## Lecture 7.1

Topics

1. Functions – Revisited
2. C Logical Operators

---

## 1. Functions – Revisited

In general, a function

- Is designed to **be reused** as needed during a program execution,
- Should be designed to handle **one task** at a time.

These functions may be grouped and used in another function to perform a more complex task.

When functions are used, the following processes would occur:

i. A called function may or may not receive control (i.e., parameters or argument values) from the calling function.
   - **`printClassInfo()`** is a **called function**. It is called inside `main()`.
   - **`main()`** is calling `printClassInfo()`; so `main()` is a **calling function**.

ii. The called function may return one and only one value to the calling function.

iii. During the execution of a function, it may produce side effects, which change the state of the program. Side effects may involve data from outside of the program, sending data to external files (monitor included), or variable values.

In working with functions, one should complete the following tasks:

- Specifying "**Function Prototype**"
- Implementing "**Function Definition**" (in C of course)

Note that the implementation of a function is also called function definition**.**

Example

```
/**
 *Program Name: cis26L0711.c
 *Discussion:   Functions with no arguments but may
 *              return a value.
 */
#include <stdio.h>

/*Function prototypes*/
void printClassInfo( void );

void printSquare( void );
int squareInt( void );

void printSumTwoInt( void );
int sumTwoInt( void );

/*Application driver*/
int main() {
  int iResult;

  printClassInfo();

  printf( "\nCalling squareInt() : " );
  iResult = squareInt();
```

```
  printf( "\nResult returned by squareInt() : %d\n", iResult );

  printf( "\nCalling sumTwoInt() : " );
  iResult = sumTwoInt();
  printf( "\nResult returned by sumTwoInt() : %d\n", iResult );

  return 0;
}

/*Function definitions*/
/**
 *Function Name: printClassInfo()
 *Description  : Printing the class information
 *Pre          : Nothing (nothing is sent to this function)
 *Post         : None
 */
void printClassInfo( void ) {
  printf( "\n\tCIS 26 : C Programming\n" );
  printf( "\n\tLaney College.\n" );

  return;
}

/**
 *Function Name: printSquare()
 *Description  : Computing and displaying the square of
                 an integer
 *Pre          : Nothing
 *Post         : Displaying square value
 */
void printSquare( void ) {
  int iValue;
  printf( "\n  Computing square of int -- printSquare():\n\
    \tEnter an integer + ENTER: " );
  scanf( "%d", &iValue );

  printf( "\n\tThe square of %d is %d\n", iValue,
          iValue * iValue );

  return;
}

/**
 *Function Name: squareInt()
 *Description  : Computing the square of an integer
 *Pre          : Nothing
 *Post         : Returning square value
 */
int squareInt( void ) {
  int iValue;
  printf( "\n  Computing square of int -- squareInt():\n\
    \tEnter an integer + ENTER: " );
  scanf( "%d", &iValue );

  return ( iValue * iValue );
}

/**
 *Function Name: printSumTwoInt()
 *Description  : Computing and displaying the sum of
                 two integers
 *Pre          : Nothing
 *Post         : None
 */
void printSumTwoInt( void ) {
  int i1;
```

```
  int i2;
  printf( "\n  Computing sum of two int's -- printSumTwoInt():\n\
    \tEnter an integer + ENTER: " );
  scanf( "%d", &i1 );

  printf( "\n\tEnter an integer + ENTER: " );
  scanf( "%d", &i2 );

  printf( "\n\tThe sum of %d and %d is %d\n", i1, i2, i1 + i2 );

  return;
}

/**
 *Function Name: sumTwoInt()
 *Description  : Computing the sum of two integers
 *Pre          : Nothing
 *Post         : Returning sum
 */
int sumTwoInt( void ) {
  int i1;
  int i2;
  printf( "\n  Computing sum of two int's -- sumTwoInt():\n\
    \tEnter an integer + ENTER: " );
  scanf( "%d", &i1 );

  printf( "\n\tEnter an integer + ENTER: " );
  scanf( "%d", &i2 );

  return ( i1 + i2 );
}
```

## OUTPUT

```
        CIS 26 : C Programming


        Laney College.


Calling squareInt() :
  Computing square of int -- squareInt():
        Enter an integer + ENTER: 5


Result returned by squareInt() : 25


Calling sumTwoInt() :
  Computing sum of two int's -- sumTwoInt():
        Enter an integer + ENTER: 6


        Enter an integer + ENTER: -7


Result returned by sumTwoInt() : -1
```

## 2. C Logical Operators

Recall that logical data of values **true** and **false** are used in expressions that involve with **yes/no** answer.

### 2.1 Logical Data

In some programming languages, a data value of zero (**0**) is considered as **false**, and data value of nonzero is considered as **true**.

A numeric representation of the **true-false** behavior is depicted in **Figure 1** below.

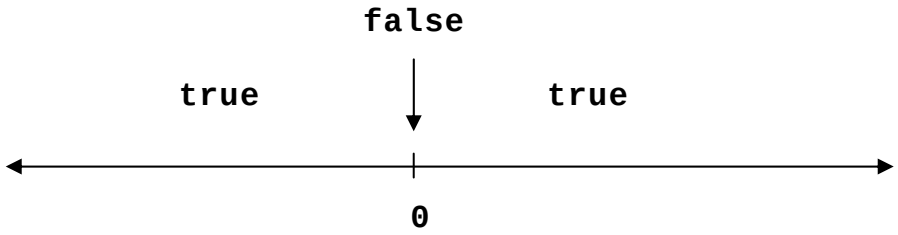


**Figure 1** True-False scale in a numeric representation

Expressions can be written using several logical data and logical operators.

### 2.2 Logical Operators

There are three logical operators of **AND, OR,** and **NOT**. The ranking and meaning of these operators are given in **Table 1** below.

| Operator | Meaning | Rank |
|---|---|---|
| ! | Logical **NOT** | 2 |
| && | Logical **AND** | 11 |
| \|\| | Logical **OR** | 12 |

**Table 1** Logical Operators

### 2.3 Truth Table

The results of logical expressions are also logical value **true-false**. The above three operators will have:

- One single operand as in the **NOT (!)** operator, and
- Two operands for the **AND (&&)** and **OR (||)** operators.

Note that the operators enclosed in parentheses are the operator notations used in C programs.

The truth tables for these logical operators are given below.

**NOT : Logical Expression**

| a | !a |
|---|---|
| T | F |
| F | T |

**NOT : C Programming**

| a | !a |
|---|---|
| 0 | Nonzero |
| Nonzero | 0 |

**AND : Logical Expression**

| a | b | a && b |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

**AND : C Programming**

| a | b | a && b |
|---|---|---|
| Nonzero | Nonzero | Nonzero |
| Nonzero | 0 | 0 |
| 0 | Nonzero | 0 |
| 0 | 0 | 0 |

**OR : Logical Expression**

| a | b | a \|\| b |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

**OR : C++ Programming**

| a | b | a \|\| b |
|---|---|---|
| Nonzero | Nonzero | Nonzero |
| Nonzero | 0 | Nonzero |
| 0 | Nonzero | Nonzero |
| 0 | 0 | 0 |

**Table 2** `True-False` truth tables

Using these logical operators, complex test-expressions can be formed.

For examples,

(1) To test if the integer **`iValue`** is positive **and** even, the expression may be written as

```
( iValue > 0 ) && ( iValue % 2 == 0 )
```

(2) To test if the integer is greater than 10 **and** odd, the expression may be written as

```
( iValue > 10 ) && ( iValue % 2 )
```