Fundamentals of Programming CS-114

Lecture 6

Function

module

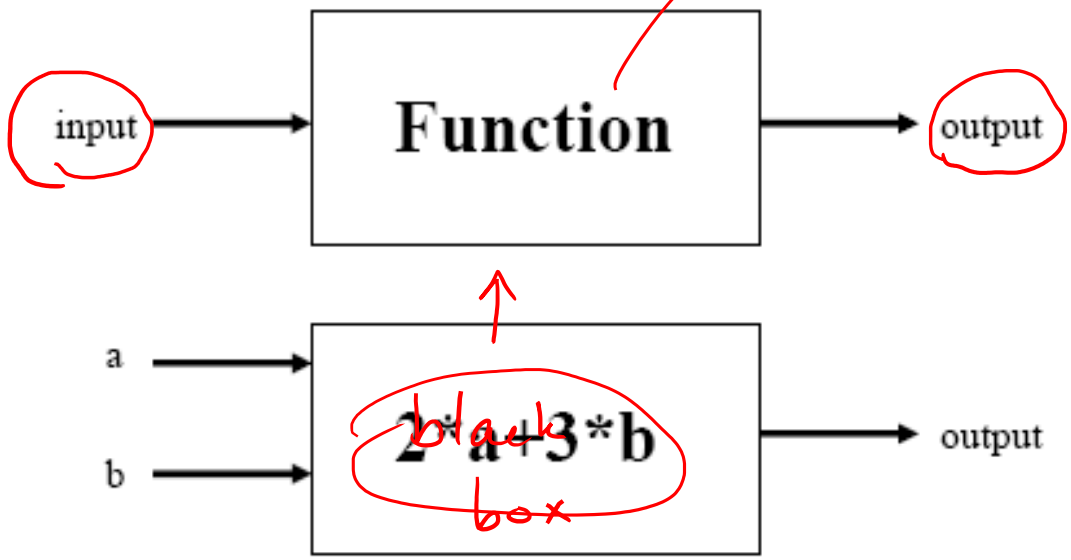
int main( )
{

}

Calculator ( )
{

}

Functions



Output = f(a,b)

Motivation for using Functions

- The divide-and-conquer approach
  - It makes program development more manageable by constructing programs from small, simple pieces.
- Software reusability
  - Using existing functions as building blocks to create new programs.

Motivation for using Functions

- Avoids repeating code.
- Dividing a program into meaningful functions makes the program easier to debug and maintain.

Functions

- Every function should be limited to performing a single, well-defined task
  - E.g sum and difference operations should make 2 functions, not one.
- Name of the function should be sensible. Such functions make programs easier to write, test, debug and maintain.
  - Calling your functions “func” or “A” will make your program listing difficult to understand and follow.

