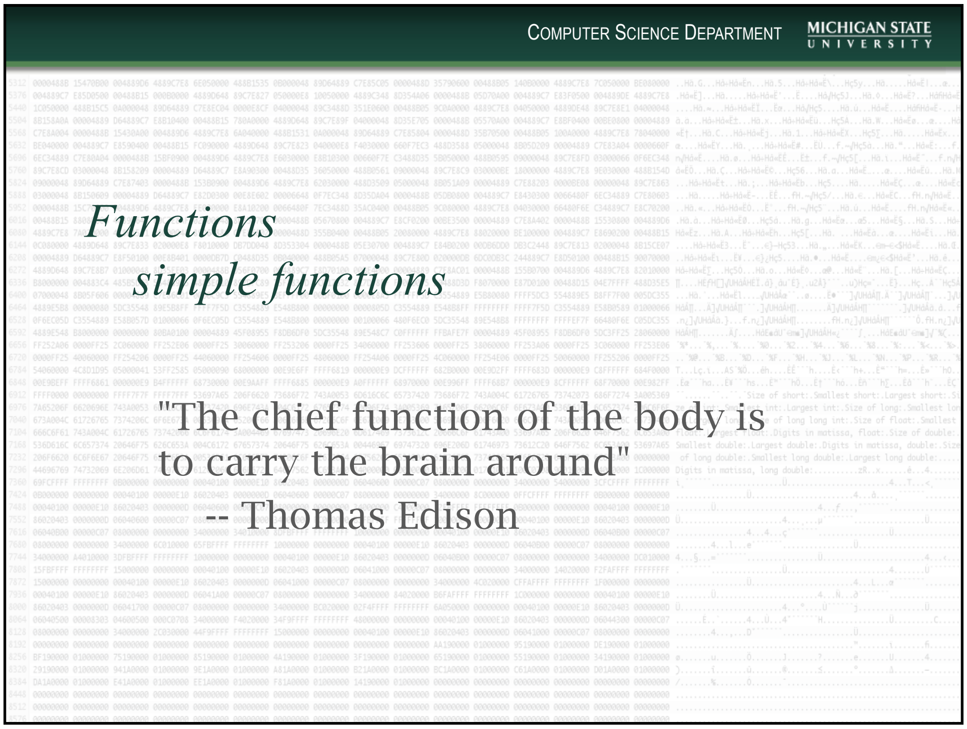
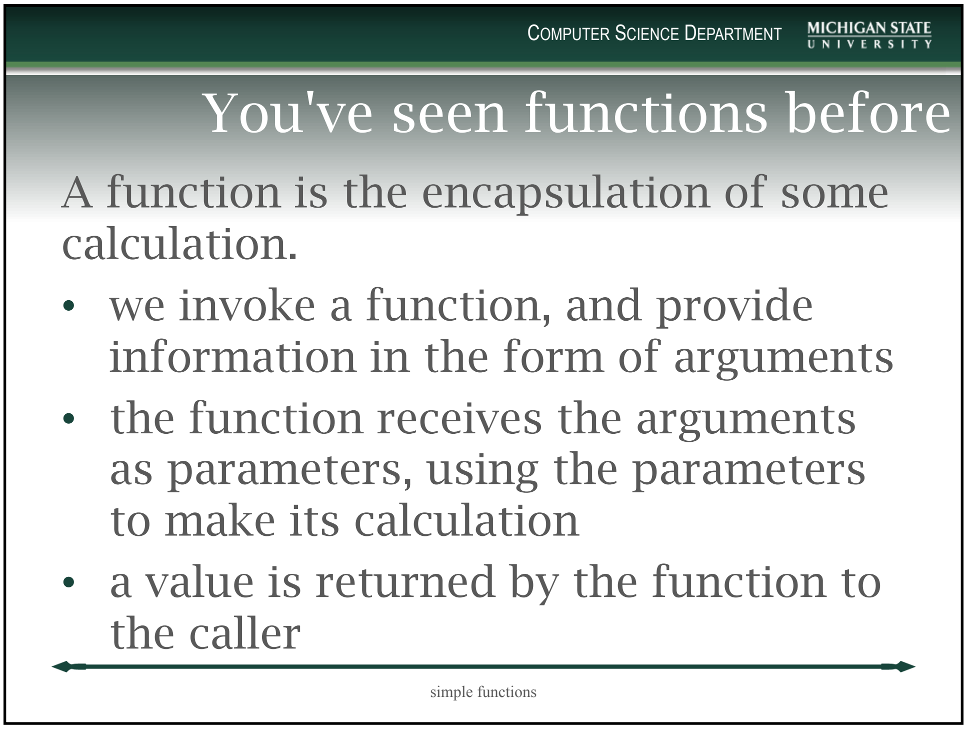


