

# C Programming

## Lecture 8-1 : Function (Basic)

## What is a Function?

- A small program(subroutine) that **performs a particular task**
  - Input : parameter / argument
  - Perform what ? : function body
  - Output : return value
- Modular programming design
  - Large and complex task can be divided into smaller and simple task which is more easily solved(implemented).
  - Also called
    - structured design
    - Top-down design
    - Divide-and-Conquer

## Function Example

```
#include <stdio.h>

int f(int val);           // function prototype declaration
                        // int f(int);   is also OK!

int main()
{
    int x,y;
    scanf("%d",&x);
    y = f(x);             // function call. x is argument.
    printf("y=%d", y);
    return 0;
}

int f(int val)           // function definition. val is parameter.
{
    int k;
    k = 2*val - 3 ;
    return k;
}
```

## Function Definition

### ■ Syntax

```
return_type function_name (data_type variable_name, ...)
{
    local declarations;           // local variables
    function statements;
}
```

### ■ Example

```
int factorial (int n)
{
    int i,product=1;
    for (i=2; i<=n; ++i)
        product *= i;
    return product;
}
```

## void type

- Example)

```
void print_info(void)
{
    printf("Navier-Stokes Equations Solver ");
    printf("v3.45\n");
    printf("Last Modified: ");
    printf("12/04/95 - viscous coefficient added\n");
}
```

- return type is void
- No parameter

## Variables

- Global variable
  - Declared outside function block
  - Accessible everywhere
  - Global variable is destroyed only when a program is terminated.
- Local variable (automatic variable ?)
  - Declared inside function body
  - Accessible only in the function
  - Local variable is created when a function is called and is destroyed when a function returns.
- Static variable (declared in a function)
  - (Usually) accessible in the function
  - Static variable persists until the program is terminated

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

void useLocalScope( void );           // function prototype
void useStaticLocalScope( void );     // function prototype
void useGlobalScope( void );         // function prototype

int x = 1;                             // global variable

int main()
{
    int x = 5; // local variable to main
    printf("local x in main's outer block is %d\n",x);

    { // start new block

        int x = 7;
        printf("local x in main's inner block is %d\n",x);

    } // end new block
```

```
printf("local x in main's outer block is %d\n",x);

useLocalScope();
useStaticLocalScope();
useGlobalScope();
useLocalScope();
useStaticLocalScope();
useGlobalScope();

printf("\nlocal x in main's outer block is %d\n",x);

return 0;

} // end main
```

```

// useLocalScope
void useLocalScope( void )
{
    int x = 25; // initialized each time this function is called.

    printf("local x is %d on entering useLocalScope()\n",x);
    ++x;
    printf("local x is %d on exiting useLocalScope()\n",x);
} // end function useLocalScope

```

```

void useStaticLocalScope( void )
{
    // x is initialized only first time useStaticLocalScope is called.
    // It's value is kept till the next call.
    static int x = 50;

    printf("local static x is %d on entering useStaticLocalScope()\n",x);

    ++x; // increment x

    printf("local static x is %d on exiting useStaticLocalScope()\n",x);
} // end function useStaticLocal

```

```

// useGlobal modifies global variable x during each call
void useGlobalScope( void ) // modifies global variable x during each call.
{
    printf("global x is %d on entering useGlobalScope()\n",x);
    x *= 10; // multiply 10 to x
    printf("global x is %d on exiting useGlobalScope()\n",x);
} // end function useGlobalScope()

```

```

local x in main's outer block is 5
local x in main's inner block is 7
local x in main's outer block is 5

local x is 25 on entering useLocalScope()
local x is 26 on exiting useLocalScope()

local static x is 50 on entering useStaticLocalScope()
local static x is 51 on exiting useStaticLocalScope()

global x is 1 on entering useGlobalScope()
global x is 10 on exiting useGlobalScope()

local x is 25 on entering useLocalScope()
local x is 26 on exiting useLocalScope()

local static x is 51 on entering useStaticLocalScope()
local static x is 52 on exiting useStaticLocalScope()

global x is 10 on entering useGlobalScope()
global x is 100 on exiting useGlobalScope()

local x in main's outer block is 5

```

output

## Variables

- You must understand the difference between
  - Global vs Local variables
  - Static vs Global variables
  - Static vs Local(Automatic) variables

## Considerations (1)

- The number of arguments in the function call must match the number of arguments in the function definition.
- The type of the arguments in the function call must match the type of the arguments in the function definition.
- The type of actual return value must match the type of return type in function prototype.
- Before calling a function, either function definition or function prototype declaration must be done.

## Considerations (2)

- The actual arguments in the function call are matched up in-order with the dummy arguments in the function definition.
- The actual arguments are passed by-value to the function. The dummy arguments in the function are initialized with the present values of the actual arguments. *Any changes made to the dummy argument in the function will NOT affect the actual argument in the main program.*

## Why use functions?

- Many, many reasons
  - Don't have to repeat the same block of code many times. Make that code block a function and call it when needed.
  - Reuse : useful functions can be used in a number of programs.
  - top-down technique : Make an outline and hierarchy of the steps needed to solve your problem and create a function for each step.
  - Easy to debug : Get one function working well then move on to the others.
  - Easy to modify and expand : Just add more functions to extend program capability
  - Readability : Make program self-documenting and readable.

# Math Library Functions

(example)

```
#include <stdio.h>
#include <math.h> // you must include <math.h>
                // to use math functions

int main()
{
    double c, a, b;
    scanf("%lf %lf",&a,&b);
    c=sqrt(pow(a,2)+pow(b,2));
    printf("a^2+b^2=%lf\n",c);
    return 0;
}
```

| Method       | Description  | Example   |
|--------------|--|---|
| ceil( x )    | rounds $x$ to the smallest integer not less than $x$   | ceil( 9.2 ) is 10.0<br>ceil( -9.8 ) is -9.0                       |
| cos( x )     | trigonometric cosine of $x$ ( $x$ in radians)          | cos( 0.0 ) is 1.0   |
| exp( x )     | exponential function $e^x$                             | exp( 1.0 ) is 2.71828<br>exp( 2.0 ) is 7.38906                    |
| fabs( x )    | absolute value of $x$                                  | fabs( 5.1 ) is 5.1<br>fabs( 0.0 ) is 0.0<br>fabs( -8.76 ) is 8.76 |
| floor( x )   | rounds $x$ to the largest integer not greater than $x$ | floor( 9.2 ) is 9.0<br>floor( -9.8 ) is -10.0                     |
| fmod( x, y ) | remainder of $x/y$ as a floating-point number          | fmod( 13.657, 2.333 ) is 1.992                                    |
| log( x )     | natural logarithm of $x$ (base $e$ )                   | log( 2.718282 ) is 1.0<br>log( 7.389056 ) is 2.0                  |
| log10( x )   | logarithm of $x$ (base 10)                             | log10( 10.0 ) is 1.0<br>log10( 100.0 ) is 2.0                     |
| pow( x, y )  | $x$ raised to power $y$ ( $x^y$ )                      | pow( 2, 7 ) is 128<br>pow( 9, .5 ) is 3                           |
| sin( x )     | trigonometric sine of $x$ ( $x$ in radians)            | sin( 0.0 ) is 0   |
| sqrt( x )    | square root of $x$                                     | sqrt( 900.0 ) is 30.0<br>sqrt( 9.0 ) is 3.0                       |
| tan( x )     | trigonometric tangent of $x$ ( $x$ in radians)         | tan( 0.0 ) is 0   |

[Math library functions.](#)