (&) and indirection operator (*).
- E.g.

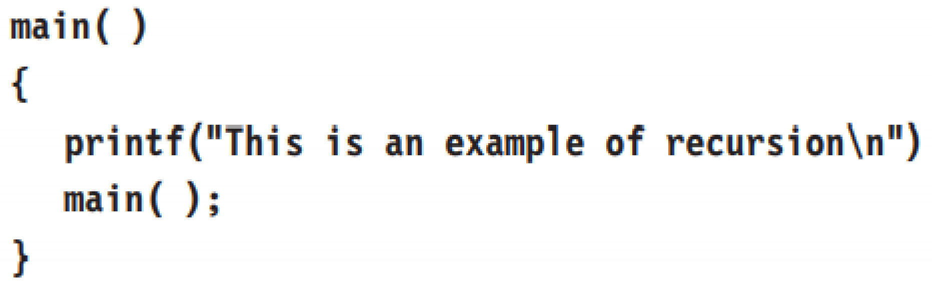
```
void mathoperation (int x, int y, int *s, int *d);
main( )
{
     int x = 20, y = 10, s, d;
     mathoperation(x,y, &s, &d);

     printf("s=%d\n d=%d\n", s,d);
}
void mathoperation (int a, int b, int *sum, int *diff)
{
     *sum  = a+b;
     *diff = a-b;
}
```

NESTING OF FUNCTIONS

- C permits nesting of functions.
- It means main() can call fun1(), fun1() can call fun2(), fun2() can call fun3() and so on.

RECURSION

- When a called function in turns call another function, a process of chaining (nesting) occurs.
- Recursion- special case of this process where a function calls itself.
- E.g.



```
main( )
{
   printf("This is an example of recursion\n")
   main( );
}
```