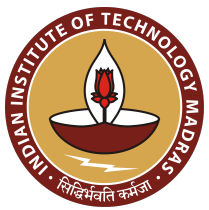
Returning Values From Functions

Saurav Samantaray

Department of Mathematics

Indian Institute of Technology Madras

September 18, 2024



Returning Values

- When a function completes its execution, it can return a single value to the calling program.
- When a function returns a value, the data type of this value must be specified.
- See ”convert.cpp”
- The function declaration does this by placing the data type, `float` in this case, before the function name in the declaration and the definition.
- Functions in programs which return no value, so the return type would be `void`.
- In the CONVERT program, the function `lbstokg()` (pounds to kilo- grams, where lbs means pounds) returns type `float`, so the declaration is

```
float lbstokg(float);
```

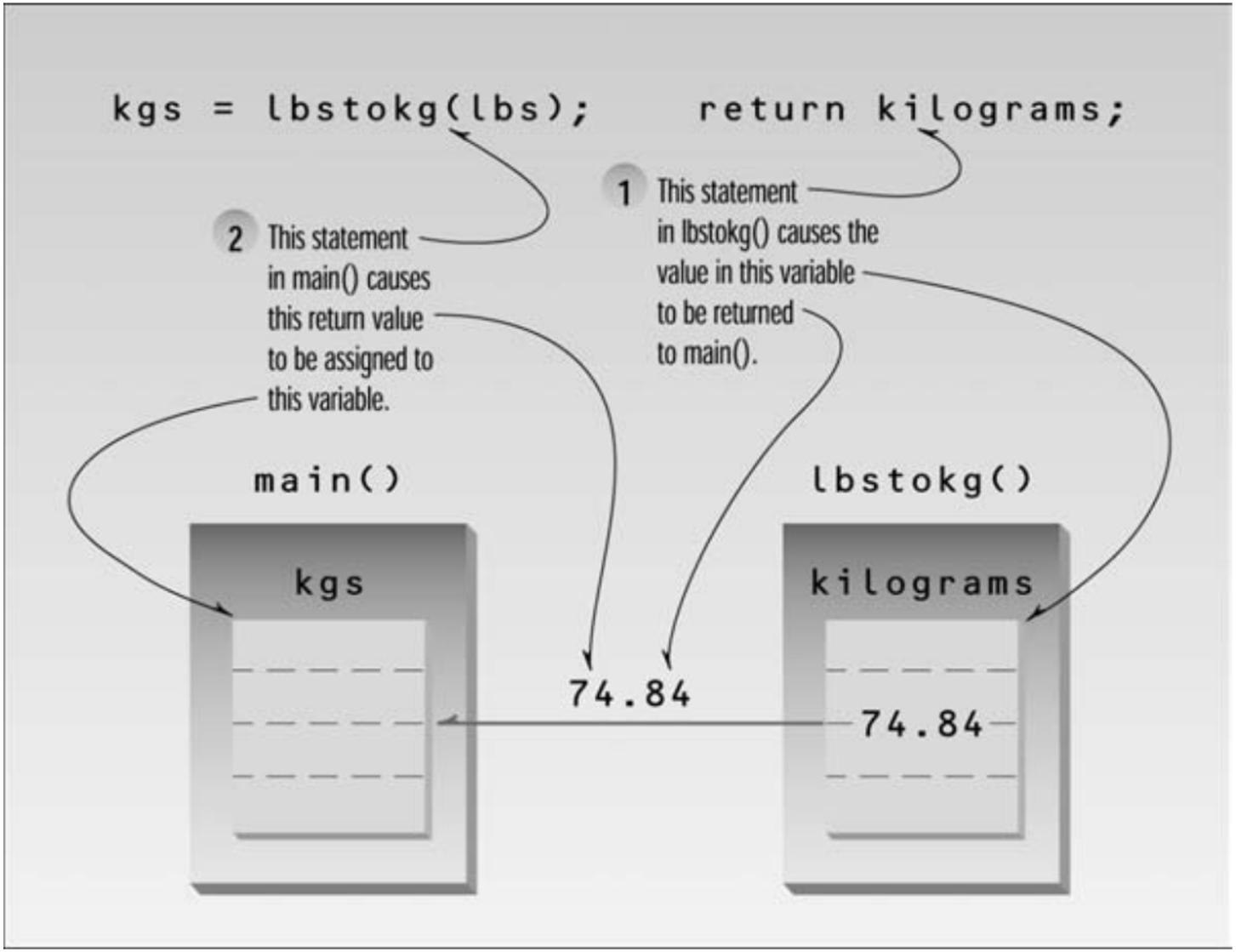
Returning Values

- The first `float` specifies the return type.
- The `float` in parentheses specifies that an argument to be passed to `lbstokg()` is also of type `float`.
- When a function returns a value, the call to the function
  `lbstokg(lbs)`
  is considered to be an expression that takes on the value returned by the function.
- We can treat this expression like any other variable;
  `kgs = lbstokg(lbs);`
- This causes the variable `kgs` to be assigned the value returned by `lbstokg()`.

The return Statement

- The function `lbstokg()` is passed an argument representing a weight in pounds, which it stores in the parameter pounds.
- It calculates the corresponding weight in kilograms by multiplying this pounds value by a constant; the result is stored in the variable `kilograms`.
- The value of this variable is then returned to the calling program using a return statement:
  `return kilograms;`
- Notice that both `main()` and `lbstokg()` have a place to store the `kilogram` variable:
  - `kgs` in `main()`, and
  - `kilograms` in `lbstokg()`.
- When the function returns, the value in `kilograms` is copied into `kgs`.
- The calling program does not access the `kilograms` variable in the function; only the value is returned.

Returning Values



Returning Values

- While many arguments may be sent to a function, only one argument may be returned from it.
- This is a limitation when we need to return more information.
- There are other approaches to returning multiple variables from functions:
  - One is to pass arguments by reference
  - Another is to return a structure with the multiple values as members
- We should always include a function's return type in the function declaration.
- If the function doesn't return anything, use the keyword `void` to indicate this fact.
- If you don't use a return type in the declaration, the compiler will assume that the function returns an `int` value.
- For example, the declaration
  `somefunc();` // declaration
  – assumes return type is `int` tells the compiler that `somefunc()` has a return type of `int`.
