# Ch. 5 - Functions & Recursion

Sometimes a program gets bigger in size & it's not possible for a programmer to track which piece of code is doing what.
Function is a way to break our code into chunks so that it is possible for a programmer to reuse them.

## What is a function?

A function is a block of code which performs a particular task.
A function can be reused by the programmer in a given program any number of times

## Example & Syntax of a Function

```
#include <stdio.h>
Void display(); ⇒ Function prototype
int main(){
  int a;
  display();  ⇒ Function Call
  return 0;
}
Void display(){  ⇒ Funct^n definition
  printf("Hi I am display");
}
```

## Function Prototype

Function prototype is a way to tell the compiler about the function we are going to define in the program.
Here void indicates that the function returns nothing

## Function call

Function call is a way to tell the compiler to execute the function body at the time the call is made.
Note that the program execution starts from the main function in the sequence the instructions are written.

## Function definition

This part contains the exact set of instructions which are executed during the function call. When a function is called from main(), the main function falls asleep & gets temporarily suspended. During this time the control goes to the function being called. When the function body is done executing main() resumes.

Code:

```
# include <stdio.h>
void display(); // Function prototype

int main(){
  int a;
  printf("Initializing display function\n");
```

```
display();  // function call
printf("Display function finished its work\n");

return 0;
}

// function defenition
void display(){
    printf("This is display\n");
}
```

Ans:
```
Initializing display function
This is display
Display function finished its work
```

Quiz:
Write a program with the 3 functions
1) Good morning function which prints "Good Morning".
2) Good afternoon function which prints "Good afternoon".
3) Good night function which prints "Good night".

main() should call all of these in order 1→2→3

Code:
```
#include <stdio.h>
void goodMorning();
void goodAfternoon();
void goodNight();
int main(){
```

```
int main(){
    return 0;
}
goodMorning();
goodAfternoon();
goodNight();
return 0;
}

void goodMorning(){
    printf("Good Morning Harry \n");
}

void goodAfternoon(){
    printf("Good Afternoon Harry \n");
}

void goodNight(){
    printf("Good Night Harry \n");
}
```

Ans

Good Morning Harry
" Afternoon "
" Night "

## Imp. Points

1) Execution of a C program starts from main().

2) A C program can have more than one function.

3) Every function gets called directly & indirectly from main().

4) There are 2 types of functions in C.

# Types of Functions

1) Library functions → Commonly required functions grouped together in a ~~for~~ library file on a disk

2) User defined functions → These are the functions declared & defined by the user

Why use functions?

1) To avoid rewriting the same logic again & again.

2) To keep track of what we are doing in a program.

3) To test & check logic independently

For e.g.

```
#include <stdio.h>
/* sum is a function which takes a & b as input & returns an integer as an output */
int sum(int a, int b); // function ~~call~~ definition
                                  prototype
```

```
int main (){
    int c;
    c= sum (2, 15); // function call
printf("The value of c is %d \n", c);

    return 0;
}
int sum (int a, int b){
    int result;          // function defintion
    result= a+b;
    return result;
}
```

Ans:
The value of c is 17

Passing values to functions
We can pass values to a function & can get a value in return from a function:

```
int sum (int a, int b)
```

The above prototype means that sum is a function which takes values a (of type int) & b (of type int) & returns a value of type int.

function defintion of sum can be:

```
int sum (int a, int b){
        int c;          a & b are parameters
        c= a+b;
        return c;
}
```

Now we can call sum(2,3); from main to get 5 in return → Here 2 & 3 are arguments

int d = sum(2,3); d becomes 5

Note:

1) Parameters are the values or variable placeholders in the function definition. E.g. a, b

2) Arguments are the actual values passed to the function to make a call. E.g. 2 & 3

3) A function can return only one value at a time

4) If the passed variable is changed inside the function, the function call doesn't change the value in the calling function.

```
int change (int a){
    a = 77;        → Mesnomer
    return 0;
}
```

change is a function which changes a to 77. Now if we call it from ~~the~~ main like this

```
int b = 22
change(b);   ⇒ The value of b remains 22
printf("b is %d", b);
        ⇒ prints "b is 22".
```

This happens because a copy of b is passed to the change function

Quiz: Use the library functions to calculate the area of a square with side a

```
#include <stdio.h>
#include <math.h>
int main()
{
    int side;
printf("Enter the value of side \n");
scanf("%d", &side);
printf("The value of area is %f", pow(side,2));
return 0;
}
```

Ans:

Enter the value of side

4

The value of area is 16

# Recursion

A function defined in C can call itself. This is called recursion.

A function calling itself is also called 'recursive' function.

