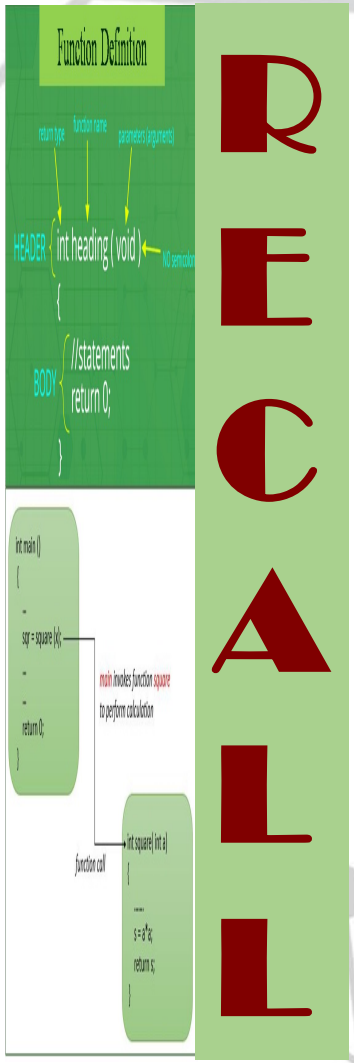
UNIVERSITY of SAN CARLOS
SCIENTIA • VIRTUS • DEVOTIO

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

CIS 1101 – PROGRAMMING 1

PASS BY COPY & PASS BY ADDRESS

C FUNCTION: C FUNCTION ASPECTS



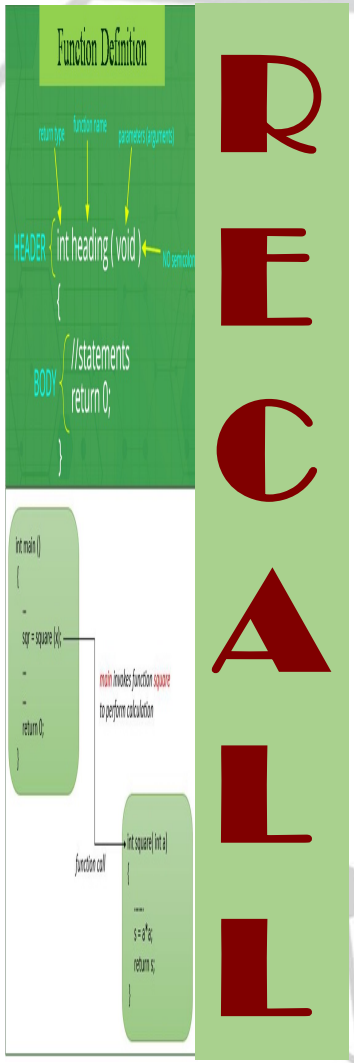
RECALL

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: STEPS IN FUNCTION CREATION

