VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102L- Structured and object-oriented programming

Module 2

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

BCSE102P- Structured and object-oriented programming Laboratory

Indicative Experiments	
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

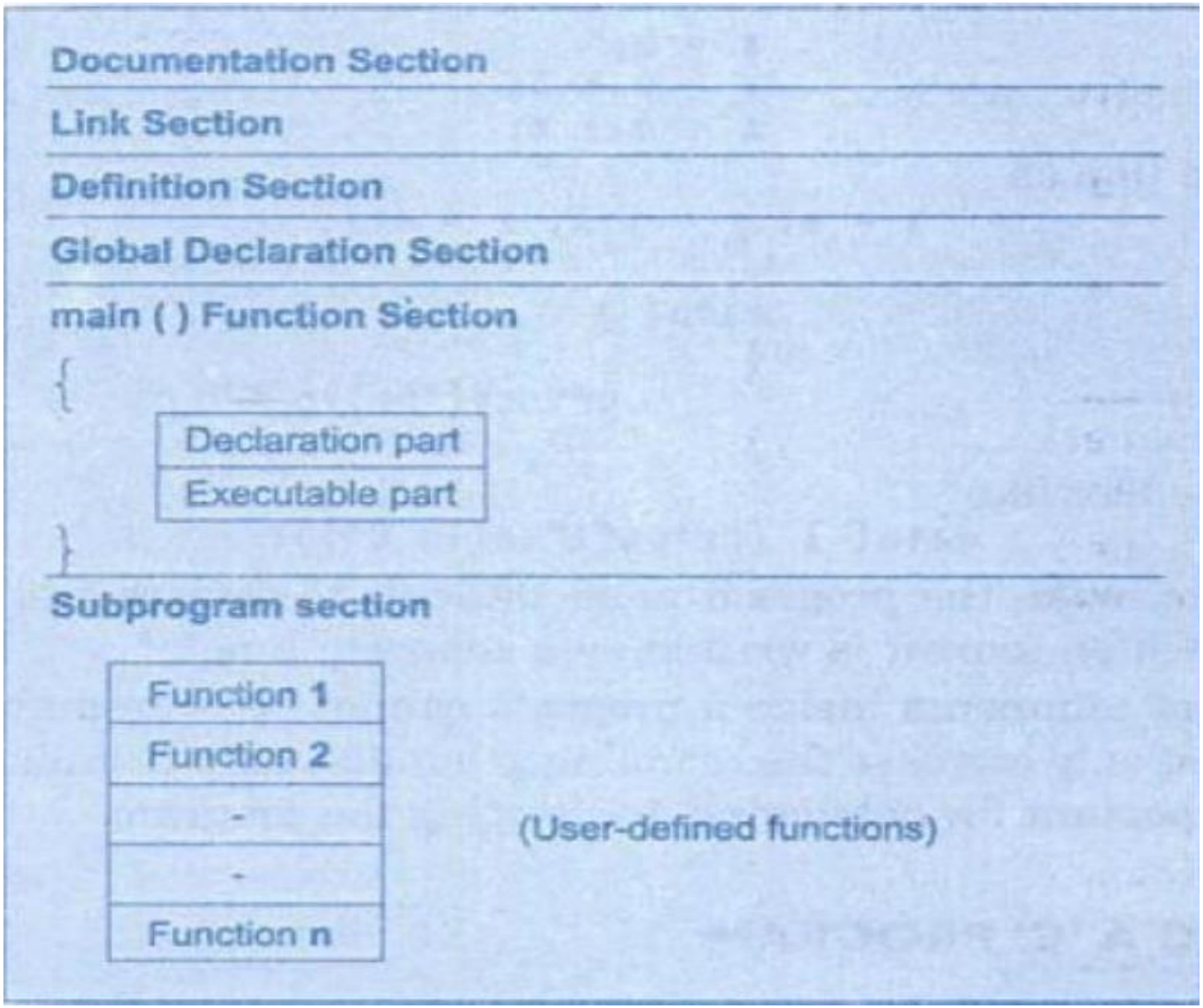
BCSE102L- Structured and Object-Oriented Programming

