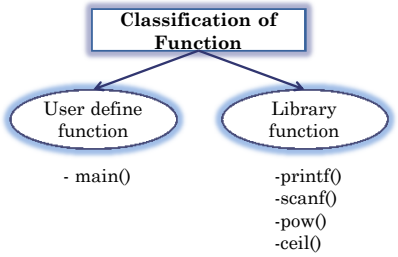
C PROGRAMMING-FUNCTION

PAWAN KUMAR SINGH
DEPARTMENT OF INFORMATION TECHNOLOGY
JADAVPUR UNIVERSITY
KOLKATA

What is a function

- A large program in c can be divided to many subprogram
- The subprogram posses a self contain components and have well define purpose.
- 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().



Advantages of function

- It is much easier to write a structured program where a large program can be divided into a smaller, simpler task.
- Allowing the code to be called many times
- Easier to read and update
- It is easier to debug a structured program where there error is easy to find and fix

Example

```
1:  #include <stdio.h>
2:
3:  long cube(long x);  /* Function prototype*/
4:
5:  long input, answer;
6:
7:  int main( void )
8:  {
9:    printf("Enter an integer value: ");
10:   scanf("%d", &input);
11:   answer = cube(input);   /* calling function*/
12:    printf("\nThe cube of %ld is %ld.\n", input, answer);
13:
14:    return 0;
15: }
16:
17: long cube(long x) /* Function definition*/
18: {
19:    long x_cubed;
20:
21:    x_cubed = x * x * x;
22:    return x_cubed;
23: }
```

Arguments/formal parameter

Return data type

Actual parameters

- Function names is cube
- Variable that are requires is long
- The variable to be passed on is X(has single arguments)—value can be passed to function so it can perform the specific task. It is called arguments

Output

Enter an integer value:4

The cube of 4 is 64.

How the function works

- C program doesn't execute the statement in function until the function is called.
- When function is called the program can send the function information in the form of one or more argument.
- When the function is used it is referred to as the ***called function***
- Functions often use data that is passed to them from the ***calling function***
- Data is passed from the calling function to a called function by specifying the variables in a argument list.
- **Argument** list cannot be used to send data. Its only copy data/value/variable that pass from the calling function.
- The called function then performs its operation using the copies.

Function prototypes

- Provides the compiler with the description of functions that will be used later in the program
- Its define the function before it 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 return

Syntax for Function Prototype

return-type function_name(arg-type name-1,...,arg-type name-n);

Function Prototype Example:

- double squared(double number);
- void print_report(int report_number);
- int get_menu_choice(void);

Function Definitions

- It is the actual function that contains the code that will be execute.
- Should be identical to the function prototype.

Syntax of Function Definition

return-type function_name(arg-type name-1,...,arg-type name-n) ---- **Function header**

```
{
declarations;
statements;
return(expression);
}
```

Function Body

Function Definition Example:

```
float conversion (float celsius)
 {
      float fahrenheit;
      fahrenheit = celcius*33.8
      return fahrenheit;
 }
```

The function name's is conversion

This function accepts arguments celcius of the type float. The function return a float value.

So, when this function is called in the program, it will perform its task which is to convert fahrenheit by multiply celcius with 33.8 and return the result of the summation.

Note that if the function is returning a value, it needs to use the keyword return.

Function return types

Can be any of C's data type:

- char
- int
- float
- long.....