E.g.

factorial(n) = 1 x 2 x 3 x ... (n-1) x n

= [factorial(n-1)] x n

Since we can write factorial of a number in terms of itself, we can program it using recursion

E.g.) int factorial (int x){
int f;
if (x==0 || x==1)
return 1;
else
f = x * factorial (x-1);
return f;
}

⇒ A program to calculate factorial using recursion

Code:
```
#include <stdio.h>
int factorial(int x);

int main(){
int a=3;
printf("The value of factorial: %d is %d", a, factorial(a));
return 0;
}

int factorial(int x){
if (x==1 || x==0){
return 1;
}
else{
return x* factorial (x-1);
}
}
```

// You can change value of a

Ans: The value of factorial of 3 is 6

How does it work?

factorial(3)

3 x factorial(2)

= 3 x 2 x factorial(1)

= 3 x 2 x 1

Imp. Notes:

1) Recursion is sometimes the most direct way to code an algorithm.
2) The condition which doesn't call the function any further in a recursive function is called as the base condition.
3) Sometimes, due to a mistake made by the programmer, a recursive function can keep running without returning resulting in a memory error.

| Constant | Meaning |
|---|---|
| '\a' | audible alert (bell) |
| '\b' | backspace |
| '\f' | form feed |
| '\n' | new line |
| '\r' | carriage return |
| '\t' | horizontal tab |
| '\v' | vertical tab |

| | |
|---|---|
| '\?' | question mark |
| '\\' | backslash |
| '\0' | null |

| Characters | ASCII values |
|---|---|
| 0-9 | 48-57 |
| A-Z | 65-90 |
| a-z | 97-122 |
| Special Symbols | 0-47, 58-64, 91-96, 123-127 |

## Bitwise Operators

Bitwise operators are used for manipulation of data at the bit (binary digit) level. Bitwise operators cannot be applied for float or double. The use of bitwise operators is for testing the bits, or shifting them to the right or left

| Operator | Meaning |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive |
| << | shift left |
| >> | shift right |
| ~ | complement |

The 'sizeof' operator is used to obtain the number of bytes that the operand occupies on memory. The operand can be a variable, constant or a data type qualifier

e.g. z = sizeof (average);

| Data Type | | Conversion Character |
|---|---|---|
| Integer | short signed | %d or %i |
| | short unsigned | %u |
| | long signed | %ld |
| | long unsigned | %lu |
| | unsigned Hexadecimal | %x |
| | unsigned octal | %o |
| Real | float | %f |
| | double | %lf |
| character | signed | %c |
| | unsigned | %c |
| string | | %s |

## Ch. 5 - Practice Set

Q) Write a program using functions to find average of three numbers.

```
#include <stdio.h>
float average(int a, int b, int c);
```

```
int main(){
  int a, b, c;
  printf("Enter the value of a\n");
  scanf("%d", &a);

  printf("Enter the value of b\n");
  scanf("%d", &b);

  printf("Enter the value of c\n");
  scanf("%d", &c);

  printf("The value of average is %f\n",
                    average(a, b, c));

  return 0;
}

float average(int a, int b, int c){
    float result;
    result = (float)(a+b+c)/3;

    return result;
}
```

Ans:

Enter the value of a

1

Enter the value of b

2

Enter the value of c

3

The value of average is 2

Q2 Write a function to convert celsius temperature to farenheit fahrenheit?

Q2 Write a function to calculate force of attaraction on a body of mass m exerted by earth (g = 9.8 m/s²)

```
#include<stdio.h>
float force(float mass);
int main(){
    float m;
    printf("Enter the value of mass in kg\n");
    scanf("%d", &m);

    printf("The value of force in Newton is %f\n", force(m));
    if we write %.nf to get n digits after decimal)
    return 0;
}
float force(float mass){
    float result = mass * 9.8;
    return result;
}
```

Ans:

Enter the value of mass in kg
10

The value of force in Newton is 98.0

Q3) Write a program using recursion to calculate $n^{th}$ element of fibonacci series.

hint: $fib(n) = fib(n-1) + f(n-2)$

Q.4) What will be the foll. line produced in a C program:

a=3 printf("%d %d %d \n", a, ++a, a++);

Code:
```
#include <stdio.h>

int main(){
    int a=3
    printf("%d %d %d, a, ++a, a++);
    return 0;
}
```

Ans
5 5 3

∵ gcc compiler runs arguments from right to left

Q5) Write a program using functions to print the following pattern (first n lines)

```
*
***
*****
```

Code:
```
#include <stdio.h>
void print_pattern(int n);

int main(){
    int n=3
    print_pattern(n);
    return 0;
```

// for n = 3

//  *  1 → 1

// * * *  2 → 3

// * * * * *  3 → 5

// 2n-1

```
void printPattern(int n){
    if(n==1){
        printf("*\n");
        return;
    }

printPattern(n-1);
for(int i=0; i<(2*n-1); i++){
    printf("*");
}
printf("\n");
}
```

Ans:

*

* * *

* * * * *