7

# C How to Program

## Functions

Dr. mhd. Mazen Al Mustafa

# Lecture 7 - Functions

**Outline**

7.1 **Program Modules in C**

7.2 **Math Library Functions**

**7.3** **What is a function**

**7.4** **Functions**

7.5 **Function Definitions**

7.6 **Function Prototypes**

7.7 **functionsCalling Functions: Call by Value and Call by Reference**

7.8 **Header Files**

7.9 **Passing Arrays to**

# 7.1 Program Modules in C

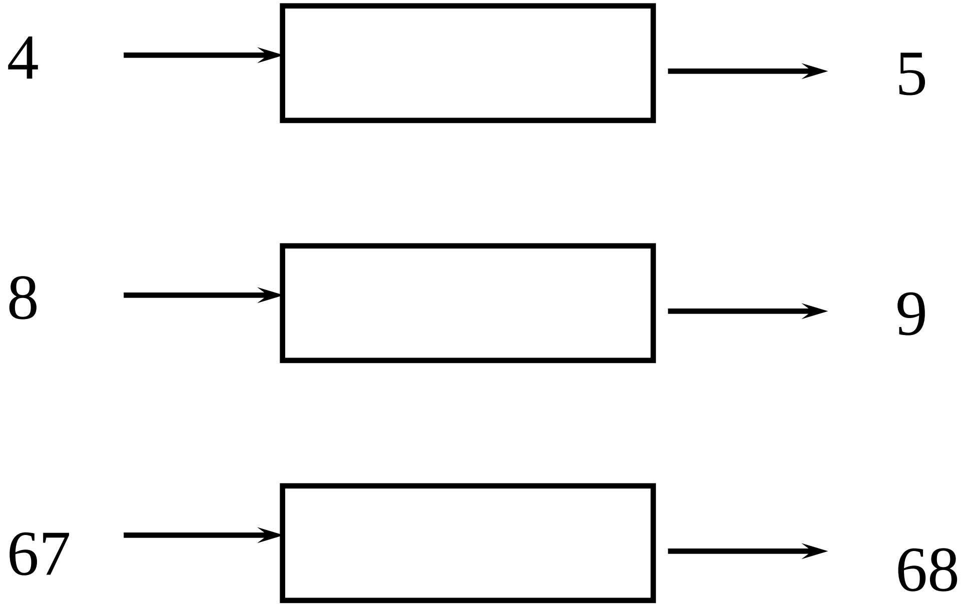
- **Functions**
  - Modules in C
  - Programs combine user-defined functions with library functions
    - C standard library has a wide variety of functions
- **Function calls**
  - Invoking functions
    - Provide function name and arguments (data)
    - Function performs operations or manipulations
    - Function returns results
  - Function call analogy:
    - Boss asks worker to complete task
      - Worker gets information, does task, returns result
      - Information hiding: boss does not know details

# 7.2 Math Library Functions

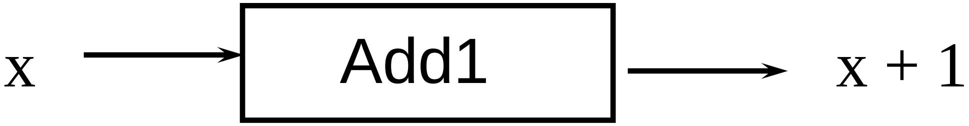
- Math library functions
  - perform common mathematical calculations
  - **#include <math.h>**
- Format for calling functions
  - **FunctionName(** *argument* **);**
    - If multiple arguments, use comma-separated list
  - **printf( "%.2f", sqrt( 900.0 ) );**
    - Calls function **sqrt**, which returns the square root of its argument
    - All math functions return data type **double**
  - Arguments may be constants, variables, or expressions

# 7.3 What is a function?

- **This is an area where computer science has been stolen from the mathematical community.**
- **First Definition (not the final one): A Function is a “black box” that takes input, operates on it, and produces output.**
- **Here’s a simple function:**



# What is a function? (cont)



x

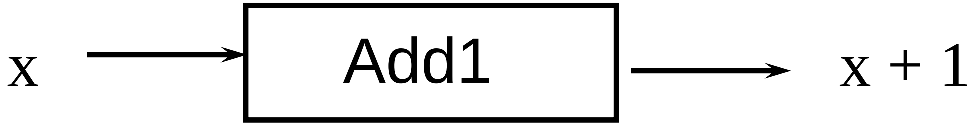
Add1

x + 1

- **Functions have 3 parts:**
  - **The input.**
  - **The output.**
  - **The operation.**

