UNIVERSITY of SAN CARLOS

SCIENTIA • VIRTUS • DEVOTIO

DCISM

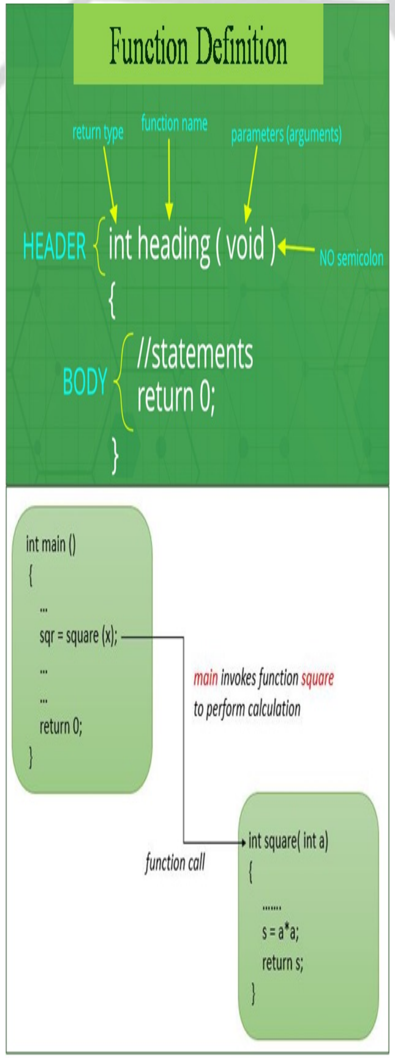
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

CIS 1101 – PROGRAMMING 1

FUNCTION IN C

Part 1

C FUNCTION: DEFINITION



```
int heading ( void )
{
    //statements
    return 0;
}
```

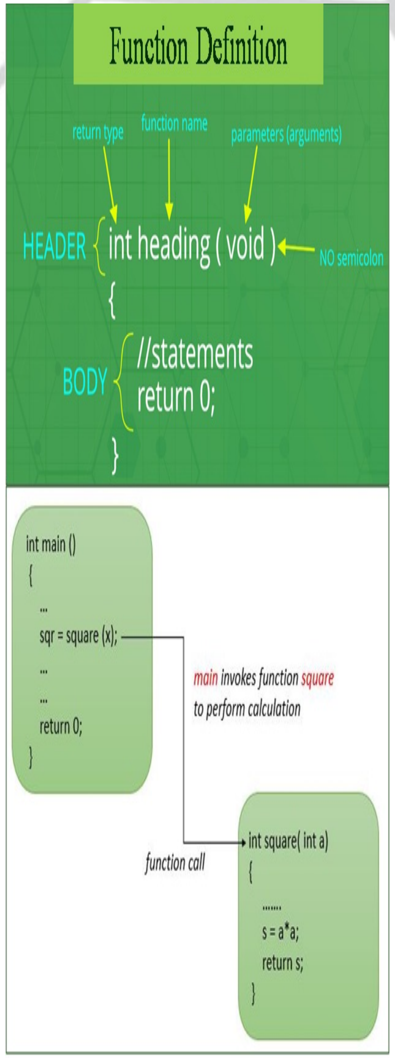
```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- A function is a set of statements that take inputs, do some specific computation or operation and produces output.

- It is a block of code that performs a specific task.

C FUNCTION: OTHER INFORMATION



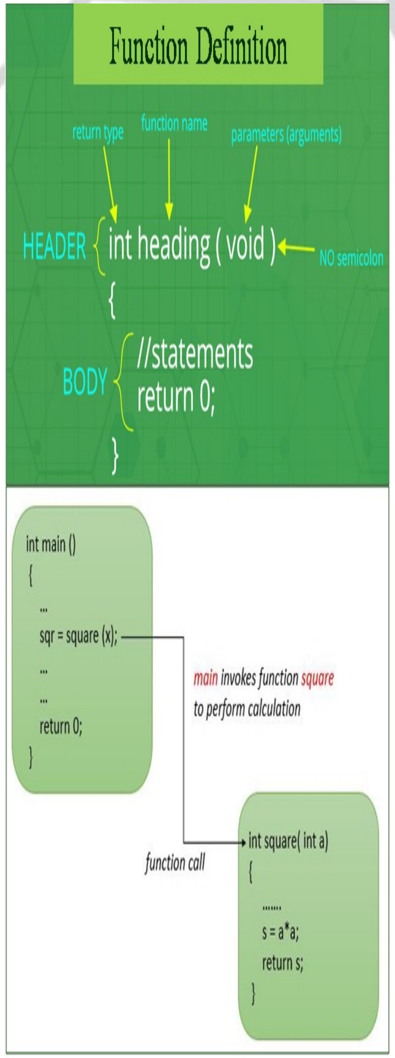
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- It contains set of instructions enclosed by "{ }" which performs specific operation in a C program.
- It has a name and it is reusable (can be executed from as many different parts in a C Program as required).
- It also optionally returns a value to the calling program.
- All C programs are written using functions to improve re-usability, understandability and to keep track on them.

C FUNCTION: OTHER INFORMATION



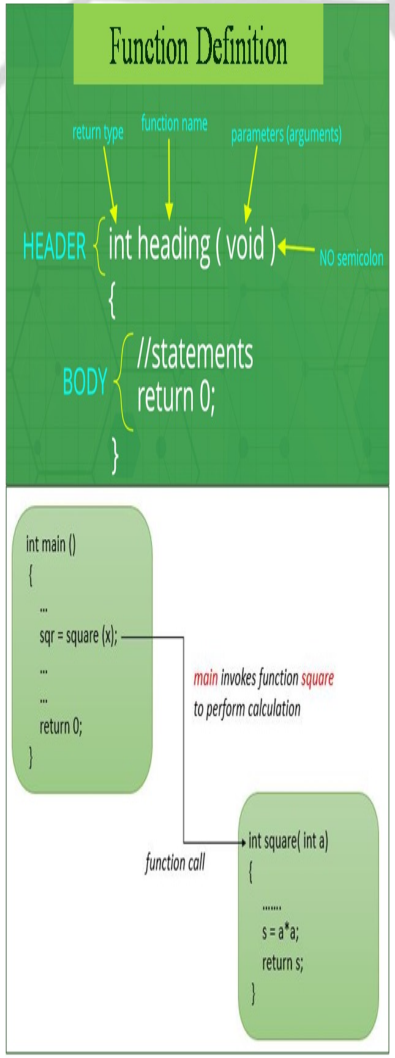
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- Collection of these functions creates a C program.
- C allows you to define functions
  - according to your need and
  - these functions are known as **user-defined functions**.
- It is known with various names like
  - a **method** or
  - a **sub-routine** or
  - a **procedure**, and
  - **others**.

C FUNCTION: USES



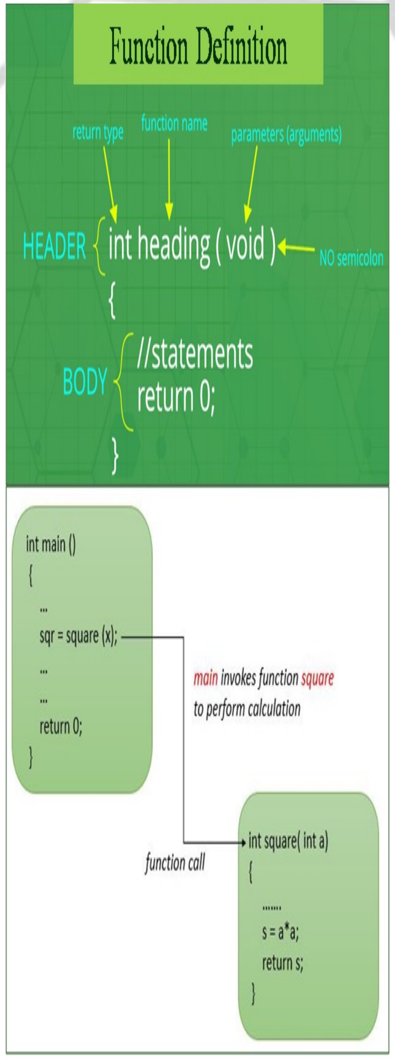
```
int heading ( void )
{
   //statements
   return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- C functions are used to avoid rewriting same logic/code again and again in a program.
- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.
- A large C program can easily be tracked when it is divided into functions.