COMPUTER SCIENCE DEPARTMENT MICHIGAN STATE UNIVERSITY

Functions

simple functions

"The chief function of the body is to carry the brain around"
-- Thomas Edison



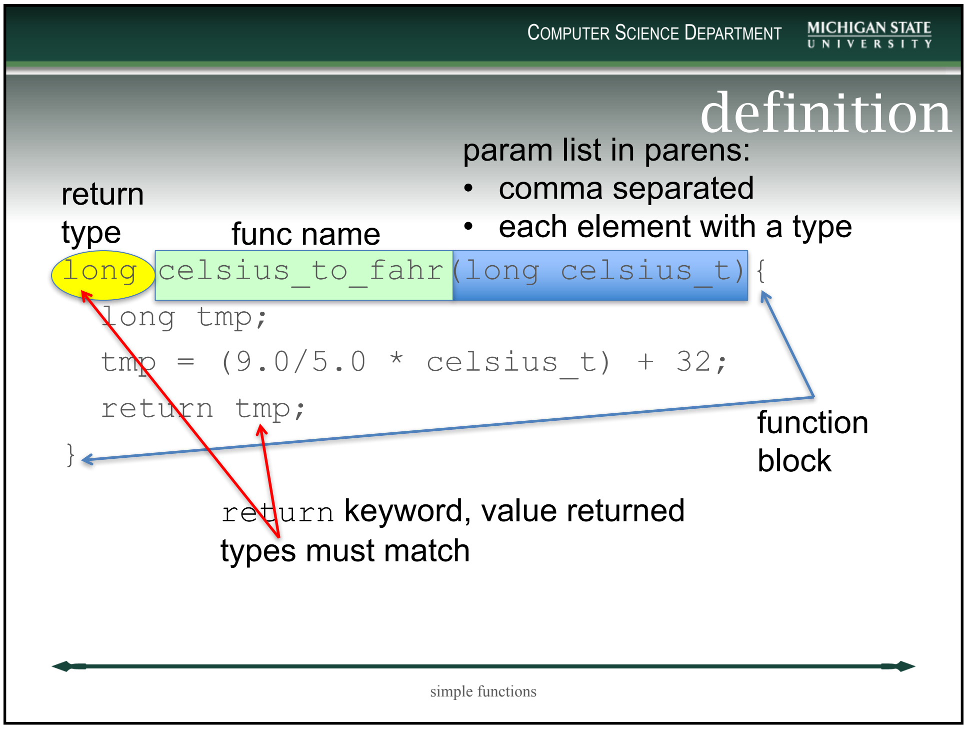
COMPUTER SCIENCE DEPARTMENT MICHIGAN STATE UNIVERSITY

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

simple functions



Computer Science Department MICHIGAN STATE UNIVERSITY

definition

param list in parens:
- comma separated
- each element with a type

return type

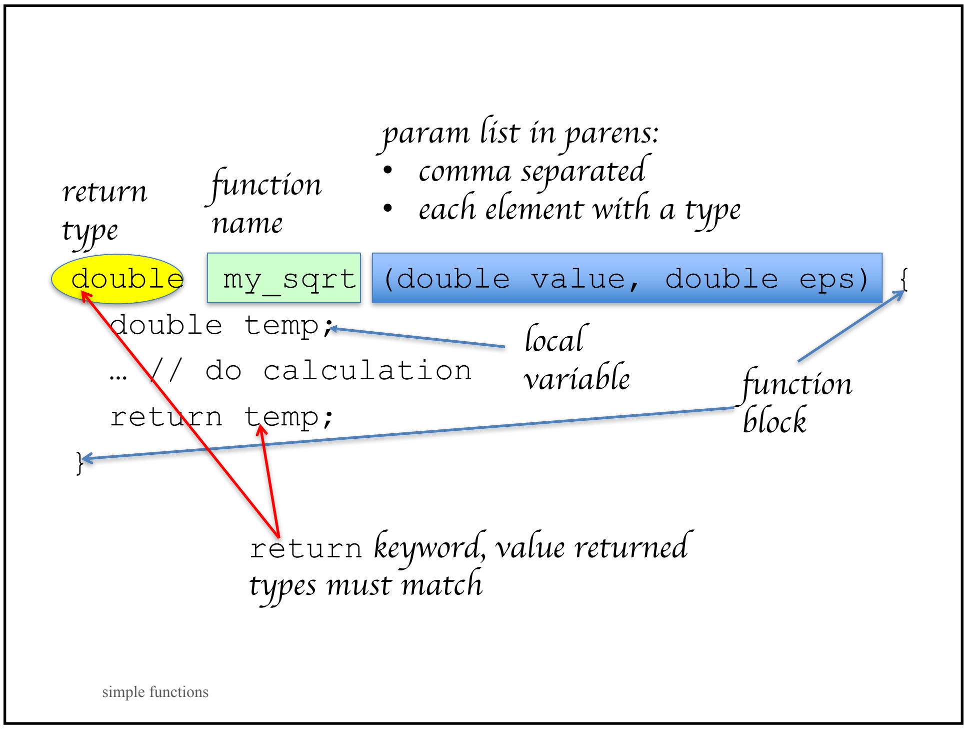
func name

```
long celsius_to_fahr(long celsius_t){
  long tmp;
  tmp = (9.0/5.0 * celsius_t) + 32;
  return tmp;
}
```

function block

`return` keyword, value returned types must match

simple functions



param list in parens:
- *comma separated*
- *each element with a type*

return type

function name

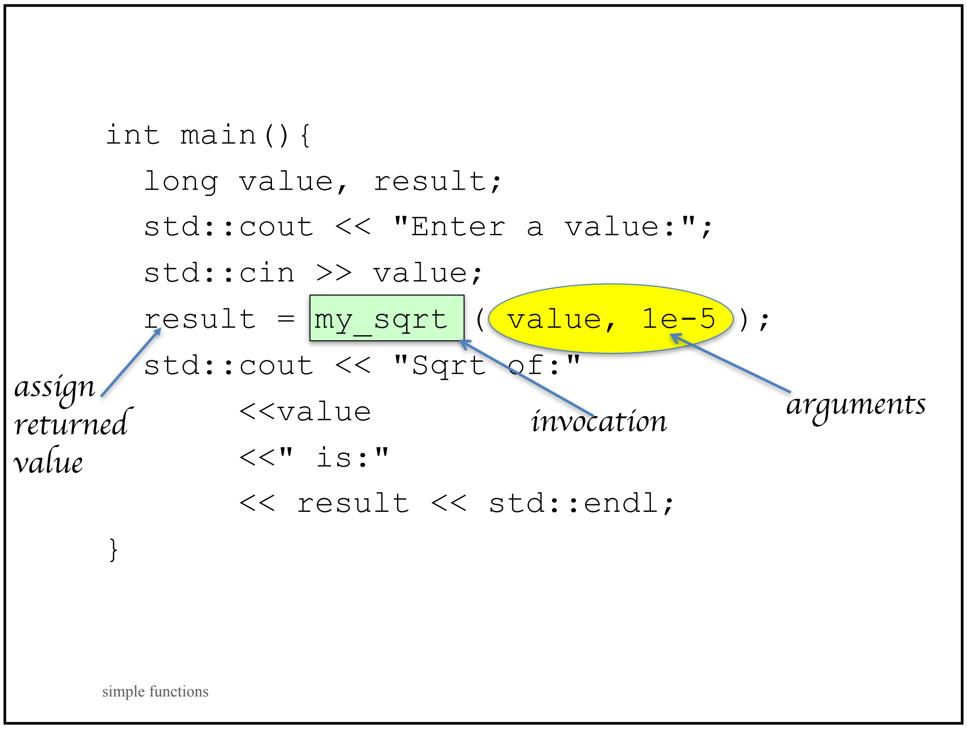
```
double  my_sqrt  (double value, double eps) {
  double temp;
  … // do calculation
  return temp;
}
```