- **Module-2: ARRAYS AND FUNCTIONS**
  - **Arrays – 1D & 2D**
  - **Strings and its operations**
  - **User defined Functions**
    - **Declaration and Definition**
    - **Call by Value and Call by reference**
    - **Types of Functions- Recursive Functions**
  - **Storage Classes**
    - **Scope, Visibility and lifetime of variables**

USER DEFINED FUNCTIONS

- A **function** is a block of code that performs a specific and well defined task.
- Self-contained program segment that is placed separately from the main program to perform some specific well defined task is called **function**.
- Dividing complex problem into small components makes program easy to understand and use.
- The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.
- **Functions are the building blocks of a C program.**
- Two types: Library and User-defined functions

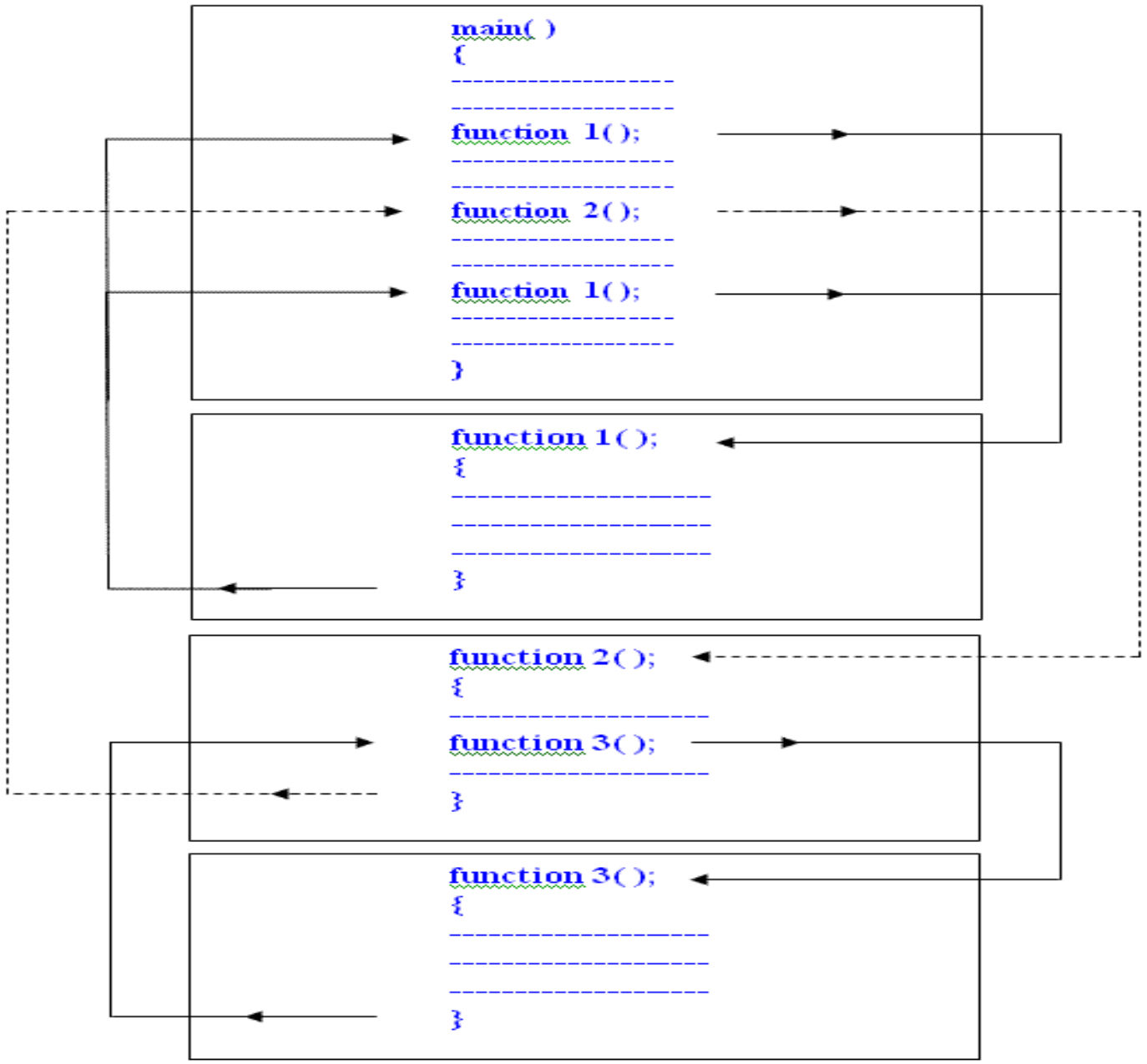
STRUCTURE OF C PROGRAM



About Function & its Advantages

- Every function has a unique name. This name is used to call function from **"main()"** or other function.
- A function can be called from within another function.
- A function is independent and it can perform its task without intervention from or interfering with other parts of the program.
- It reduces the length of source program.
- Breaks the complexity of a program.
- It is easy to maintain, modify and understand.
- A program can be divided into smaller subprograms.
- It facilitates modular programming.
- Saves time and space.
- Avoid rewriting the same sequence of code at two or more locations in a program.

Flow of Control



Agenda

- How a function is designed?
- How a function is integrated into a program?
- How two or more functions are put together?
- How they communicate with one another?

Categories of Functions

- **User defined functions-** Designed and developed by the user while writing and compiling a C program. (**Example: Main function).**
- **Library functions are also known as Built-in functions.**
  - Input/Output functions.
  - Boolean functions.
  - Conversion functions.
  - Memory functions.
  - String functions.