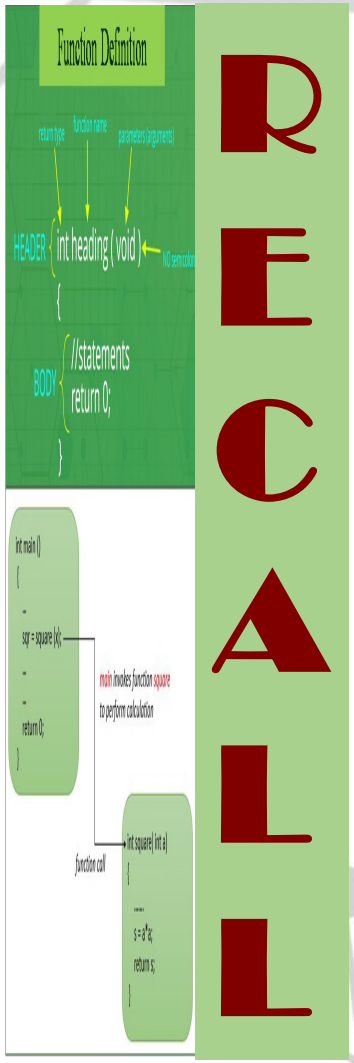


RECALL

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

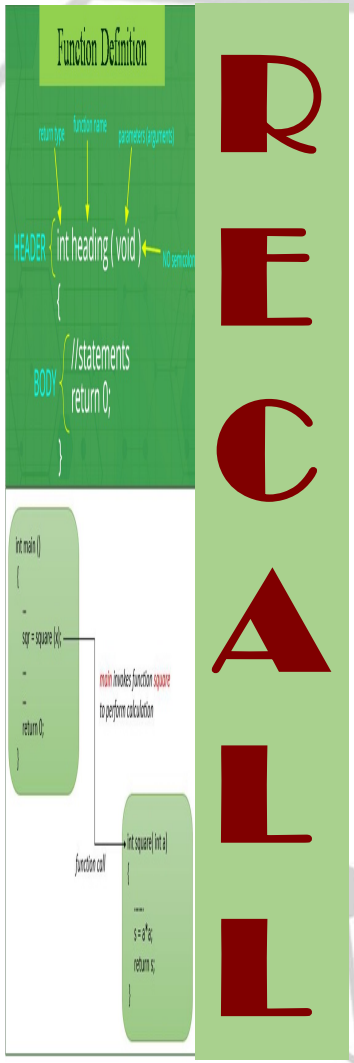


RECALL

Sample Function Call

```
int  a, b;
int total;
a = 75;
b = 86;

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

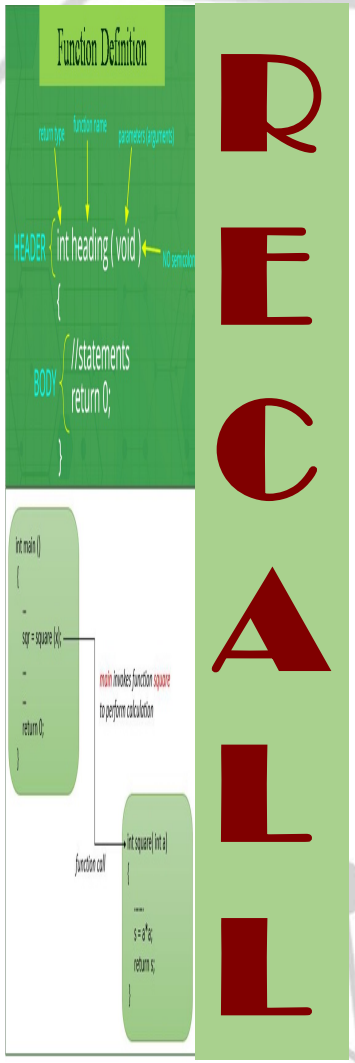
C FUNCTION: STEPS IN FUNCTION CREATION



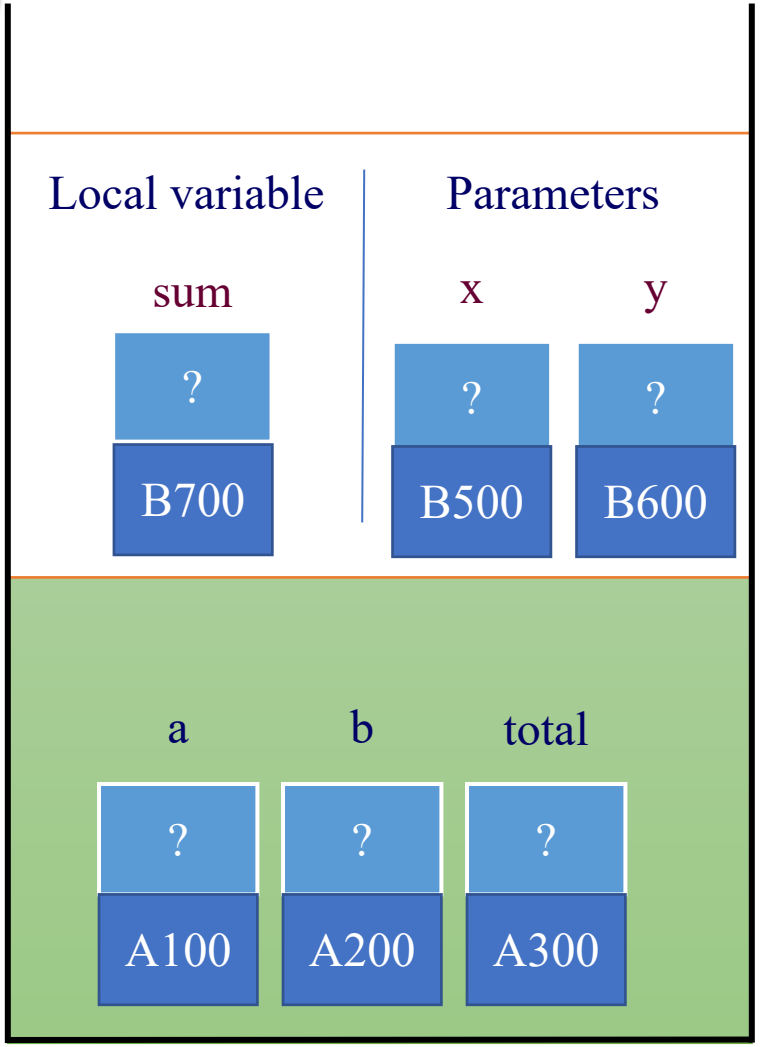
RECALL

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



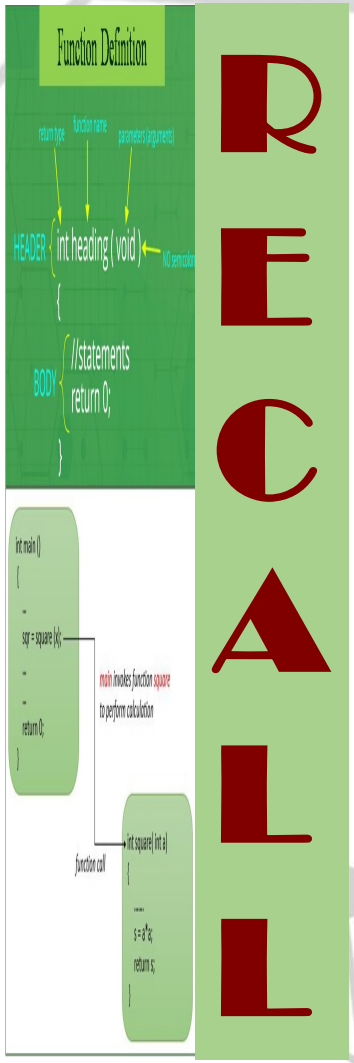
RECALL



- Activation Record of calcSum()
  - Pass-By-Copy (Call-By-Value)

- Activation Record of main()
  - total = calcSum (a, b);

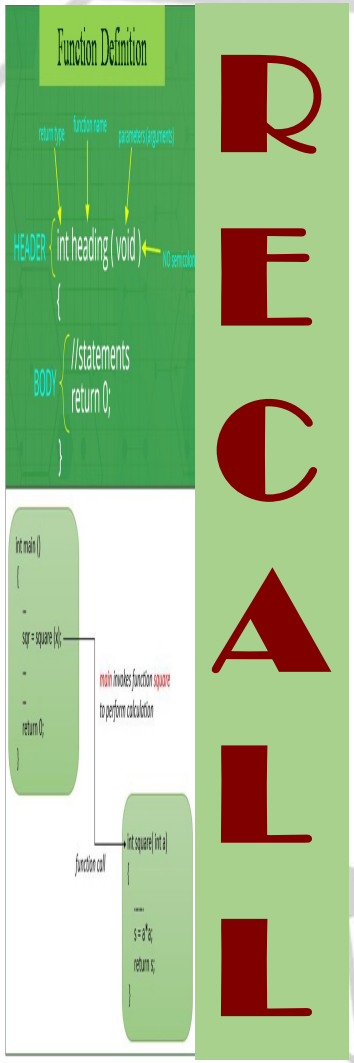
C FUNCTION: STEPS IN FUNCTION CREATION



