



SLIIT

Discover Your Future

IT1010 – Introduction to Programming

Lecture 6 - Functions



Objectives

- At the end of the Lecture students should be able to
 - Construct programs using functions.
 - Use math functions in C standard library.
 - Use assertions to test whether the values of expressions are correct.

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
<code>sqrt(x)</code>	square root of x	<code>sqrt (900.0)</code> is 30.0
<code>pow(x, y)</code>	x raised to power y (x^y)	<code>pow (2, 7)</code> is 128.0
<code>exp(x)</code>	exponential function e^x	<code>exp (1.0)</code> is 2.718282
<code>log (x)</code>	natural logarithm of x	<code>log(2.718282)</code> is 1.0
<code>ceil (x)</code>	rounds x to the smallest integer not less than x	<code>ceil (9.2)</code> is 10.0 <code>ceil (- 9.8)</code> is – 9.0
<code>floor (x)</code>	rounds x to the smallest integer not greater than x	<code>floor (9.2)</code> is 9.0 <code>floor (- 9.8)</code> 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

```
# include <stdio.h>
# include <math.h>
int main (void)
{
    float c1 = 13;
    float d = 3.0;
    float f = 4.0;
    printf("%.2f", sqrt(c1 + d * f));
    return 0;
}
```

output

5.00

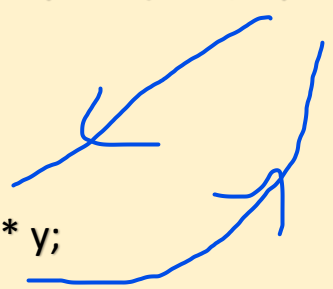
Quiz

- Write the value of x after each of the following statements is performed:
 - a) `x = floor (7.5)`
 - b) `x = floor (0.0)`
 - c) `x = ceil (-6.4)`
 - d) `x = pow (5, 2)`

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;
}
```



1 4 9 16 25 36 49 64 81 100

Calling and Called Functions

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

```
int square ( int y );
```

Function prototype

```
int main ( void )
```

```
{
```

```
    int x;
```

Calling Function /
Caller

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

```
        printf("%d ", square( x ));
```

```
    puts("");
```

```
}
```

Called Function

```
int square (int y)
```

```
{
```

```
    return y * y;
```

```
}
```

- 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.

e.g:

```
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

```
return-value-type function-name( parameter-  
list)  
{  
    definitions  
    statements  
}
```

- 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.

Exercise 1

- Write a function that displays a solid square of asterisks whose side is specified in integer parameter side. For example, if side is 4, the function displays:

```
****  
****  
****  
****
```

Exercise 2

- Write a function called *max* to determine and return the largest of two integers. The integers should be input from the key board in the main program and pass to *max* function.

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.

Passing Arguments By Value - Example

```
# include <stdio.h>
int cubeByValue ( int n );

int main ( void )
{
    int number = 5;

    number = cubeByValue (number);
}
```

number

5

```
int cubeByValue (int n)
{
    return n * n * n;
}
```

n

undefined

Passing Arguments By Value - Example

```
# include <stdio.h>
int cubeByValue ( int n );

int main ( void )
{
    int number = 5;

    number = cubeByValue (number);
}
```

number

5

```
int cubeByValue ( int n )
{
    return n * n * n;
}
```

n

5

Passing Arguments By Value - Example

```
# include <stdio.h>
int cubeByValue ( int n );

int main ( void )
{
    int number = 5;

    number = cubeByValue (number);
}
```

number

5

```
int cubeByValue ( int n )
{
    return n * n * n;
}
```

125

n

5

Passing Arguments By Value - Example

```
# include <stdio.h>
int cubeByValue ( int n );

int main ( void )
{
    int number = 5;

    number = cubeByValue (number);
}
```

Diagram illustrating the state of the variable `number` in the `main` function:

- Before the function call: `number` is 5.
- During the function call: `number` is 125 (highlighted in blue).
- After the function call: `number` is 125.

```
int cubeByValue ( int n )
{
    return n * n * n;
}
```

Diagram illustrating the state of the parameter `n` in the `cubeByValue` function:

- Before the function call: `n` is undefined.
- During the function call: `n` is 5.
- After the function call: `n` is undefined.

Passing Arguments By Value - Example

```
# include <stdio.h>
int cubeByValue ( int n );

int main ( void )
{
    int number = 5;
    number = cubeByValue (number);
}
```

number

125

125

125

```
int cubeByValue ( int n )
{
    return n * n * n;
}
```

n

undefined

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
```


Block scope example

```
{ //start of outer block  
  int a = 39;  
  int b = 6;  
  printf( "a= %d and b= %d \n", a, b);
```

a

39

b

6

a = 39 and b = 6

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
```

a	26.25	b	6	c	30
---	-------	---	---	---	----

a = 39 and b = 6

Now a = 26.25 and b = 6 and c = 30

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
```

a

39

b

6

a = 39 and b = 6

Now a = 26.25 and b = 6 and c = 30

Finally a = 39 and b = 6

File scope

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

```
#include <stdio.h>

int x = 1; // global variable

int main(void )
{
    printf("%d", x);
    return 0;
}
```

output

1

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 1

- 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

Assert – Example 2

- Write a function called `grade()` which takes a mark as a argument and return the grade according to the following table. Write another function called `test_grade()` which contain test cases to debug the `grade()` function.

Marks Range	Grade
0 to 39	F
40 to 59	C
60 to 74	B
75 to 100	A
Mark < 0 and Mark > 100	X

Assert – Example 2 – grade() function

```
char grade(int marks) {  
    char result;  
    if (marks < 0)  
        result = 'X';  
    if (marks < 40)  
        result = 'F';  
    else if (marks < 60)  
        result = 'C';  
    else if (marks < 75)  
        result = 'B';  
    else if (marks <= 100)  
        result = 'A';  
    else  
        result = 'X'; // Error (invalid mark)  
    return result;  
}
```


Assert – Example 2 – test_grade() function

```
void test_grade() {  
    assert(grade(20) == 'F');  
    assert(grade(50) == 'C');  
    assert(grade(70) == 'B');  
    assert(grade(78) == 'A');  
    assert(grade(-10) == 'X');  
    assert(grade(110) == 'X');  
    // boundary conditions  
    assert(grade(0) == 'F');  
    assert(grade(40) == 'C');  
    assert(grade(60) == 'B');  
    assert(grade(75) == 'A');  
    assert(grade(100) == 'A');  
    printf("grade() unit tests passed\n");  
}
```

Assert – Example 2 – main function

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

char grade(int marks);
void test_grade();

int main( void ) {

    test_grade();
    return 0;
}
```

Output

Assertion 'grade(-10) == 'X' failed.

Modify the grade() function as follows and run the program

```
if (marks < 0)
    result = 'X';
else if (marks < 40)
    result = 'F';
.....
.....
```

Summary

- C math library functions
- User-defined functions
- Scope of a variable
- Parameter passing by value
- Assert statement