  - Miscellaneous functions.
  - Math or Arithmetic functions and
  - Time functions.

Elements of User Defined Functions

- **Function definition (Function implementation)**
  - Independent program module that is specially written to implement the requirements of the function. In order to use this we need to invoke it at a required place in the program
- **Function call**
  - Program that calls the function is referred to as the calling program or calling function
- **Function declaration**
  - Calling program should declare any function that is to be used in the program

Terminologies in Functions

- Function name
- Function type
- List of parameters

Function Header

- Local variable declarations
- Function statements
- Return statement

Function Body

General Format of Function definition

Function Header

```
function-type function-name(parameter list)
{
        local variable declaration;
        executable statement1;
        executable statement2;
        ……….
        return statement
}
```

Function Body

Function Header

Function Type

- Specifies the type of the value (like int, float or double) that the function is expected to return to the program calling the function.
- If the return type is not explicitly specified, C will assume that it is an integer.
- If the function is not returning anything, then we need to specify the return type as void.

Function Name

- Any valid C identifier that must follow the same rules of formation as other variable names in C.

Formal Parameter List

- Declares the variables that will receive the data sent by the calling program

Examples

```
int circle(){------}                          // No argument

int area(int p,int r){------}                 //Two argument

int rectangle(int h, int b, int w){---}       //Three argument.

float quadratic( int a, int b, int c){---}    //Three argument
```

Function Body

Local Declaration

- It is used to specify the variables necessary for a function. Variables declared inside the main function is called as Local variables and Variables declared outside the main function is called Global variables.