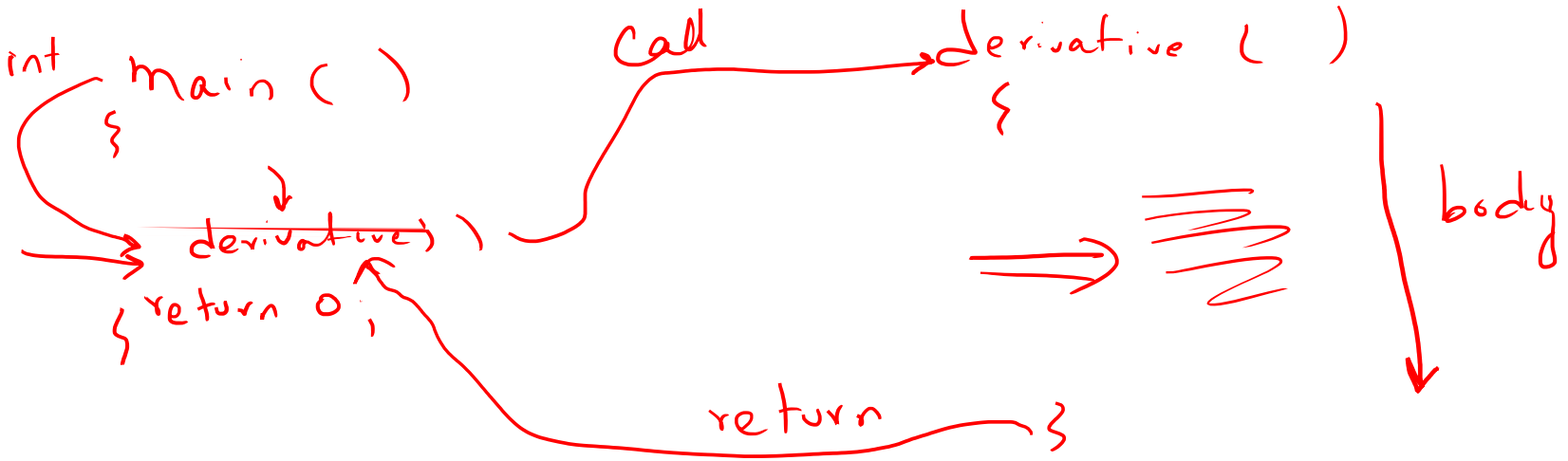
Functions

- Any portion of the code that carries out a specific task or is to be repeated several times can be converted into a function.
- For example, the summing operation takes 2 inputs and returns their sum as output.
  - sum() function can be called every time to calculate the sum

- Programs written by
  - combining new functions with “prepackaged” functions in the C++ standard library.
  - The standard library provides a rich collection of functions.
    - Input/output (getline, read etc.)
    - Common mathematical calculations (pow, sqrt, exp, etc.)
    - String manipulations(strcpy, strcat, etc.
    - Many more

- Functions are invoked by a function call
  - A function call specifies the function name and provides information (as arguments) that the called function needs
  - Boss to worker analogy:

    A boss (the calling function or caller) asks a worker (the called function) to perform a task and return (i.e., report back) the results when the task is done.



- Function definitions
  - Only written once
  - These statements are hidden from other functions.
  - Boss to worker analogy:

    The boss does not know how the worker gets the job done; he just wants it done

Math Library Functions

- Math library functions
  - Allow the programmer to perform common mathematical calculations
  - Are used by including the header file **<math.h>**
- Functions called by writing

  functionName (*argument*)
- Example

```
cout << sqrt( 900.0 );
```

  - Calls the **sqrt** (square root) function. The preceding statement would print **30**
  - The **sqrt** function takes an argument of type **double** and returns a result of type **double**, as do all functions in the math library

Math Library Functions

- Function arguments can be
  - Constants

```
sqrt( 4 );
```

  - Variables

```
sqrt( x );
```

  - Expressions

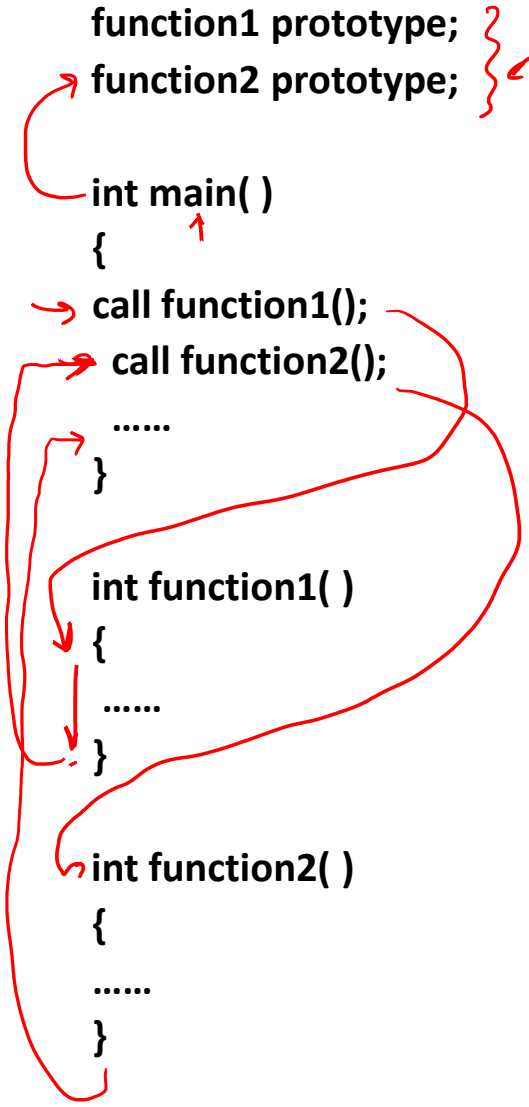
```
sqrt( sqrt( x ) );
sqrt( 3 - 6x );
```

