Part I  Functions - Summary

• C Program
  - A C program consists of one or more functions
  - All C programs MUST have a main() function
  - A function in C performs a particular task
    - e.g. print info on the screen, compute, etc
  - Execution of the program begins at the first statement of main() function
  - main() function usually invokes (call) other functions to perform its job. Some functions are defined in the same program, others are provided by libraries
    - e.g. \( \text{sin(\text{angle_param})} \) function is provided by \text{math.h} library

• C Program Structure

```c
my_program.c

// Include libraries

// Function definitions

int main ()
{
    ...
}

OR

my_program.c

// Include libraries

// Function declarations

int main ()
{
    ...
}

// Function definitions
```
- **Include libraries**
  - C standard libraries:  `
``
#include <stdio.h>
```
  - User defined libraries:  `
```
#include "age.h"
```
  - Programmer has to create for each own library
    - *header file* (e.g. age.h)
    - *library implementation file* (e.g. age.c)

- **Function Definitions**

```c
return_type function_name (type name_param1,
                          type name_param2,...)
{
    // function_body
}
```

**Example 1:**

```c
int sum (int no1, int no2)
{
    int res;

    res = no1 + no2;
    return res;
}
```

**Example 2:**

```c
void print_hello (void)
{
    printf (" hello! \n")
    return;
}
```
Example 3:
```c
void print_array (int sIDs[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d 
, sIDs[i]);
}
```

- **Function Declarations**

```
return_type function_name(type name_param1,
                             type pame_param2,...);
```

Example 1:
```
int sum (int no1, int no2);
```

Example 2:
```
void print_hello (void);
```

Example 3:
```
void print_array (int sIDs[], int size);
```

- **Using the Functions (Function Call)**
  - A function can be called (invoked) from the `main()` function or within the body of another function
  - Call a function that receives a list of parameters and returns a data

```
variable_name = function_name (param1, param2,...);
```
Example 1:
int val;
val = sum(2, 3);

Example 2:
int val, x, y;
x = 100;
y = 250;
val = sum(x, y);

• Call a function that receives a list of parameters and DOES NOT return any data

function_name (param1, param2,...);

Example:
int students[4] = { 1200, 1766, 120, 30};
print_array (students, 4);

• Call a function that DOES NOT receive a list of parameters and DOES NOT return any data

function_name ();

Example:
#include <stdlib.h>
#include <stdio.h>
.
.
int main ()
{
    print_hello();
    return (EXIT_SUCCESS);
}
**Formal and actual function parameters**

Example:

```c
/* include libraries */
#include <stdlib.h>
#include <stdio.h>

/* function definitions */
int sum (int no1, int no2)
{
    int res;
    res = no1 + no2;
    return res;
}

/* main function */
int main ()
{
    int val;
    val = sum(2, 3);
    printf ("sum = %d", val);
    return (EXIT_SUCCESS);
}
```

Actual Parameter is temporarily bound to the Formal Parameter ONLY when the function is called.