local variable

function block

`return` *keyword, value returned types must match*

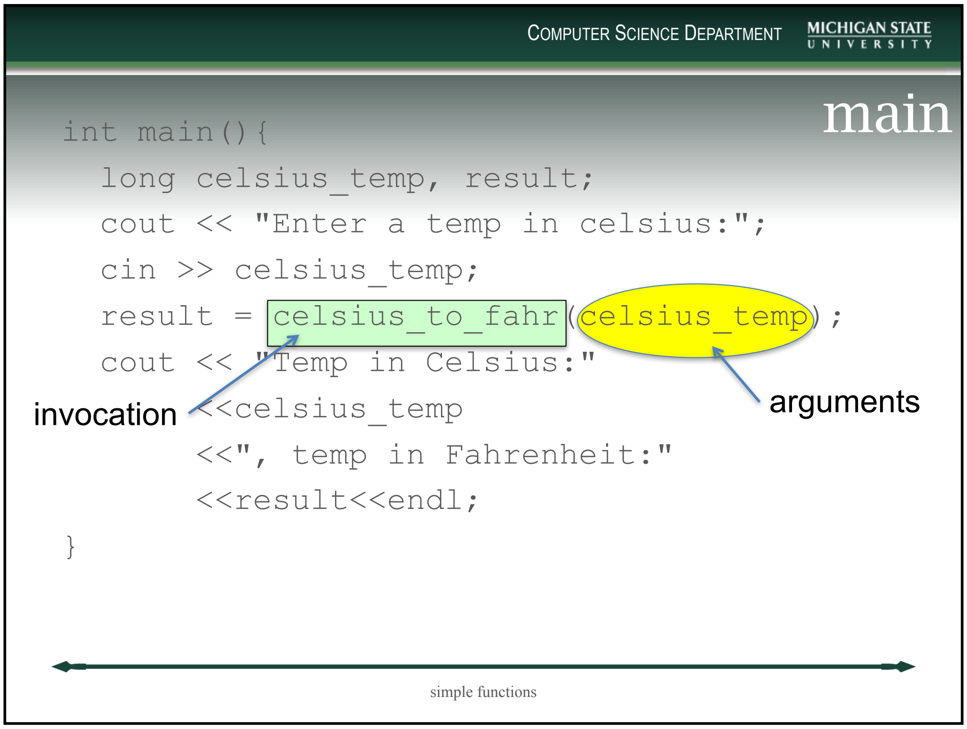
simple functions



```
int main(){
  long 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;
}
```

assign returned value
invocation
arguments

simple functions



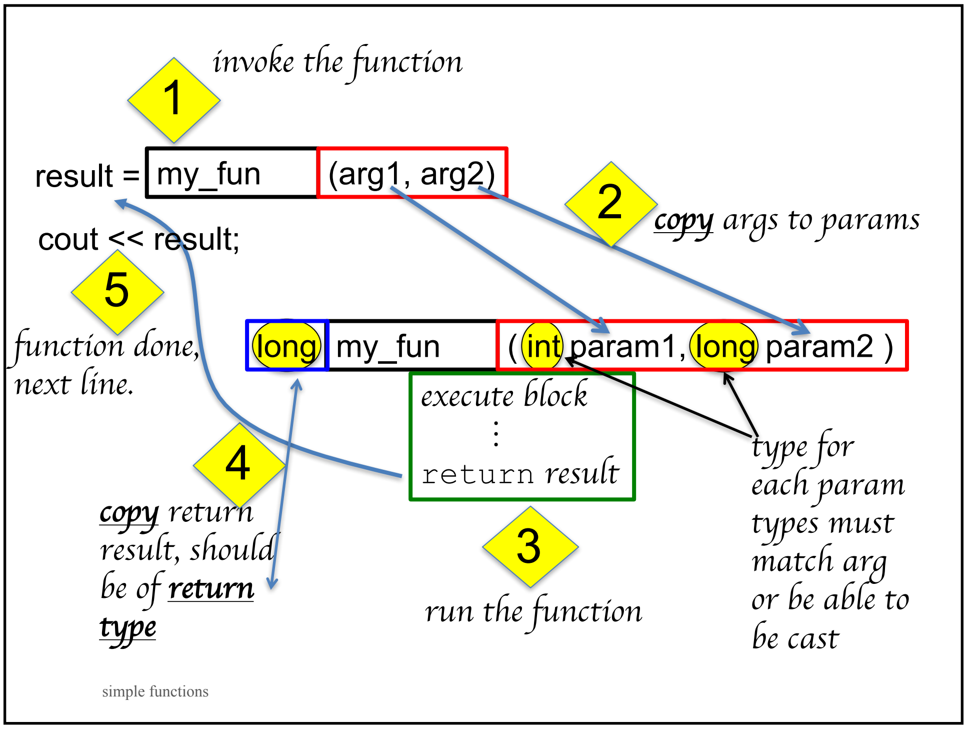
COMPUTER SCIENCE DEPARTMENT MICHIGAN STATE UNIVERSITY

main

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

invocation
arguments

simple functions



1 invoke the function

result = my_fun (arg1, arg2)

cout << result;

2 *copy* args to params

long my_fun (int param1, long param2)

