Tabular Representation to illustrate the difference between library and user-define function:

User-defined Functions	Library Functions
These functions are not predefined in the Compiler.	These functions are predefined in the compiler of C language.
These function are created by user as per their own requirement.	These functions are not created by user as their own.
User-defined functions are not stored in library file.	Library Functions are stored in special library file.
There is no such kind of requirement to add the particular library.	In this if the user wants to use a particular library function then the user have to add the particular library of that function in header file of the program.
Execution of the program begins from the user-define function.	Execution of the program does not begin from the library function.
Example: sum(), fact(),...etc.	**Example:** printf(), scanf(), sqrt(),...etc.

Structure of a User Defined Function

A general form of a C function looks like this:

```
<return type> FunctionName (Argument1, Argument2, Argument3……)
{
Statement1;
Statement2;
Statement3;
}
```

Functions in C

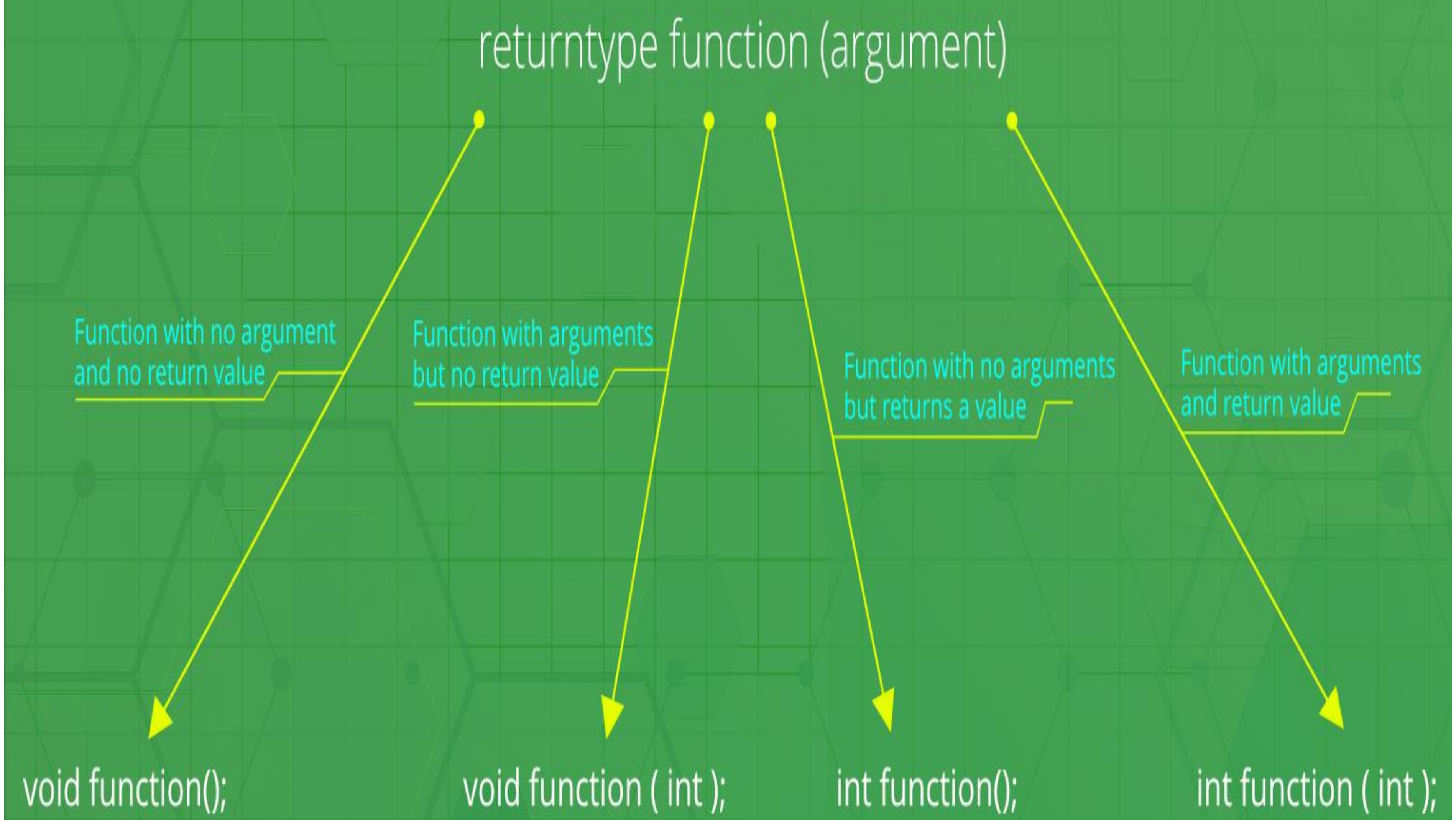
- A function in C can be called either **with arguments or without arguments**.
- These function **may or may not return values to the calling functions**.
- All C functions can be called either **with arguments or without arguments** in a C program.
- Also, **they may or may not return any values**.

Type of User-defined Functions in C

There can be 4 different types of user-defined functions, they are:

1. Function with **no arguments** and **no return value**
2. Function with **arguments** and **no return value**
3. Function with **no arguments** and **a return value**
4. Function with **arguments** and **a return value**

Types of Functions in C


returntype function (argument)
Function with no argument and no return value
Function with arguments but no return value
Function with no arguments but returns a value
Function with arguments and return value
void function();
void function (int);
int function();
int function (int);

1. Function with **no arguments** and **no return value**

Functions with no arguments and no return value.