C FUNCTION: BENEFITS



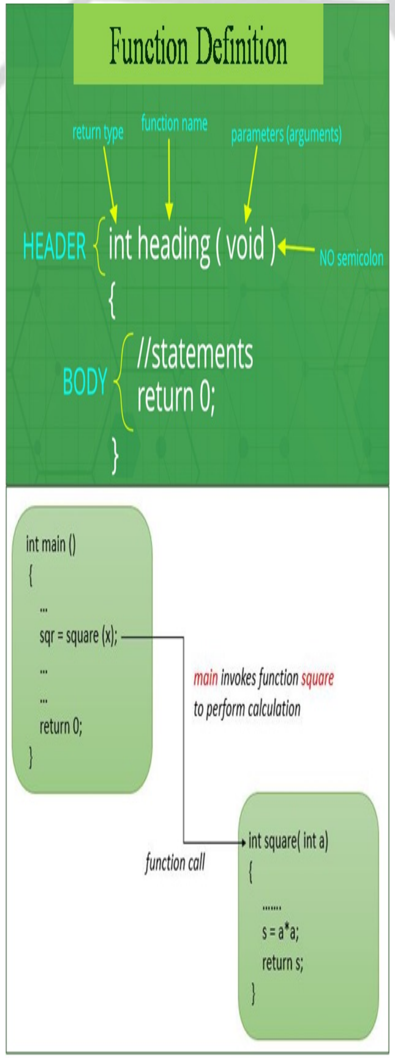
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    ......
    s = a*a;
    return s;
}
```

- Provides modularity
- Provides reusable code
- Make debugging and editing tasks in large programs easy
- Program can be modularized into smaller parts
- Separate function can be developed according to the needs

TYPES OF C FUNCTION: LIBRARY FUNCTIONS



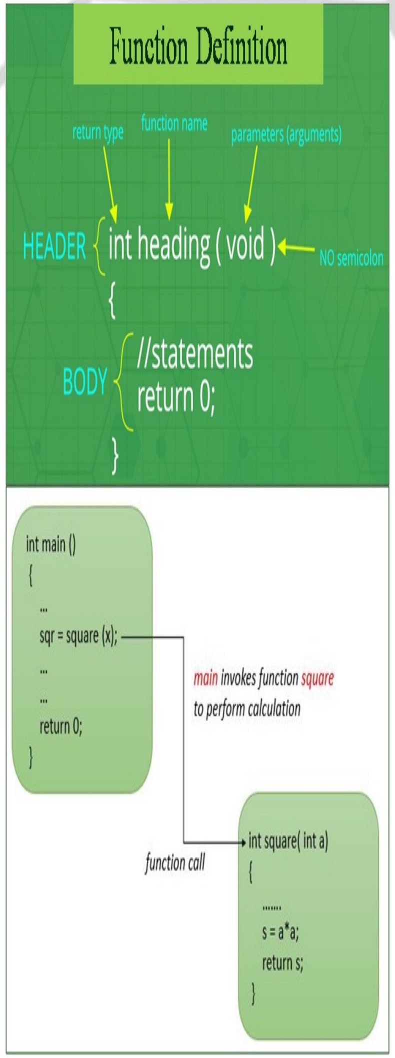
```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- These are built-in functions which are
  - grouped together and placed in a common place called library.
- Each library function in C
  - performs a specific operation.
- One can make use of these library functions
  - to get the pre-defined output instead of writing own code to get those outputs.

TYPES OF C FUNCTION: LIBRARY FUNCTIONS



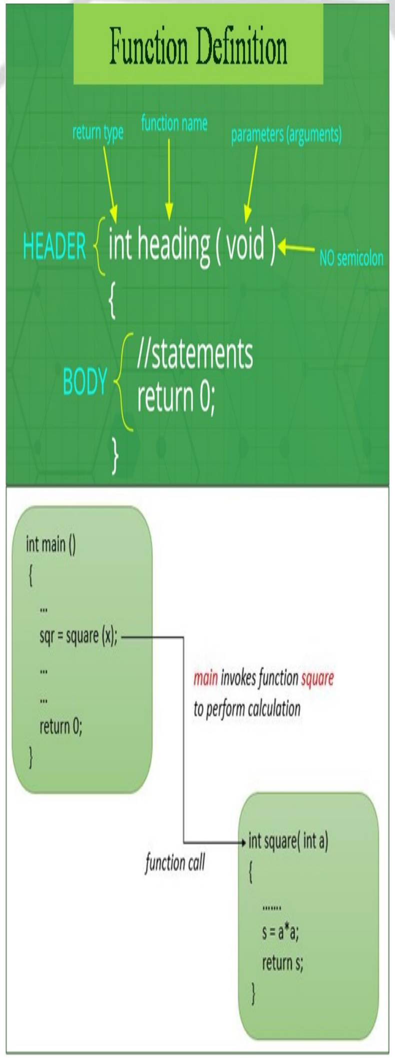
```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ..
  sqr = square (x);
  ..
  ..
  return 0;
}
```

```
int square( int a)
{
  ......
  s = a*a;
  return s;
}
```

- These library functions are
  - created by the persons who designed and created C compilers.
- All C standard library functions are
  - declared in many header files which are saved as file_name.h.
- Function declaration and definition for macros
  - are given in all header files.

TYPES OF C FUNCTION: LIBRARY FUNCTIONS



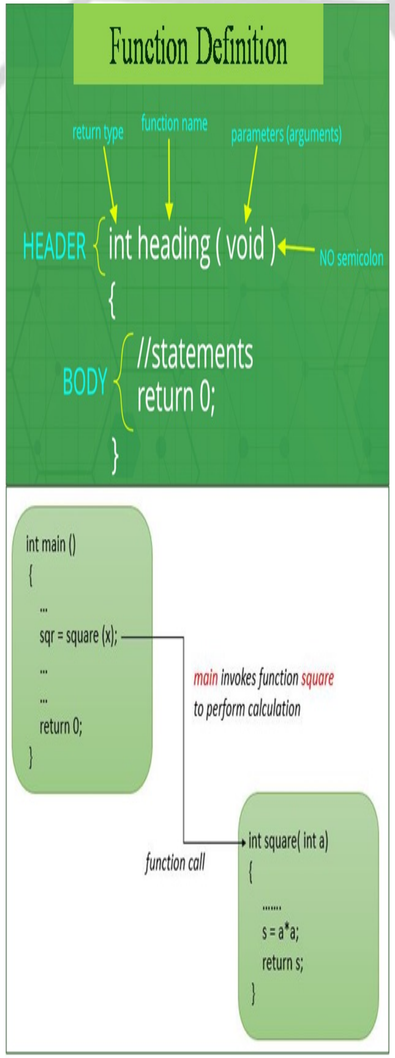
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- These header files are
  - included in C program using "#include<file_name.h>" command to make use of the functions that are declared in the header files.
- When header files are included in C program
  - using "#include<filename.h>" command, all C code of the header files are included in the said C program, then, this C program is compiled by compiler and executed.

C LIBRARY FUNCTIONS & HEADER FILES



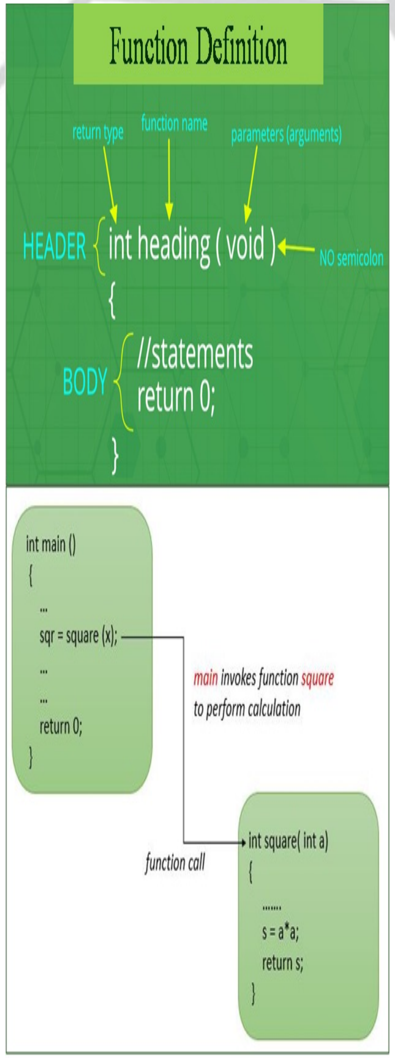
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    ......
    s = a*a;
    return s;
}
```

Header File	Description
stdio.h	This is standard input/output header file in which Input/Output functions are declared
conio.h	This is console input/output header file
string.h	All string related functions are defined in this header file
stdlib.h	This header file contains general functions used in C programs

C LIBRARY FUNCTIONS & HEADER FILES



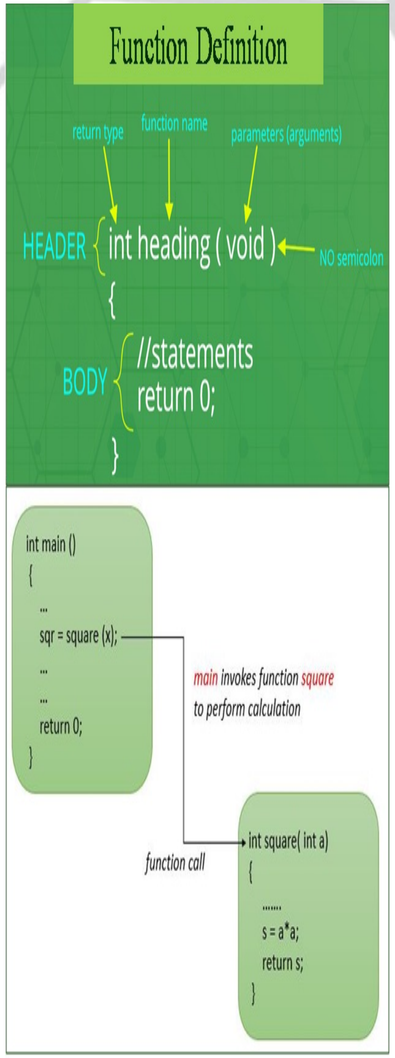
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

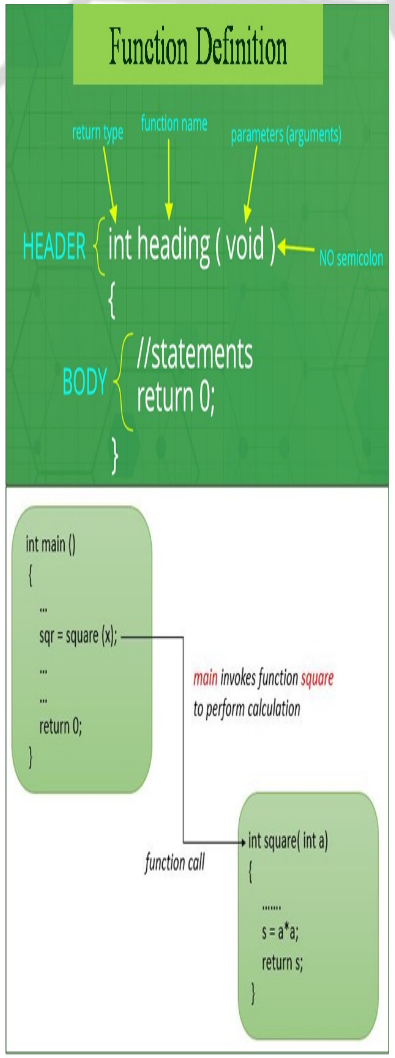
Header File	Description
math.h	All math related functions are defined in this header file
time.h	This header file contains time and clock related functions
ctype.h	All character handling functions are defined in this header file
stdarg.h	Variable argument functions are declared in this header file

C LIBRARY FUNCTIONS & HEADER FILES



Header File	Description
signal.h	Signal handling functions are declared in this file
setjmp.h	This file contains all jump functions
locale.h	This file contains locale functions
errno.h	Error handling functions are given in this file
assert.h	This contains diagnostics functions

TYPES OF C FUNCTION: USER-DEFINED FUNCTIONS



