Introduction to Functions

Introduction to Functions

- A complex problem is often easier to solve by dividing it into several smaller parts, each of which can be solved by itself.
- This smaller part of the program can be referred as ***subprograms***.
- Dividing a complex program into ***subprograms*** is called ***structured*** programming.

INTRODUCTION

- The ***subprogram*** posses a self contain components and have well define purpose.
- In C, the ***subprogram*** is called as a ***function*** .
- Basically a job of ***function*** is to do something.
- C program contain at least one function which is **`main()`**
- **`main()`** then uses these functions to solve the original problem.

Introduction

- C language is collection of various **inbuilt functions**.
- If you have written a program in C then it is evident that you have used C's inbuilt functions.
- printf, scanf, clrscr etc. all are C's inbuilt functions.

Introduction

- A ***function*** in C language is a block of code that performs a specific task.
- It has a ***name*** and it is ***reusable*** i.e. it can be executed from as many different parts in a C Program as required.

1/1! + 2/2! + 3/3! + …..7/7!

Properties

- ***Function*** in a C program has some properties:
- Every function has a unique ***name***.
  - This name is used to ***call*** function from “main()” 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.

Properties

- ***Function*** in a C program has some properties:
- A ***function*** performs a **specific task**.
  - A task is a distinct job that your program must perform as a part of its overall operation,
  - such as:
    - adding two or more integer,
    - Finding the number is Prime or Not, or
    - calculating a cube root etc.

Properties

- ***Function*** in a C program has some properties:
- A ***function*** **returns** a value to the calling program.
  - This is optional and depends upon the task your function is going to accomplish.
  - Suppose you want to just show few lines through function then it is not necessary to return a value.
  - But if you are calculating area of rectangle and wanted to use result somewhere in program then you have to send back (return) value to the calling function.

CONTENT

- **What is C function and what are its uses?**
- **Types of C functions**
  - **Library functions in C**
  - **User defined functions in C**
- **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**

What is a 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.

Example

```
# include <stdio.h>
void message( ) ; /* function prototype declaration */
int main( )
{
message( ) ; /* function call */
printf ( "Cry, and you stop the monotony!\n" ) ;
return 0 ;
}
void message( ) /* function definition */
{
printf ( "Smile, and the world smiles with you...\n" ) ;
}
```

output...
Smile, and the world smiles with you...
Cry, and you stop the monotony!

Function Prototype Declaration

- The first is the function prototype declaration and is written as:

  void message( ) ;

- This prototype declaration indicates that **message( )** is a function which after completing its execution does not return any value.
- This ‘does not return any value’ is indicated using the keyword **void**.
- It is necessary to mention the prototype of every function that we intend to define in the program.

Function Prototypes

- Provides the compiler with the description of functions that will be used later in the program.
- It defines the function before it's been used/called.
- Function prototypes need to be written at the beginning of the program.
- The function prototype must have :
  - A return type indicating the variable that the function will be returning
  - It can be – void, int, float, char etc.

Function Definition

- The second usage of **message** is…

```
void message( )
{
printf ( "Smile, and the world smiles with you...\n" ) ;
}
```

- This is the function definition. In this definition right now we are having
- only **printf( )**, but we can also use **if**, **for**, **while**, **switch**, etc., within this function definition

Calling the Function

- The third usage is…

  message( ) ;

- Here the function **message( )** is being called by **main( )**.
- What does it mean when we say that **main( )** 'calls' the function **message( )**?
- It means that the control passes to the function **message( )**.
- The activity of **main( )** is temporarily suspended; it falls asleep while the **message( )** function wakes up and goes to work.
- When the **message( )** function runs out of statements to execute, the control returns to **main( )**, which comes to life again and begins executing its code at the exact point where it left off.
- Thus, **main( )** becomes the 'calling' function, whereas **message( )** becomes the 'called' function.

Function prototype declaration

```
void message( ) ;

main( )
{
    message( ) ;
    printf ( "\nCry, and you stop the monotony!" ) ;
}
message( )
{
    printf ( "\nSmile, and the world smiles with you..." ) ;
}
```

Function Call

```
main( )
{
    message( ) ;
    printf ( "\nCry, and you stop the monotony!" ) ;
}
message( )
{
    printf ( "\nSmile, and the world smiles with you..." ) ;
}
```

Function definition

OUTPUT:

```
Smile, and the world smiles with you...
Cry, and you stop the monotony!
```

Summarize

(a) C program is a collection of one or more functions.

