2. Define function and its list its advantages? Describe the difference between passing arguments by value and passing arguments by address with suitable program.

Function:

A function is defined as self-contained block of statements that performs a particular specified job in a program. They are the modules of code composed of a number of statements grouped to perform or accomplish a specific task. Function, usually, ‘takes in’ data, process it and ‘return’ a result. A function can be used over and over again without having to write all the codes in it and simply calling it by name. A function can be called from inside other functions as well.

Advantages of Function:

1. The use of function to perform specific job in a program makes easier to write small program blocks and keep track of what they do.
2. A single function can be used multiple times in a single program from different places; that is same code can be used again and again without having to rewrite.
3. The use of function avoids the redundant codes.
4. When series of function calls are used, the main program is comparatively smaller and logically clear to understand.
5. The workload can be divided among multiple programmers as one may write a function that does one thing and other may write a function that does another thing, and third programmer may write a function which uses the previous two functions. Then can do this all without breaking the whole program.

Part 2:

Passing arguments by value	Passing arguments by Address
When values of actual variables are passed to function as arguments, it is known as function call by value i.e., passing arguments by value.	When the address of the variable is passed as to a function as arguments, it is known as function call by address i.e., passing arguments by address.
Syntax: function_name(value_of_arg1, value_of_arg2, …)	Syntax: function_name(address_of_arg1, address_of_arg2, …)
Function gets the access to copy of the variables.	Function gets the access to original variable’s content.
Changes made to variables inside the function are not reflected in the original value.	Changes made to variables inside the function are reflected in the original value.

Example:
`#include <stdio.h>`
`void swap(int, int);`
`int main()`
`{`
`    int a, b;`
`    a = 55; b = 72;`
`    printf("Before swap:\ta = %d \t b = %d\n",a, b);`
`    swap(a, b);`
`    printf("After swap:\ta = %d \t b = %d\n",a,b);`
`    return 0;`
`}`
`void swap(int a, int b)`
`{`
`    int temp;`
`    temp = a;`
`    a = b;`
`    b = temp;`
`    printf("From function:\ta = %d \t b = %d\n",a,b);`
`}`
 	
`Output:`
`Before swap:     a = 55    b = 72`
`From function:   a = 72    b = 55`
`After swap:      a = 55    b = 72`
	Example:
	`#include <stdio.h>`
	`void swap(int *, int *);`
	`int main()`
	`{`
	`    int a, b;`
	`    a = 55; b = 72;`
	`    printf("Before swap:\ta = %d \t b = %d\n",a, b);`
	`    swap(&a, &b);`
	`    printf("After swap:\ta = %d \t b = %d\n",a,b);`
	`    return 0;`
	`}`
	`void swap(int *a, int *b)`
	`{`
	`    int temp;`
	`    temp = *a;`
	`    *a = *b;`
	`    *b = temp;`
	`    printf("From function:\ta = %d \t b = %d\n",*a,*b);`
	`}`
	
	`Output:`
	`Before swap:     a = 55    b = 72`
	`From function:   a = 72    b = 55`
	`After swap:      a = 72    b = 55`