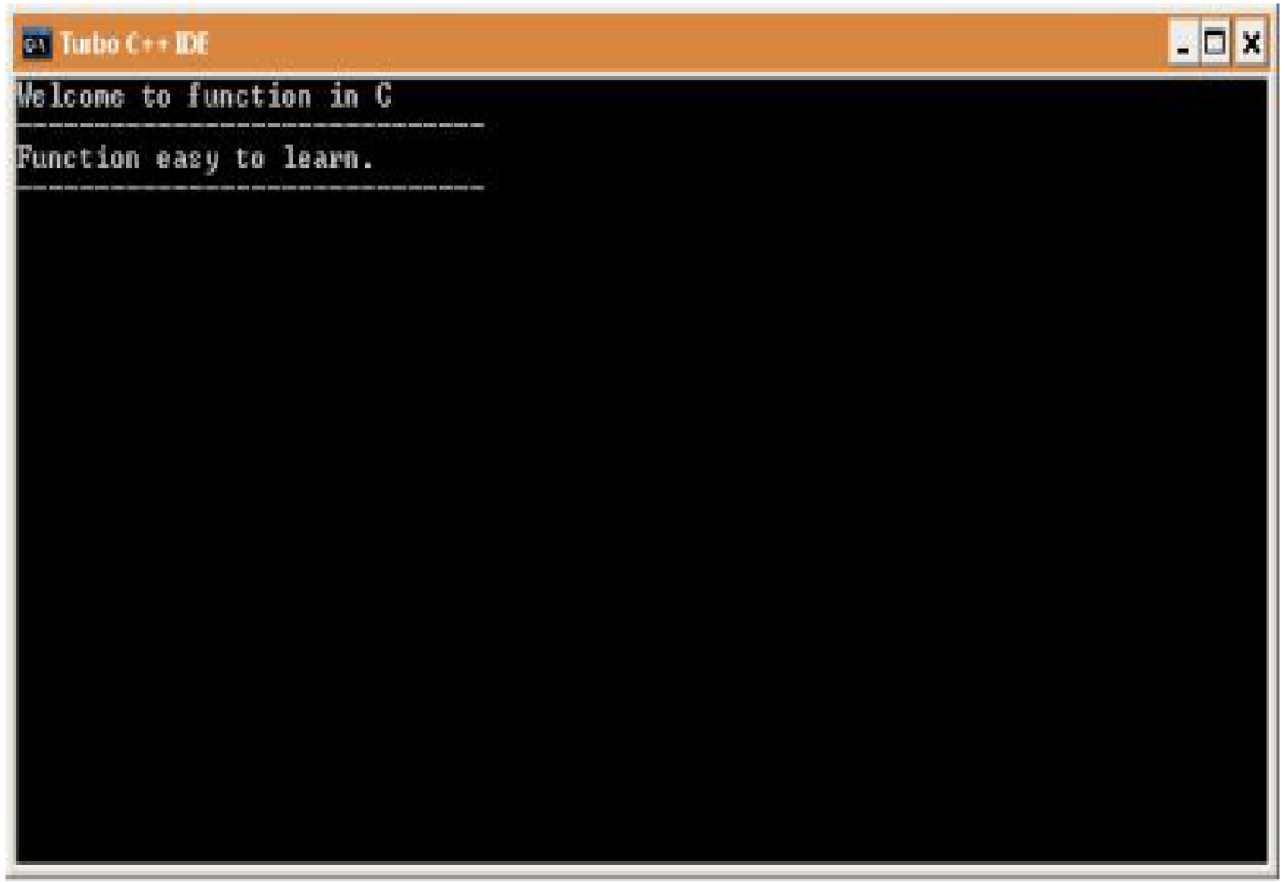
- In our previous example what we have noticed that “main()” function has no control over the “printfline()”, it cannot control its output.
- Whenever “main()” calls “println()”, it simply prints line every time.
- So the result remains the same.

Example

```
void printline()
{  int i;
for(i=0;i<30;i++)
{  printf("-");
}
printf("\n");
}
```

```
#include<stdio.h>
void printline();
void main()
{
printf("Welcome to function in
C");
printf("\n");
printline();
printf("Function easy to learn.");
printline();
}
```

Output:



```
Welcome to function in C
------------------------------
Function easy to learn.
------------------------------
```

Output of above program.

Functions with no arguments and no return value.

- A C function **without any arguments** means you **cannot pass data** (values like int, char etc) to the called function.
- Similarly, function with **no return type** does **not pass back data to the calling function**.
- It is one of the simplest types of function in C.
- It cannot be used in an expression.
- It can be used only as independent statement.

Function with **no arguments** and **no return value**

```
include<stdio.h>
void greatNum(); // function declaration
int main()
{ greatNum(); // function call
 return 0;
}
void greatNum() // function definition
{ int i, j;
printf("Enter 2 numbers that you want to compare...");
scanf("%d %d", &i, &j);
if(i > j)
{ printf("The greater number is: %d", i); }
else { printf("The greater number is: %d", j); }
}
```

Functions with no arguments and no return value.

- In our previous example what we have noticed that “main()” function has no control over the User Defined Function “`greatNum()`”, it cannot control its output.
- Whenever “main()” calls “`greatNum()`”, it simply scans two number in the function, and finds and prints the greater number every time.

An example of function

```
void evenodd()
{
int n;
scanf(“%d”,&n);
if(n%2==0)
        printf(“EVEN”);
else
        printf(“ODD”);
}
```

An example of function

```
#include<stdio.h>
void evenodd();
void main()
{
evenodd();
}
```

```
void evenodd()
{
int n;
scanf("%d",&n);
if(n%2==0)
		printf("EVEN");
else
		printf("ODD");
}
```

2. Functions with Arguments and No return value.

- A C function with arguments can perform much better than previous function type.
- This type of function can accept data from calling function.
- In other words, you send data to the called function from calling function but you cannot send result data back to the calling function. Rather, it displays the result on the terminal.
- But we can control the output of function by providing various values as arguments. Let’s have an example to get it better.

1/1! + 2/2! + 3/3!

1/fact(1) + 2/fact(2)

An example of function

```
void evenodd()
{
int n;
scanf("%d",&n);
if(n%2==0)
        printf("EVEN");
else
        printf("ODD");
}
```

```
void evenodd(int n)
{
int n;
scanf("%d",&n);
if(n%2==0)
        printf("EVEN");
else
        printf("ODD");
}
```

Not Required

An example of function

```
void evenodd()
{
int n;
scanf("%d",&n);
if(n%2==0)
        printf("EVEN");
else
        printf("ODD");
}
```

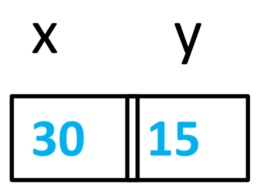
```
void evenodd(int n)
{
if(n%2==0)
        printf("EVEN");
else
        printf("ODD");
}
```

An example of function

```
#include<stdio.h>
void evenodd();
void main()
{ evenodd();
}
void evenodd()
{ int n;
scanf("%d",&n);
if(n%2==0)
        printf("EVEN");
else
        printf("ODD");
}
```

```
#include<stdio.h>
void evenodd( int n );
void main()
{ int n;
scanf("%d",&n);
evenodd(n);
}
void evenodd(int n)

if(n%2==0)
        printf("EVEN");
else
        printf("ODD");
}
```

