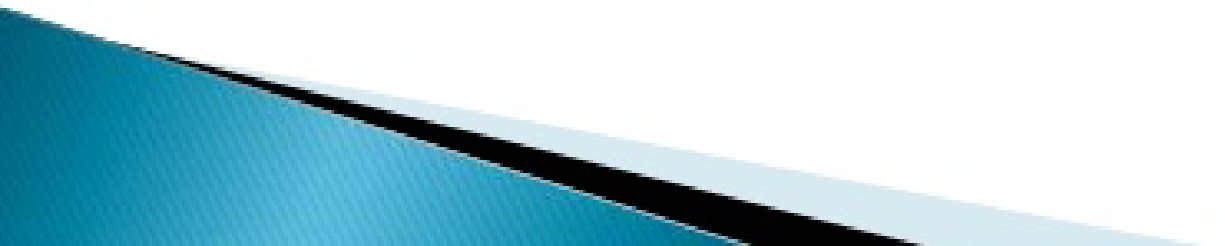


Function



- ▶ What is C function?
- ▶ Uses of C functions
- ▶ Types of C functions
 - Library functions in C
 - User defined functions in C
 - Creating/Adding user defined function in C library
- ▶ C function declaration, function call and definition with example program
- ▶ How to call C functions in a program?
 - Call by value
 - Call by reference
- ▶ C function arguments and return values
 - C function with arguments and with return value
 - C function with arguments and without return value
 - C function without arguments and without return value
 - C function without arguments and with return value

1. What is C function?

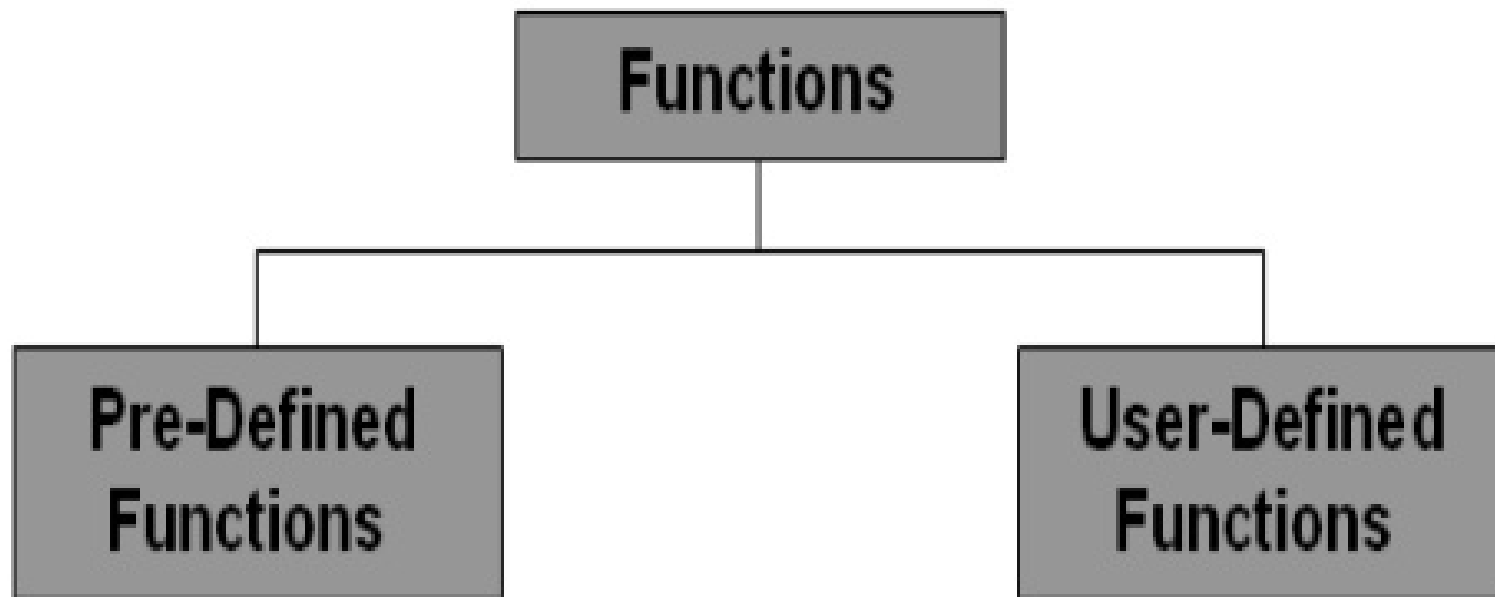
- ▶ A large C program is divided into basic building blocks called C function.
 - ▶ C function contains set of instructions enclosed by “{ }” which performs specific operation in a C program.
 - ▶ Actually, Collection of these functions creates a C program.
- 

2.Uses of C functions:

- 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.
- The core concept of C functions are, **re-usability**, **dividing a big task into small pieces** to achieve the functionality and to **improve understandability** of very large C programs.

