Practical-8

To study about User defined functions

Function

- A function is a block of code that performs a specific task.
- Types:
  - Library Functions
    - Predefined like scanf(), printf(), gets(), puts()
    - declared inside Header files
    - Body part in .dll file
  - User defined functions
    - Created by user
    - To reduce complexity of program

Function

S.no	C function aspects	syntax
1	function definition	return_type function_name (arguments list) { Body of function; }
2	function call	function_name (arguments list);
3	function declaration	return_type function_name (argument list);

Function in Program

1. **Function Declaration/Prototype**
2. Void main()
3. {
4. ……..
5. ……..
6. **Function calling**
7. }
8. **Function Definition()**
9. {

10. }

1. **Function Definition**
2. {

3. }
4. Void main()
5. {

6. **Function calling**
7. }

Function Declaration

- The **function** prototypes are used to tell the compiler about **function's** name, parameters and return type.
- By this information, the compiler cross-checks the **function** signatures before calling it.
- Syntax:

```
return type function_name( parameter list );
```

- E.g

```
int Add(int x , int y);
```

Function Calling

- To use a function, you will have to call that function to perform the defined task.
- To call a function, you simply need to
    1. pass the required parameters along with the function name,
    2. and if the function returns a value, then you can store the returned value.

Function Calling

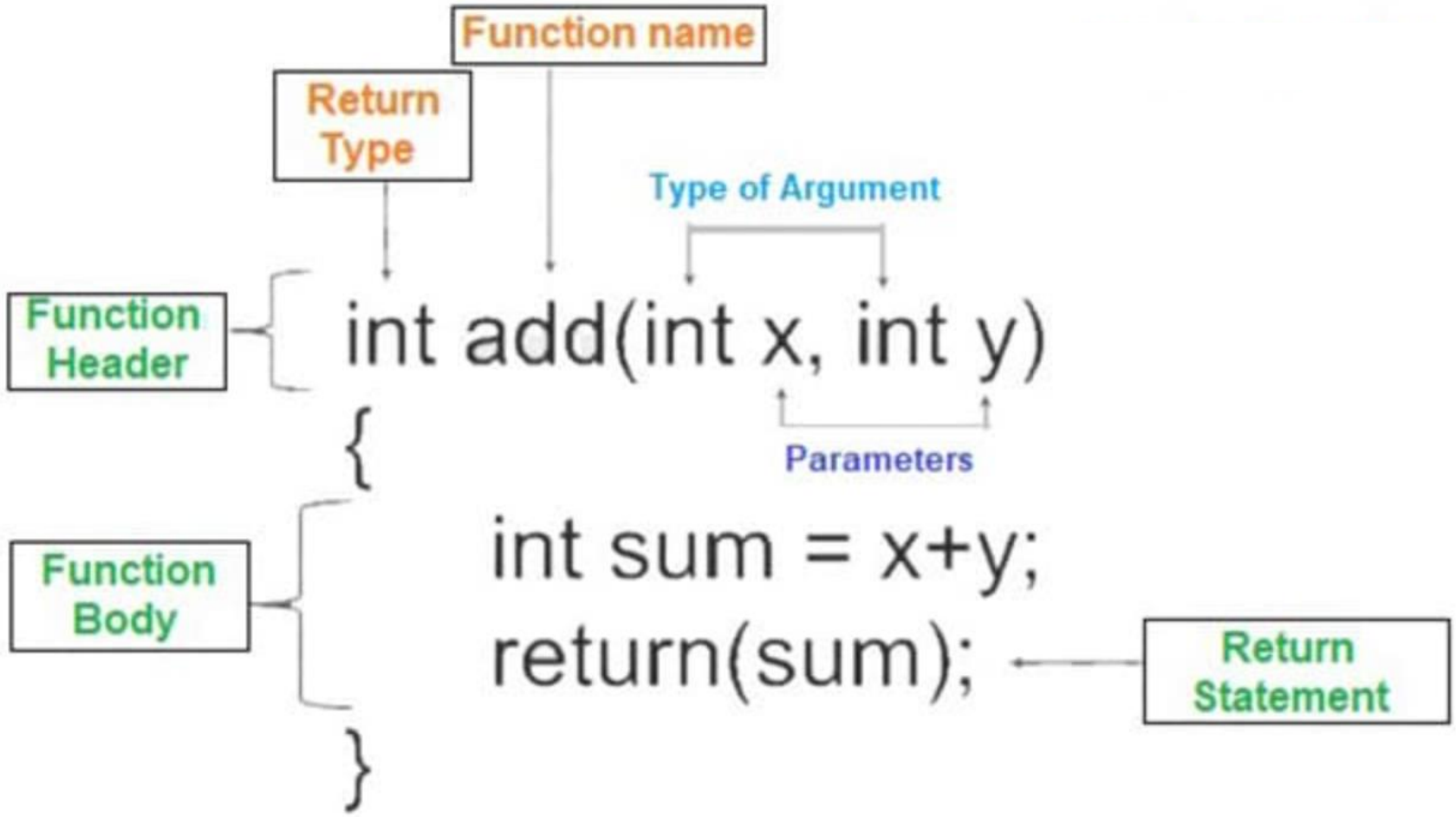
- Example:
- Int Add(int x ,int y); //Declaration
- void main()

```
{
        Int a=10,b=20,Ans;
        Ans=Add(a,b);
        Printf("%d",Ans);
}
```

