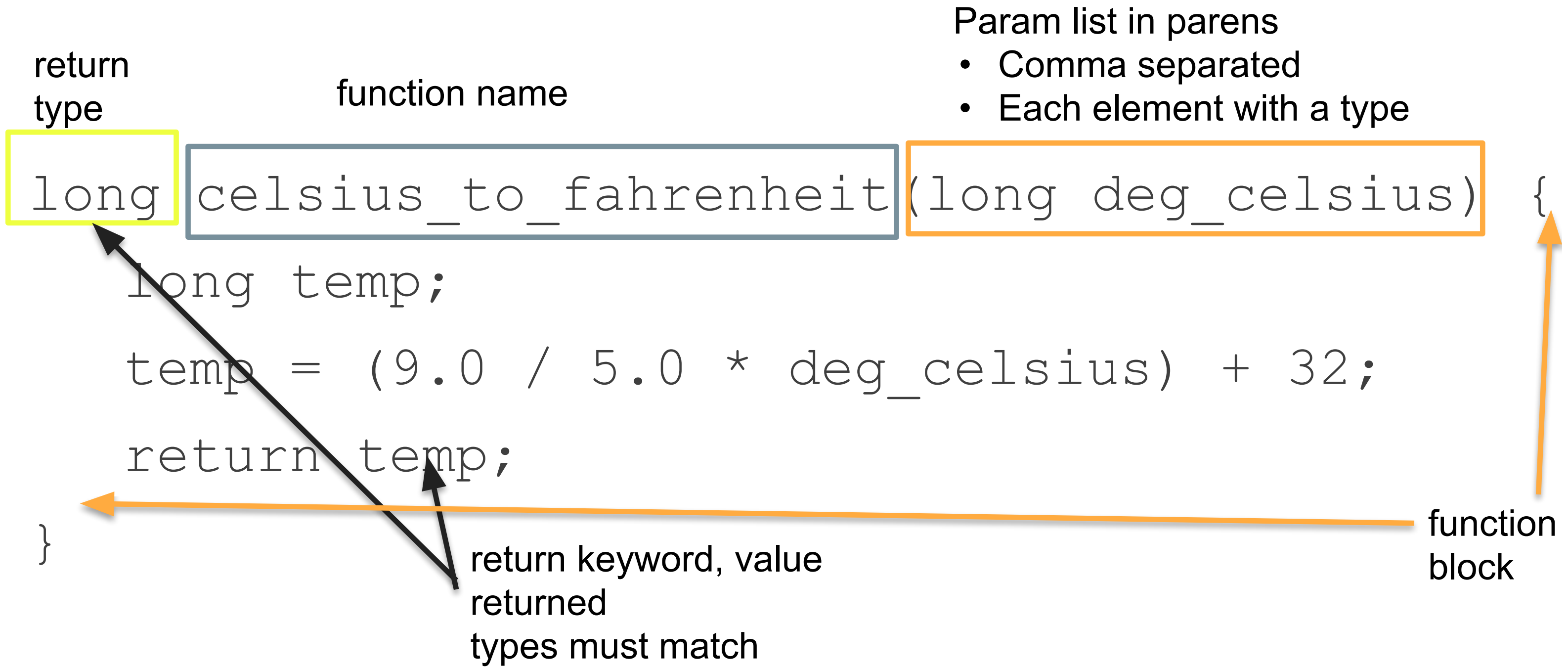
Functions

You’ve seen functions before

- A function is the encapsulation of some calculation.
  - We invoke a function and provide information in the form of arguments
  - The function receives the arguments as parameters, using the parameters to make its calculation
  - A value is returned by the function to the caller