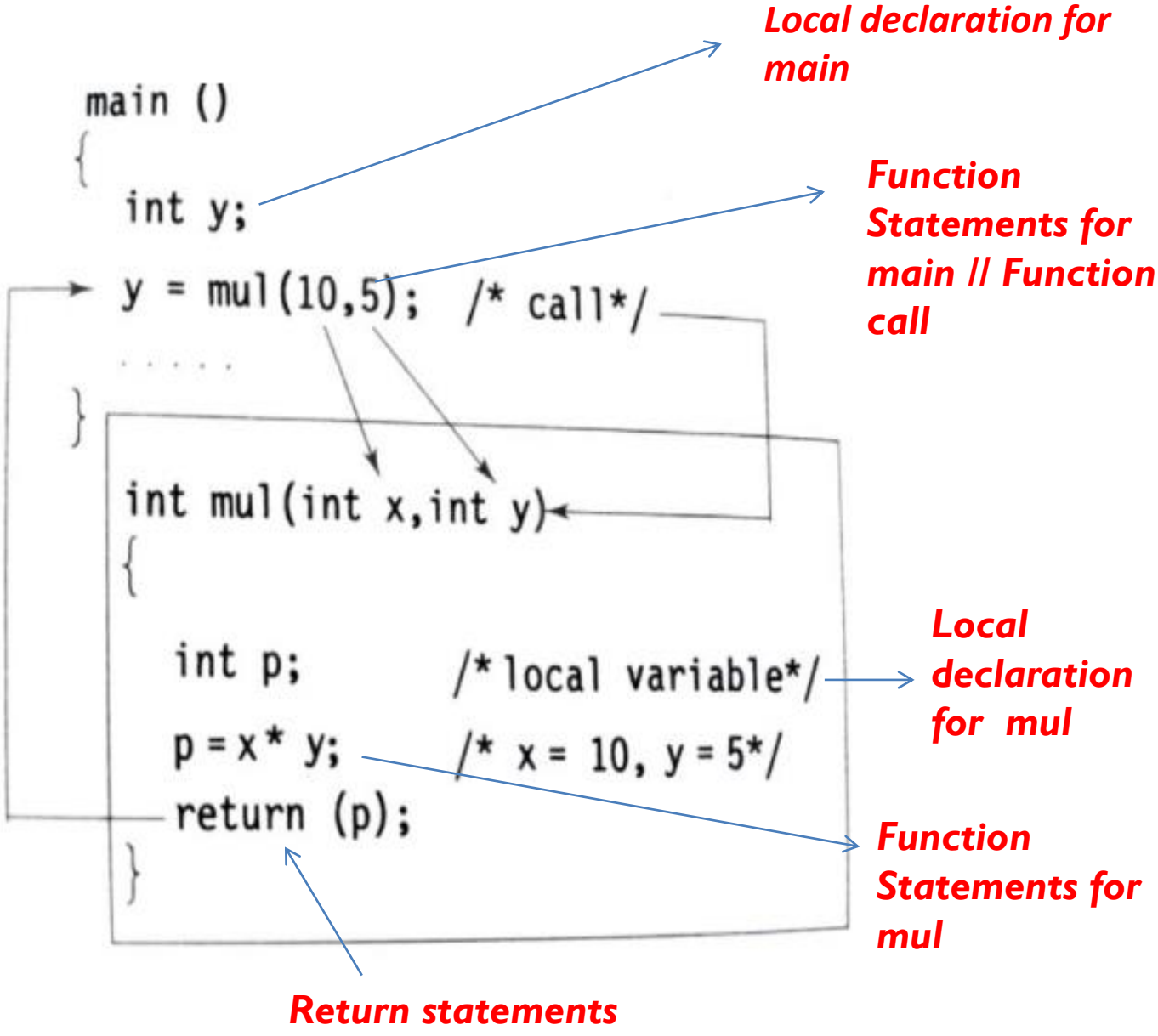
Function Statement

- Used to perform task for the function

Return Statement

- 'Return' statement is used to return the processed value to the main function.
- If a function does not return any value to the calling function, we can omit the return statement, then its return type is 'void'.
- If a function does not receive any value from the sub function then there is no need of 'return' statement.

Function call - Example



Return Values

- A function may or may not return send back any value to the calling function
- It return one value per call
- All functions by default return **int** types

Syntax:

```
return; // does not return anything

return (expression);
```

Example:

```
return(p);

return(x*y);

return(2.5 * 3.0); // return the value 7
```

Function Declaration

- Like variables, all functions in a C program must be declared, before they are invoked. It was also known as Function prototype. It consists of four parts.
  - **Function type (return type)**
  - **Function name**
  - **Parameter list**
  - **Terminating semicolon**

Examples
double sqroot(int num);
void prod(int a, int b);
int mul(int m, int n);

- **General form:**

  datatype function name (argument list);

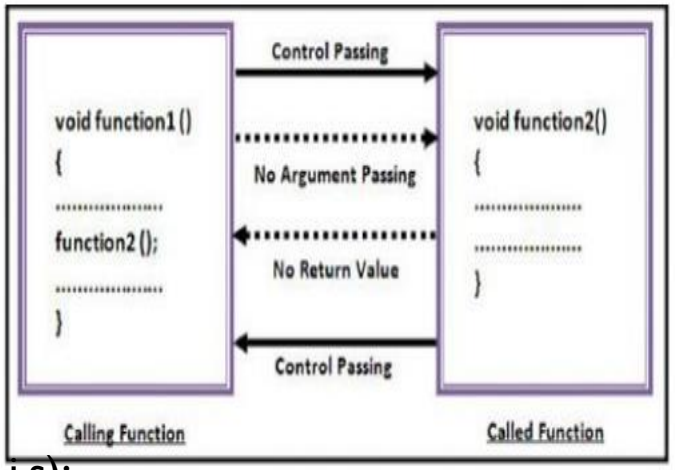
This is very similar to the function header line except the terminating semicolon.