Examples:

```
int func1(...)        /* Returns a type int.   */
float func2(...)      /* Returns a type float. */
void func3(...)       /* Returns nothing.      */
```


Types of Functions

Function can be divided into 4 categories:

A function with no arguments and no return value
A function with no arguments and a return value
A function with an argument or arguments and returning no value
A function with arguments and returning a values

A function with no arguments and no return value

- Called function does not have any arguments
- Not able to get any value from the calling function
- Not returning any value
- There is no data transfer between the calling function and called function.

```
#include<stdio.h>
#include<conio.h>
void printline();

void main()
{
   printf("Welcome to function in C");
   printline();
   printf("Function easy to learn.");
   printline();
   getch();
}

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

```
Welcome to function in C
------------------------------
Function easy to learn.
------------------------------
```


A function with no arguments and a return value

- Does not get any value from the calling function
- Can give a return value to calling program

```
#include <stdio.h>
#include <conio.h>
int send();

void main()
{
   int z;
   z=send();
   printf("\nYou entered : %d.",z);
   getch();
}

int send()
{
   int no1;
   printf("Enter a no: ");
   scanf("%d",&no1);
   return(no1);
}
```

```
Enter a no: 46

You entered : 46.
```

A function with an argument or arguments and returning no value

- A function has argument/s
- A calling function can pass values to function called , but calling function not receive any value
- Data is transferred from calling function to the called function but no data is transferred from the called function to the calling function
- Generally Output is printed in the Called function
- A function that does not return any value cannot be used in an expression it can be used only as independent statement.


```
#include<stdio.h>
#include<conio.h>
void add(int x, int y);

void main()
{
   add(30,15);
   add(63,49);
   add(952,321);
   getch();
}

void add(int x, int y)
{
   int result;
   result = x+y;
   printf("Sum of %d and %d is %d.\n\n",x,y,result);
}
```

```
Sum of 30 and 15 is 45.
Sum of 63 and 49 is 112.
Sum of 952 and 321 is 1273.
```


A function with arguments and returning a values

- Argument are passed by calling function to the called function
- Called function return value to the calling function
- Mostly used in programming because it can two way communication
- Data returned by the function can be used later in our program for further calculation.


```
#include <stdio.h>
#include <conio.h>
int add(int x,int y);

void main()

{
   int z;

   z=add(952,321);
   printf("Result %d. \n\n",add(30,55));
   printf("Result %d.\n\n",z);

   getch();
}
int add(int x,int y)

{
   int result;
   result = x + y;
   return(result);
}
```

```
Result 85.

Result 1273.
```

Send 2 integer value x and y to add()

Function add the two values and send back the result to the calling function

int is the return type of function

Return statement is a keyword and in bracket we can give values which we want to return.

Pointers

Variable that declared occupies a memory according to it size

It has address for the location so it can be referred later by CPU for manipulation

The '*' and '&' Operator

Int x= 10

x	→	Memory location name
10	→	Value at memory location
76858	→	Memory location address

We can use the address which also point the same value.


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


void main()

{
   int i=9;

   printf("Value of i : %d\n",i);
   printf("Adress of i %d\n", &i);

   getch();
}
```

```
Value of i : 9
Adress of i 2686732
```

& show the address of the variable


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


void main()

{
   int i=9;

   printf("Value of i : %d\n",i);
   printf("Address of i %d\n", &i);
   printf("Value at address of i: %d", *(&i));

   getch();
}
```

```
Value of i : 9
Address of i 2686732
Value at address of i: 9_
```

* Symbols called the value at the address

Passing arguments by value or reference

```
#include <stdio.h>
#include <conio.h>
void callByValue(int, int);
void callByReference(int *, int *);

int main()
{
   int x=10, y =20;
   printf("Value of x = %d and y = %d. \n",x,y);

   printf("\nCAll By Value function call...\n");
   callByValue(x,y);
   printf("\nValue of x = %d and y = %d.\n", x,y);

   printf("\nCAll By Reference function call...\n");
   callByReference(&x, &y);
   printf("Value of x = %d and y = %d.\n", x,y);

   getch();
   return 0;
}
```

```
void callByValue(int x, int y)

{
   int temp;
   temp=x;
   x=y;
   y=temp;

   printf("\nValue of x = %d and y = %d inside callByValue function",x,y);
}

void callByReference(int *x, int *y)

{
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;

}
```



```
Value of x = 10 and y = 20.