- In practice, we shouldn't take advantage and always specify the return type explicitly, even if it actually is `int`.
- This keeps the listing consistent and readable.

Returning Structure Variables

- Structures can be used as arguments to functions.
- We can also use them as return values.

Returning by Reference

- Besides passing values by reference, we can also return a value by reference.
- Why you would want to do this may seem obscure.
- A reason is to allow us to use a function call on the left side of the equal sign.
- This is a somewhat bizarre concept, so let's look at an example.
- See "retref.cpp".
- In this program the function `setx()` is declared with a reference type, `int&`, as the return type:
  ```
  int& setx();
  ```
- This function contains the statement
  ```
  return x;
  ```
  where `x` has been defined as a global variable.
- Now—and this is what looks so strange—you can put a call to this function on the left side of the equal sign:
  ```
  setx() = 92;
  ```

- The result is that the variable returned by the function is assigned the value on the right side of the equal sign.
- That is, `x` is given the value 92. The output from the program
  `x=92`
  verifies that this assignment has taken place.

Function Calls on the Left of the Equal Sign

- Does this still sound obscure?
- Remember that an ordinary function—one that returns a value—can be used as if it were a value:
  `y = squareroot(x);`
- Here, whatever value `squareroot(x)` has is assigned to `y`.
- The function is treated as if it were a value.
- A function that returns a reference, on the other hand, is treated as if it were a variable.

Function Calls on the Left of the Equal Sign

- It returns an alias to a variable, namely the variable in the function's return statement.
- The function `setx()` returns a reference to the variable `x`.
- When this function is called, it's treated as if it were the variable `x`.
- Thus it can be used on the left side of an equal sign.
- There are two corollaries to this.
  1. we can't return a constant from a function that returns by reference.
  2. we can't return a reference to a local variable

Returning by Reference

- In `setx()`, we can't say

```
int& setx() {
    return 3; }
```

- If we try this the compiler will complain that you need an lvalue, that is, something that can go on the left side of the equal sign: a variable and not a constant.
- returning a reference to a local variable:

```
int& setx() {
    int x = 3;
    return x;      // error }
```

- What's wrong with this?
- The problem is that a function's local variables are probably destroyed when the function returns, and it doesn't make sense to return a reference to some- thing that no longer exists.

why function call on left of equal sign?

- In procedural programming there probably isn't too much use for this technique.
- "Operator Overloading," we'll find that returning by reference is an indispensable technique