Types of Functions

- Functions with no arguments and no return values.
- Functions with no arguments and return values.
- Functions with arguments and no return values.
- Functions with arguments and return values.

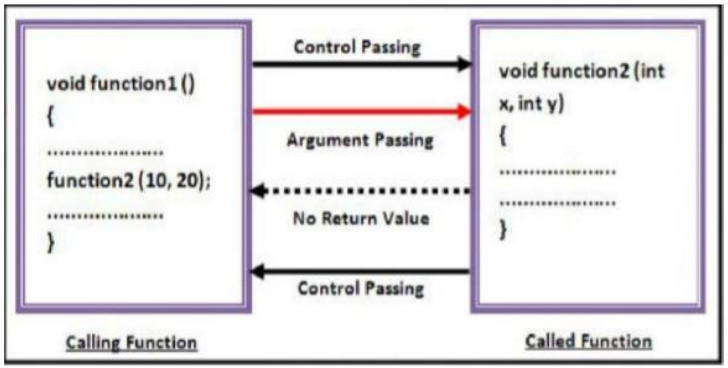
Functions with no arguments and no return values

```
int main()              // main function
{
        int i, j,x=1;           //  local variable declaration
        int square();                   // Function declaration
        printf("\n Enter a number :"); // output the text
        scanf("%d",&i);                 // read number i
        for(j=1;j<=i;j++)
                x=j*x;
        printf("\nThe Factorial of %ld is %ld",i,x);
        square();                       // Function call
}
int square() // Function definition
{
        int i,n,s,sum=0;   // local variable
        printf("Enter the range:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                s=i*i;
                sum+=s;
                printf("Square of %d is %d",i,s);
        }
printf("\n Sum of squares is %d",sum);
}
```



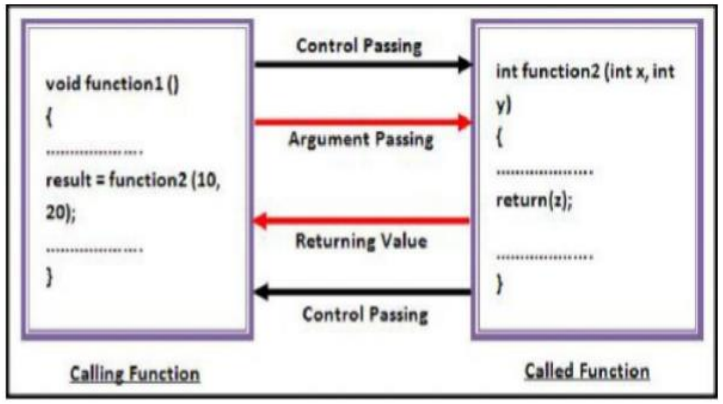
Functions with arguments and no return values

```
int main()              // main function
{
        int i, j,x=1, n=10;          //  local variable declaration
        int square();                // Function declaration
        printf("\n Enter a number :"); // output the text
        scanf("%d",&i);              // read number i
        for(j=1;j<=i;j++)
                x=j*x;
        printf("\nThe Factorial of %ld is %ld",i,x);

        square(n);                   // Function call
}
int square(n) // Function definition
{
        int i,s,sum=0;   // local variable
        for(i=1;i<=n;i++)
        {
                s=i*i;
                sum+=s;
                printf("Square of %d is %d",i,s);
        }
printf("\n Sum of squares is %d",sum);
}
```