- **Example: The Add1 Function is fully described as follows –**
  - **Input: An Integer, x.**
  - **Output: An Integer, F(x).**
  - **Operation: F(x) = x + 1.**

# What is a function? (cont)



x

Add1

x + 1

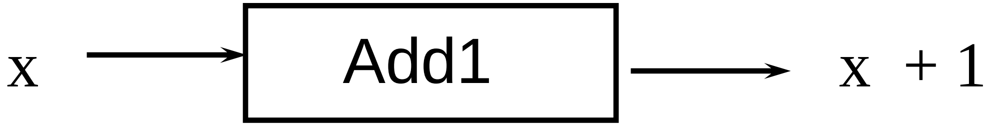
**Function: A function is a “black box” that takes input, operates on it, and produces output. A function is fully defined by three things: input, output, and a description relating the output to the input.**

**Example: The Add1 function:**
**Input - a number, x.**
**Output - a number, F(x)**
**Operation - F(x) = x + 1**

# What is a function? (cont)

- When we speak of functions, we focus on WHAT is being done.
- Functions have three components:
  - The input
  - The output.
  - The description of the operation (this hides the details of the algorithm).

# Examples of Notation



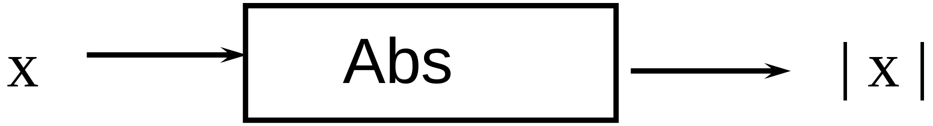
x

Add1

x + 1

- **Typical notation for a function is borrowed from math:**
  - **Add1(0) = 1**
  - **Add1(1) = 2**
  - **Add1(2) = 3, etc**

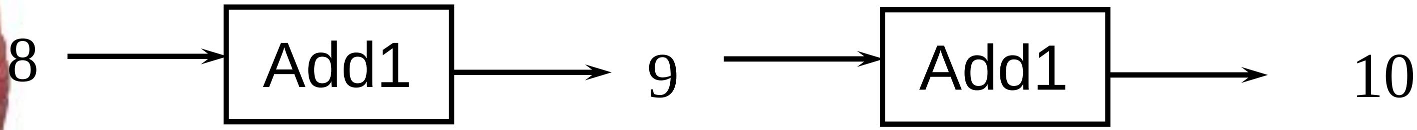
# Another Function Example



- The Abs function takes the absolute value of its input:
  - Input:  An Integer x.
  - Output:  An Integer, F(x).
  - Operation: F(x) = | x |.

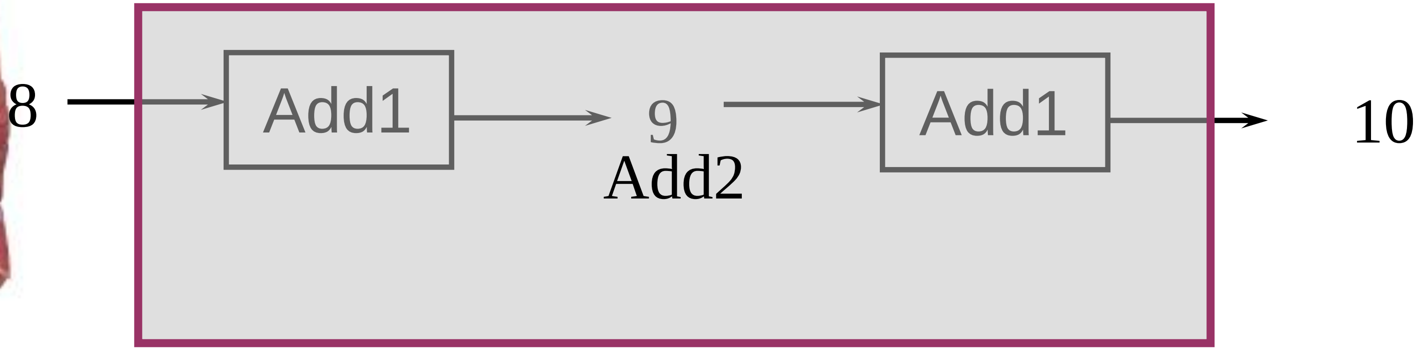
- Example notation and values:
  - Abs(3) = 3.
  - Abs(-3) = 3.

# Function Composition



- What happens when we take the output of one function and use it as the input of another?
- Once again, we steal from math, and use the concept of function composition.
- Function composition occurs when two functions are combined to form (or compose) a new function.