Function definition



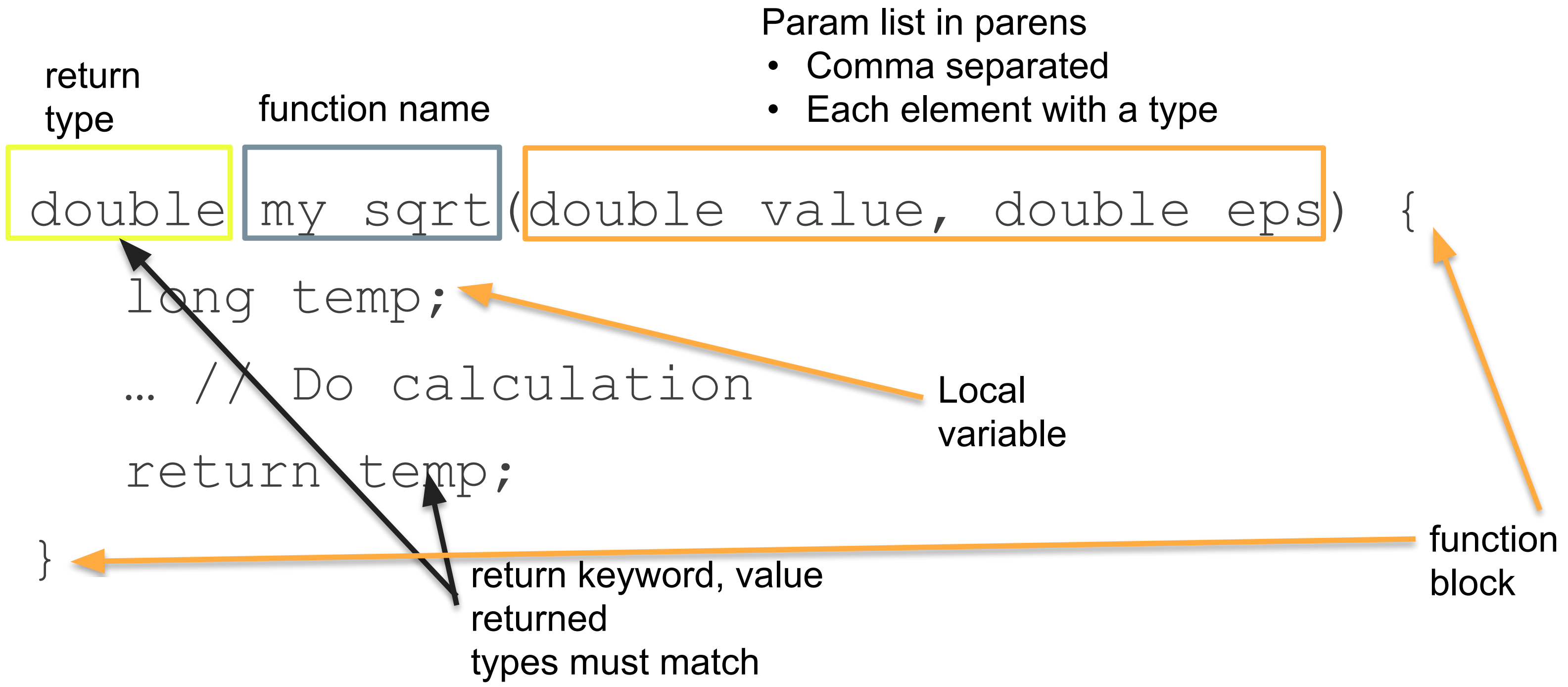
```
long celsius_to_fahrenheit(long deg_celsius) {
    long temp;
    temp = (9.0 / 5.0 * deg_celsius) + 32;
    return temp;
}
```

What is the return type of this python function:

```
def add(a, b):
    print(a)
    print(b)
    return a + b
```

- string
- int
- float
- I don't know

Function definition



```
double my_sqrt(double value, double eps) {
    long temp;
    … // Do calculation
    return temp;
}
```

Calling Example

```
int main() {
    double value, result;
    std::cout << “Enter a value:”’
    std::cin >> value;
    result = my_sqrt(value, 1e-5);
    std::cout << “Sqrt of:” << value << “ is:”
            << result << std::endl;
}
```