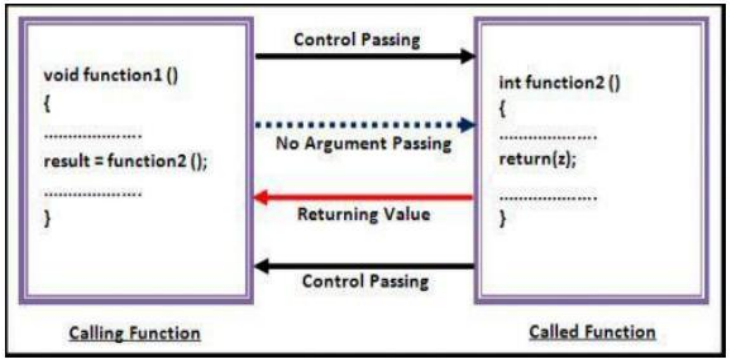
Functions with arguments and with return values

```
int main()              // main function
{
        int i, j, x=1, n=10, z;          //  local variable declaration
        int square();                    // Function declaration
        printf("\n Enter a number :");   // output the text
        scanf("%ld",&i);                 // read number i
        for(j=1;j<=i;j++)
                x=j*x;
        printf("\nThe Factorial of %ld is %ld",i,x);

        z=square(n);        // Function call
printf("\n Sum of squares is %d",z);
}
int square(n) // Function definition
{
        int i,s,sum=0;   // local variable
        for(i=1;i<=n;i++)
        {
                s=i*i;
                sum+=s;
                printf("Square of %d is %d",i,s);
        }
return sum;
}
```



Functions with no arguments and with return values

```
int main()                  // main function
{
        int i, j, x=1, z;                   //  local variable declaration
        int square();                       // Function declaration
        printf("\n Enter a number :");      // output the text
        scanf("%ld",&i);                    // read number i
        for(j=1;j<=i;j++)
                x=j*x;
        printf("\nThe Factorial of %ld is %ld",i,x);
        z=square();          // Function call
printf("\n Sum of squares is %d",z);
}
int square() // Function definition
{
        int i,s,n,sum=0; // local variable
        printf("Enter the range:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                s=i*i;
                sum+=s;
                printf("Square of %d is %d",i,s);
        }
        return sum;
}
```



Parameter Passing to Functions

The parameters passed to function are called **actual parameters.**

The parameters received by function are called **formal parameters.**

- **Pass/Call by Value:** In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.
- **Pass/Call by Reference:** Both actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

Pass/Call by Value

```
int main()
{
        int m = 6, n = 10;
        printf("Before swapping value of m and n is :%d,%d\n",m,n);
        printf("Before swapping address of m and n is :%u,%u\n",&m,&n);
        swap(m,n);
        printf("After swapping m,n value is : %d,%d\n",m,n);
        printf("After swapping (m,n) address is :%u,%u\n",&m,&n);
        return;
}
int swap(int x, int y)
{
        int tmp = x;
        x = y;
        y = tmp;
        printf("Swapped values (x,y) is : %d,%d\n",x,y);
        printf("Swapped values address (x,y) is : %u,%u\n",&x,&y);
}
```

```
Before swapping value of m and n is :6,10
Before swapping address of m and n is :6422044,6422040
Swapped values (x,y) is : 10,6
Swapped values address (x,y) is : 6422000,6422008
After swapping m,n value is : 6,10
After swapping (m,n) address is :6422044,6422040
```

Pass/Call by Reference

```
void swap(int *x, int *y);
void main ()
{
    int a = 100, b = 200;
    printf("Before swap, value of a : %d\n", a );// Value of a printed
    printf("Before swap, address of a : %u\n", &a ); // Address of a printed
    printf("Before swap, value of b : %d\n", b );// // Value of b printed
    printf("Before swap, address of b : %u\n", &b );// Value of b printed
    swap(&a, &b);
    printf("After swap, value of a : %d\n", a );  //Value of a printed
    printf("After swap, value of b : %d\n", b );// Value of b printed
}
Void int swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    printf("After swap, value of x : %d\n", x );
    printf("After swap, value of y : %d\n", y );
}
```

```
Before swap, value of a : 100
Before swap, address of a : 6422044
Before swap, value of b : 200
Before swap, address of b : 6422040
After swap, value of x : 6422044
After swap, value of y : 6422040
After swap, value of a : 200
After swap, value of b : 100
```

Recursion

- When a function is called repeatedly until specific condition is satisfied, then it is known as Recursion.
- A function is called recursive if a statement within the body of a function calls the same function.