# Function Composition (cont)



- In this case we can take the composition of the two Add1 functions, and describe a new function - Add2.

- Definition of Add2:
  - Input:  A number, X
  - Output: A number, F(x)
  - Operation: F(x) = x + 2

# 7.4 Functions

- **Functions**
  - Modularize a program
  - All variables declared inside functions are local variables
    - Known only in function defined
  - **Parameters**
    - Communicate information between functions
    - Local variables

# Why are Functions Important?

- Benefits of functions
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoid code repetition
  - Using the concept of functions allows us to relate mathematics to computation.
    - By treating computer programs as functions we can draw on mathematical principles to discuss properties of programs.
  - We can build large programs by composing small functions.
  - We can describe large programs by describing the functions that make them up and how they are composed.

# Algorithm **for the Add1 Function: Add1()**

- **Inputs:** **Number x**
- **Other Data:** **None**
- **Initialization:** **X is given to us**
- **Computation: X++;**
- **Outputs:** **return (x);**

# 7.5 Function Definitions

- Function definition format

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

  – Function-name: any valid identifier

  – Return-value-type: data type of the result (default **int**)

    - **void** – indicates that the function returns nothing

  – Parameter-list: comma separated list, declares parameters

    - A type must be listed explicitly for each parameter unless, the parameter is of type **int**

# Function Definitions (Cont.)

- **Function definition format (continued)**

  *return-value-type function-name( parameter-list )*<br>
  **{**<br>
  *declarations and statements*<br>
  **}**

  - Declarations and statements: function body (block)
    - Variables can be declared inside blocks (can be nested)
    - Functions can not be defined inside other functions
  - Returning control
    - If nothing returned
      - **return;**
      - or, until reaches right brace
    - If something returned
      - **return** *expression***;**

# How do they look as Functions?

- Remember that functions are concerned with input and output.

- When we write the function that corresponds to an algorithm, we need to give the function some input.
  - The "thing" inside the bracket does that.
  - Can be a constant or a variable or another function.
- We need to store the output in memory somewhere.
  - The assignment operation does that ("=").

- Add1(x)

  As used in a program:

  y = Add1(4);     <= this will set y to 5

  y = Add1(x);     <= this will take whatever value is stored at location 'x' and add 1 to it, then store the result in location 'y'.

# Algorithm to Find Area of a Circle: CircleArea()

- **Inputs: Number radius;**
- **Other Data: Number area; Number pi;**
- **Initialization: Assume radius is given to us.**
  **pi = 3.14159;**
- **Computation: area = pi * radius * radius**
- **Outputs: return (area);**

# Algorithm to Find Circumference of a Circle: CircleCircumference()

- **Inputs: Number radius;**
- **Other Data: Number circ; Number pi;**
- **Initialization: Assume radius is given to us.**
  **pi = 3.14159;**
- **Computation: circ = 2*pi * radius;**
- **Outputs: return (circ);**

# Now use the Circle Algorithms as Functions: CircleProps()

- **Inputs: Number radius;**
- **Other Data: Number area; Number circ; Number pi;**
- **Initialization: Assume radius is given to us.**
  **pi = 3.14159;**
- **Computation: area = CircleArea( radius);**
  **circ = CircleCircumference(radius);**
- **Outputs: return (area, circ );**

```c
/* Fig. 5.4: fig05_04.c
   Finding the maximum of three integers */
#include <stdio.h>

int maximum( int, int, int );   /* function prototype */

int main()
{
   int a, b, c;

   printf( "Enter three integers: " );
   scanf( "%d%d%d", &a, &b, &c );
   printf( "Maximum is: %d\n", maximum( a, b, c ) );

   return 0;
}

/* Function maximum definition */
int maximum( int x, int y, int z )
{
   int max = x;

   if ( y > max )
      max = y;

   if ( z > max )
      max = z;

   return max;
}
```

```
Enter three integers: 22 85 17
Maximum is: 85
```

1. Function prototype (3 parameters)

2. Input values

2.1 Call function

3. Function definition

# 7.6 Function Prototypes

- **Function prototype**
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default **int**)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program
  - The function with the prototype

    **int maximum( int, int, int );**

    - Takes in 3 **int**s
    - Returns an **int**
- Promotion rules and conversions
  - Converting to lower types can lead to errors

# 7.7 Calling Functions: Call by Value and Call by Reference

- **Used when invoking functions**
- **Call by value**
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- **Call by reference**
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- **For now, we focus on call by value**