(b) A function gets called when the function name is followed by a semicolon. For example,

```
main( )
{
    argentina( ) ;
}
```

Function Call

(c) A function is defined when function name is followed by a pair of braces in which one or more statements may be present. For example,

```
argentina( )
{
    statement 1 ;
    statement 2 ;
    statement 3 ;
}
```

Function Definition

EXAMPLE

```
main( )
{
    printf ( "\nI am in main" ) ;
    italy( ) ;
    brazil( ) ;
    argentina( ) ;
}
```

```
italy( )
{
    printf ( "\nI am in italy" ) ;
}
brazil( )
{
    printf ( "\nI am in brazil" ) ;
}
argentina( )
{
    printf ( "\nI am in argentina" ) ;
}
```

EXAMPLE

```
main( )
{
    printf ( "\nI am in main" ) ;
    italy( ) ;
    brazil( ) ;
    argentina( ) ;
}
```

```
italy( )
{
    printf ( "\nI am in italy" ) ;
}
brazil( )
{
    printf ( "\nI am in brazil" ) ;
}
argentina( )
{
    printf ( "\nI am in argentina" ) ;
}
```

OUTPUT:

```
I am in main
I am in italy
I am in brazil
I am in argentina
```

Properties

- Any C program contains at least one ***function***.
- If a program contains **only one function**, it must be ***main( ).***
- If a C program contains **more than one function**, then **one (and only one) of these functions must be main( )**, because **program execution always begins with main( ).**
- There is **no limit** on the **number of functions** that might be present in a C program.
- Each **function** in a program is **called in the sequence** specified by the function calls in main( ).
- **After each function has done its thing**, ***control returns to main( ).***
- When main( ) runs out of function calls, **the program ends**.

EXAMPLE

```
main( )
{
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}
italy( )
{
    printf ( "\nI am in italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in italy" ) ;
}
brazil( )
{
    printf ( "\nI am in brazil" ) ;
    argentina( ) ;
}
argentina( )
{
    printf ( "\nI am in argentina" ) ;
}
```

EXAMPLE

```
main( )
{
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}
italy( )
{
    printf ( "\nI am in italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in italy" ) ;
}
brazil( )
{
    printf ( "\nI am in brazil" ) ;
    argentina( ) ;
}
argentina( )
{
    printf ( "\nI am in argentina" ) ;
}
```

OUTPUT:

```
I am in main
I am in italy
I am in brazil
I am in argentina
I am back in italy
I am finally back in main
```

Summarize

(a) A function gets called when the function name is followed by a semicolon (;).

For example,

```
argentina( ) ;
```

(b) A function is defined when function name is followed by a pair of braces ({ }) in which one or more statements may be present.

For example,

```
void argentina( )
{
statement 1 ;
statement 2 ;
statement 3 ;
}
```

Summarize

(c) Any function can be called from any other function. Even **main( )** can be called from other functions.

For example,

```
# include <stdio.h>
void message( ) ;
int main( )
{
message( ) ;
return 0 ;
}
void message( )
{
printf ( "Can't imagine life without C\n" ) ;
main( ) ;
}
```

Summarize

(d) A function can be called any number of times. For example,

```
# include <stdio.h>
void message( ) ;
int main( )
{
message( ) ;
message( ) ;
return 0 ;
}
void message( )
{
printf ( "Jewel Thief!!\n" ) ;
}
```

Summarize

(e) The order in which the functions are defined in a program and the order in which they get called need not necessarily be same. For example,

```
# include <stdio.h>
void message1( ) ;
void message2( ) ;
int main( )
{
message1( ) ;
message2( ) ;
return 0 ;
}
void message2( )
{
printf ( "But the butter was bitter\n" ) ;
}
void message1( )
{
printf ( "Mary bought some butter\n" ) ;
}
```

Summarize

(f) A function can call itself. Such a process is called '**recursion**'. Higher concepts of C functions. (not part of syllabus).

(g) **A function can be called from another function**, but a function cannot be defined in another function.

Thus, the following program code would be wrong, since **argentina( )** is being defined inside another function, **main( )**:

```
int main( )
{
printf ( "I am in main\n" ) ;
void argentina( )
{
printf ( "I am in argentina\n" ) ;
}
}
```