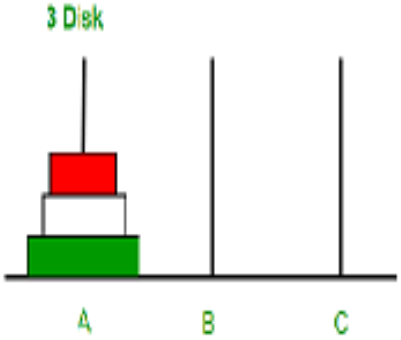
Simple Example

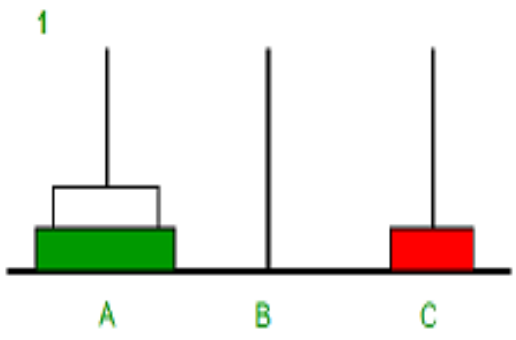
```
void main()
{
        printf("Welcome to BCSE102L & BCSE102P");
        main();
}
```

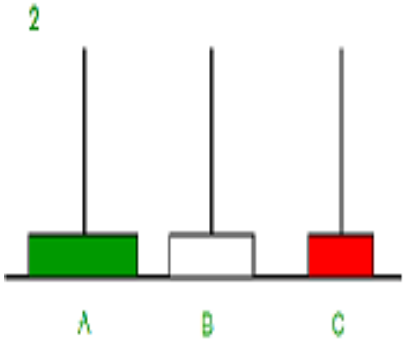
Recursion- Example

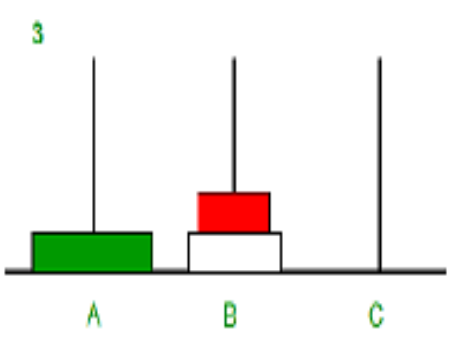
```
long fact(long); // Function Declaration
void main()
{
        long num, fac;
        printf("Enter a number :");
        scanf("%ld",&num);
        fac=fact(num);     // Function calling with arguments
        printf("The factorial of %ld is %ld",num,fac);
}

long fact(long x) // Function Definition
{
        if(x<=1)  // base case condition checking
        return(1);
else
        return(x*fact(x-1)); // Recursive Call
}
```