3.Types of functions

- ▶ C functions can be classified into two categories
 - Library functions
 - User-defined functions



LIBRARY FUNCTIONS

- Library functions are not required to be written by us
- printf and scanf belong to the category of library function

Examples:

Printf(),scanf(),Sqrt(), cos(), strcat(),rand(), etc are some of library functions

- ▶ Consider the following example.

```
#include <stdio.h>
```

```
#include <math.h>
```

```
main( )
```

```
{
```

```
    float x,y ;
```

```
    scanf("%f", &x);
```

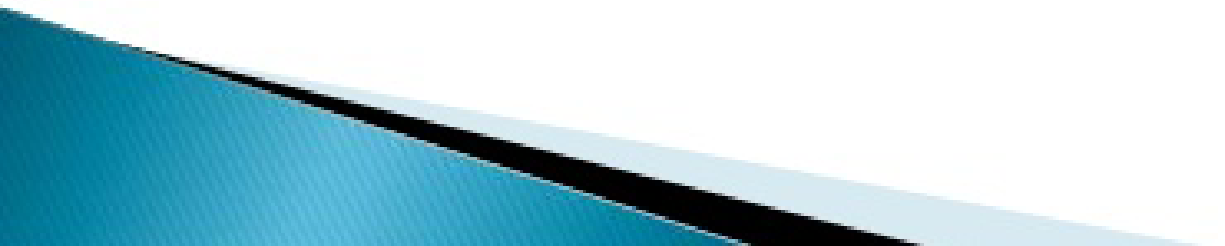
```
    y=sqrt(x);
```

```
    printf("Square root of %f is %f\n", x,y);
```


```
}
```

main() calls 3 built-in functions:
scanf(), sqrt() & printf()

NEED FOR USER-DEFINED FUNCTIONS

- ▶ Every program must have a main function
 - ▶ It is possible to code any program utilizing only main function, it leads to a number of problems
 - ▶ The program may become too large and complex and as a result the task of debugging, testing, and maintaining becomes difficult
 - ▶ If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit
 - ▶ These subprograms called 'functions' are much easier to understand, debug, and test
- 

NEED FOR USER-DEFINED FUNCTIONS


- ▶ There are times when some types of operation or calculation is repeated at many points throughout a program
 - ▶ In such situations, we may repeat the program statements whenever they are needed
 - ▶ Another approach is to design a function that can be called and used whenever required
 - ▶ This saves both time and space
- 

ELEMENTS OF USER-DEFINED FUNCTIONS

- ▶ **Function declaration or prototype** – informs compiler about the function name, function parameters and return value's data type.
- ▶ **Function call** – This calls the actual function
- ▶ **Function definition** – This contains all the statements to be executed.

Sno	C Function aspects	Syntax
1	Function definition	<code>return_type function_name(arguments list) { Body of function; }</code>
2	function call	<code>function_name (arguments list);</code>
3	function declaration	<code>return_type function_name (argument list);</code>

ELEMENTS OF USER-DEFINED FUNCTIONS

- ▶ Functions are classified as one of the derived data types in C
 - ▶ Can define functions and use them like any other variables in C programs.
 - ▶ Similarities between functions and variables in C
 - Both function name and variable names are considered identifiers and therefore they must adhere to the rules for identifiers.
 - Like variables, functions have types (such as int) associated with them
 - Like variables, function names and their types must be declared and defined before they are used in a program
- 

ELEMENTS OF USER-DEFINED FUNCTIONS

- ▶ There are **three elements** related to functions
 - **Function definition**
 - **Function call**
 - **Function declaration**
- ▶ **The function definition** is an independent program module that is specially written to implement the requirements of the function
- ▶ To use this function we need to invoke it at a required place in the program. This is known as **the function call**.
- ▶ The program that calls the function is referred to as the **calling program or calling function**.
- ▶ The calling program should declare any function that is to be used later in the program. This is known as the **function declaration or function prototype**.