```
int heading ( void )
{
    //statements
    return 0;
}
```

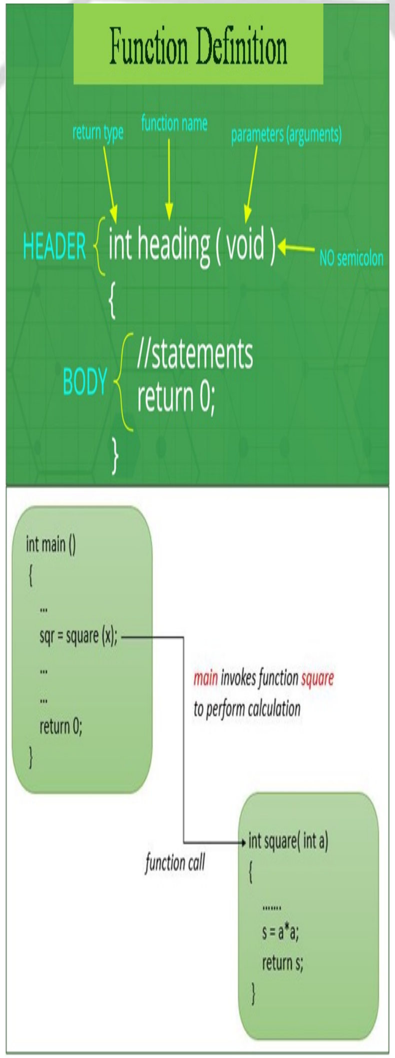
```
int main ()
{
   ...
   sqr = square (x);
   ...
   ...
   return 0;
}
```

```
int square( int a)
{
   .......
   s = a*a;
   return s;
}
```

- These are the functions
  - that are self-contained blocks of statements which are written by the user to compute or perform a task.

- These functions
  - can be called by the main program repeatedly as per the requirement.

TYPES OF C FUNCTION: USER-DEFINED FUNCTIONS



```
int heading ( void )
{
  //statements
  return 0;
}
```

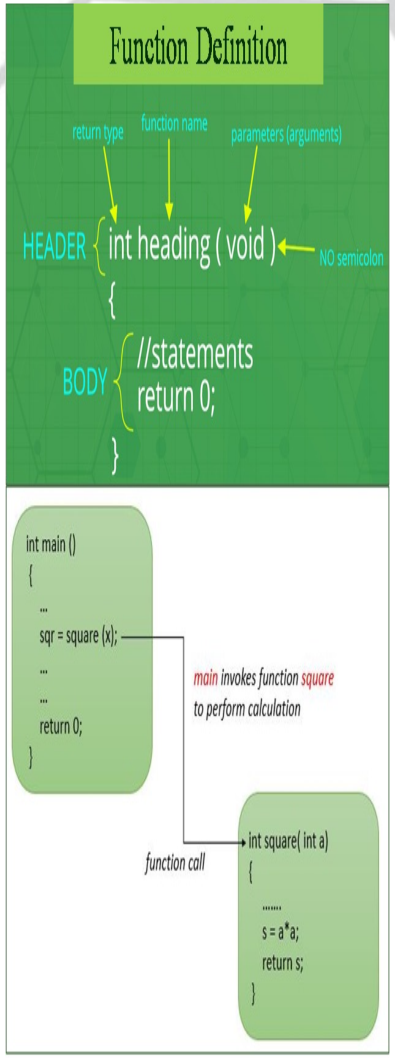
```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- One can add
  - own user defined functions in C library.

- It is possible to
  - add, delete, modify and access own user defined function **to or from** C library.