execute block
⋮
return result

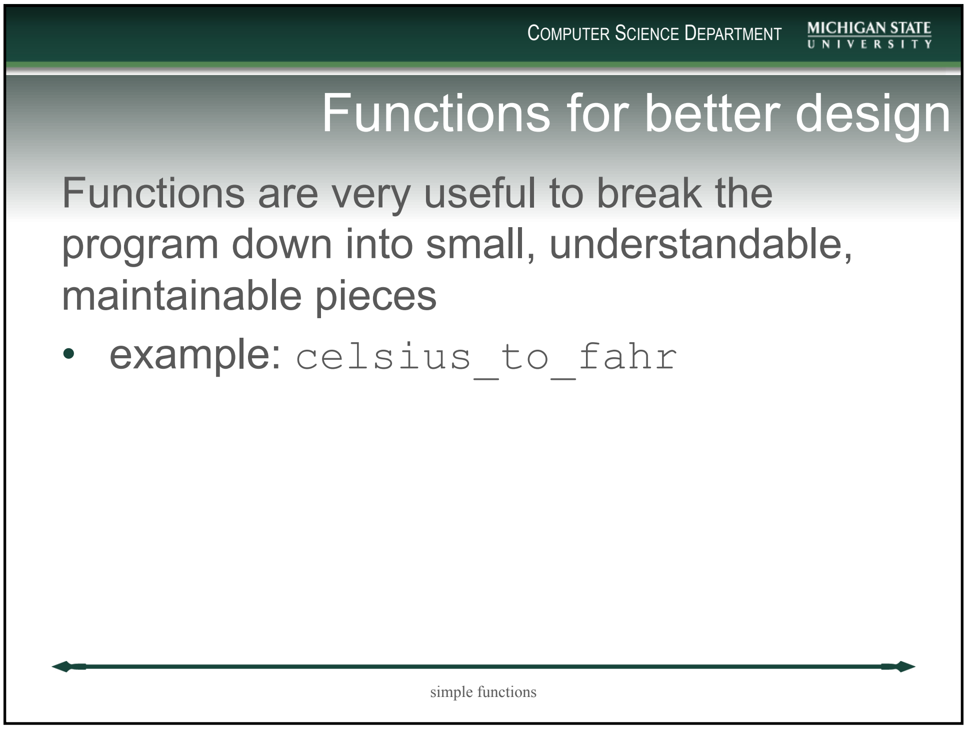
3 run the function

4 *copy* return result, should be of **return type**

5 function done, next line.

type for each param types must match arg or be able to be cast

simple functions



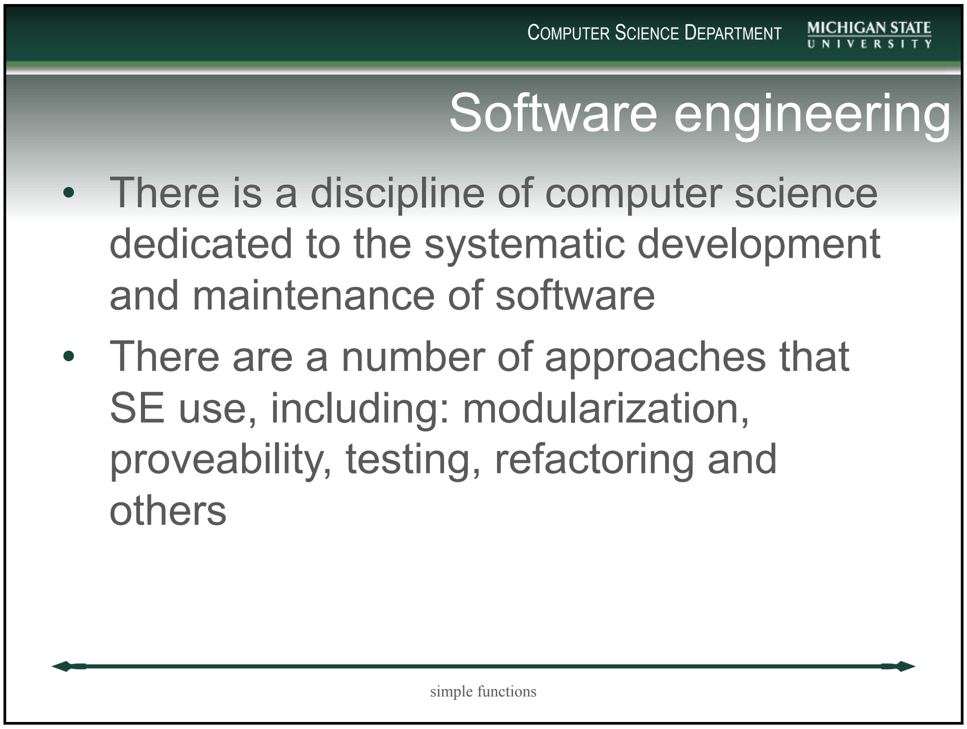
Computer Science Department Michigan State University

Functions for better design

Functions are very useful to break the program down into small, understandable, maintainable pieces