Function Definition

- Comprises whole description and code of the function
- The structure tells what function is doing and what are its input and output.
- A function definition consists of a ***function header*** and a ***function body.***
- **Return Type** – A function may return a value.
- **Function Name** – This is the actual name of the function.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter.
- **Function Body** – The function body contains a collection of statements that define what the function does.

Function Definition



Example

```
int a = 10, b = 5, c;

int product(int x, int y);

int main(void)
{
    c = product(a,b);

    printf("%i\n",c);

    return 0;
}

int product(int x, int y)
{
    return (x * y);
}
```

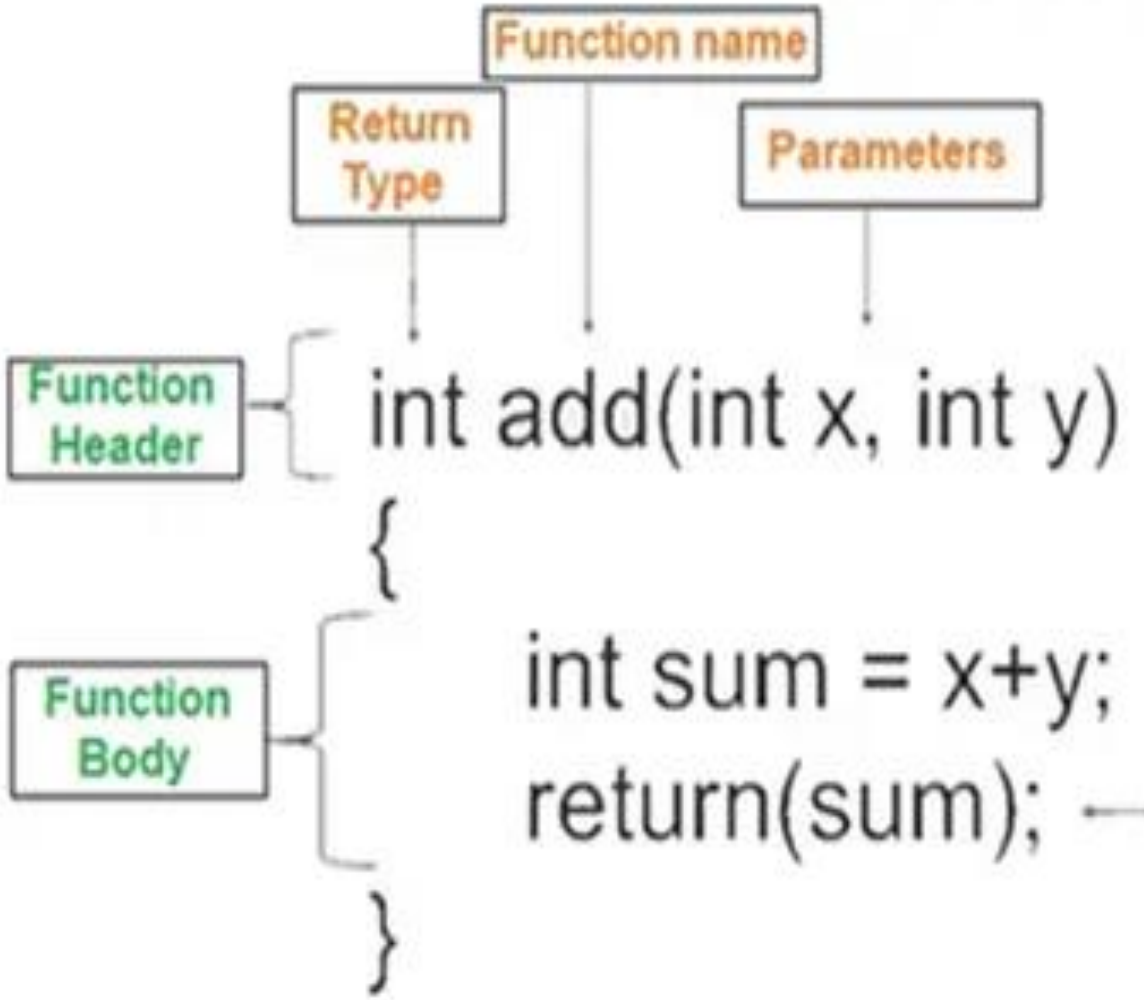
Function Prototype

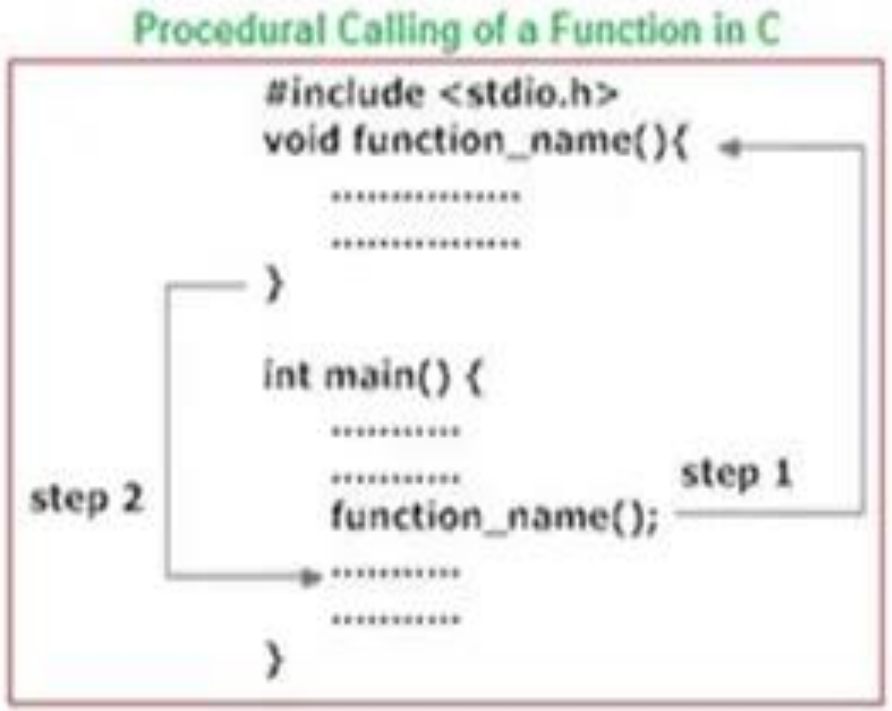
Main Function

Function call

Function Definition

Example


Function name
Return Type
Parameters
Function Header
int add(int x, int y)
{
Function Body
int sum = x+y;
return(sum);
}
Return Statement


Procedural Calling of a Function in C
#include <stdio.h>
void function_name(){
.....
.....
}
int main() {
.....
.....
function_name();
.....
.....
}
step 1
step 2