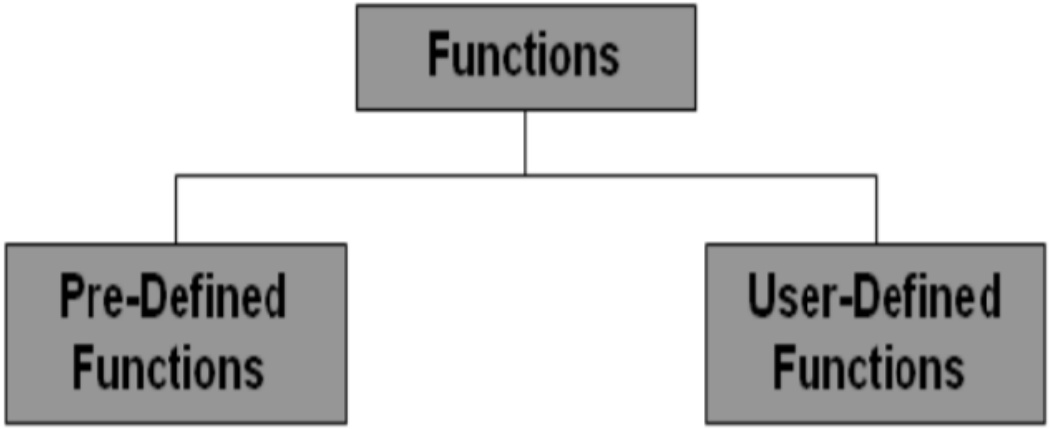
Summarize

(h) There are basically two types of functions:

- **Library functions** Ex. **printf( )**, **scanf( )**, etc.
- **User-defined functions** Ex. **argentina( )**, **brazil( )**, etc.

Types of C functions

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



Library functions

- As the name suggests, **library functions** are nothing but **commonly required functions grouped together and stored in a Library file on the disk.**
- These library of functions come ready-made with development environments like Turbo C, Visual Studio, NetBeans, gcc, etc.
- The procedure for calling both types of functions is exactly same.

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

Common Library Functions

`#include <math.h>`

- `sin(x) // radians`
- `cos(x) // radians`
- `tan(x) // radians`
- `atan(x)`
- `atan2(y,x)`
- `exp(x)` // e^x
- `log(x)` // $\log_e x$
- `log10(x)` // $\log_{10} x$
- `sqrt(x)` // $x \geq 0$
- `pow(x, y)` // x^y
- `...`

`#include <stdio.h>`

- `printf()`
- `fprintf()`
- `scanf()`
- `sscanf()`
- `...`

`#include <string.h>`

- `strcpy()`
- `strcat()`
- `strcmp()`
- `strlen()`
- `...`

- 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()

Note:

1. the inbuilt function to calculate square root is sqrt()

2. This function takes one argument x.

3. The sqrt() function is defined in math. h header file.

4. The sqrt function takes a single argument in float and returns it in float.

5. The returned value is stored in variable y which is float

User Defined Functions

- These functions are **designed by the user** when they are writing any program because **for every task we do not have a library of functions where their definitions are predefined.**
- To perform according to the **requirement of user** the **user have to develop some functions by itself**, these functions are called **user-defined functions**.
- For such functions the user have to define the **proper definition of the function**.

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	return_type function_name(arguments list) { Body of function; }
2	function call	function_name (arguments list);
3	function declaration	return_type function_name (argument list);

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 Function

- 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**.

Why Use Functions

- Why write separate functions at all?
- Why not squeeze the entire logic into one function, **main( )**?

Why Use Functions

Two reasons:

(a) **Writing functions avoids rewriting the same code over and over.**

- Suppose you have a section of code in your program that calculates **area of a triangle**.
- If later in the program you want to calculate the area of a different triangle, you will have to write the same instructions all over again.
- **Instead**, you would prefer to jump to a 'section of code' that calculates area and then jump back to the place from where you left off.
- This section of code is nothing but a **function**.

Example

Let’s have an example to illustrate this

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
printf("Welcome to function in C");
printf("\n");
for(i=0;i<30;i++)
{  printf("-");
}
printf("\n");
printf("Function easy to learn.");
for(i=0;i<30;i++)
{  printf("-");
}
printf("\n");
}
```

Example

```
#include<stdio.h>
#include<conio.h>
void printline();
void main()
{
printf("Welcome to function in C");
printf("\n");

