**Functions**

**Introduction**

* Most computer programs that solve real-world problems are much larger than the programs that we discuss in the class.
* The best way to develop and maintain large program is to construct it from smaller pieces or modules.
* This technique is called divide and conquer.
* In C language these modules are called functions.
* Another motivation to use functions is software reusability.

**C standard library**

* C standard library provides a rich collection of functions for performing common mathematical calculations, string manipulations, input/output.
* e.g: printf

scanf

pow

**Math Library Functions**

* Allows the user to perform certain common mathematical calculations.

|  |  |  |
| --- | --- | --- |
| **Function** | Description | Example |
| sqrt( x ) | square root of x | sqrt (900.0) is 30.0 |
| pow( x, y ) | x raised to power y (xy) | pow ( 2, 7) is 128.0 |
| exp( x ) | exponential function ex | exp (1.0) is 2.718282 |
| log ( x ) | natural logarithm of x | log(2.718282) is 1.0 |
| ceil ( x ) | rounds x to the smallest  integer not less than x | ceil (9.2) is 10.0  ceil ( - 9.8 ) is – 9.0 |
| floor ( x ) | rounds x to the smallest integer not greater than x | floor (9.2) is 9.0  floor ( - 9.8 ) is – 10.0 |

**Using math functions in C programs**

# include <stdio.h>

# include <math.h>

int main (void)

{

printf(“%.2f”, sqrt(900.0);

return 0;

}

**Output**

30.00

**Programmer – defined functions**

# include <stdio.h> int square ( int y ) ; int main ( void )

{

int x;

for (x =1; x <= 10; ++x)

printf(%d “, square( x ));

puts(“”);

}

int square (int y)

{

return y \* y;

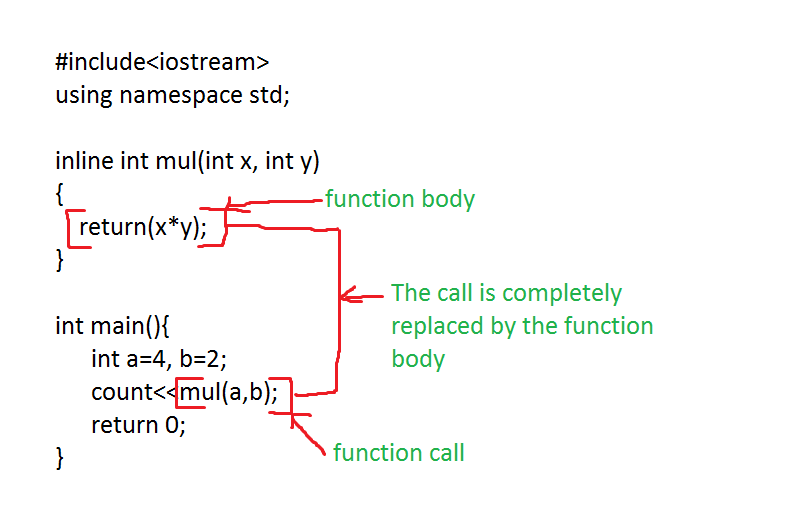
}

**Output**

1 4 9 16 25 36 49 64 81 100

**Calling and Called Functions**

* Functions are invoked by a function call.
* The function which invokes a function is called the calling function or caller.
* The function being activated is referred to as the called function.
* A function prototype gives all of the information needed by the calling function to invoke the called function.
* The function prototype is the declaration of the function and must appear before the function is invoked.



**Local variables**

* All variables defined in function definitions are local variables.
* They can be accessed only in the function in which they are defined.
* Example : int x in the square function

**Parameter List**

* The parameter list is a comma-separated list that specifies the parameters received by the function when it’s called.
* If a function does not receive any values, parameter list is void.
* A type must be listed explicitly for each parameter.
* Example : int y in the square function is a parameter

**Function prototype**

* The compiler uses function prototypes to validate function calls.

Eg: int square (int y)

* The *int* in parenthesis informs the compiler that square expects to receive an integer.
* The int to the left of the function name informs the compiler that square returns an integer result to the caller.

**return statement**

* return statement helps the called function to return a value to the calling function.
* If a function does not return a value, the statement

return;

* If a function does return a result, the statement

return expression;

**The format of a function**

* Function name is any valid identifier
* The return-value-type is the data type of the result returned to caller.
* Definitions and statements within the braces form the function body.
* If a function does not return a value, the return-value-type should be indicated as void.

return-value-type function-name( parameter- list)

{

definitions

statements

}

**Passing Arguments By Value**

* When arguments are passed by value, a copy of the argument’s value is made and passed to the called function.
* Changes to the copy do not affect an original variable’s value in the caller.

**Block Scope**

* The scope of an identifier is the portion of the program in which the identifier can be referenced.
* Identifiers defined inside a block have a block scope.
* Block scope ends at the terminating right brace.
* Local variables defined at the beginning of a function have block scope.
* when blocks are nested and inner and outer blocks both have the same identifier name,identifier in the outer block is hidden until the inner block terminates.

**Block scope example**

{ //start of outer block

int a = 39;

int b = 6;

printf( “a= %d and b= %d \n“, a, b);

{ // start of inner block

float a = 26.25;

int c = 30;

printf(“Now a= %.2f and b= % d and c= %d\n”, a, b, c);

} //end inner block

printf( “Finally a= %d and b = %d \n, a, b);

} // end of outer block

**File scope**

* An identifier declared outside any function has file scope.
* Such identifies are known to all the functions in the program
* Global variables, function definitions and function prototypes has file scope.

**Assert**

* assert.h contains information for adding diagnostics that aid program debugging.
* Assert test the value of an expression at execution time.
* If the value is false (0) , assert print an error message and terminate the program.

**Assert – Example**

* Write a program which print numbers greater than 10.

# include <stdio.h>

# include <assert.h>

int main(void )

{

int x;

printf(“Pls input a number”);

scanf(“%d”, &x);

assert(x >= 10);

printf(“The value of x is %d”, x);

return 0;

}

Output

Pls input a number : 12

The value of x is 12

Pls input a number : 8

Assertion ‘x>=10’ failed