C FUNCTION: 3 IMPORTANT ASPECTS



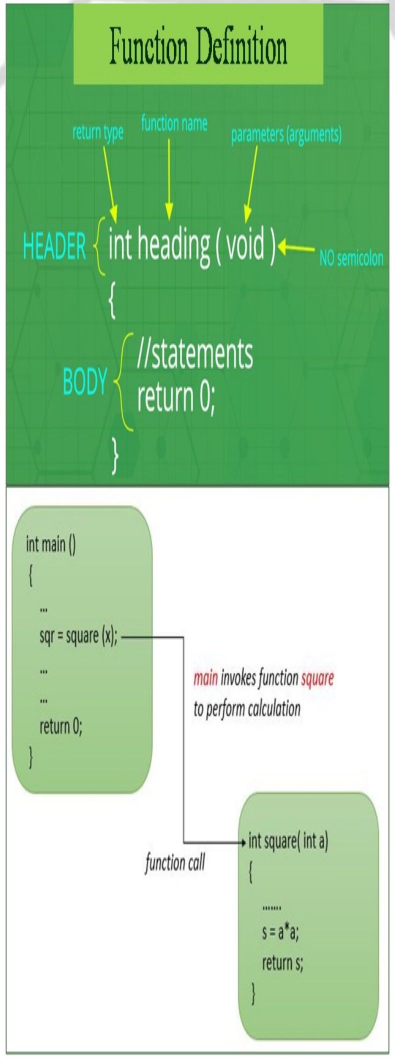
```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- **Function declaration (or prototype):**
  - This informs the compiler about the
    - ✓ **function name**,
    - ✓ **function parameters**
      - (the type of data it expects to receive), and
    - ✓ **return value type**
      - (the type of data it will return to the calling function).

C FUNCTION: 3 IMPORTANT ASPECTS



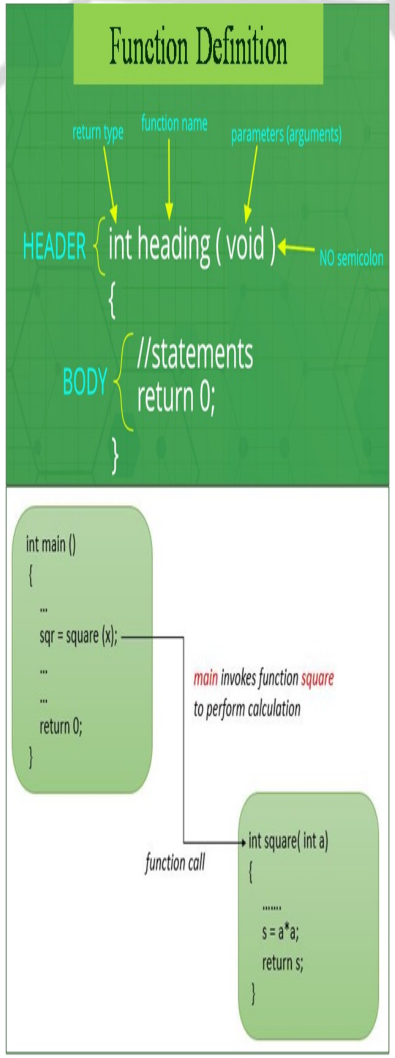
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- **Function call:**
  - This **calls** the actual function.
- **Function definition:**
  - This comprises the function header and the body of the function.

C FUNCTION: 3 IMPORTANT ASPECTS



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

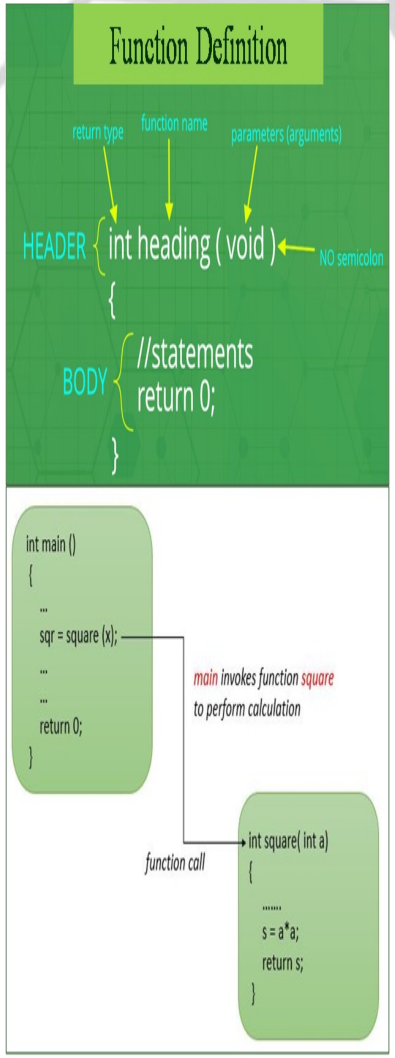
```
int square( int a)
{
   .......
   s = a*a;
   return s;
}
```

C Function Aspects	Syntax

Function definition:
▪ Function Header
▪ Function Body
	Return_type function_name (parameters list)
	{
	    Body of the function;
	}

Function call	Function_name (arguments list);
Function declaration (or prototype)	Return_type function_name (parameters list);

C FUNCTION DEFINITION: FUNCTION HEADER



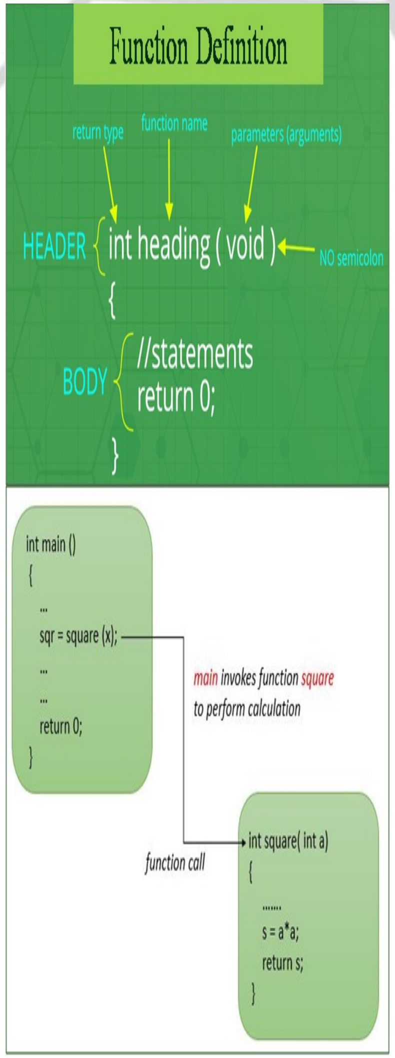
```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  ......
  s = a*a;
  return s;
}
```

- Provides the **information** about the
  - type of function,
  - name of function, and
  - a list of parameters (formal parameters).
- The **list of parameters** comprises
  - the types of parameters and
  - names of parameters;
  - enclosed in parentheses and
  - separated by commas.

C FUNCTION: RETURN TYPE



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

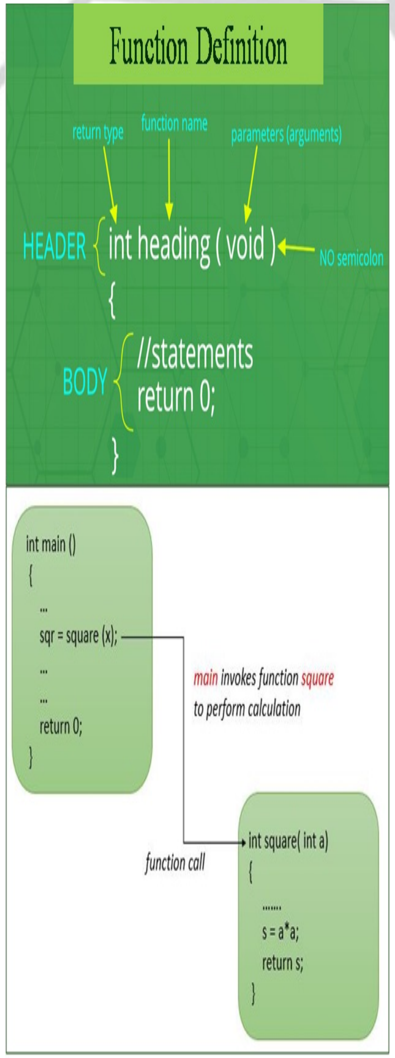
```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- A function **may** return a value.

- The function return_type is
  - the data type of the value that
  - the function returns.

- Some functions
  - perform the desired operations without returning a value and
  - its return_type is the keyword **void.**

C FUNCTION: FUNCTION NAME



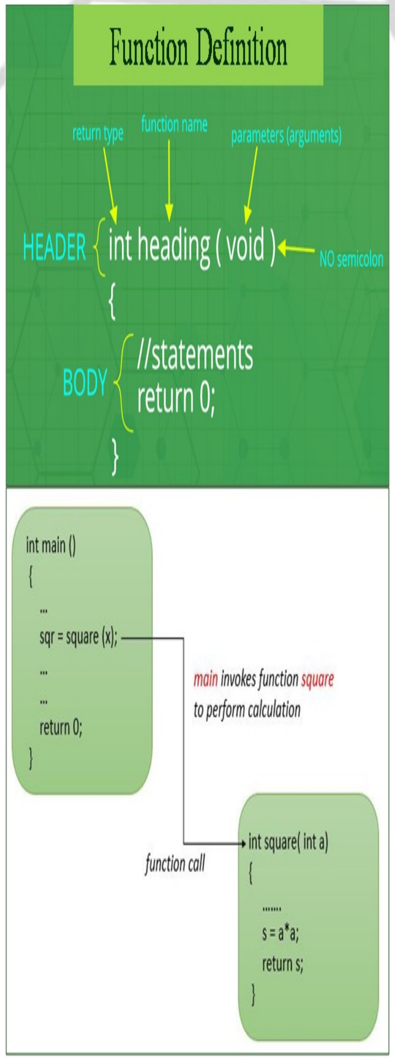
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    ......
    s = a*a;
    return s;
}
```

- The function name is based on the task the function will do.
- Since it is a task, it should be an action word (verb)
  - Examples:
    - ✓findPositive
    - ✓getNum
    - ✓computeSalary
- Naming a function will follow a camel notation

C FUNCTION: PARAMETER & ARGUMENT



```
int heading ( void )
{
    //statements
    return 0;
}
```

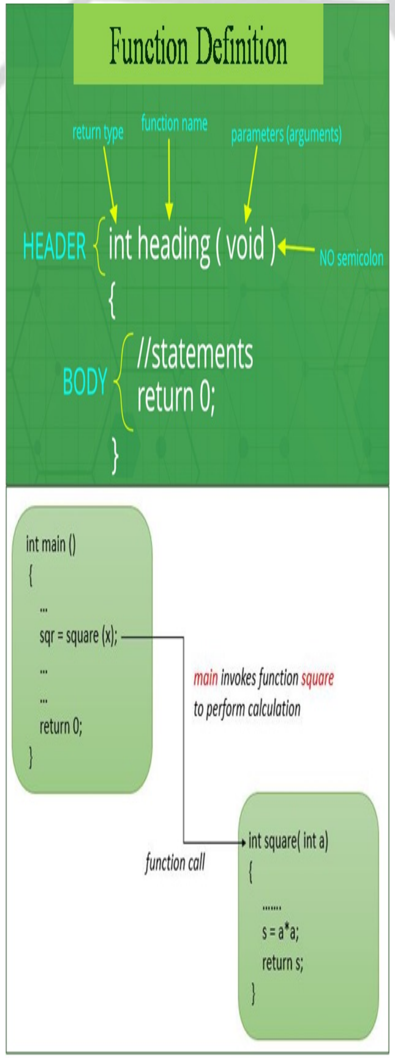
```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- **Argument:**
  - The actual value that is passed to the function as input when it is called.

- **Parameter:**
  - The variables that are used in the function declaration (or definition) to represent the arguments that were passed to the function during the function call.