Components of a Function

- Prototype or Function Declaration ← 1
- Function Call ← 2
- Function Definition or Function Body ← 3

derivative ( )

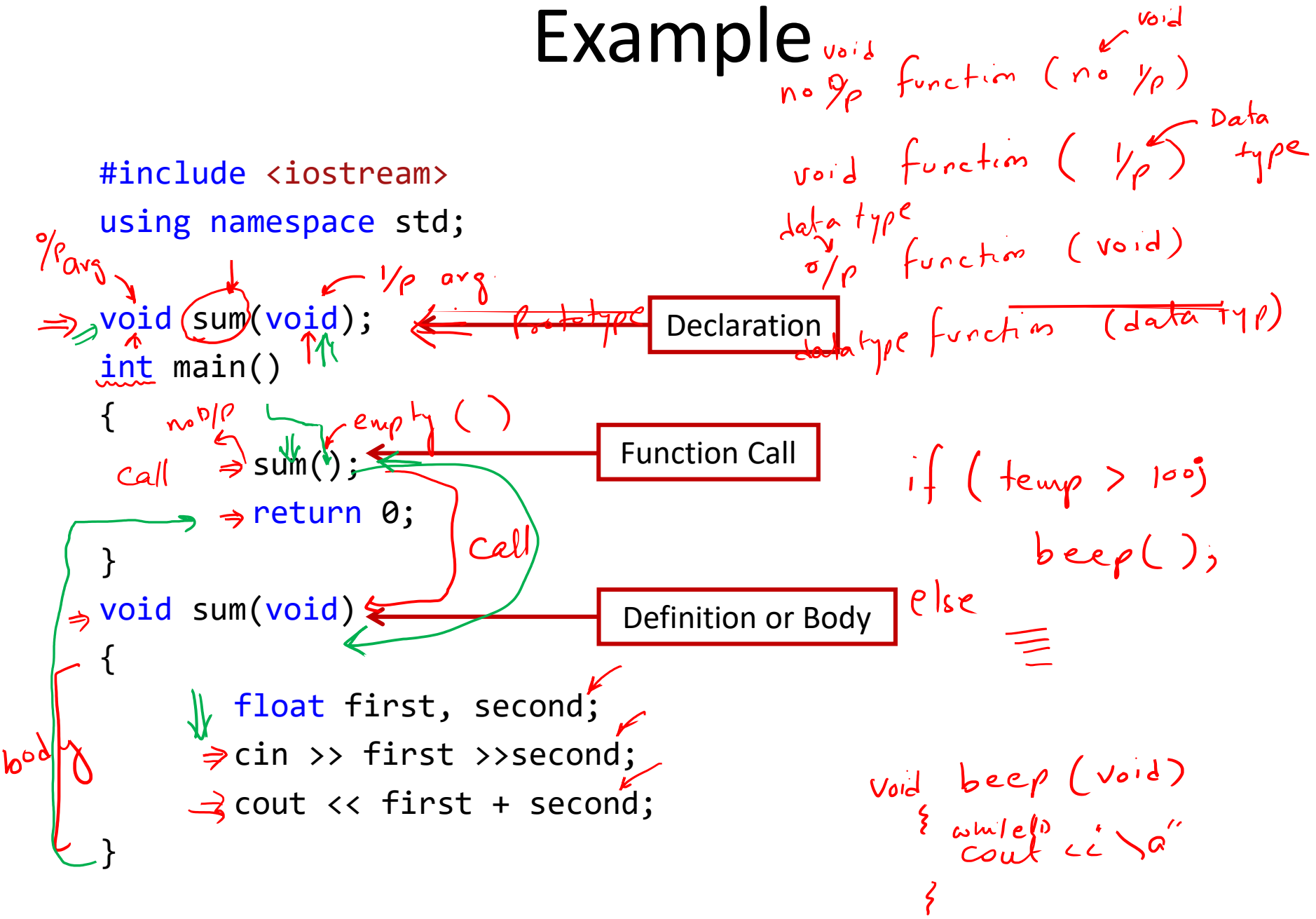
```
Preprocesssor lines

function1 prototype;          //function prototype
function2 prototype;

int main( )
{
call function1();             //function call
 call function2();
  ……
}

int function1( )              //function definition
{                             //function body
 ……
}

int function2( )
{
……
}
```