Function with return and without return

Functions

With return type

```
int main()
{
   int n = adder(25, 17);
   printf("adder's result is = %d", n);
}

int adder(int a, int b)
{
   int c = a + b;
   return c;
}
```

Without return type

Function does not return a Value

```
void area();
```

Nothing specified in Brackets - Function does not take any argument

Function without return

```
#include<stdio.h>
void myfun();

int main()
{
        printf("Hello I'm main function");

        myfun();

        printf("Hello I'm back to main function");
}

void myfun()
{
        printf("Hello I'm myfun function");
}
```

1

2

Function with argument and without argument

with argument

- declared and defined with parameter list
- values for parameter passed during function call

Eg:

```
int sum(int x, int y);  // declaration
sum(10, 20);            // call
```

without argument

- No parameter list
- No value passed during function call

Eg:

```
int show();   // declaration
show();       // call
```

Category of user defined function

- Function with argument
  - with Return Type
  - without Return Type
- Function without Argument
  - with Return Type
  - without Return Type

Practical-8.1

- Write a function line() to print a line (80 hyphens (_)).
- No Argument No Return

1. **Declaration : void line(void);**
2. **Calling– line();**
3. **Definition-**

```
Void line(void)
{
//print _ 80 times
}
```

Practical-8.2

Modify practical-8.1 rename function as lineprint() that will take

1. two argument- a character and an integer.
2. The character should be printed the no of times specified by the integer value.

Practical-8.2

- No Return- with Argument
- Definition :

```
void lineprint(char ch, int n)
        {
                //logic to print ch n times
        }
```

- Calling : Main()

```
{
    int x=5 ; char c;
    lineprint(c,5);
}
```

Practical-8.3

Write a function sortarray() to sort an array passed as an argument.

- Passing individual array elements

  to a function is similar to passing variable to function.

Practical-8.3

```
Int add(int a[],int len)  //x=&x[0]
  {
            int sum=0;
            for(i=0;i<len;i++) { sum+=a[i]; }
            return sum;
   }
 Int main()
            {
                        int x[ ]={1,2,3,4,5};  int Ans;
                  //Name of array is pointer point base address of array
                        Ans=add( x , 5 )   ; //passing only base address(x) of array not whole array
            }
```

Example

```
void display(int age1, int age2)
    {
     printf("%d  and %d", age1,age2);
     }
int  main()     {
    int ageArray[] = {2, 8, 4, 12};
    display(ageArray[1], ageArray[2]);
    return 0;
    }
```

Example

```
float calculateSum(float age[]);

int main() {

    float result;

    float  age[] = {2.4, 5, 2.6,3,4.5,8};

    Result = calculateSum(age);

    printf("Result = %.2f", result);

    return 0;

}
```

```
float calculateSum(float age[])

{

    float sum = 0.0;

    for (int i = 0; i < 6; i++)

        {        sum += age[i];

        }

        return sum;

}
```

Practical-8.3

- void sortarray(int a[])

```
{
        //logic to sort Array and print array
}
```

- Int main()

```
 {
        //declare array a[10] and variable

        //read array elements

        //call array    sortarray(a);

}
```

Practical-8.4

write a Program using function to Find out Cube of any given number N.

- **Function Input** Number N
- **Function Output** Cube of N
- **Function Type** : with Return and With Argument/without return and with argument

Practical-8.4

- Int findcube(int n);

