Chapter 6 : Functions

6.1 : Introduction

- self containing subprogram that performs some task
- Advantages : ① divide & conquer c program to reduce difficulty
  ② use repeatative code more than once (reusability)
  ③ easy to debug

- C Functions → User defined functions
  → library functions

* Library function: provides built in function through include files such as math.h stdlib.h strings.h stdio.h

* User defined functions : functions created by user to perform a certain task

Need. ① function definition
② function declaration
③ function call

```
void function (parameters);   → function declaration
int main()
{
    function call;            → function call
}
void function (param)
{
    Body                      → function definition
}
```

6.4 : Function definition

- description plus main code of a function

```
return_type function_name (parameters)   → Function Header
{
    local variable;
    statement                            → Function Body
    return parameter
}
```

- there can be no return type or zero parameters in that case provide void
  else by default int will be considered

imp - No nesting (function definition inside function definition) is allowed

Note: Formal Arguments Passed to function definition
Actual Argument : Passed to function call

6.5 : Function call :

- used where program actually needs logic to be implemented!
- syntax : func_name (arg 1, arg 2, ... , arg N)
  actual arguments
- when function is called following things happens :
  (a) compiler allocates storage space for param's in function definition
  (b) values are assigned to formal parameters
  (c) if mismatch in type of formal & actual arguments, some default conversions takes place
  (d) Perform code body
  (e) Function terminates when return statement

behind the scenes of what happens if function is called

- if function have return but in fuction call we are not assigning any value return value is just discarded.

return discarded if not used

```
int calculate ( x, y )    declaration
calculate (3, 2)   ✓   No error return discarded
```

- function call on RHS is invalid

```
calculate (3,2) = 2   ✗   invalid
```

6.6 : Return statement

- to return from calling function and getting back to called function
- if no return, function execute till closing curly brace ({)
- return value can be constant, variable, expression like

```
return 1    return x++    return (x+y+z)    return (8 * sum(2,3))
```

when confliciting data type
- when confliciting data type, compiler tries to correct. F-examp. if return expected is double, but value returned is int, compiler will convert int to float.

No return to non void
- if return type is not void, but function doesnt return anything, value could be garbage or anything

multiple return
- we could have multiple return statement, but as soon as 1st return is encountered function returns

multiple value in return
- return 1, 2, 3   ✗   invalid - only one value can be returned

6.7 :- Function Parameters And Arguments :

call by value
- Function parameters are called by value
- changes to them doesnt affect actual arguments

call by reference
- could also be called by reference, where changes could affect actual parameters (Done using pointers)

Data type mismatch
- If there is datatype mismatch, compiler corrects with legal type conversion or garbage value is passed

6.8 : Order of evaluation of function arguments

- undefined in C
- compiler dependent
- example `int k, a=3`
  `k= multiply (a, a++)` } output undefined
  a++ could be first or a could be 1st

6.9 : Function Declaration

`return_type function_name(type1 par1, type2 par2, ... ,typeN parN)`

- Function declaration informs compiler about following
  (a) Name of function
  (b) type and no of parameters to be passed
  (c) type of value of return
- names of parameters are optional
  `int multiply (int, int);` ☑ correct, no need for names

6.10 : main() Function :

- User defined function but name, number and type predefined by compiler
- return value 1 → program completed successfully
  — 0 — → error
  No return → garbage value
- main()'s return ; ≅ exit()
- Definition of main(): By compiler
  declaration of main(): By programmer
  call of main(): By OS

6.15 : Local, Global And Static Variable

6.15.1 : Local variables

- variables declared inside function / block & local to function / block
- created when function is called, destroyed when function returns
- could be access only by that function / block
- initialised to garbage value

6.15.2 : Global variables

- defined outside any function
- could be accessed by all functions within code
- initialised to zero

Note:- whenever there is conflict between local & global variable, local variable gets precedence

6.15.3 : Static variables

- declared with variable keyword " static variable_name "
- every time variable retains previous value rather than reinitialising