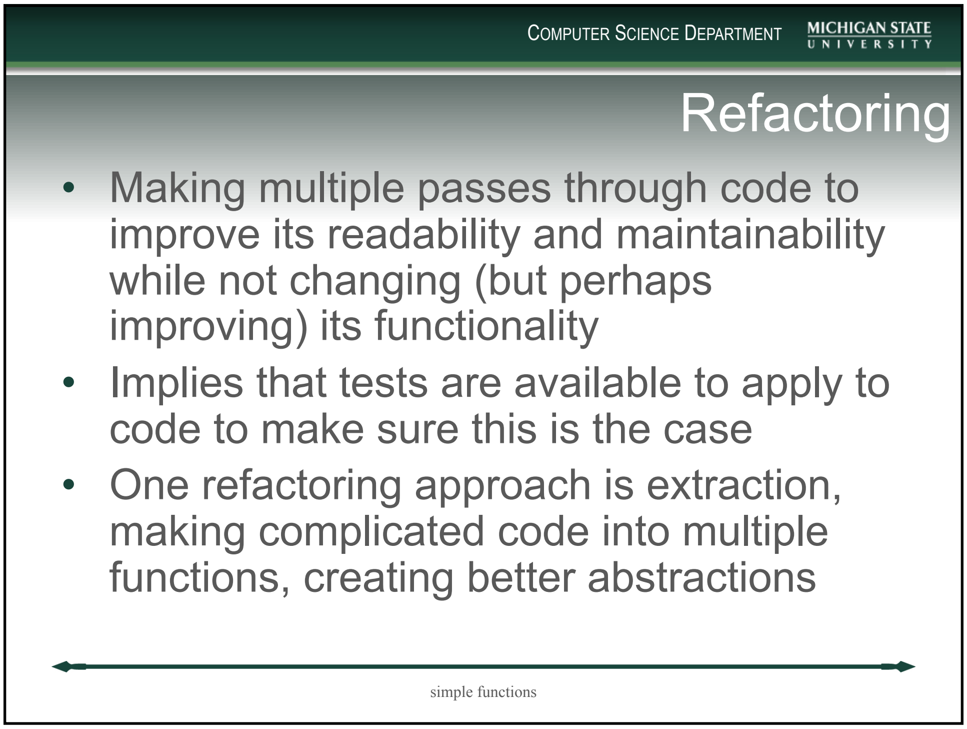
- example: `celsius_to_fahr`

simple functions



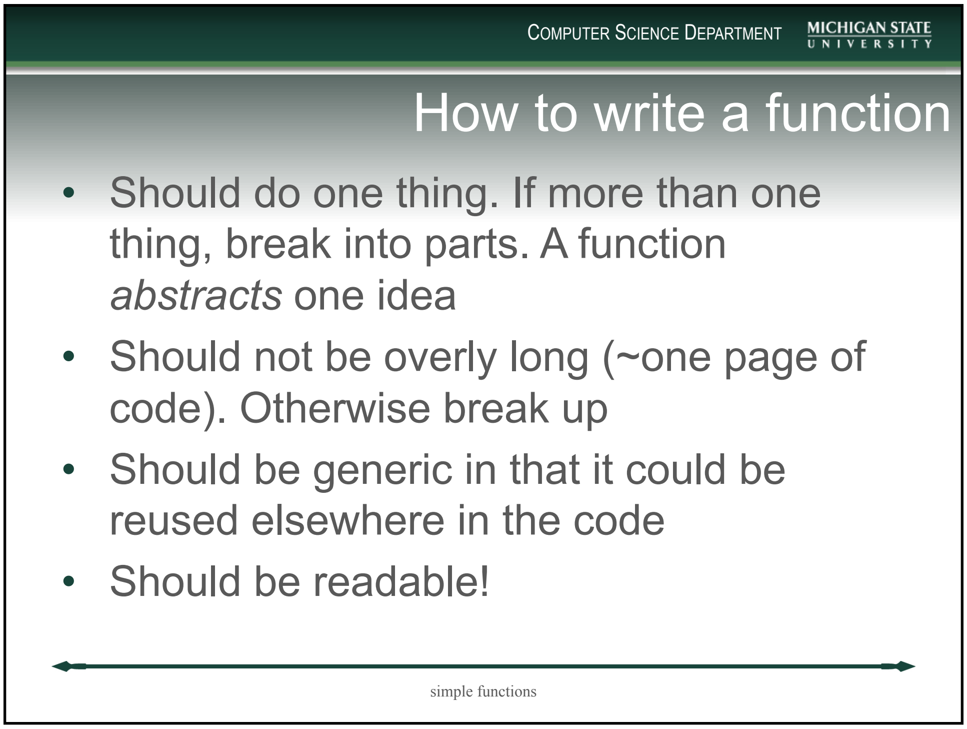
Software engineering

- 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, proveability, 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



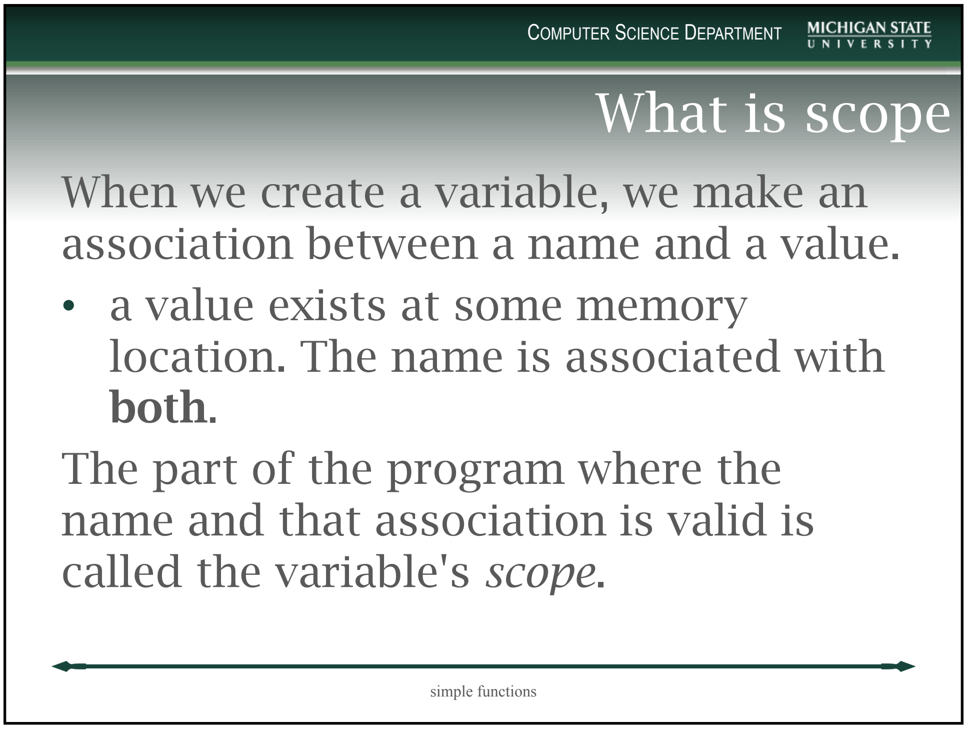
How to write a function

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

Scope

"Still this planet's soil for noble deeds grants scope abounding."

-- Goethe

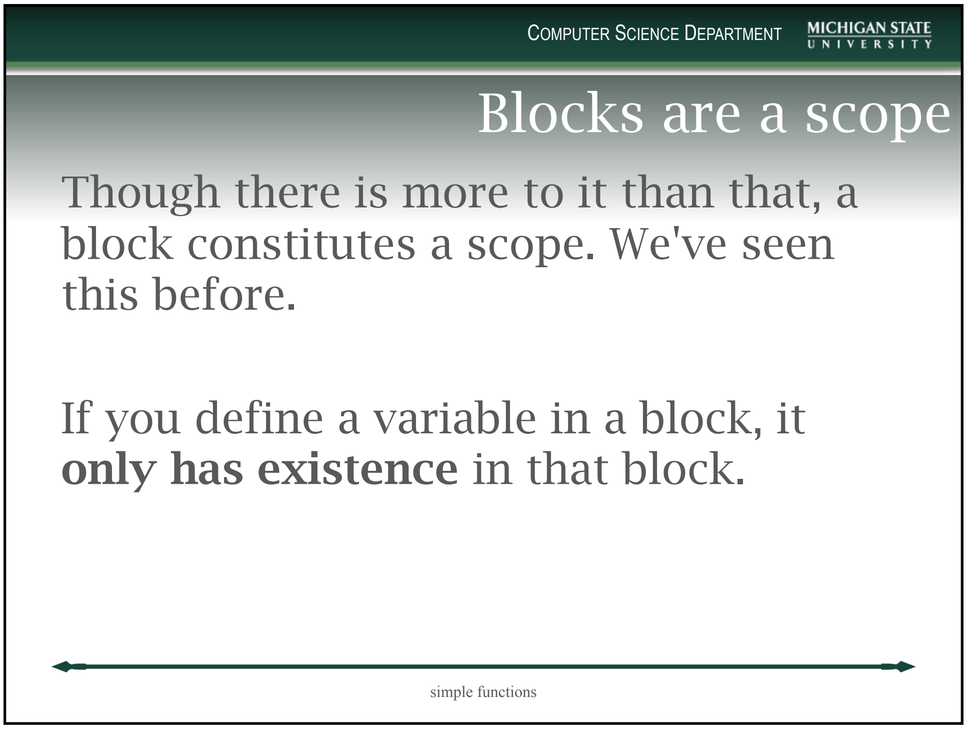


What is scope

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**.