Functions

- Function Prototype (Declaration)
  - Tells the compiler that a function of this kind is defined somewhere in the program
- Function Call
  - Takes the program control to the called function
- Function Header and Body
  - Contains the body of the function i.e. all the commands that make up the function

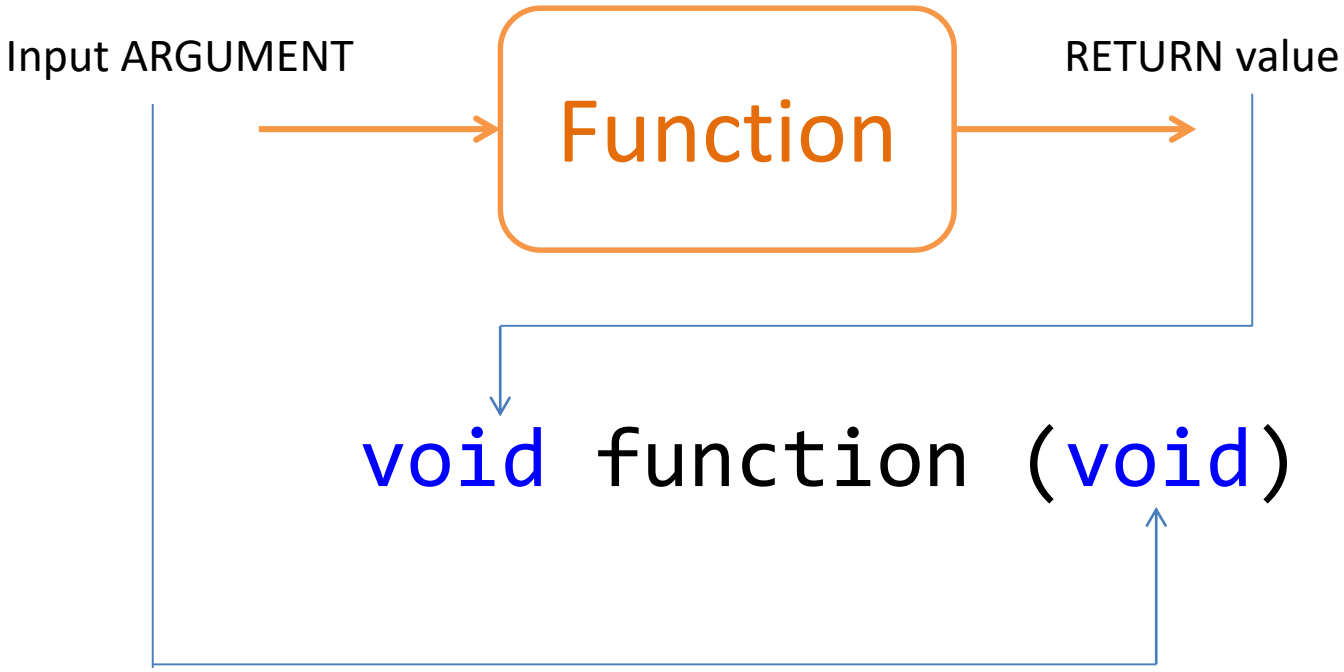
Example


#include <iostream>
using namespace std;
void sum(void);
int main()
{
sum();
return 0;
}
void sum(void)
{
float first, second;
cin >> first >>second;
cout << first + second;
}
Declaration
Function Call
Definition or Body
call
call
body
o/p arg
i/p arg
no i/p
empty ( )
void no o/p function (no i/p)
void function (i/p) Data type
data type o/p function (void)
data type function (data typ)
if (temp > 100)
beep( );
else
Void beep (void)
{ while
cout << "\a"
}

Types of Functions

- Functions with no input and no output
- Functions with input but no output
- Functions with no input but output
- Functions with input and output

Functions



Fns with no Input/Output


```
void function (void)
```

- A void function does not return any value (no output)
- It does not take any parameters
- It performs a specific task .