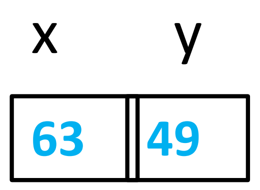
Example



```
#include<stdio.h>


void add(int x, int y)

{

int result;

result = x+y;

printf("Sum of %d and %d is %d.\n\n",x,y,result);

}
```

```
void main()
{
add(30,15);
add(63,49);
add(952,321);
}
```

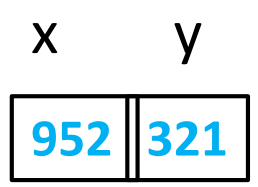
Example



```
#include<stdio.h>

void add(int x, int y)
{
int result;
result = x+y;
printf("Sum of %d and %d is %d.\n\n",x,y,result);
}
```

```
void main()
{
add(30,15);
add(63,49);
add(952,321);
}
```

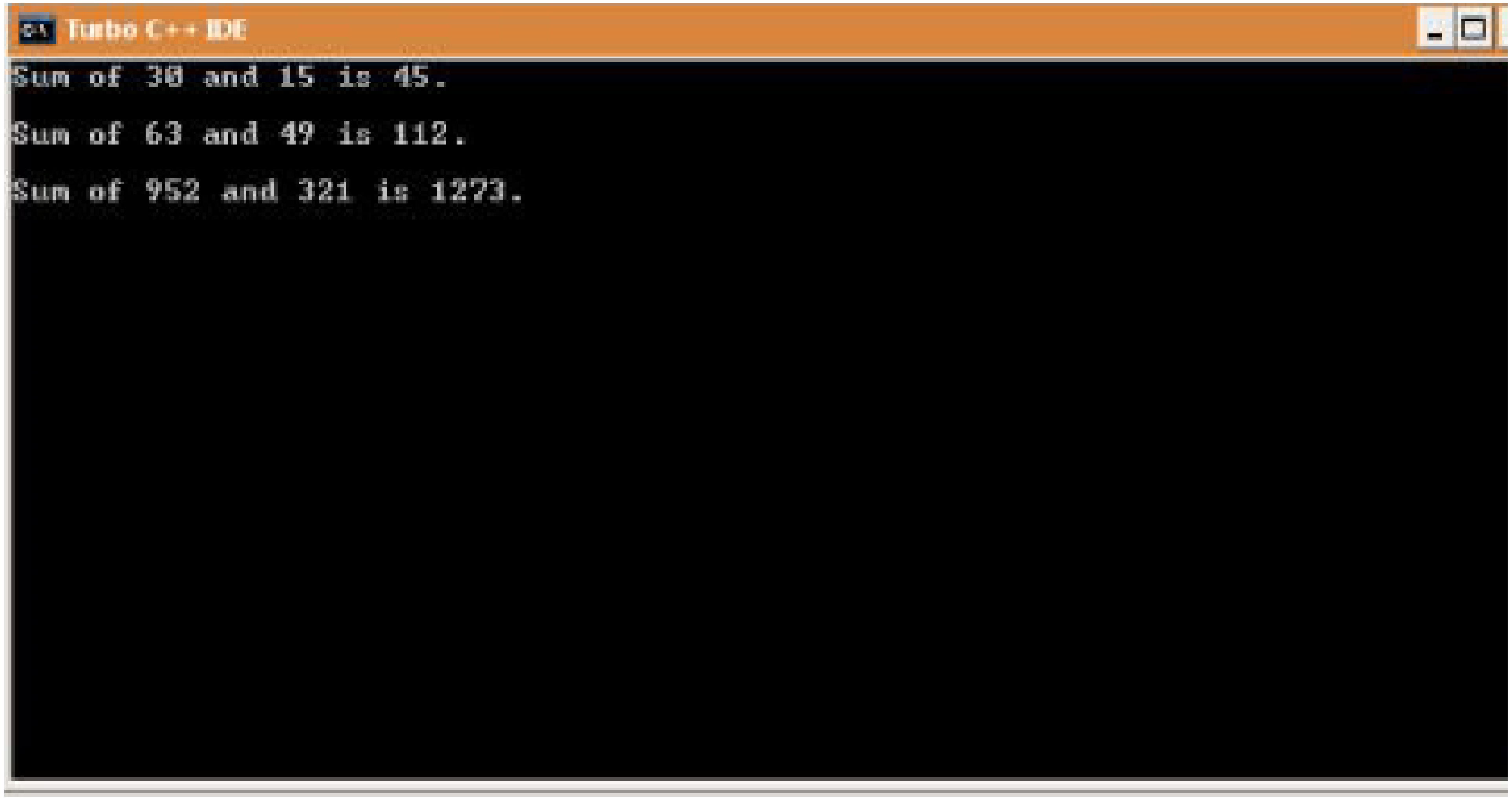
Example



```
#include<stdio.h>

void add(int x, int y)
{
int result;
result = x+y;
printf("Sum of %d and %d is %d.\n\n",x,y,result);
}
```

```
void main()
{
add(30,15);
add(63,49);
add(952,321);
}
```

OUTPUT



```
Sum of 30 and 15 is 45.

Sum of 63 and 49 is 112.

Sum of 952 and 321 is 1273.
```

Example : Argument passed but no return value

```
#include <stdio.h>
void checkPrimeAndDisplay(int n);
int main() {
int n;
printf("Enter a positive integer: ");
scanf("%d",&n); // n is passed to the function
checkPrimeAndDisplay(n);
return 0; } // return type is void meaning doesn't return any value

void checkPrimeAndDisplay(int n)
{ int i, flag = 0;
for(i=2; i <= n/2; ++i)
{
if(n%i == 0)
{ flag = 1;
break; } }
if(flag == 1) printf("%d is not a prime number.",n);
else printf("%d is a prime number.", n);
}
```

The input from the user is passed to the checkPrimeNumber() function.

The checkPrimeNumber() function checks whether the passed argument is prime or not.

3. Function with **no arguments** and **a return value**

- There could be occasions where we may need to design functions that may not take any arguments but returns a value to the calling function.
- A example for this is **getchar** function it has no parameters but it returns an integer an integer type data that represents a character.