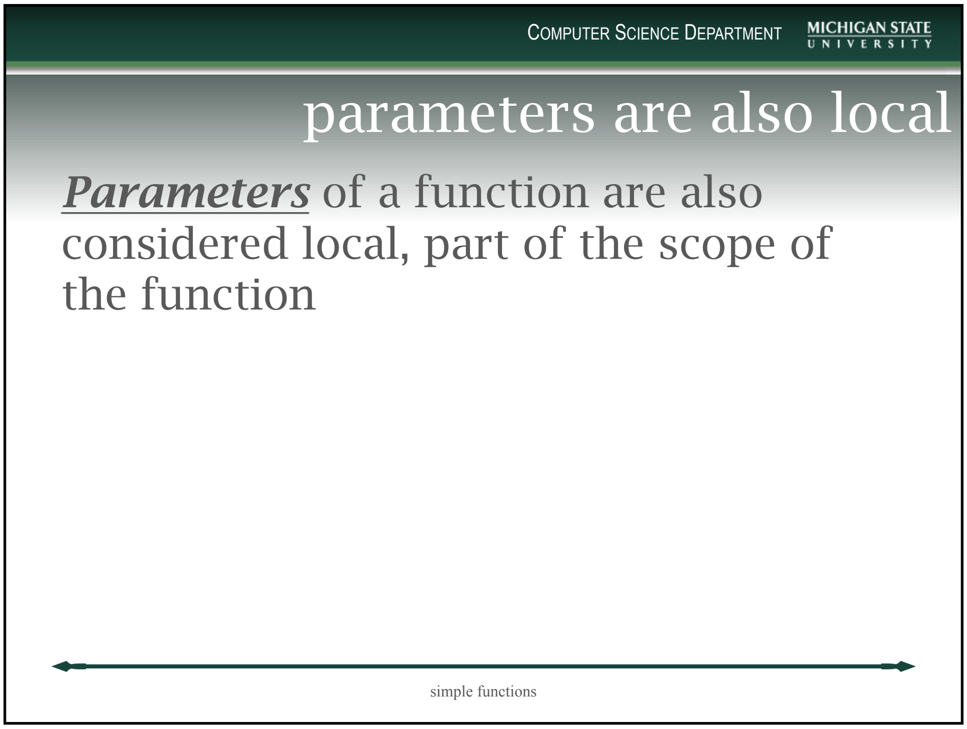
The part of the program where the name and that association is valid is called the variable's *scope*.



Blocks are a scope

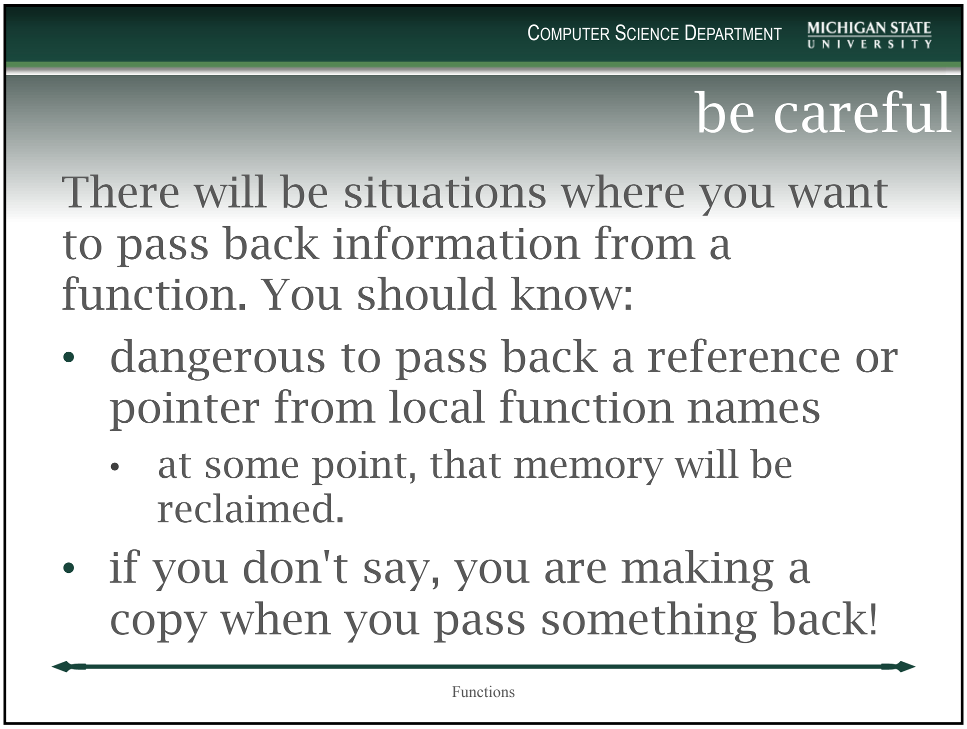
Though there is more to it than that, a block constitutes a scope. We've seen this before.