C FUNCTION: PARAMETER & ARGUMENT



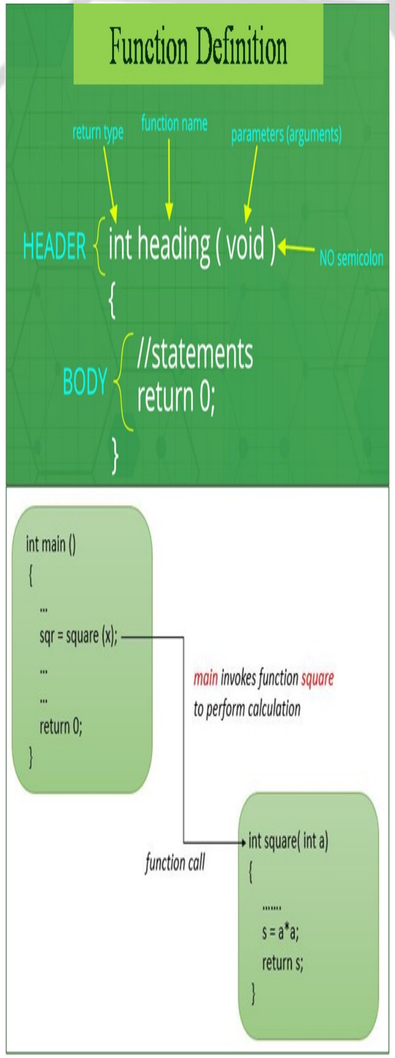
```
int heading ( void )
{
   //statements
   return 0;
}
```

```
int main ()
{
   ...
   sqr = square (x);
   ...
   ...
   return 0;
}
```

```
int square( int a)
{
   .......
   s = a*a;
   return s;
}
```

- When a function is invoked,
  - a value is passed to the parameter and this value is referred to as actual parameter (or argument).
- The parameter list refers to
  - the type, order, and number of the parameters of a function.
- Parameters are optional:
  - A function may contain no parameters.

C FUNCTION DEFINITION: BODY OF A FUNCTION



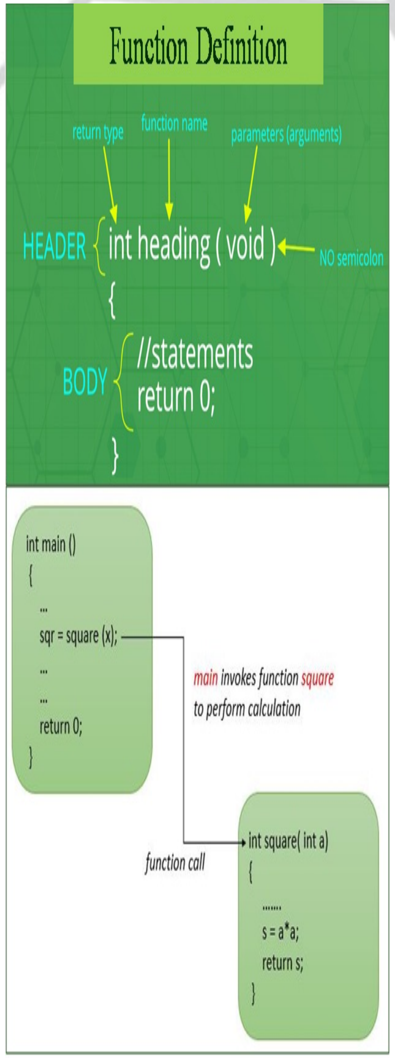
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- A **part of a function** which includes
  - the declarations of its local variables and the statements that determine what the function does.

CALLING A C FUNCTION: EXAMPLE PROBLEM



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

Given:

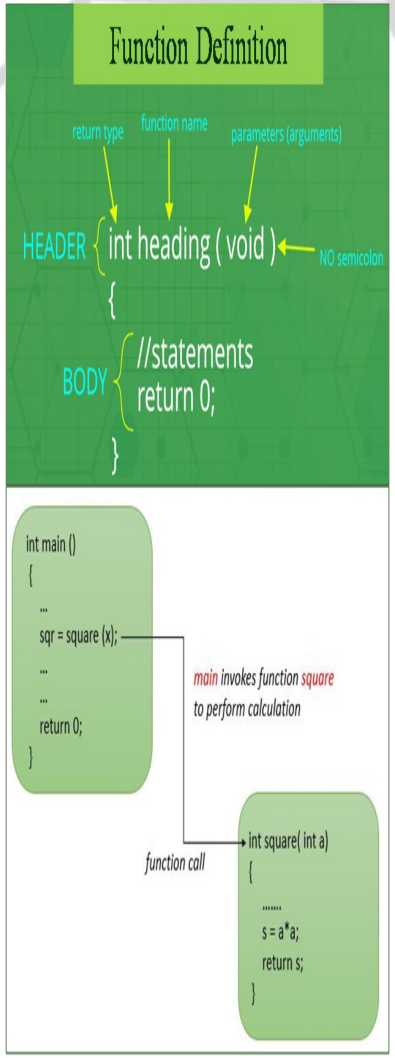
A program will ask the user to enter two integers and will calculate the sum of the integers. Then it will print the sum.

Asked:

a) Create a function that will accept two integers as parameters and will calculate their sum. Then it will return the sum.

b) Write the code of the program using the created C function.

C FUNCTION: FUNCTION SPECIFICATIONS

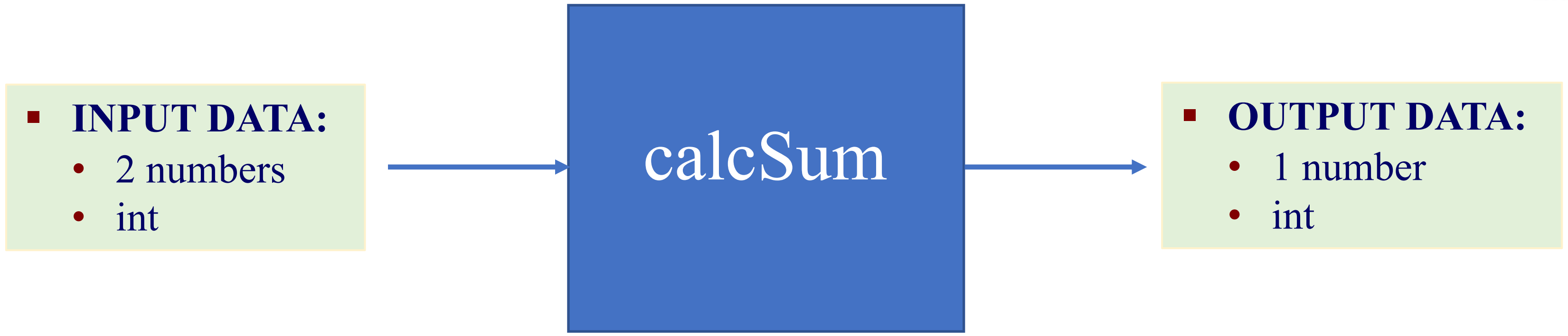


```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
   ...
   sqr = square (x);
   ...
   ...
   return 0;
}
```

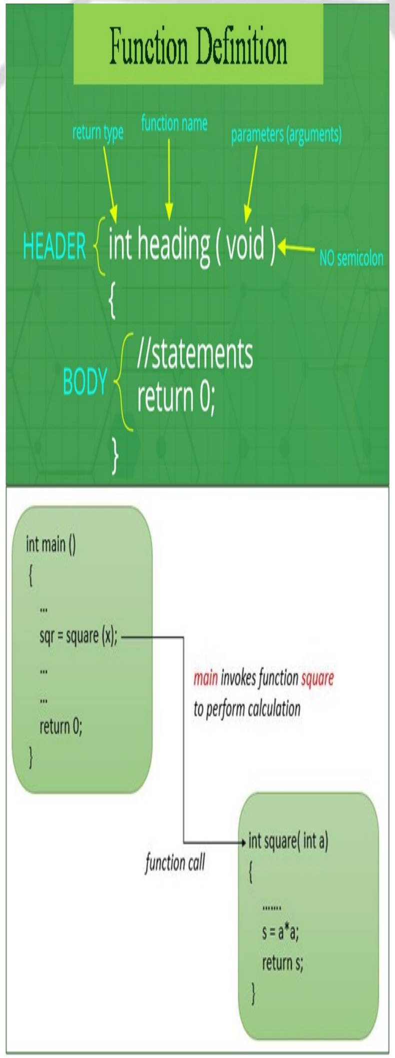
```
int square( int a)
{
   .......
   s = a*a;
   return s;
}
```

- Precision is specified:



- Function name:
  - calcSum (use camel notation)
- Formal Parameters:
  - int x, int y
- Return type:
  - int

C FUNCTION: STEPS IN FUNCTION CREATION



```
int heading ( void )
{
  //statements
  return 0;
}
```