Functions returning a value

- c = getch();
- ans = pow(x,y);

Functions with no arguments but returns value

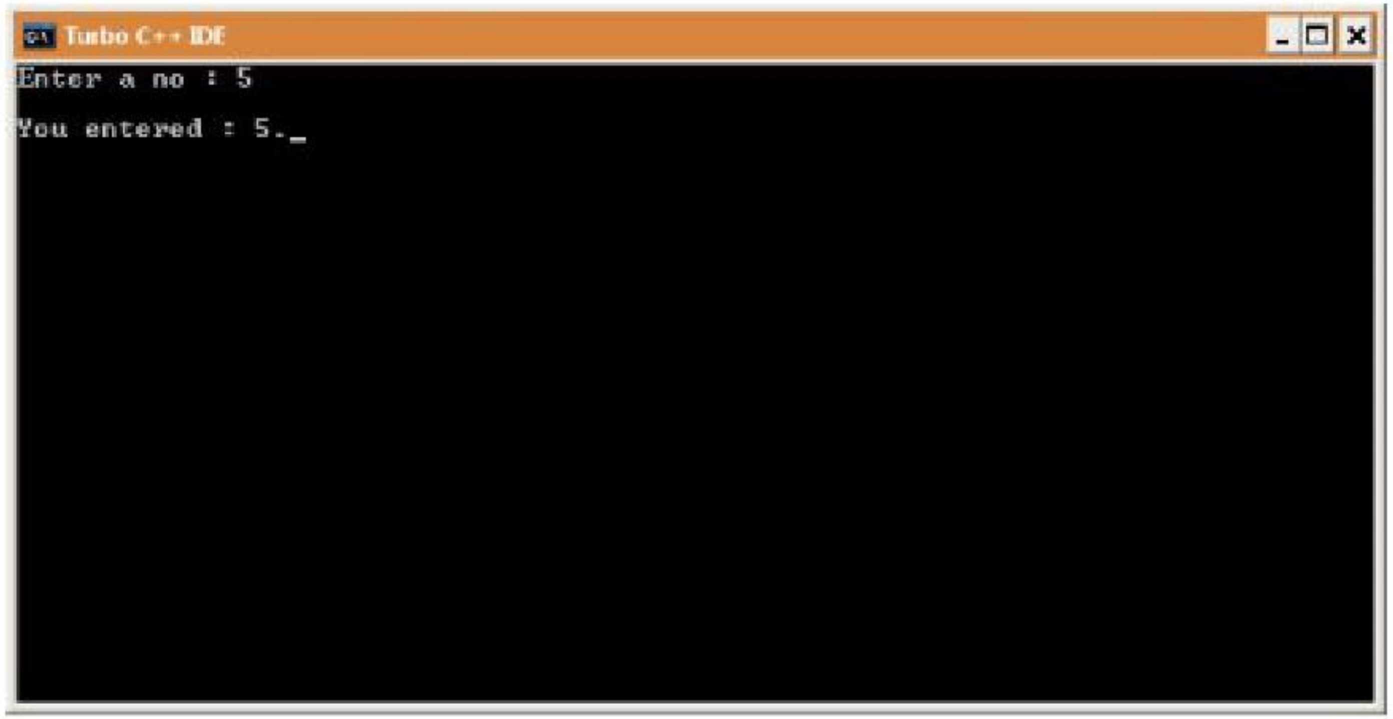
- We can create similar functions of our own.

Example

```
#include<stdio.h>
int send()
{
int no1;
printf("Enter a no : ");
scanf("%d",&no1);
return(no1);
}
```

```
void main()
{
int z;

z = send();

 printf("\nYou entered : %d.", z);
}
```

OUTPUT



```
Enter a no : 5

You entered : 5._
```

Output of the above program.

RETURN VALUES AND THEIR TYPES

- A function may or may not send back any value to the calling function
- Done through return statement
- It is possible to send any number of values to the called function
- The called function can only return one value per call
- SYNTAX:

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

Function with **no arguments** and **a return value**

```c
#include <stdio.h>

int sum();
int main()
{
    int num;
    num = sum();
    printf("\nSum of two given values = %d", num);
    return 0;
}

int sum()
{
    int a = 50, b = 80, s;
    s = a + b;
    return s;
}
```

Output:

```
Sum of two given values = 130
```

Example 2: No arguments passed but a return value

```
#include <stdio.h>
int getInteger();
int main() {
int nx, i, flag = 0; // no argument is passed
nx= getInteger();
for(i=2; i<=n/2; ++i)
{ if(n%i==0)
{ flag = 1;
break; } }
if (flag == 1)
printf("%d is not a prime number.", n);
else
printf("%d is a prime number.", n);
return 0; } // returns integer entered by the user
```

```
int getInteger()
{
 int n;
printf("Enter a positive integer: ");
 scanf("%d",&n);
return n;
}
```

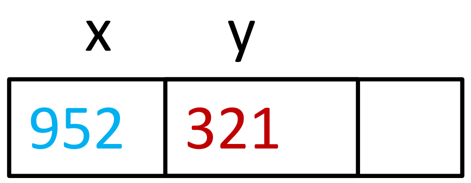
Here, the **getInteger()** function takes input from the user and returns it. The code to check whether a number is prime or not is inside the main() function.

The empty parentheses in the **n = getInteger();** statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to **n**.

4. Functions with arguments and return value

- This type of function can send arguments (data) from the calling function to the called function and wait for the result to be returned back from the called function back to the calling function.
- This type of function is mostly used in programming world because it can do two way communications; it can accept data as arguments as well as can send back data as return value.
- **The data returned by the function can be used later in our program for further calculations.**

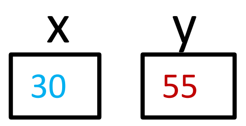
Example



```
#include<stdio.h>
#include<conio.h>
int add(int x, int y)
{  int result;
result = x+y;
return(result);
}
```

```
void main()
{
int z;
z = add(952,321);

printf("Result %d.\n\n",z);

}
```

Example



```
#include<stdio.h>
#include<conio.h>
int add(int x, int y)
{  int result;
result = x+y;
return(result);
}
```

```
void main()
{
int z;
z = add(952,321);

printf("Result %d.\n\n",z);

printf("Result %d.\n\n",add(30,55));
}
```