If you define a variable in a block, it **only has existence** in that block.



parameters are also local

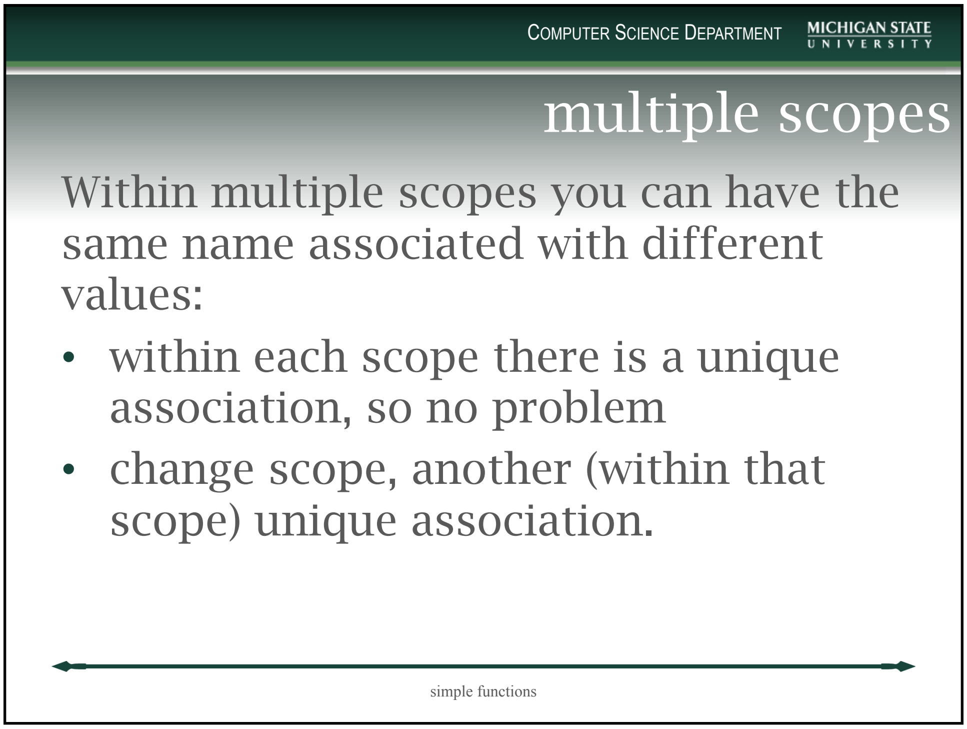
Parameters of a function are also considered local, part of the scope of the function



be careful

There will be situations where you want to pass back information from a function. You should know:

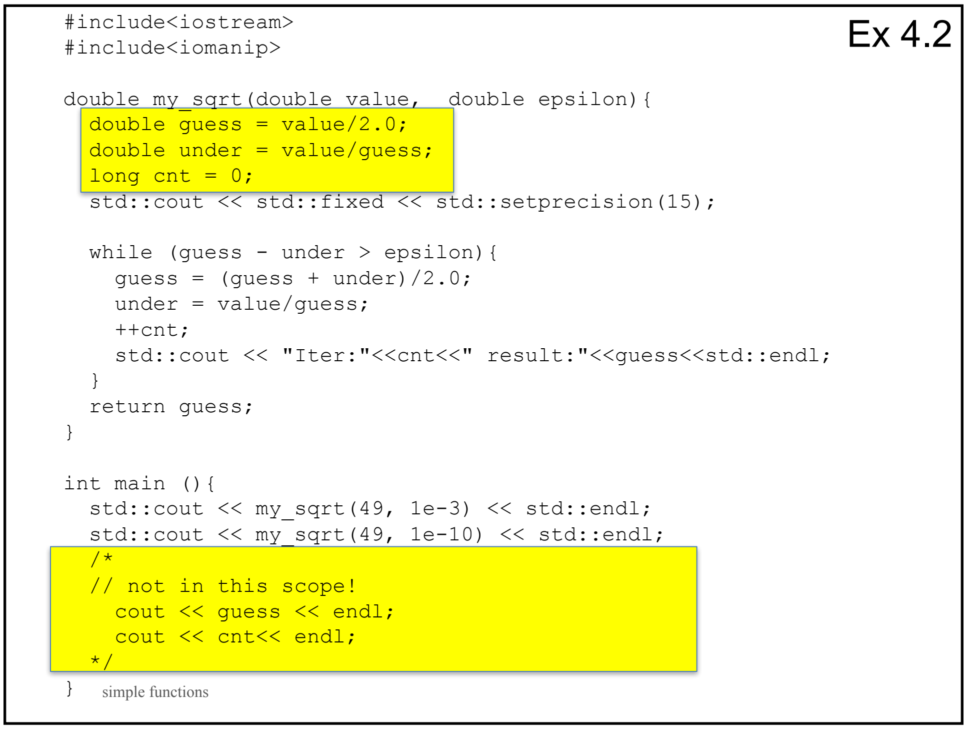
- dangerous to pass back a reference or pointer from local function names
  - at some point, that memory will be reclaimed.
- if you don't say, you are making a copy when you pass something back!



multiple scopes

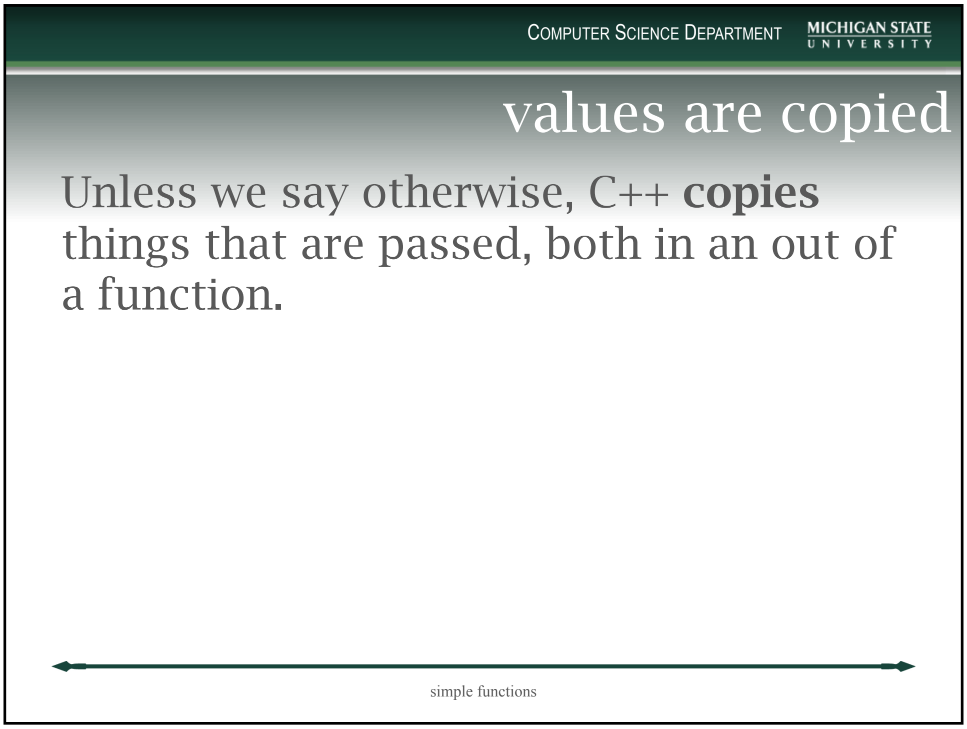
Within multiple scopes you can have the same name associated with different values:

- within each scope there is a unique association, so no problem
- change scope, another (within that scope) unique association.