RECALL

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.

CALLING A C FUNCTION: CALL-BY-VALUE/PASS-BY-VALUE



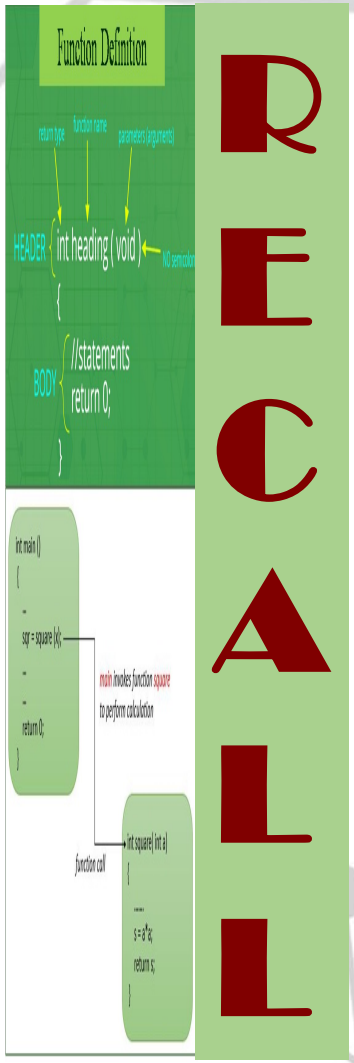
RECALL

- 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-Copy.

CALLING A C FUNCTION: CALL-BY-VALUE/PASS-BY-VALUE

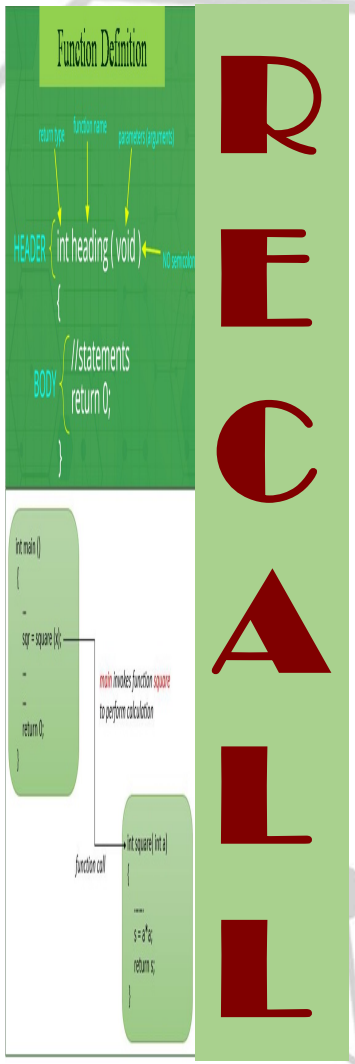


RECALL

- The value of the actual parameter
  - **cannot 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.

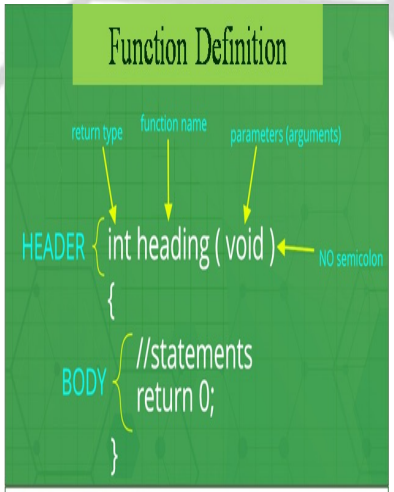
ACTUAL PARAMETER(S) VERSUS FORMAL PARAMETER(S)

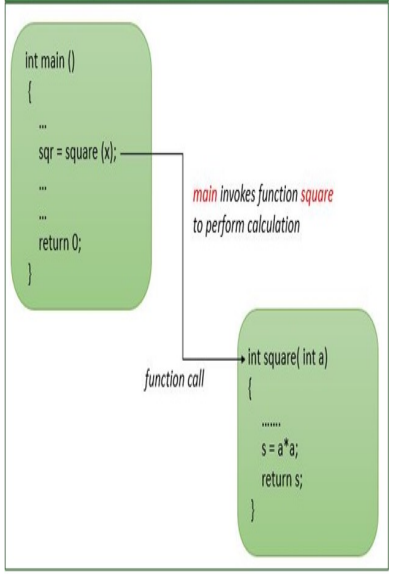


- **Actual parameter:**
  - The actual value that is passed into the function by a caller, and it is often called as argument.