Fns with no Input/Output

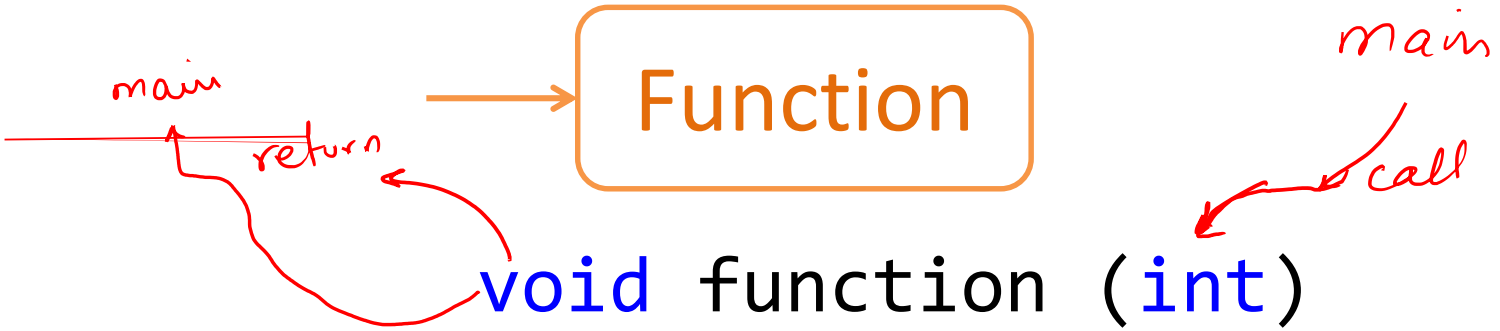
- It may use data stored internally within the function body or only display the number of times the function is called

- An example is a ***Beep( )*** function (inbuilt) that sounds an audible alarm to alert the user. It needs no input parameters and returns no o/p.