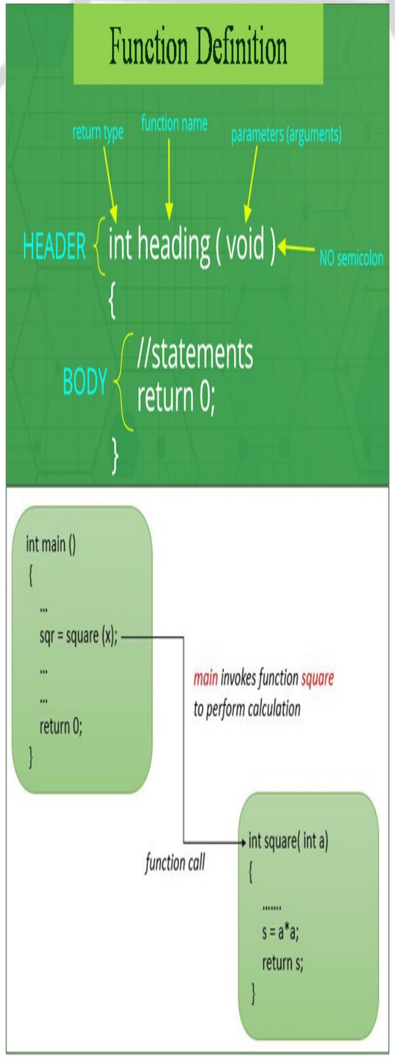
```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}

int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

1. Write the Function Header.
   - int calcSum(int x, int y)

2. Write a sample Function Call.
   - Declare and initialize (if needed) the variables used in the call and place them before the call.

C FUNCTION: FUNCTION CALL



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

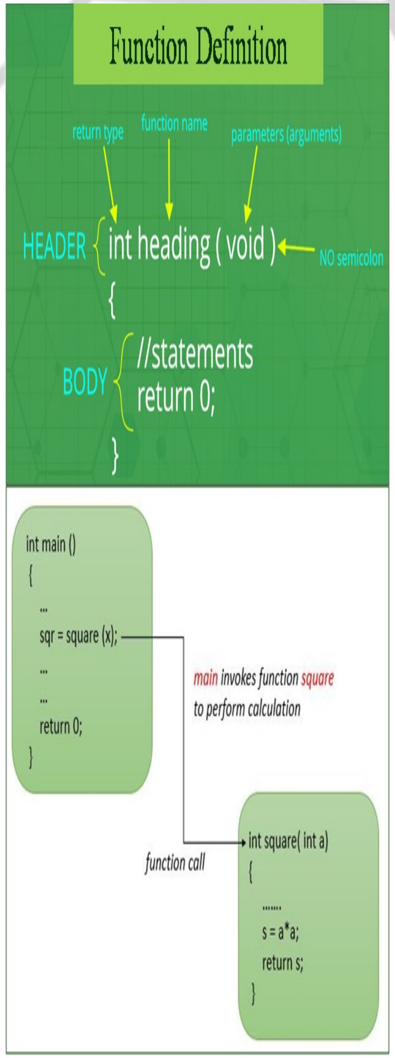
```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

Function Call

```
int  a, b;
int total;
a = 75;
b = 80;


total = calcSum(a,b);
```

C FUNCTION: STEPS IN FUNCTION CREATION



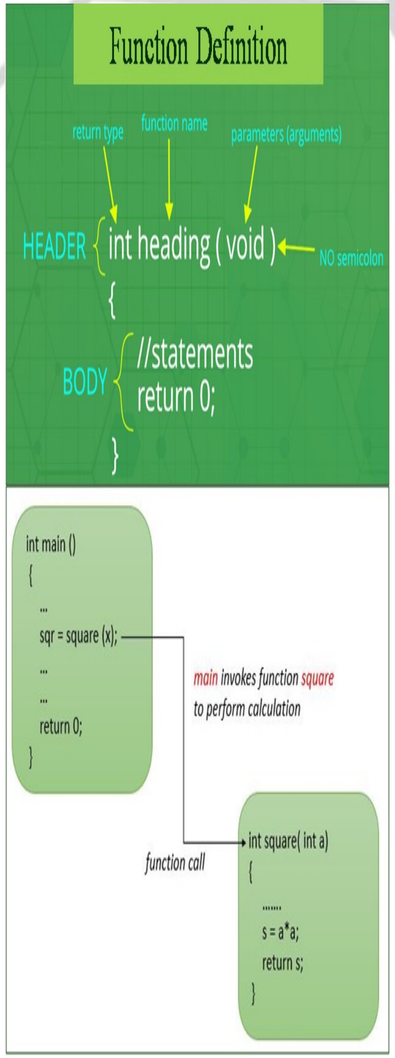
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

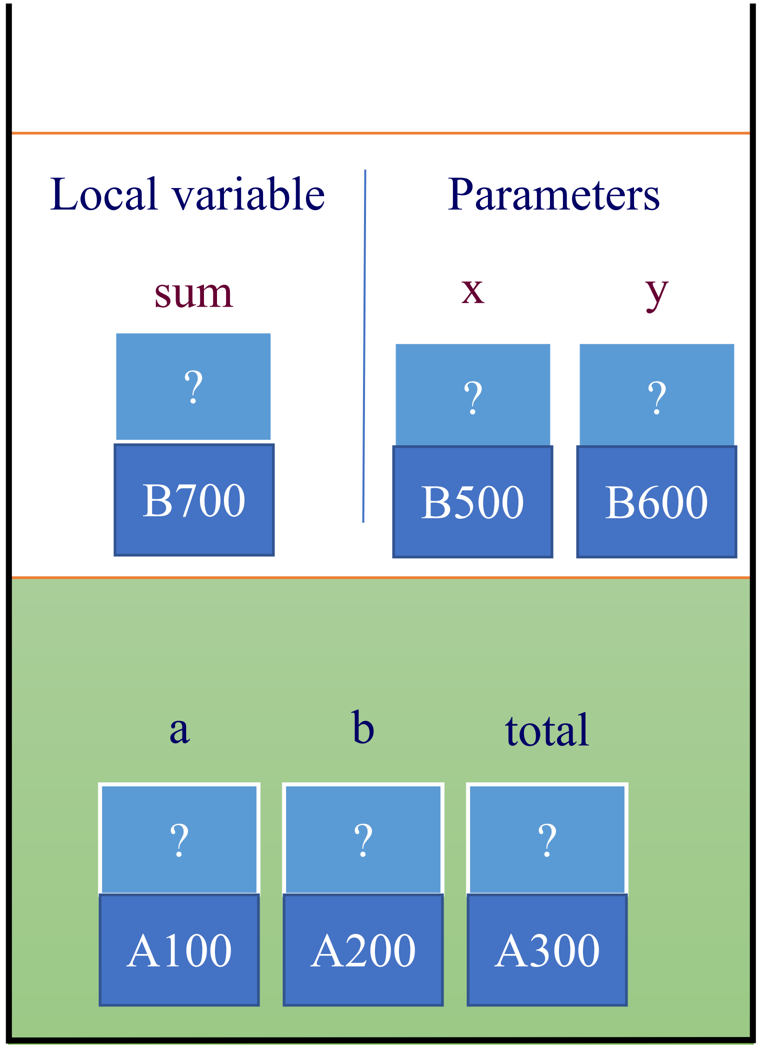
3. Assume that the function call is in **main()**.
   - Draw the execution stack.
   - Label the variables with names, values, and addresses.

C FUNCTION: EXECUTION STACK



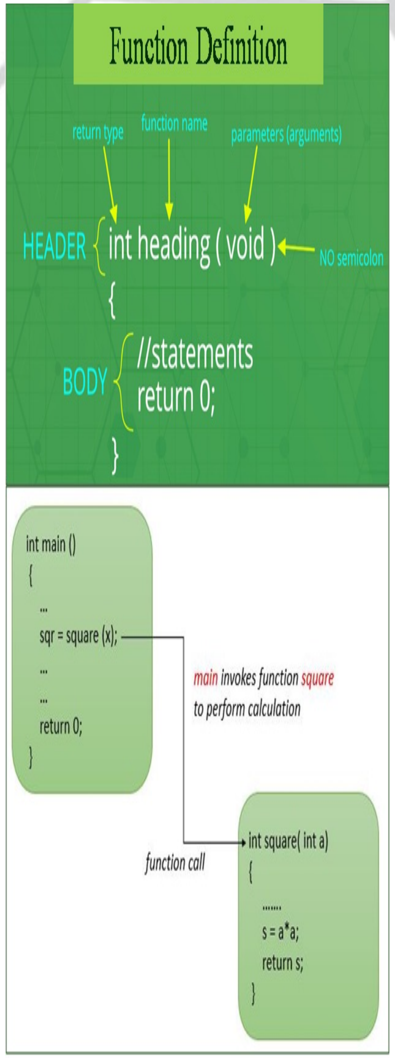
```
int main ()
{
   ...
   sqr = square (x);
   ...
   ...
   return 0;
}
```

```
int square( int a)
{
   .......
   s = a*a;
   return s;
}
```



- Activation Record of calcSum()
  - Pass-By-Copy (Call-By-Value)
- Activation Record of main()
  - total = calcSum (a, b);