- **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: CALL-BY-REFERENCE



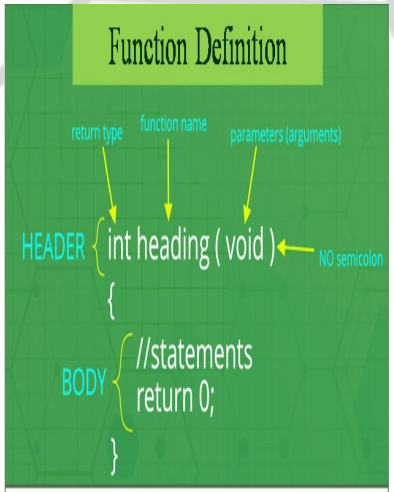


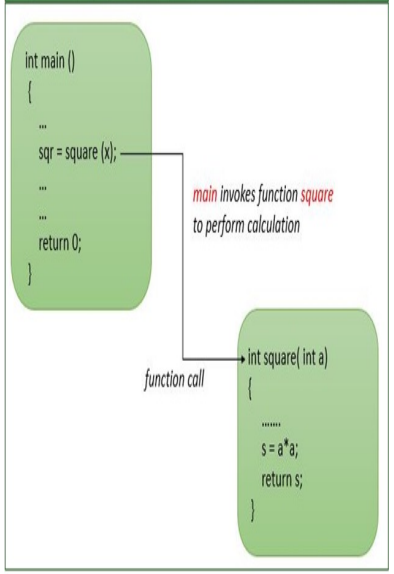
- The **address of the variable**
  - is **passed to the function as parameter**.

- The value of the actual parameter
  - can be modified by formal parameter.

- Same memory is used
  - for both actual and formal parameters since only address is used by both parameters.

CALLING A C FUNCTION: CALL-BY-REFERENCE





- The *location* **(reference or address)**
  - of the **actual parameter variable** is ***passed*** **(or copied)** into the **formal parameter variable.**

- It is also known as Pass-By-Reference.

CALLING A FUNCTION: SAMPLE

Given:

A program uses two variables **d** and **e** and it assigns **d** with a value of **30** and **e** with **50**. Then it will swap those values.

Tasks:

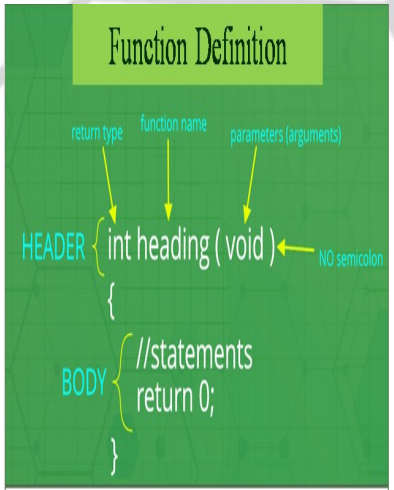
a) Create a function that will swap the values assigned to the given two variables **d** and **e** using **Pass-By-Value**.

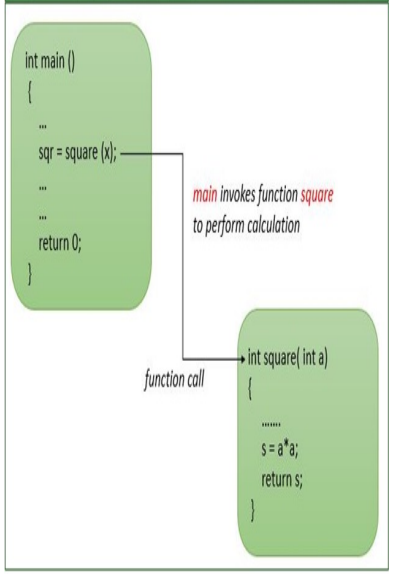
b) Write the code of the program described above using the created function and verify its output.

c) Create a function using pointers that will swap the values assigned to the given two variables **d** and **e** using **Pass-By-Reference**.

d) Write the code of the program described above using the created function and verify its output.

CALLING A FUNCTION: SAMPLE





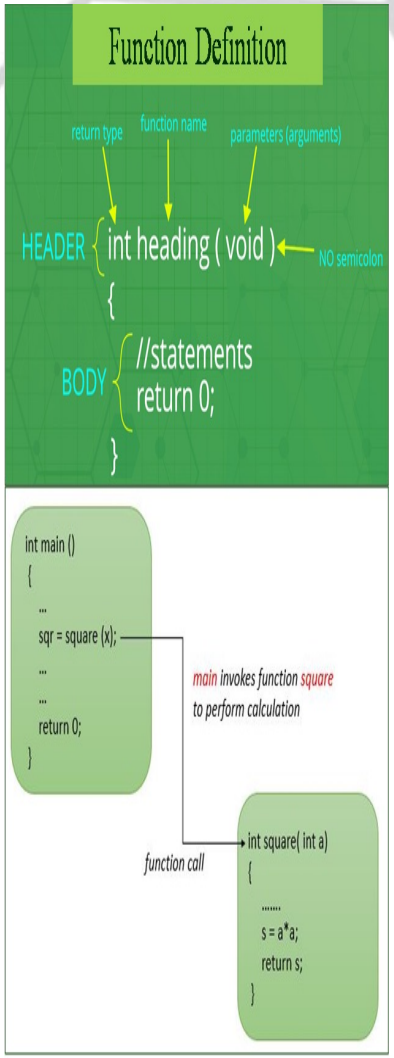

