ICS103 Programming in C

Lecture 5: Introduction to Functions

Outline

- Introduction to Functions
- Predefined Functions and Code Reuse
- User-defined void Functions without Arguments
  - Function Prototypes
  - Function Definitions
  - Placement of Functions in a Program
  - Program Style

Introduction to Functions

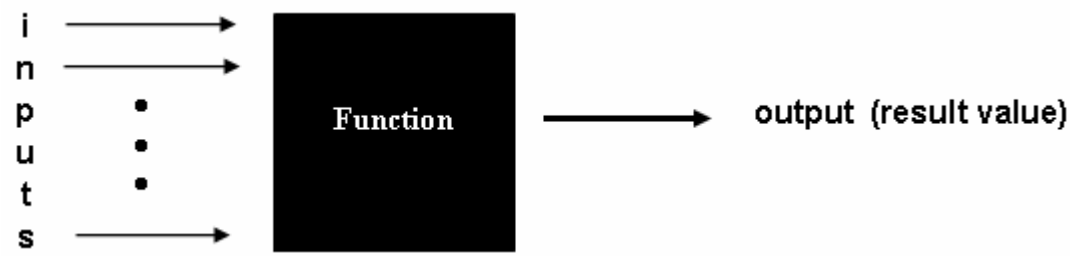
- So far, we have learnt how to use operators, +, -, *, / and % to form simple arithmetic expressions.
- However, we are not yet able to write many other mathematical expressions we are used to.
- For example, we cannot yet represent any of the following expressions in C:

$\sqrt{x}$ $\quad |q+z| \quad \left(\frac{h}{12.3}\right)^3 \quad \log m$

- C does not have operators for “square root”, “absolute value”, sine, log, etc.
- Instead, C provides program units called functions to carry out these and other mathematical operations.

Introduction to Functions …

- A function can be thought of as a black box that takes one or more input arguments and produces a single output value.



- For example, the following shows how to use the sqrt function that is available in the standard math library:

```
y = sqrt (x);
```

- If x is 16, the function computes the square root of 16. The result, 4, is then assigned to the variable y.
- The expression part of the assignment statement is called function call.
- Another example is: z = 5.7 + sqrt (w);

  If w = 9, z is assigned 5.7 + 3, which is 8.7.

Example

```
/* Performs three square root computations  */
#include <stdio.h> /* definitions of printf, scanf */
#include <math.h>  /* definition of sqrt */

int main(void)  {
      double first, second, /* input - two data values       */
            first_sqrt,    /* output - square root of first input */
            second_sqrt,   /* output - square root of second input */
            sum_sqrt;      /* output - square root of sum     */

      printf("Enter a number> ");
      scanf("%lf", &first);
      first_sqrt = sqrt(first);
      printf("Square root of the number is %.2f\n", first_sqrt);

      printf("Enter a second number> ");
      scanf("%lf", &second);
      second_sqrt = sqrt(second);
      printf("Square root of the second number is %.2f\n", second_sqrt);

      sum_sqrt = sqrt(first + second);
      printf("Square root of the sum of the 2 numbers is %.2f\n",sum_sqrt);
      return (0);
}
```

Predefined Functions and Code Reuse

- The primary goal of software engineering is to write error-free code.
- Reusing code that has already been written & tested is one way to achieve this. --- ”Why reinvent the wheel?”
- C promotes reuse by providing many predefined functions. e.g.
  - Mathematical computations.
  - Input/Output: e.g. `printf`, `scanf`

Predefined Functions and Code Reuse …

- The next slide lists some commonly used mathematical functions (Table 3.6 in the book)
- Appendix B gives a more extensive lists of standard library functions.
- In order to use a function, you must use `#include` with the appropriate library.
  - Example, to use function `sqrt` you must include `math.h`.
- If a functions is called with a numeric argument that is not of the argument type listed, the argument value is converted to the required type before it is used.
  - Conversion of type `int` to type `double` cause no problems
  - Conversion of type `double` to type `int` leads to the loss of any fractional part.
- Make sure you look at documentation for the function so you use it correctly.