Assigned return value

Invocation

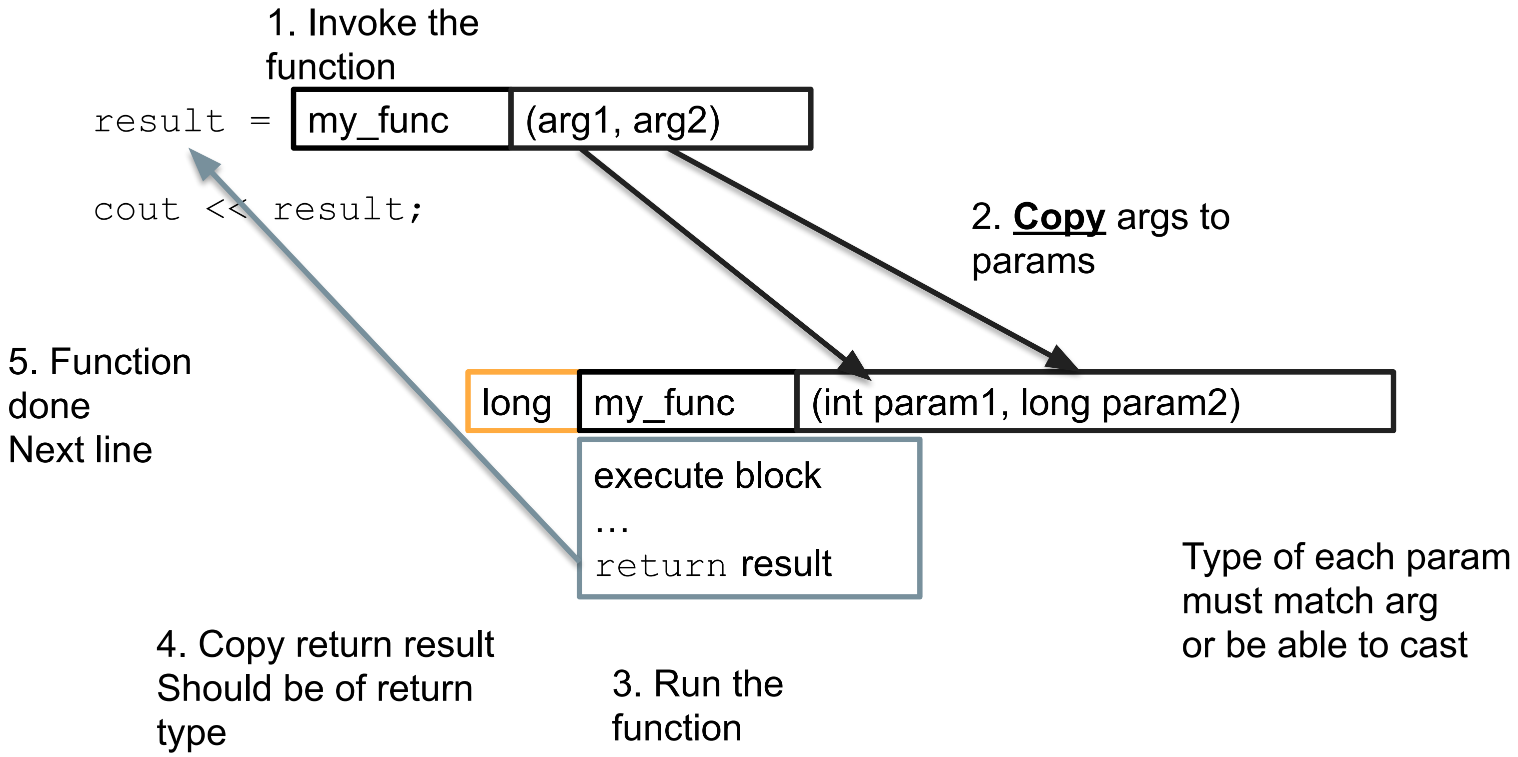
Arguments

Calling Example

```
int main() {
    long celsius_temp, result;
    cout << "Enter a temp in Celsius:";
    cin >> celsius_temp;
    result = celsius_to_fahrenheit(celsius_temp);
    cout << "Temp in celsius:" << celsius_temp
        << ", temp in Fahrenheit:" << result << endl;
}
```

Invocation

Arguments


1. Invoke the function
result = my_func (arg1, arg2)
cout << result;
2. Copy args to params
5. Function done Next line
long my_func (int param1, long param2)
execute block
…
return result
Type of each param must match arg or be able to cast
4. Copy return result Should be of return type
3. Run the function

Functions for better design

- Functions are very useful to break the program down into small, understandable maintainable pieces
  - Example: `celsius_to_fahrenheit`

Software Engineering

- There is a discipline of computer science dedicated to the systematic development and maintenance of software
- There are a number of approaches that SE use including: modularization, provability, testing, refactoring, and others

Refactoring

- Making multiple passes through code to improve its readability and maintainability while not changing (but perhaps improving) its functionality
- Implies that tests are available to apply to code to make sure this is the case
- One refactoring approach is extraction, making complicated code into multiple functions, creating better abstractions

How to write a function

- Should do one thing. If more than one thing, break it into parts.
  - A function abstracts one idea
- Should not be overly long (~one page of code).
  - Otherwise break it up!
- Should be generic in that it could be reused elsewhere in the code
- Should be readable!

What is probably too long for functions according to the Google Style Guide?

- `> 1 page`
- `> 40 lines`
- `> 3 ideas`
- `I don't know`

Scope

What is scope?

- When we create a variable, we make an association between a name and a value
  - A value exists at some memory location
  - The name is associated with **both**
- The part of the program where the name and that association is valid is called the variable's *scope*

Blocks are a scope

- Blocks constitute a scope
  - A variable declared within a block is only valid within that block
  - We’ve seen this before
- If you define a variable in a block, it **only has existence** within that block

Parameters are also local

- **Parameters** of a function are also considered local, part of the scope of the function

Be careful

- There will be situations where you want to pass back information from a function
- You should know:
  - It’s dangerous to pass back a reference or a pointer from local function names
    - At some point that memory will be reclaimed
  - If you don’t say otherwise, you are making a copy when you pass something back

Multiple Scopes

- Within multiple scopes you can have the same name associated with different values:
  - Within each scope there is a unique association, so no problem
  - Change scope, another (within that scope) unique association

Example 4.2

Should you reuse names in different scopes?

- Generally no
- Generally yes
- I dislike subjective questions
- I don't know

Values are copied

- Unless we say otherwise, C++ **copies** things that are passed, both in and out of a function.

More function examples

- Example 4.3
- Example 4.4

Are functions allowed to call themselves?

- `Yes`
- `No`
- `Maybe?`
- `I don't know`