Example

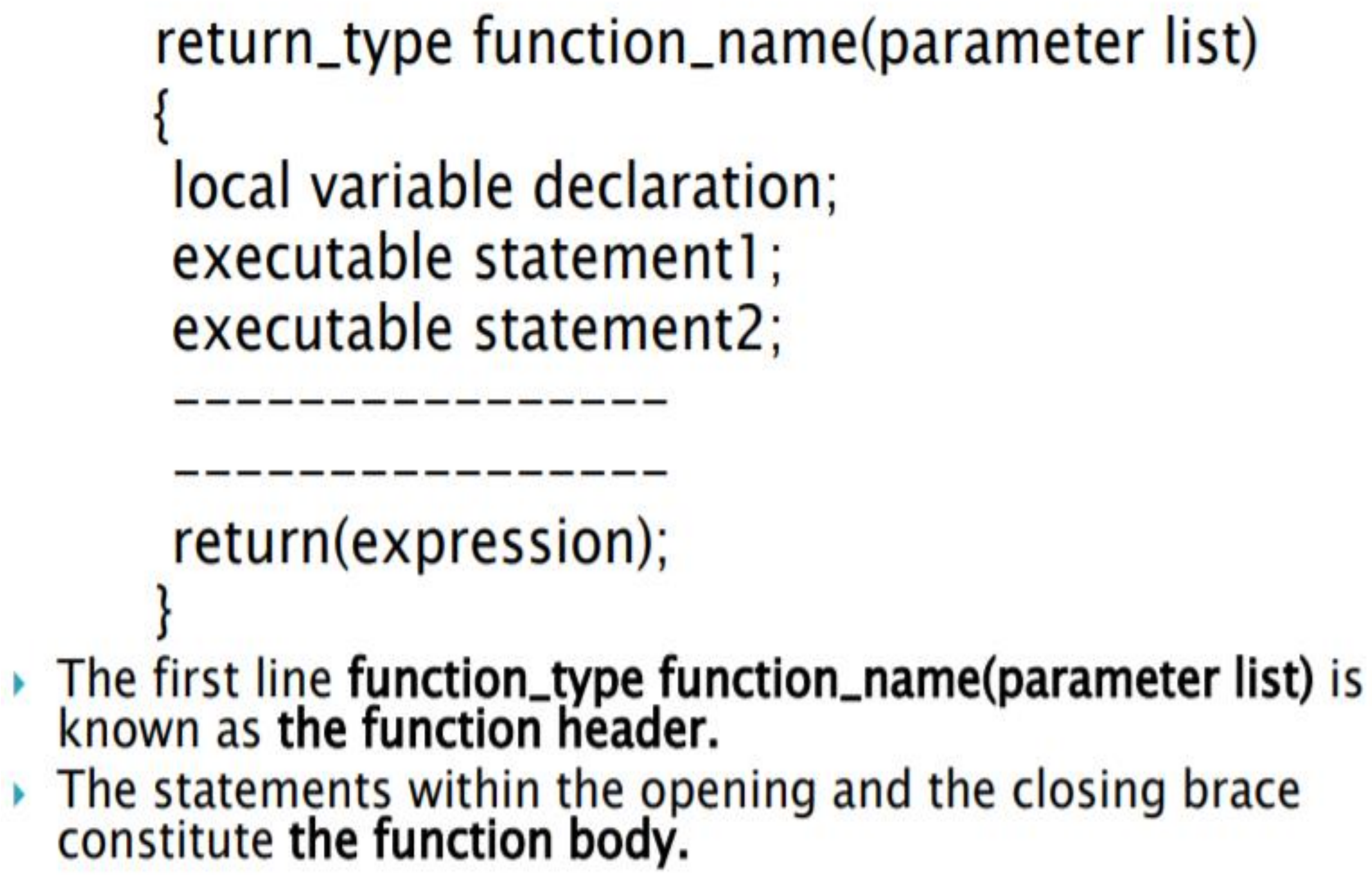
```
int sum (int x, int y)
{ int result;
result = x + y;
return (result);
}
```

```
int main ()
{ int a,b,ans;
scanf(“%d %d”,&a,&b);
ans = sum(a,b);
printf(“Ans =  %d”,ans);
return (0);
}
```

More on Functions

- Function Header
- Types of Parameters/Arguments:
    - Actual arguments
    - Formal arguments

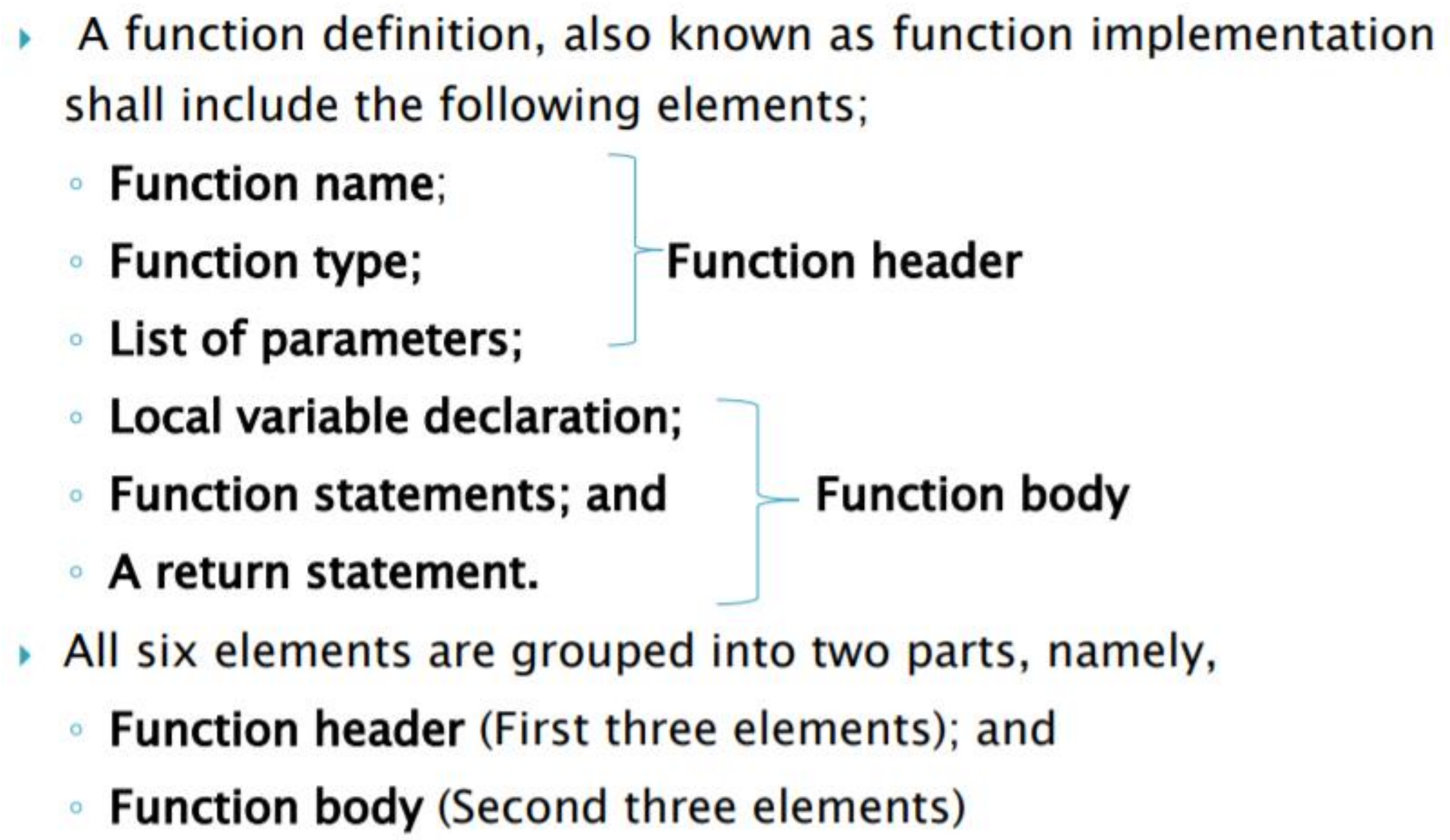
Function Header



```
return_type function_name(parameter list)
{
 local variable declaration;
 executable statement1;
 executable statement2;
 ----------------

 ----------------

 return(expression);
}
```