CAll By Value function call...

Value of x = 20 and y = 10 inside callByValue function
Value of x = 10 and y = 20.

CAll By Reference function call...
Value of x = 20 and y = 10.
```


LOCAL AND GLOBAL VARIABLES

- **Local:**

These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

- **Global:**

These variables can be accessed by any function comprising the program. They do not get recreated if the function is recalled. To declare a global variable, declare it outside of all the functions. There is no general rule for where outside the functions these should be declared, but declaring them on top of the code is normally recommended for reasons of scope. If a variable of the same name is declared both within a function and outside of it, the function will use the variable that was declared within it and ignore the global one.

FIBONACCI SERIES USING FOR LOOP

```
#include<stdio.h>
int main()
{
  int n, first = 0, second = 1, next, c;
  printf("Enter the number of terms\n");
  scanf("%d",&n);
  printf("First %d terms of Fibonacci series are :-\n",n);
  for ( c = 0 ; c < n ; c++ )
  {
    if ( c <= 1 )
          next = c;
    else
    {     next = first + second;
          first = second;
          second = next;
    }
     printf("%d\n",next);
  }
  return 0;
}
```

Fibonacci series using recursion

```
#include<stdio.h>
#include<conio.h>
int Fibonacci(int);
main()
{
   clrscr();
   int n, i = 0, c;
   printf("Enter any number\n");
   scanf("%d",&n);
   printf("Fibonacci series\n");
   for ( c = 1 ; c <= n ; c++ )
   {
   printf("%d\n", Fibonacci(i));
      i++;
   }
   return 0;
}
```

```
int Fibonacci(int n)
{
  if ( n == 0 )
    return 0;
  else if ( n == 1 )
    return 1;
  else
    return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```


Factorial program using for loop

```
#include <stdio.h>
#include<conio.h>
void main()
{
 int c, n, fact = 1;
  printf("Enter a number to calculate it's factorial\n");
 scanf("%d", &n);
 for (c = 1; c <= n; c++)
   fact = fact * c;
 printf("Factorial of %d = %d\n", n, fact);
getch();
 }
```


Factorial program using function

```
#include <stdio.h>
#include<conio.h>
long factorial(int);
void main()
{
 int number;
 long fact = 1;
  printf("Enter a number to calculate it's factorial\n");

 scanf("%d", &number);
printf("%d! = %ld\n", number, factorial(number));

getch();
}
```

```
long factorial(int n)
{
 int c;
 long result = 1;
  for (c = 1; c <= n; c++)
   result = result * c;
  return result;
}
```


Program to find HCF and LCM

```
#include <stdio.h>
#include<conio.h>
void main() {
  clrscr();
  int a, b, x, y, t, gcd, lcm;
  printf("Enter two integers\n");
  scanf("%d%d", &x, &y);
  a = x;
  b = y;
  while (b != 0) {
    t = b;
    b = a % b;
    a = t;
  }
  gcd = a;
  lcm = (x*y)/gcd;
  printf("Greatest common divisor of %d and %d = %d\n", x, y, gcd);
  printf("Least common multiple of %d and %d = %d\n", x, y, lcm);
        getch();
}
```

HCF AND LCM USING FUNCTION

```
#include <stdio.h>
#include<conio.h>
long gcd(long, long);
int main()
{
clrscr();
  long x, y, hcf, lcm;
   printf("Enter two integers\n");
  scanf("%ld%ld", &x, &y);
   hcf = gcd(x, y);
  lcm = (x*y)/hcf;
   printf("Greatest common divisor of %ld and %ld = %ld\n", x, y, hcf);
  printf("Least common multiple of %ld and %ld = %ld\n", x, y, lcm);
  getch();
}

long gcd(long x, long y) {
  if (x == 0) {
    return y;
  }
   while (y != 0) {
    if (x > y) {
      x = x - y;
    }
    else {
      y = y - x;
    }
  }
   return x;
}
```