Some Mathematical Library Functions

Function	Header File	Purpose	Arguments	Result
abs(x)	<stdlib.h>	Returns the absolute value of its integer argument x.	int	int
sin(x),cos(x), tan(x)	<math.h>	Returns the sine, cosine, or tangent of angle x.	double (in radians)	double
log(x)	<math.h>	Returns the natural log of x.	double (must be positive)	double
Log10(x)	<math.h>	Returns base 10 log of x	Double (positive)	double
pow(x, y)	<math.h>	Returns x^y	double, double	double
sqrt(x)	<math.h>	$\sqrt{x}$	double (must be positive)	double

Example

- We can use C functions ***pow*** and ***sqrt*** to compute the roots of a quadratic equation in x of the form:

$$ax^2 + bx + c = 0$$

- If the discriminant ($b^2 - 4ac$) is greater than zero, the two roots are defined as:

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- In C, these two roots are computed as:

```
/* compute two roots, root_1 and root_2, for disc > 0.0 */
disc = pow(b, 2) - 4 * a * c;
root_1 = (-b + sqrt(disc)) / (2 * a);
root_2 = (-b - sqrt(disc)) / (2 * a);
```

Simple User-defined Functions

- An advantage of using **predefined** functions is that the programmer needs to be concerned only with **what** the function does but not **how** it does it.
- In complex software systems, this principle of separating what from the how is an important aspect of managing the complexity of programs.
- C provides a mechanism for the programmer to define his own functions with the same advantages as the C's library functions.
- We now study the simplest type of user-defined functions – those that display one or more lines of output.
- These are useful for tasks such as displaying instructions to the user on how to use a program.

Example …

```
/* Performs three square root computations  */
#include <stdio.h> /* definitions of printf, scanf */
#include <math.h>  /* definition of sqrt */

void instruct (void); //displays user instruction

int main(void)  {


      double first, second, /* input - two data values         */
             first_sqrt,    /* output - square root of first input */
             second_sqrt,   /* output - square root of second input */
             sum_sqrt;      /* output - square root of sum      */

       /* Display instrctions */
      instruct();

       printf("Enter a number> ");
       scanf("%lf", &first);
       first_sqrt = sqrt(first);
       printf("Square root of the number is %.2f\n", first_sqrt);

        // continue next slide …
```

Example

```
// continue from previous slide

        printf("Enter a second number> ");
        scanf("%lf", &second);
        second_sqrt = sqrt(second);
        printf("Square root of the second number is %.2f\n", second_sqrt);

        sum_sqrt = sqrt(first + second);
        printf("Square root of the sum of the 2 numbers is %.2f\n",sum_sqrt);
        return (0);
} // end of main function

/* displays user instructions */
void instruct(void) {
    printf("This program demostrates the use of the \n");
    printf("math library function sqrt (square root). \n");
    printf("you will be asked to enter two numbers -- \n");
    printf("the program will display the square root of \n");
    printf("each number and the square root of their sum. \n\n");
}
```

Function Prototypes

- Like other identifiers in C, a function must be declared before it can be used in a program.
- To do this, you can add a **function prototype** before `main` to tell the compiler what functions you are planning to use.
- A function prototype tells the C compiler:
  1. The data type the function will return
     - For example, the `sqrt` function returns a type of double.
  2. The function name
  3. Information about the arguments that the function expects.
     - The `sqrt` function expects a double argument.
- So the function prototype for `sqrt` would be:

```
double sqrt(double);
```

Function Prototypes : void Functions

- `void instruct(void);` is a void function
  - **Void function -** does not return a value
    - The function just does something without communicating anything back to its caller.
  - If the arguments are void as well, it means the function doesn't take any arguments.
- Now, we can understand what our main function means:

```
int main(void)
```

- This means that the function `main` takes no arguments, and returns an `int`

Function Definition

- The prototype tells the compiler what arguments the function takes and what it returns, but not what it does.
- We define our own functions just like we do the `main` function
  - **Function Header –** The same as the prototype, except it is not ended by the symbol ;
  - **Function Body –** A code block enclosed by {}, containing variable declarations and executable statements.
- In the function body, we define what actually the function does
  - In this case, we call `printf` 5 times to display user instructions.
  - Because it is a void function, we can omit the return statement.
- Control returns to `main` after the instructions are displayed.