- The first line **function_type function_name(parameter list)** is known as **the function header.**
- The statements within the opening and the closing brace constitute **the function body.**

Function Header



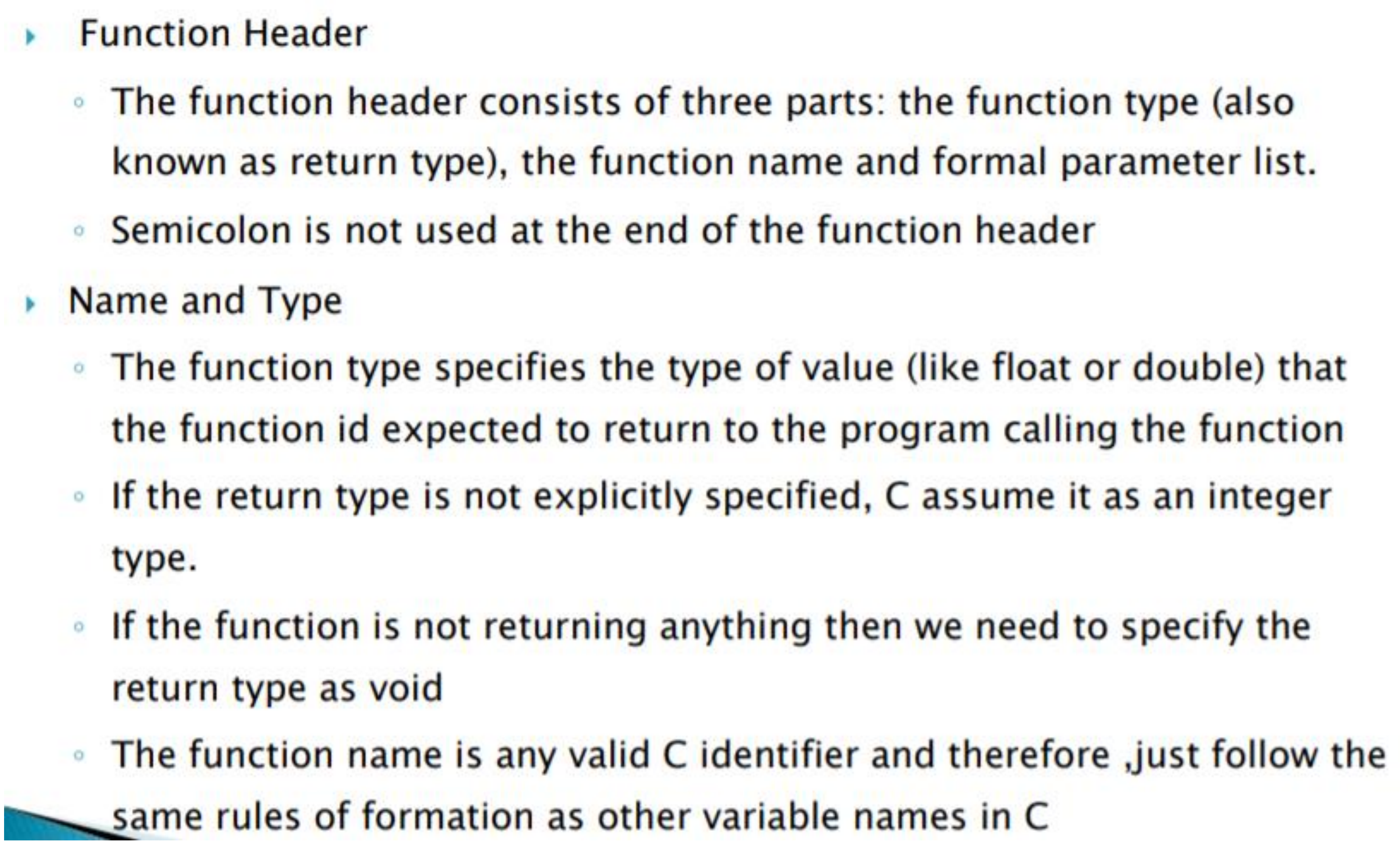
- A function definition, also known as function implementation shall include the following elements;
  - **Function name;**
  - **Function type;**
  - **List of parameters;**

  $\Big\}$ **Function header**

  - **Local variable declaration;**
  - **Function statements; and**
  - **A return statement.**

  $\Big\}$ **Function body**
- All six elements are grouped into two parts, namely,
  - **Function header** (First three elements); and
  - **Function body** (Second three elements)

Function Header



- Function Header
  - The function header consists of three parts: the function type (also known as return type), the function name and formal parameter list.
  - Semicolon is not used at the end of the function header
- Name and Type
  - The function type specifies the type of value (like float or double) that the function id expected to return to the program calling the function
  - If the return type is not explicitly specified, C assume it as an integer type.
  - If the function is not returning anything then we need to specify the return type as void
  - The function name is any valid C identifier and therefore ,just follow the same rules of formation as other variable names in C