printline();
printf("Function easy to learn.");
printline();
}
```

```
void printline()
{  int i;
for(i=0;i<30;i++)
{  printf("-");
}
printf("\n");
}
```

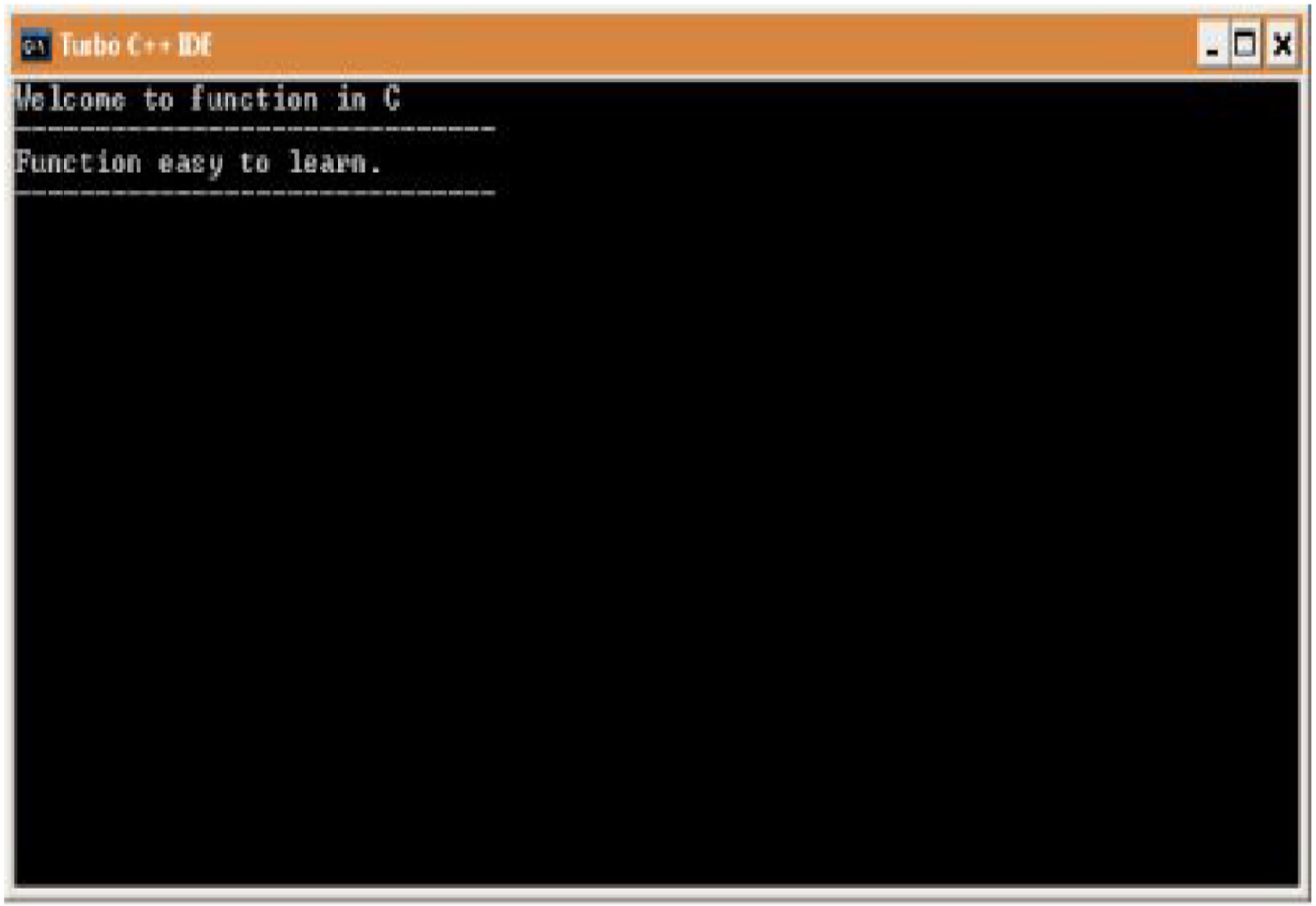
Example

```
#include<stdio.h>
#include<conio.h>
void printline();
void main()
{
printf("Welcome to function in C");
printf("\n");

printline();
printf("Function easy to learn.");
printline();
}
```

```
void printline()
{  int i;
for(i=0;i<30;i++)
{  printf("-");
}
printf("\n");
}
```

Output:



Output of above program.

Why Use Functions

Two reasons:

(b) By using functions it becomes easier to write programs and **keep track of what they are doing**.

- If the operation of a program can be divided into separate activities, and
- each activity placed in a different function,
- then each could be written and checked more or less independently.
- **Separating the code into modular functions also makes the program easier to design and understand**.

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.

Conclusion

- Break a program into small units and write functions for each of these isolated subdivisions.

Do It Yourself

1. Write a program to calculate x^y using **<u>inbuilt power function</u>** .

- Tips:

    a. Which inbuilt function is used to calculate power.

    b. Determine how many arguments does the function require for calculating power .

    c. Which header file needs to be included.

    d. What should be the data type of arguments.

Do It Yourself

2. Determine sin(x), cos(x) and tan(x) values for x=0, x=30, x=60, x=90.

The output should be displayed as follows

X=0 x=30 x=60 x=90

Sin (x)

cos(x)

tan(x)

3. getc(), getch(), getchar() and getche() are some of the functions related to characters. Determine what is their functionality. Create a program including all these inbuilt functions