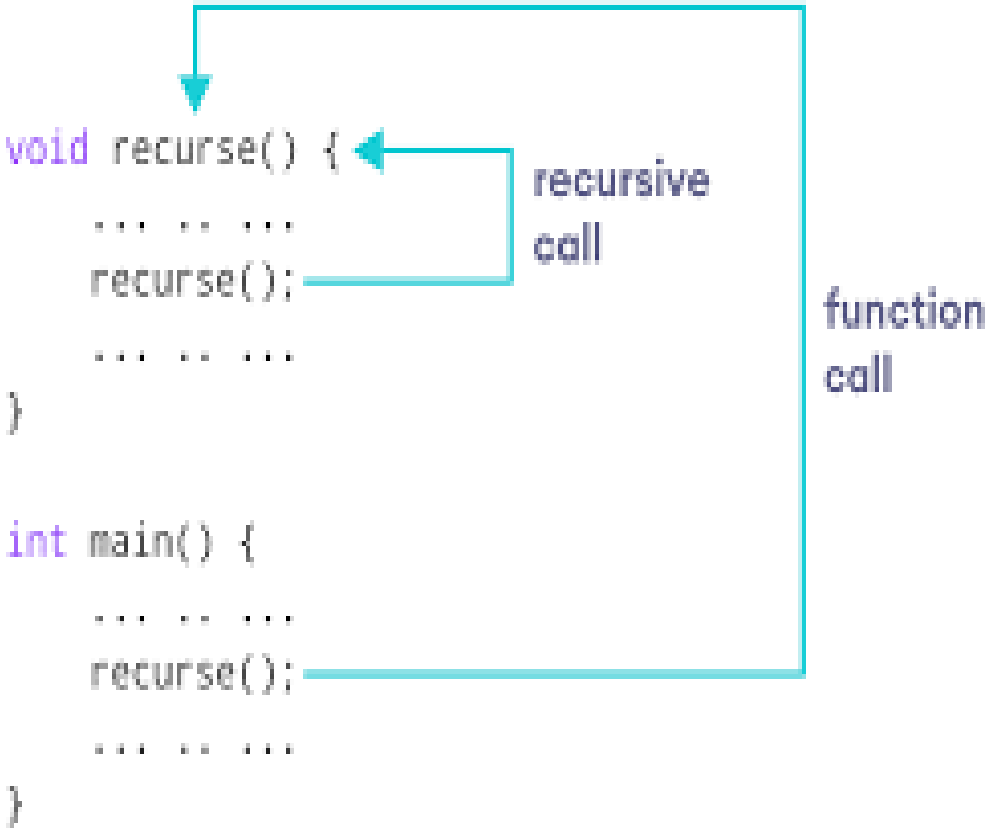
```
{
 //logic to find cube of n  and return
 }
int main()
{
 //read value  int a;
 // call function findcube(a) ;
 }
```