Structure of a Function

A general form of a C function looks like this:

```
<return type> FunctionName (Argument1, Argument2,……)
 {
Statement1;
Statement2;
Statement3;
}
```

Definitions

- *Parameter:–* a declaration of an identifier within the `'()'` of a function declaration
    - Used within the body of the function as a *variable* of that function
    - Initialized to the value of the corresponding *argument*.
- *Argument:–* an expression passed when a function is *called*; becomes the initial value of the corresponding parameter

Example

```
/*  To perform Addition of two numbers
 With Argument  and With Return values */
#include<stdio.h>
int add(int,int);     //Function prototype declaration
void  main()
{
int c;
printf("Enter two Numbers=");
scanf("%d  %d",&a,&b);
c=add(a,b);  /* Actual Arguments */
printf("The sum of two numbers is  %d",c);
}
int add(int x,int y)  /* Formal arguments */
{
int c;
c=x+y;
return(c);
}
```

Function Header

FORMAL PARAMETER LIST

- The parameter list declares the variables that will receive the data sent by the calling program.
- They serve as input data to the function to carry out the specified task.
- They represent actual input values, they are often referred to as formal parameters.
- These parameters can also be used to send values to the calling programs
- **The parameter is known as arguments.**
  - **float quadratic (int a, int b, int c) { ….. }**
  - **double power (double x, int n) { ….. }**
  - **int sum (int a, int b) { ….. }**
- There is no semicolon after the closing parenthesis
- The declaration parameter variables cannot be combined

Example

```
/*  To perform Addition of two numbers
 With Argument  and With Return values */
#include<stdio.h>
int add(int,int);     //Function prototype declaration
void  main()
{
int c;
printf("Enter two Numbers=");
scanf("%d  %d",&a,&b);
c=add(a,b);  /* Actual Arguments */
printf("The sum of two numbers is  %d",c);
}
```

Function Header

```
int add(int x,int y)  /* Formal arguments */
{
```

Here int x and int y are the formal parameter list

```
int c;
c=x+y;
return(c);
}
```

FORMAL PARAMETER LIST

- To indicate that the parameter list is empty, we use the keyword void between the parentheses as in

```
void printline (void)
{
 ...
}
```

- Many compiler accept an empty set of parentheses

```
void printline()
```

- It is good to use void to indicate a nill parameter list

Empty Parameter List

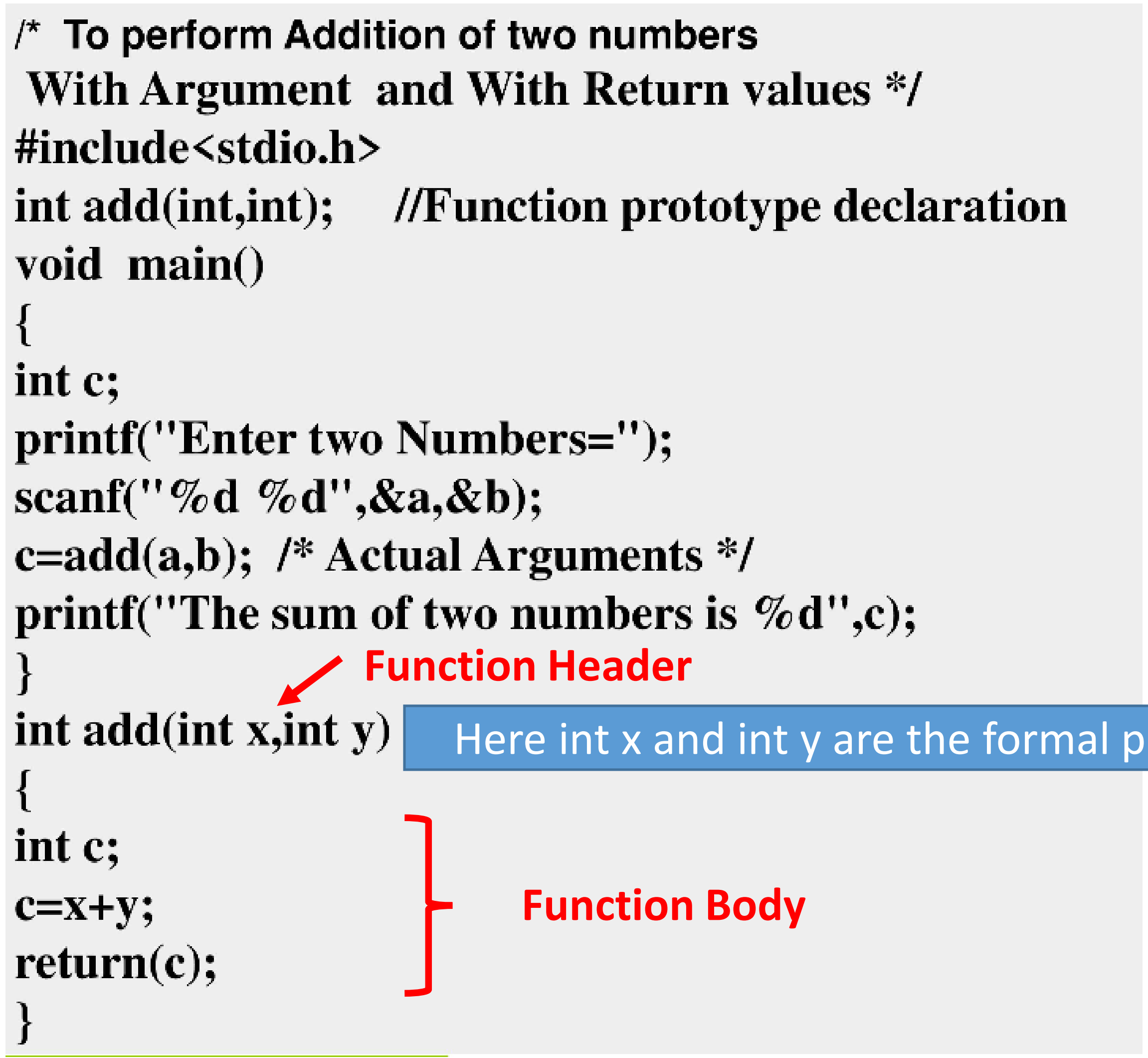
```
main( )
{
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}
italy( )
{
    printf ( "\nI am in italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in italy" ) ;
}
brazil( )
{
    printf ( "\nI am in brazil" ) ;
    argentina( ) ;
}
argentina( )
{
    printf ( "\nI am in argentina" ) ;
}
```