# 7.8 Header Files

- Header files
  - Contain function prototypes for library functions
  - **<stdlib.h>** , **<math.h>** , etc
  - Load with **#include <filename>**

    **#include <math.h>**
- Custom header files
  - Create file with functions
  - Save as **filename.h**
  - Load in other files with **#include "filename.h"**
  - Reuse functions

# 7.9 Passing Arrays to Functions

- Passing arrays
  - To pass an array argument to a function, specify the name of the array without any brackets

    ```
    int myArray[ 24 ];
    myFunction( myArray, 24 );
    ```

    - Array size usually passed to function
  - Arrays passed call-by-reference
  - Name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

# Passing Arrays to Functions

- Function prototype

```
void modifyArray( int b[], int
 arraySize );
```

  - Parameter names optional in prototype
    - **`int b[]`** could be written **`int []`**
    - **`int arraySize`** could be simply **`int`**

```c
/* Fig. 6.13: fig06_13.c
   Passing arrays and individual array elements to functions */
#include <stdio.h>
#define SIZE 5

void modifyArray( int [], int );  /* appears strange */
void modifyElement( int );

int main()
{
   int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;

   printf( "Effects of passing entire array call "
           "by reference:\n\nThe values of the "
           "original array are:\n" );

   for ( i = 0; i <= SIZE - 1; i++ )
      printf( "%3d", a[ i ] );

   printf( "\n" );
   modifyArray( a, SIZE );  /* passed call by reference */
   printf( "The values of the modified array are:\n" );

   for ( i = 0; i <= SIZE - 1; i++ )
      printf( "%3d", a[ i ] );

   printf( "\n\n\nEffects of passing array element call "
           "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
   modifyElement( a[ 3 ] );
   printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
   return 0;
}
```

Entire arrays passed call-by-reference, and can be modified

Array elements passed call-by-value, and cannot be modified

1. Function definitions

2. Pass array to a function

2.1 Pass array element to a function

3. Print

```c

void modifyArray( int b[], int size )
{
   int j;

   for ( j = 0; j <= size - 1; j++ )
      b[ j ] *= 2;
}

void modifyElement( int e )
{
   printf( "Value in modifyElement is %d\n", e *= 2 );
}
```

3.1 Function definitions

```
Effects of passing entire array call by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8

Effects of passing array element call by value:


The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

Program Output

```c
/* Fig. 6.22: fig06_22.c
   Double-subscripted array example */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

int minimum( const int [][ EXAMS ], int, int );
int maximum( const int [][ EXAMS ], int, int );
double average( const int [], int );
void printArray( const int [][ EXAMS ], int, int );

int main()
{
   int student;
   const int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
           minimum( studentGrades, STUDENTS, EXAMS ),
           maximum( studentGrades, STUDENTS, EXAMS ) );

   for ( student = 0; student <= STUDENTS - 1; student++ )
      printf( "The average grade for student %d is %.2f\n",
              student,
              average( studentGrades[ student ], EXAMS ) );

   return 0;
}
```

Each row is a particular student, each column is the grades on the exam.

3. Define functions

```

/* Find the minimum grade */
int minimum( const int grades[][ EXAMS ],
             int pupils, int tests )
{
   int i, j, lowGrade = 100;

   for ( i = 0; i <= pupils - 1; i++ )
      for ( j = 0; j <= tests - 1; j++ )
         if ( grades[ i ][ j ] < lowGrade )
            lowGrade = grades[ i ][ j ];

   return lowGrade;
}

/* Find the maximum grade */
int maximum( const int grades[][ EXAMS ],
             int pupils, int tests )
{
   int i, j, highGrade = 0;

   for ( i = 0; i <= pupils - 1; i++ )
      for ( j = 0; j <= tests - 1; j++ )
         if ( grades[ i ][ j ] > highGrade )
            highGrade = grades[ i ][ j ];

   return highGrade;
}

/* Determine the average grade for a particular exam */
double average( const int setOfGrades[], int tests )
{
```

```c
   int i, total = 0;

   for ( i = 0; i <= tests - 1; i++ )
      total += setOfGrades[ i ];

   return ( double ) total / tests;
}

/* Print the array */
void printArray( const int grades[][ EXAMS ],
                 int pupils, int tests )
{
   int i, j;

   printf( "                 [0]  [1]  [2]  [3]" );

   for ( i = 0; i <= pupils - 1; i++ ) {
      printf( "\nstudentGrades[%d] ", i );

      for ( j = 0; j <= tests - 1; j++ )
         printf( "%-5d", grades[ i ][ j ] );
   }
}
```

3. Define functions

## Program Output

```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```