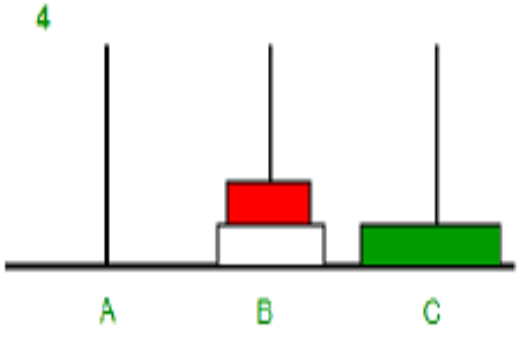
Recursion- Towers of hanoi

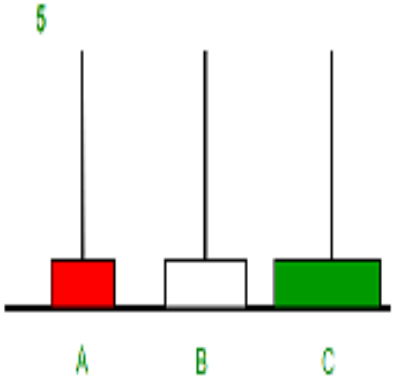


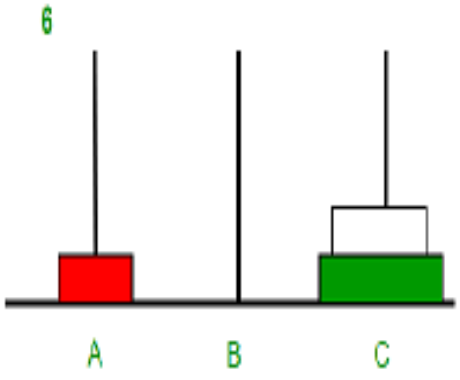


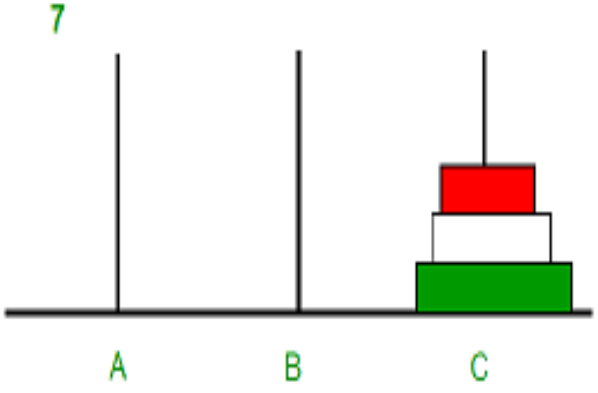








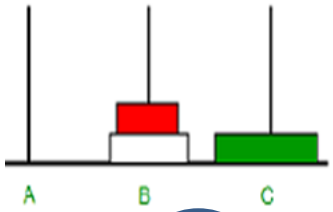





```
void towers(int, char, char, char);
 int main()
 {
        int num;
        printf("Enter the number of disks : ");
        scanf("%d", &num);
        printf("The sequence of moves involved in the Tower of
        Hanoi are :\n");
        towers(num, 'A', 'C', 'B');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
        if (num == 1)
                printf("\n Move disk 1 from peg %c to peg %c",
                frompeg, topeg);
                return;
        towers(num - 1, frompeg, auxpeg, topeg);
        printf("\n Move disk %d from peg %c to peg %c", num,
        frompeg, topeg);
        towers(num - 1, auxpeg, topeg, frompeg);
}
```

TOW(3,A,C,B)
{
if (num == 1)
FALSE
TOW(2,A,B,C)

Move disk 3 from peg A to peg C

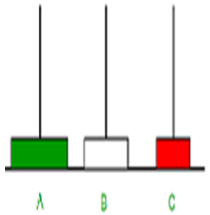


4

TOW(2,,B,C,A) REMAINING

CONTINUE

TOW(2,A,B,C)
{
if (num == 1)
FALSE
TOW(1,A,C,B)
Move disk 2 from peg A to peg B



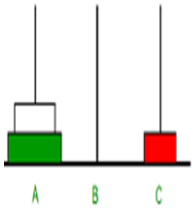
2

TOW(1,C,B,A)

}

TOW(1,A,C,B)
{
if (num == 1)
TRUE
Move disk 1 from peg A to peg C

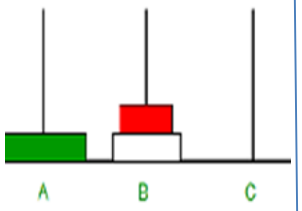
}



1

TOW(1,C,B,A)
{
if (num == 1)
TRUE
Move disk 1 from peg C to peg B

}



3

Passing Arrays to Functions

- Like Variables it is also possible to pass the values of an array to a function.
- For Example : **For Passing 1D array as an argument to function**
  - **List the name of the array without any subscripts**
  - **Size of the array**

Function Call ---- **maximum(a,n)**

Function Definition ----- **int maximum(int a[], int n)**

Simple Rules to remember:

1. Function is called by passing only name of the array
2. In Function Definition, the formal parameter must be an array type, size not required to mention

Passing Strings to Functions

- Strings are treated as character arrays in C. So the rules are same as like passing arrays to functions.

Simple Rules to remember:

1. **Function Call**

```
display(name);
```

Name of the string array/ Note: Subscript not required

2. **Function Definition**

```
void display(char name[])
```

3. **Function Declaration**

```
void display(char str[]);
```

Shows that the argument is a string.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Thank You