Example – No Input/Output

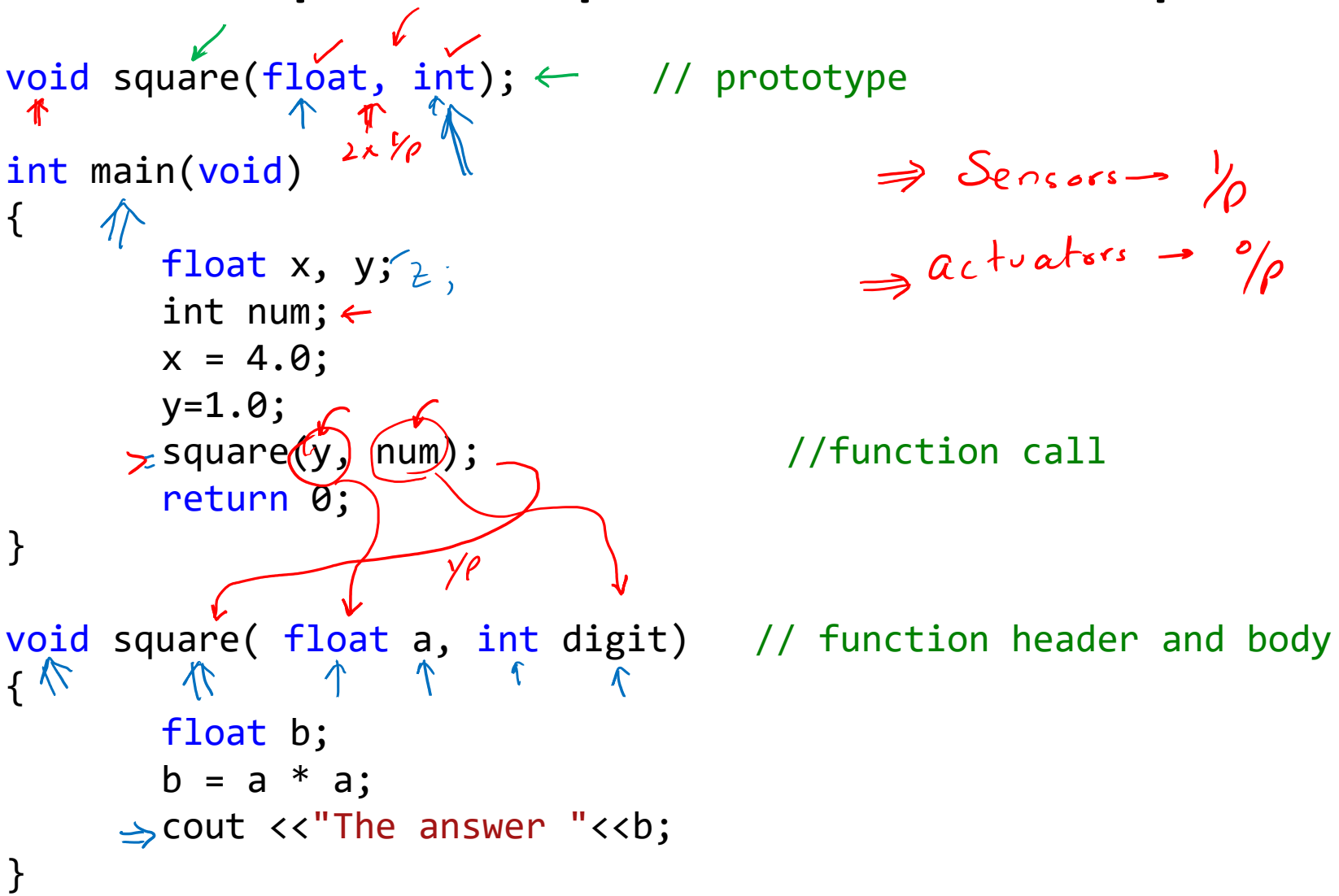
```
void errormessage(void); // prototype

int main()
{
     int x, a=0;
     while(a<10)
     {
          cout << "press 7";
          cin >> x;
          if(x!=7)
           errormessage(); // function call
          a++;
     }
     return 0;
}

void errormessage(void) // function header and body
{
     cout <<" You have entered wrong key";
}
```

Fns with Input but no Output



- Such a function does not return any value.
- However, it takes input parameters
- It performs a specific task.
- For example, it may perform some operation on the input data and simply display it on the screen.

Example – Input but no Output



```
void square(float, int);    // prototype

int main(void)
{
        float x, y;
        int num;
        x = 4.0;
        y=1.0;
        square(y, num);          //function call
        return 0;
}

void square( float a, int digit)   // function header and body
{
        float b;
        b = a * a;
        cout <<"The answer "<<b;
}
```

Fns with Output but no Input


```
int function (void)
```

- Such a function returns a value
- However it does not takes input parameters
- It performs a specific task.
- For example, it may take input values from user inside the function body and return the result of the operation to main().

Example – Output but No Input

```
float sum(void);// prototype

int main (void)
{
        float add;
        add=sum();    // function call
        cout <<"sum = " <<add;
        return 0;
}

float sum(void)     // function header and body
{
        float first, second, ans;
        cin >> first >> second;
        ans = first + second;
        return ans;
}
```

Fns with both Input and Output


```
int function (int)
```

- Such a function returns a value
- It also takes input parameter(s)
- It performs a specific task.
- For example, it may take input values from main() and return the result of the operation to main().

Example – Both Input and Output

```
float sum (float, float);       // prototype

int main (void)
{
      float first, second, add;
      first = 23.45;
      second = 87.555;
      add=sum (first, second);     // function call
      cout <<"sum = " << add;
      return 0;
}

float sum (float x, float y )  //function header and body
{
      float a;
      a = x+y;
      return a;
}
```