```c
#include<stdio.h>

/* function prototype, also called function declaration */
void swap(int, int);

int main()
{
   int d = 30, e = 50;
   /* Calling swap function by value */
   printf("\nThe values before swapping: \nd = %d \ne = %d\n", d, e);
   swap(d,e);
  /* Values will stay the same after calling the swap() function */
   printf("\nThe values after swapping: \nd= %d \ne = %d\n", d, e);
   return 0;
}
/* Swap function definition */
void swap(int a, int b)
{
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

C FUNCTION PROGRAM (USING PASS-BY-VALUE):

In this program, the values of the variables **"d"** and **"e"** are **passed to the function "swap"**.

These **values** are **copied to formal parameters "a"** and **"b"** in **swap function** and used, but the thing is, **this wont actually swap the actual parameters**. The swapping will only happen in the **swap function**.

CALLING A FUNCTION: SAMPLE




```c
#include<stdio.h>
/* Function prototype, also called function declaration */
void swap(int*, int*);

int main()
{
   int d = 30, e = 50;
   /* Calling swap function by reference */
   printf("The values before swapping: \nd = %d \ne = %d\n", d, e);
   swap(&d, &e);
   printf("\nThe values after swapping: \nd = %d \ne = %d\n", d, e);
   return 0;
}

void swap(int *a, int *b)
{
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
}
```

C FUNCTION PROGRAM (USING PASS- BY-REFERENCE):

In this program, the **address of the variables "d" and "e"** are **passed to the function "swap"**.

These **values are not copied to formal parameters "a" and "b" in swap function** because **they are just holding the address of those variables**.

This **address is used to access and change the values of the variables**.