C FUNCTION: STEPS IN FUNCTION CREATION



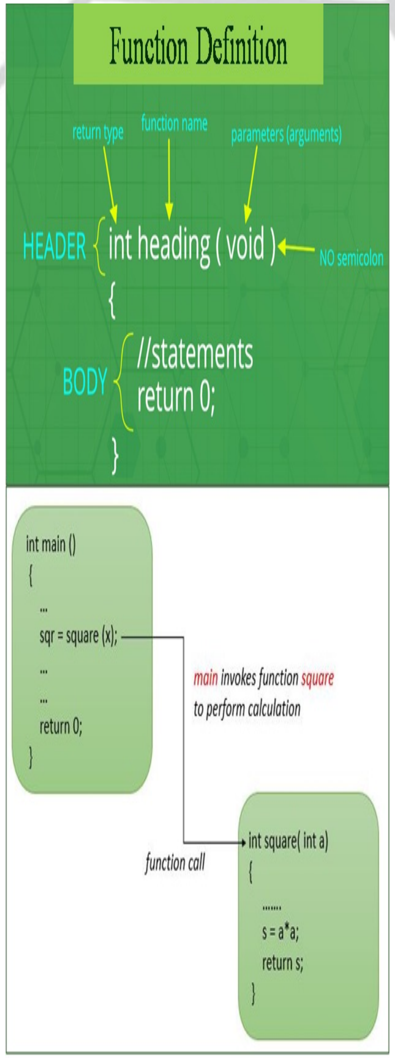
```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

4. Write the code of the function.
   - If the return type is not void, declare the local variable of the return type first then write the return statement.

BASIC FUNCTION DESIGNS



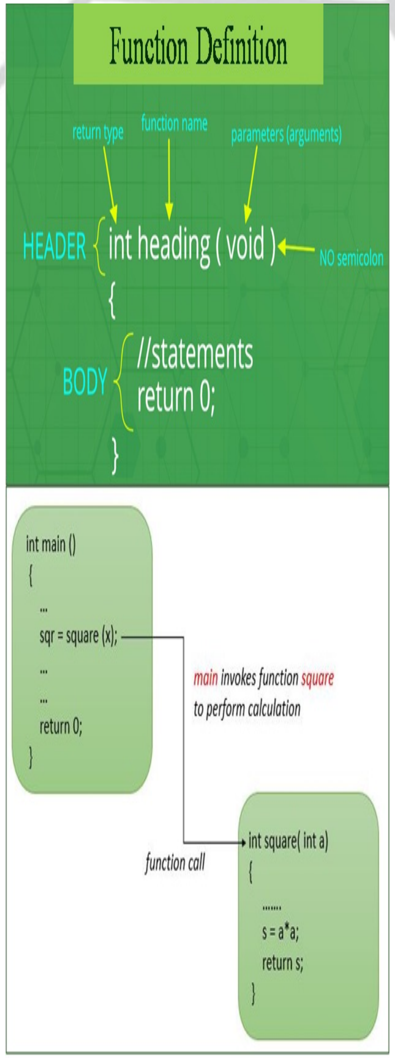
```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- **Void functions with no parameters:**
  - It is a function without parameters (or arguments) and without return value.
  - It is a function that receives nothing and returns nothing.

BASIC FUNCTION DESIGNS



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

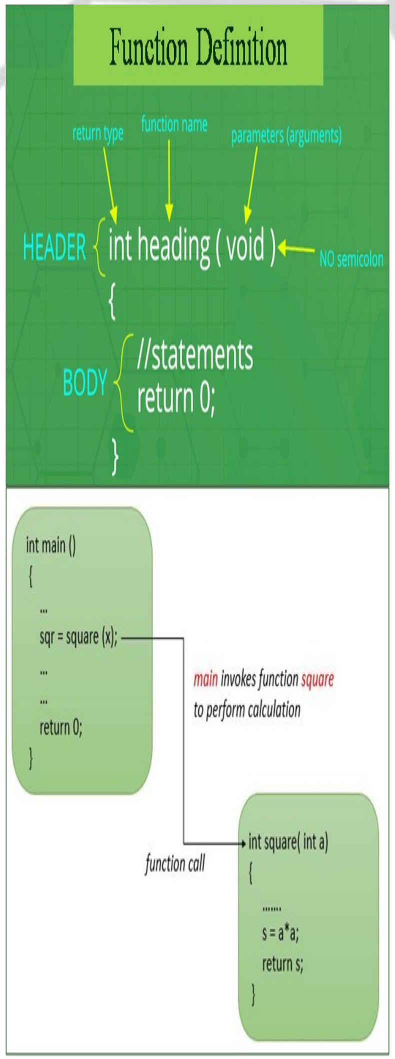
```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- **Void functions with no parameters:**
  - **function declaration:** void function();
  - **function call:** function();
  - **function definition:**

```
void function()
  {
    statements;
  }
```

  - **Example:** greeting program using function

BASIC FUNCTION DESIGNS



```
int heading ( void )
{
 //statements
 return 0;
}
```

```
int main ()
{
 ...
 sqr = square (x);
 ...
 ...
 return 0;
}
```

```
int square( int a)
{
 .......
 s = a*a;
 return s;
}
```

Void functions with no parameters:

It is a function without parameters (arguments) and without return value.

It is a function that receives nothing and returns nothing.

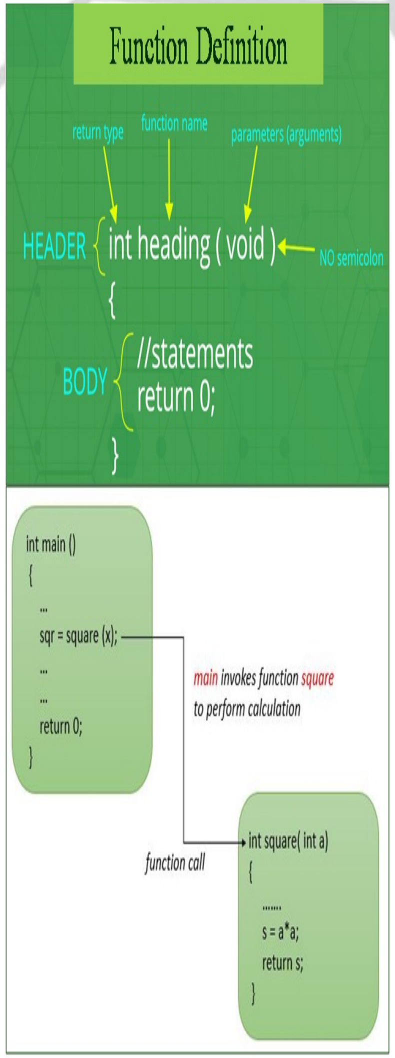
```
#include<stdio.h>

/* Function prototype or function declaration */
void greetMess(void);

/* Main function */
int main()
{
        greetMess();  /* Calling greeting function */
        return 0;
}

/* greeting function definition */
void greetMess(void)
{
        printf("\nHello, CIS1101 Studes! God bless us all!\n");
}
```

BASIC FUNCTION DESIGNS



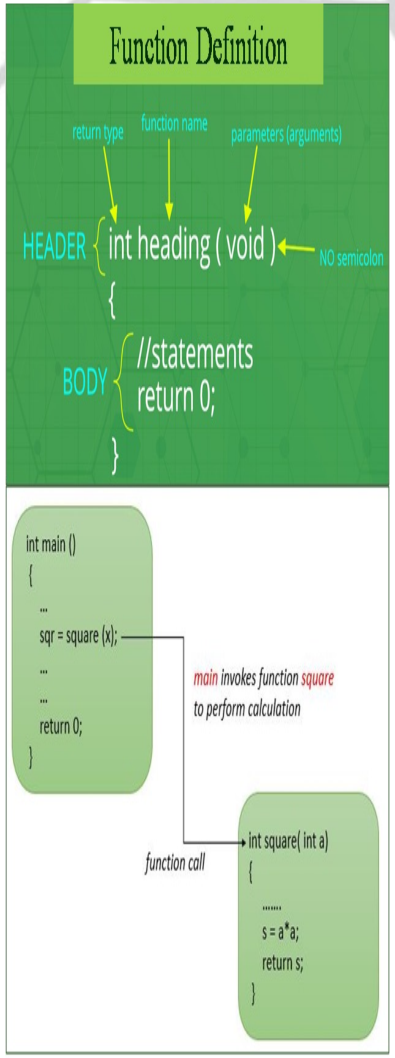
```
int heading ( void )
{
   //statements
   return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- **Non-void functions without parameters:**
  - It is a function that returns a value but have no parameters.
  - It is a function without parameters (or arguments) and with return value.