Parameter list is empty

FUNCTION BODY

- The function body contains the declarations and statements necessary for performing the required task. The body enclosed in braces, contains three parts,
  - Local declarations that specify the variables needed by the function
  - Function statements that perform the task of the function
  - A return statement that returns the value evaluated by the function
- If a function does not return any value, we can omit the return statement.
- Its return type should be specified as void

Example



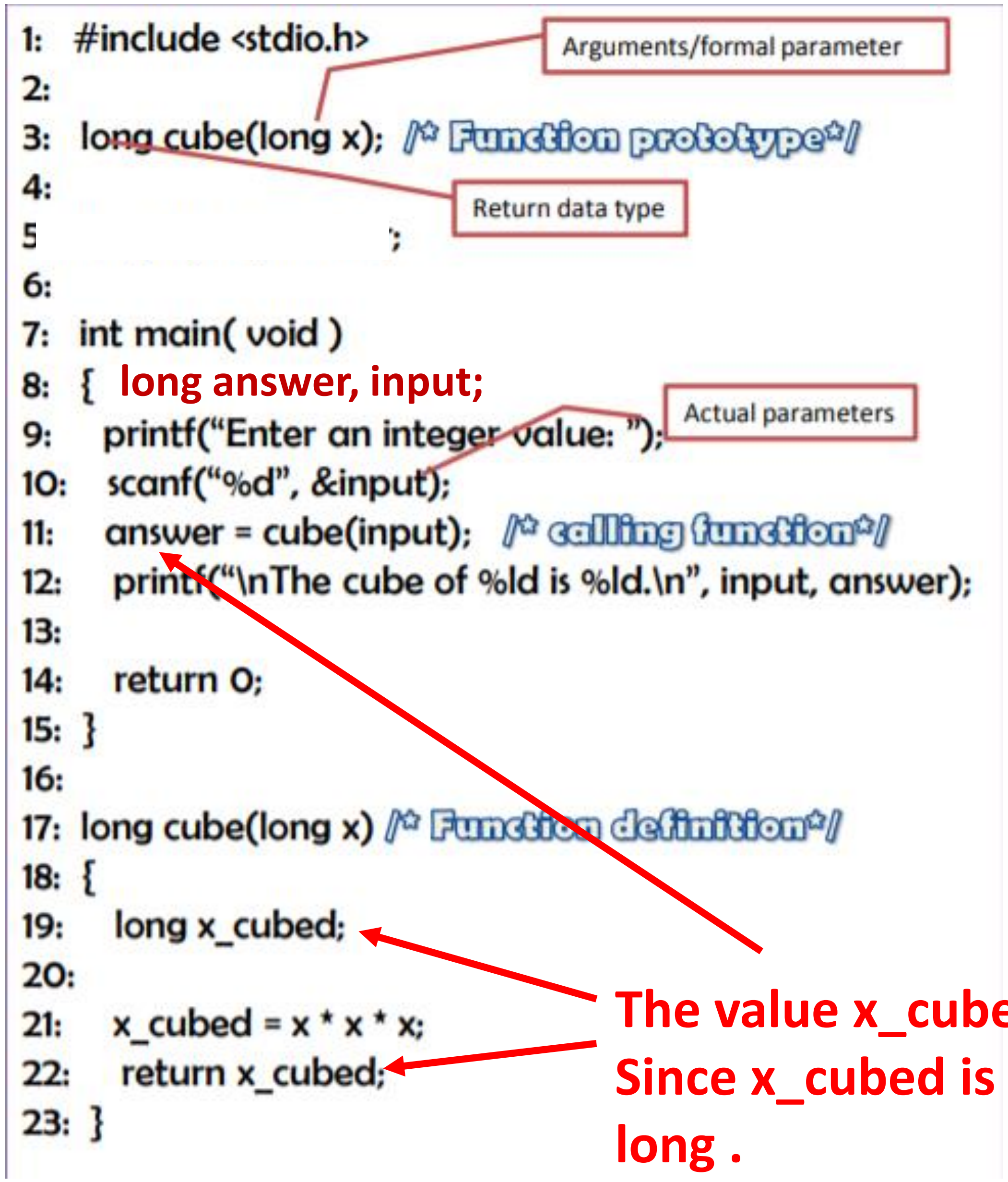
```
/*  To perform Addition of two numbers
 With Argument  and With Return values */
#include<stdio.h>
int add(int,int);     //Function prototype declaration
void  main()
{
int c;
printf("Enter two Numbers=");
scanf("%d %d",&a,&b);
c=add(a,b);  /* Actual Arguments */
printf("The sum of two numbers is %d",c);
}
int add(int x,int y)
{
int c;
c=x+y;
return(c);
}
```

Function Header

Here int x and int y are the formal parameter list

Function Body

Example



```
#include <stdio.h>

long cube(long x);  /* Function prototype*/

                          ;

int main( void )
{  long answer, input;
   printf(“Enter an integer value: ”);
   scanf(“%d”, &input);
   answer = cube(input);   /* calling function*/
    printf(“\nThe cube of %ld is %ld.\n”, input, answer);

    return 0;
}

long cube(long x) /* Function definition*/
{
    long x_cubed;

    x_cubed = x * x * x;
     return x_cubed;
}
```

Function names is cube

- Variable that are requires is long
- The variable to be passed on is X(has single arguments)—value can be passed to function so it can perform the specific task. It is called arguments

The value x_cubed being returned to main function.
Since x_cubed is declared as long so return type is declared as long .
Note: answer in the main program is also declared as long

How the function works

- C program doesn't execute the statement in function until the function is called.
- When function is called the program can send the function information in the form of one or more argument.
- When the function is used it is referred to as the called function.
- Functions often use data that is passed to them from the calling function.
- Data is passed from the calling function to a called function by specifying the variables in a argument list.
- **Argument list** cannot be used to send data. Its only copy data/value/variable that pass from the calling function.
- **The called function then performs its operation using the copies**

Exercise

- Write a program to check whether year is leap year or not using functions
- Write a program to find sum of the squares of first n numbers and to calculate its sum.
- WAP to calculate 1/1! + 2/2! + 3/3! + 4/4!......+n/n!
- (Two functions – fact(); calculateexp();
- Fact function is used within the function calculateexp()