FUNCTION

```
main()
```

```
{
```

```
.....
```

```
.....
```

```
func1();
```

```
.....
```

```
.....
```

```
}
```

```
data_type func1()
```

```
{
```

```
.....
```

```
.....
```

```
.....
```

```
.....
```

```
}
```

Calling function

Called function



DEFINITION OF FUNCTIONS

- ▶ A function definition, also known as function implementation shall include the following elements;
 - **Function name;**
 - **Function type;**
 - **List of parameters;**
 - **Local variable declaration;**
 - **Function statements; and**
 - **A return statement.**
- ▶ All six elements are grouped into two parts, namely,
 - **Function header** (First three elements); and
 - **Function body** (Second three elements)

THE FORM OF C FUNCTION

```
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 DEFINITION

- ▶ 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 is expected to return to the program calling the function
 - If the return type is not explicitly specified, C assumes 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

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

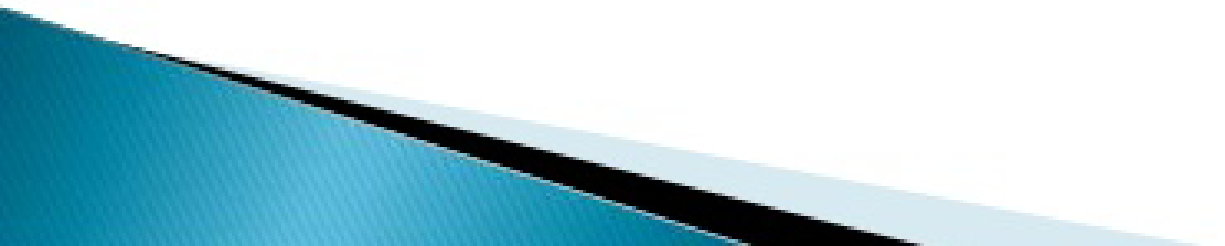
FORMAL PARAMETER LIST

- ▶ 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 null parameter list

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
- 

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);
```

4. How to call C functions in a program?

- ▶ There are two ways that a C function can be called from a program. They are,
 - Call by value
 - Call by reference

1. Call by value:

- In call by value method, the value of the variable is passed to the function as parameter.
 - 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 actual parameter is copied to formal parameter.
- ▶ Note:
- **Actual parameter** – This is the argument which is used in function call.
 - **Formal parameter** – This is the argument which is used in function definition

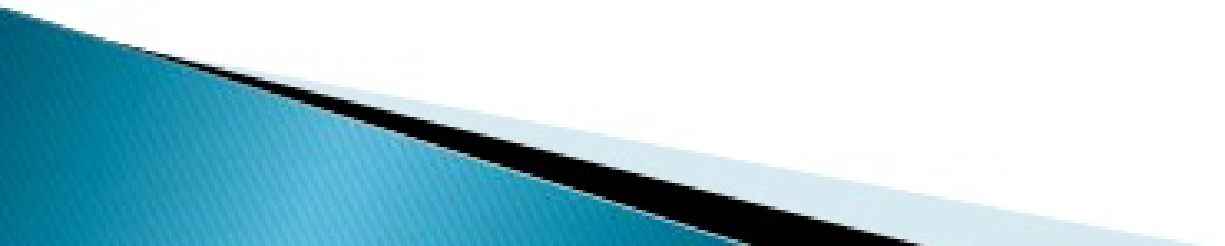
Example program for C function (using call by value):

```
▶ #include<stdio.h>
void swap(int a, int b);           // function prototype, also called function declaration
int main()
{
    int m = 22, n = 44;
    printf(" values before swap m = %d \nand n = %d", m, n);
    swap(m, n);                   // calling swap function by value
}
void swap(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
    printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

Output

values before swap m = 22 and n =44
values after swap m = 44 and n = 22

2. Call by reference:

- ▶ In call by reference method, 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.
- 

Example program for call by reference

```
#include<stdio.h>
void swap(int *a, int *b); // function prototype, also called function declaration
int main()
{
    int m = 22, n = 44;
    // calling swap function by reference
    printf("values before swap m = %d \n and n = %d",m,n);
    swap(&m, &n);
}
void swap(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    printf("\n values after swap a = %d \n and b = %d", *a, *b);
}
```

Output

values before swap m = 22 and n =44
values after swap m = 44 and n = 22

CATEGORY OF FUNCTIONS

- ▶ C function with arguments (parameters) and with return value
- ▶ C function with arguments (parameters) and without return value
- ▶ C function without arguments (parameters) and without return value
- ▶ C function without arguments (parameters) and with return value

Sno	C function	C function
1	with arguments and with return values	<pre>int function (int); // function declaration function (a); // function call int function(int a) // function definition {statements; return a;}</pre>
2	with arguments and without return values	<pre>void function (int); // function declaration function(a); // function call void function(int a) // function definition {statements;}</pre>
3	without arguments and without return values	<pre>void function(); // function declaration function(); // function call void function() // function definition {statements;}</pre>
4	without arguments and with return values	<pre>int function (); // function declaration function (); // function call int function() // function definition {statements; return a;}</pre>