BASIC FUNCTION DESIGNS



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

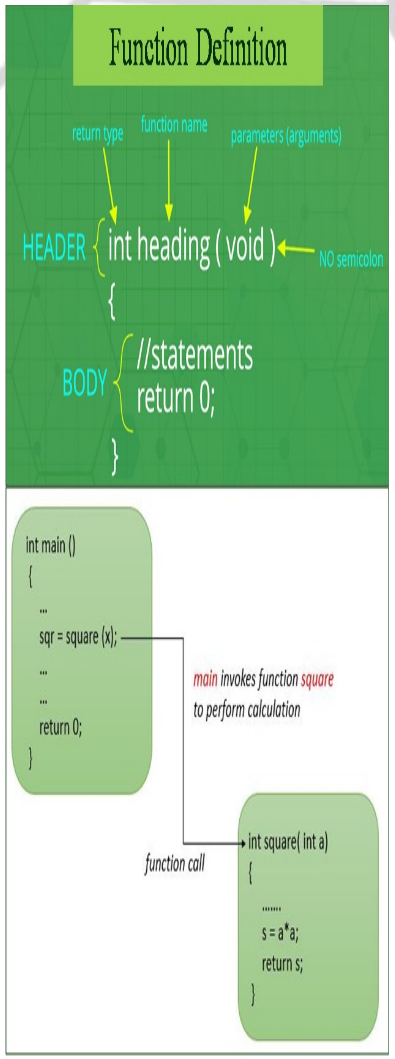
```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- **Non-void functions without parameters:**
  - **function declaration:** int function ( );
  - **function call:** function();
  - **function definition:**

```
int function( )
{
    statements;
    return a;
}
```

  - **Example:** getQuantity program using function

BASIC FUNCTION DESIGNS



```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}

int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

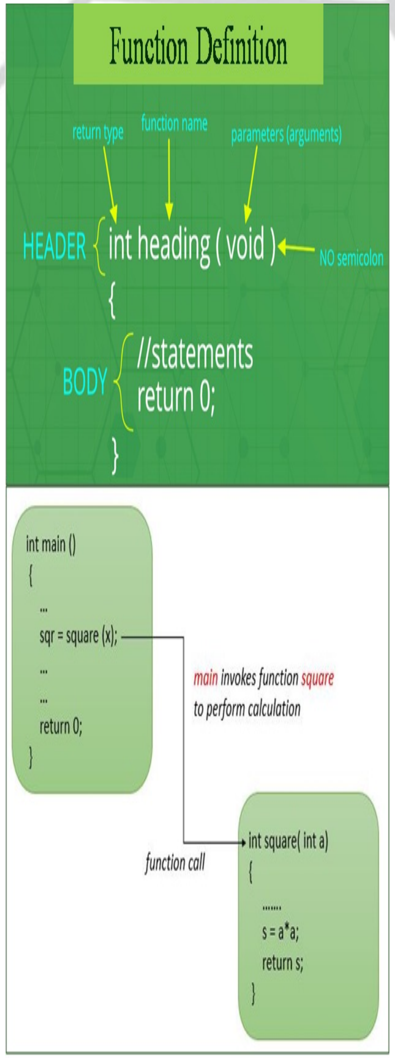
Non-void functions without parameters:

It is a function that returns a value but have no parameters.

It is a function without parameters (arguments) and with return value.

```
#include<stdio.h>
/* Function prototype or function declaration */
int getQuantity();
int main()
{
        int value, cost;
        value = getQuantity();  /* Calling getQuantity function */
        cost = value * 50;
        printf("\nThe amount payable is = %d.\n",cost);
        return 0;
}
/* Get quantity function definition */
int getQuantity()
{
        int qty;
        printf("Enter quantity: ");
        scanf("%d", &qty);
        return qty;
}
```

CALLING A C FUNCTION: CALL-BY-VALUE

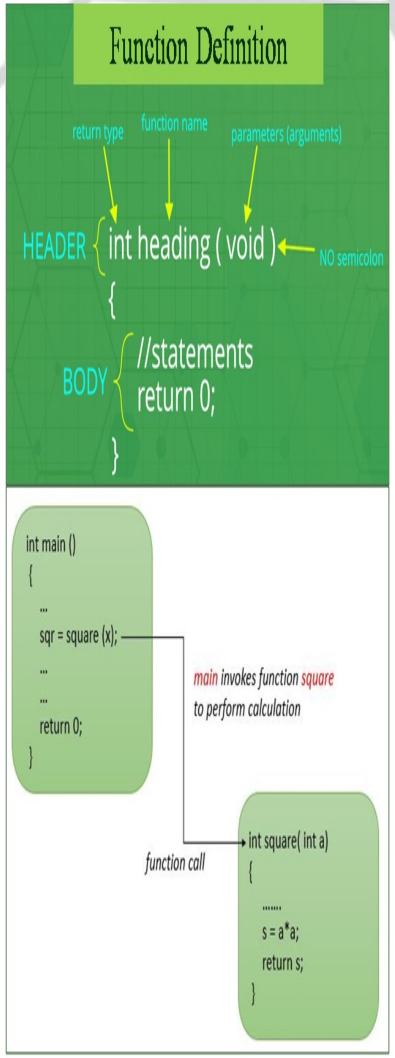


- The **value of the variable**
  - is **passed** to the function **as parameter**.

- The **value** of the
  - **actual parameter variable** is **passed (or copied)** into the **formal parameter variable**

- It is also known as
  - PASS-BY-VALUE or
  - PASS-BY-COPY

CALLING A C FUNCTION: CALL BY VALUE



```
int heading ( void )
{
    //statements
    return 0;
}
```

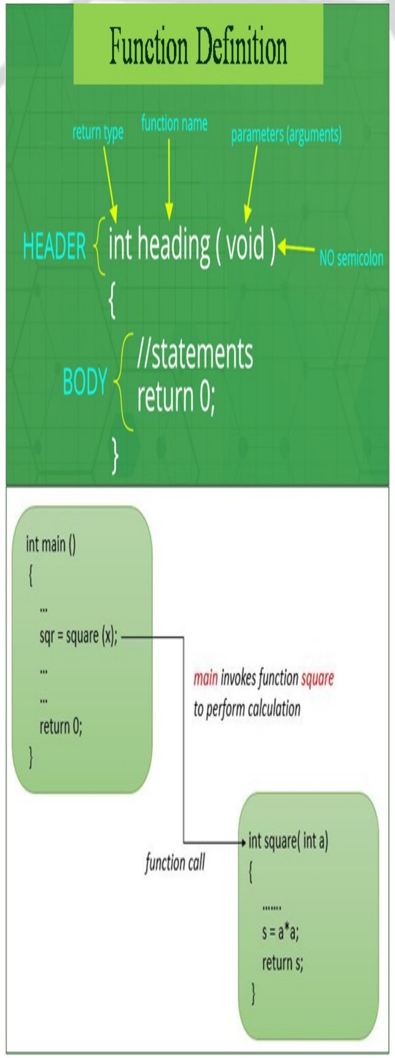
```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- The value of the actual parameter
  - **can not be modified** by formal parameter.

- Different Memory is allocated
  - for both actual and formal parameters because value of the actual parameter is copied to formal parameter.

CALLING A C FUNCTION: IMPORTANT NOTES



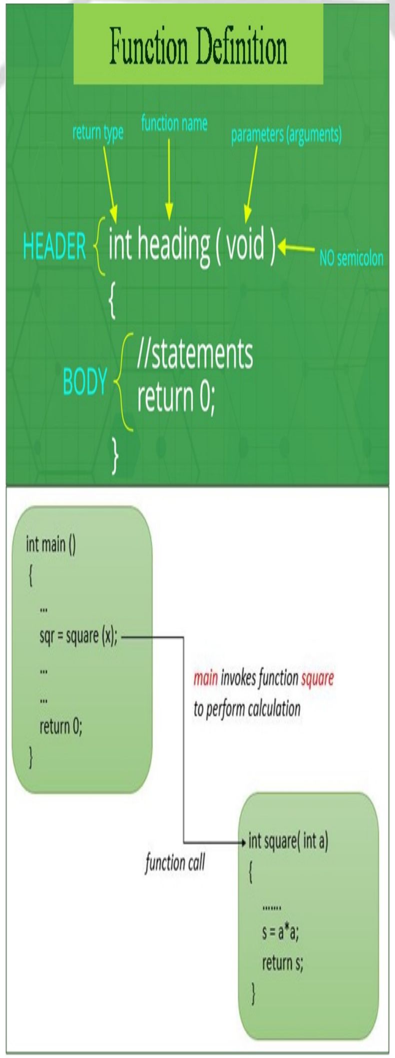
```
int heading ( void )
{
    //statements
    return 0;
}
```

```
int main ()
{
    ...
    sqr = square (x);
    ...
    ...
    return 0;
}
```

```
int square( int a)
{
    .......
    s = a*a;
    return s;
}
```

- **Actual parameter:**
  - The actual value that is passed into the function by a caller, and it is often called as argument.
- **Formal parameter:**
  - The identifier used in a function to stand for the value that is passed into the function by a caller.

CALLING A C FUNCTION: IMPORTANT NOTES



```
int heading ( void )
{
  //statements
  return 0;
}
```

```
int main ()
{
  ...
  sqr = square (x);
  ...
  ...
  return 0;
}
```

```
int square( int a)
{
  .......
  s = a*a;
  return s;
}
```

- A **parameter** cannot be both a formal and an actual parameter.

- **But** both formal parameters and actual parameters
  - **can be either value parameters or variable parameters.**