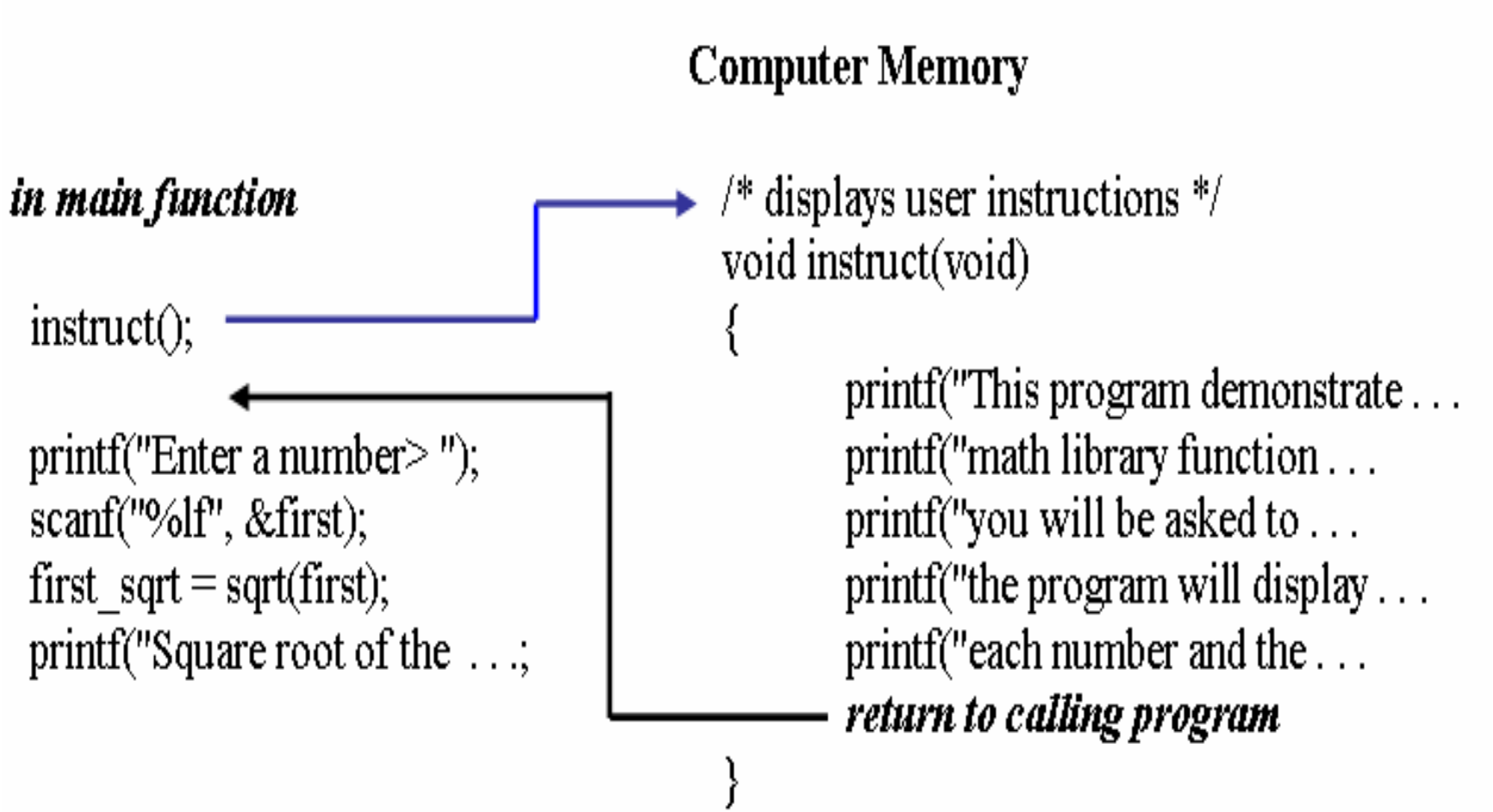
Placement of Functions in a program

- In general, we will declare all of our function prototypes at the beginning (after `#include` or `#define`)
- This is followed by the `main` function
- After that, we define all of our functions.
- However, this is just a convention.
- As long as a function's prototype appears before it is used, it doesn't matter where in the file it is defined.
- The order we define them in does not have any impact on how they are executed

Execution Order of Functions

- Execution order of functions is determined by the order of execution of the function call statements.
- Because the prototypes for the function subprograms appear before the `main` function, the compiler processes the function prototypes before it translates the `main` function.
- The information in each prototype enables the compiler to correctly translate a call to that function.
- After compiling the `main` function, the compiler translates each function subprogram.
- At the end of a function, control always returns to the point where it was called.

Flow of Control Between the main Function and a Function Subprogram



Program Style

- Each function should begin with a comment that describes its purpose.
- If the function subprograms were more complex, we would include comments on each major algorithm step just as we do in function `main`.
- It is recommended that you put prototypes for all functions at the top, and then define them all after `main`.