Practical-8.5

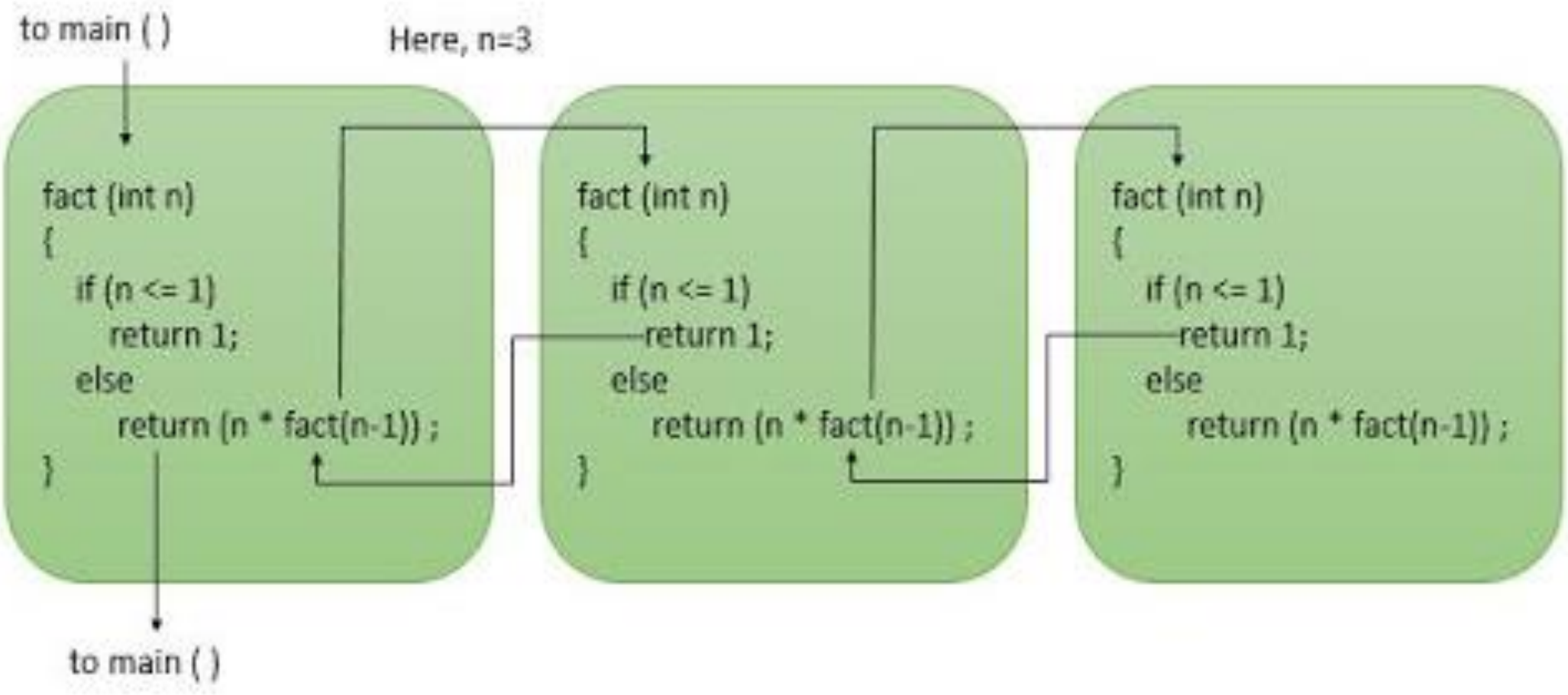
- Write a Program using function to Find out Factorial of given number.
    1. With the help recursion
    2. without recursion
- **Function Input** : Number N
- **Function Output :** Return Factorial of N
- **Function Type** : with Return and With Argument

Recursion

- **Recursion** is a process by which a function calls *itself* directly or indirectly.
- The corresponding function is called as **recursive function**.
- is used to solve complex problems by breaking them down into simpler ones



Recursion


to main ( )
Here, n=3
fact (int n)
{
if (n <= 1)
return 1;
else
return (n * fact(n-1)) ;
}
fact (int n)
{
if (n <= 1)
return 1;
else
return (n * fact(n-1)) ;
}
fact (int n)
{
if (n <= 1)
return 1;
else
return (n * fact(n-1)) ;
}
to main ( )

Practical-8.6

- Modify the program 5.9 of calculator by making user defined function for each operation.
- Hint :
- int add(a , b) return a+b;}
- Int main() {
- switch(op) {
  - Case ‘+’ : //call function
  - { printf(“%d”, Add(a,b));
  - break ; …
  - ……..
  - … }

Practical-8.7

- Write a function prime that return 1 if it‘s argument is prime and return 0 otherwise.
- **Function Input** : Number N
- **Function Output** : If N prime Return 1 else Return 0
- **Function Type** : with Return and With Argument
- Main() $\xrightarrow{\text{prime(No)}}$ int prime(int N){ //logic to find N is prime or not }

Return 1 or 0

Practical-8.8

Write a function swap() to swap values of two variables.

- **Function Input** : Number a,b
- **Function Output :** None
- **Function Type** : without Return and With Argument

- Main() $\xrightarrow{\text{swap(x,y)}}$ void Swap(int , int); { //logic to swap values and print values}

Practical-8.9

- Write a function to find maximum number out of three numbers.
- **Function Input** : Number a,b,c
- **Function Output :** Max Value
- **Function Type** : with Return and With Argument
- Main() $\xrightarrow{\text{max(x,y,z)}}$ int max(int , int,int); { //logic to find max and return}

Practical-8.10

- Write a function oddeven() to check that the number passed as a parameter is odd or even.
- **Function Input** : Number No
- **Function Output :** None
- **Function Type** : without Return and With Argument
- Main() $\xrightarrow{\text{oddeven(x)}}$ void oddeven(int n); { //logic to to check odd or even and print value }