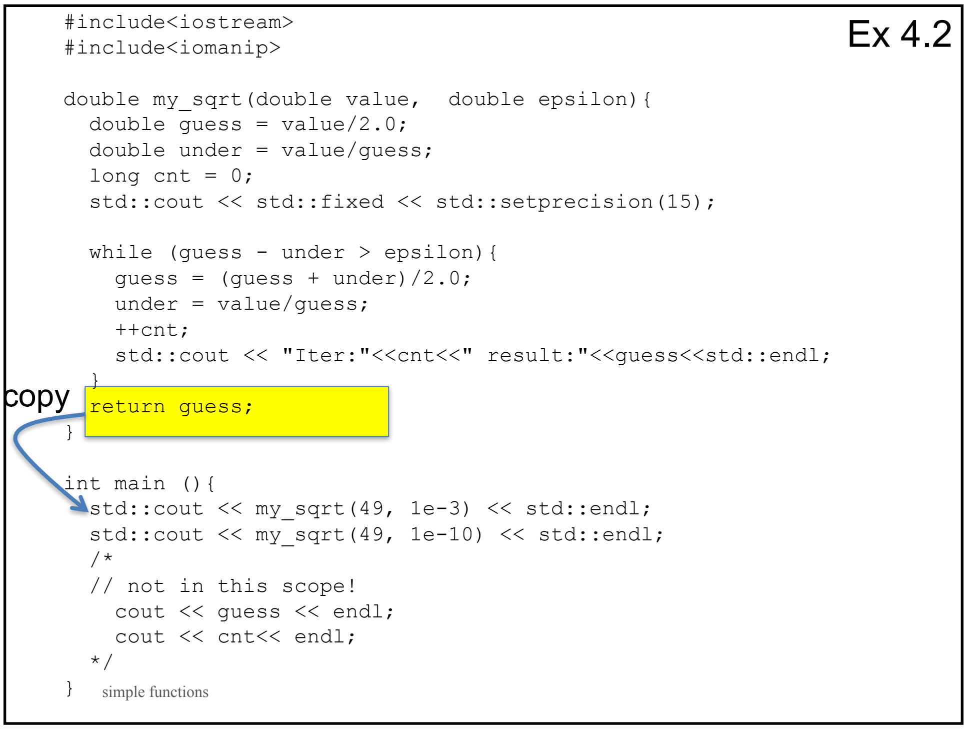
Ex 4.2

```
#include<iostream>
#include<iomanip>

double my_sqrt(double value,  double epsilon){
  double guess = value/2.0;
  double under = value/guess;
  long cnt = 0;
  std::cout << std::fixed << std::setprecision(15);

  while (guess - under > epsilon){
    guess = (guess + under)/2.0;
    under = value/guess;
    ++cnt;
    std::cout << "Iter:"<<cnt<<" result:"<<guess<<std::endl;
  }
  return guess;
}

int main (){
  std::cout << my_sqrt(49, 1e-3) << std::endl;
  std::cout << my_sqrt(49, 1e-10) << std::endl;
  /*
  // not in this scope!
    cout << guess << endl;
    cout << cnt<< endl;
  */
}
```



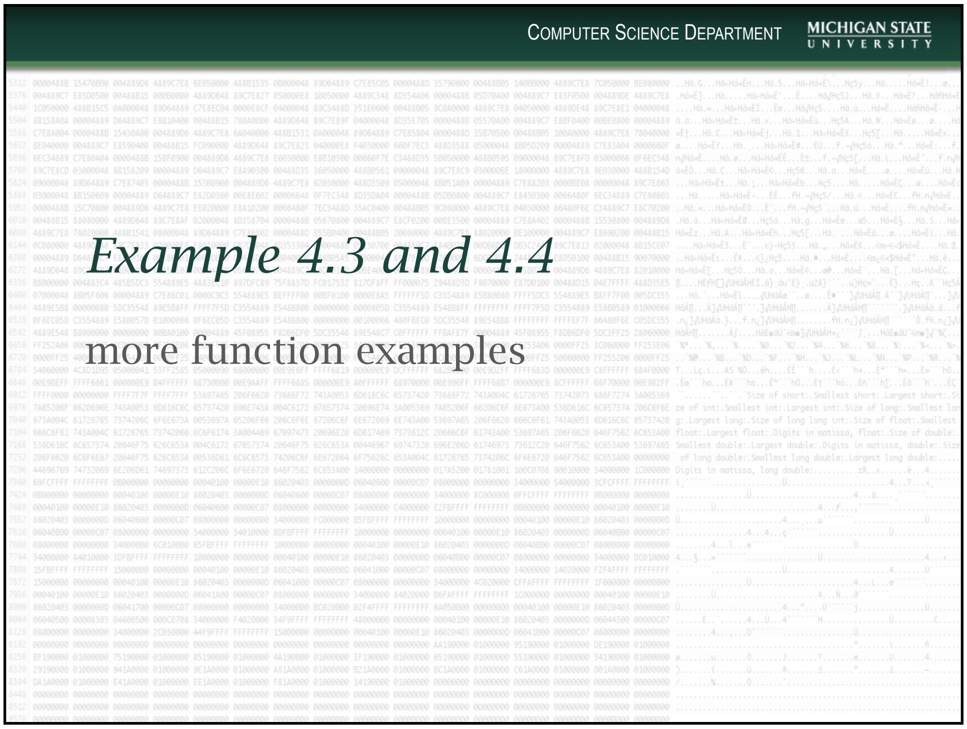
values are copied

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



Ex 4.2

```
#include<iostream>
#include<iomanip>

double my_sqrt(double value,  double epsilon){
  double guess = value/2.0;
  double under = value/guess;
  long cnt = 0;
  std::cout << std::fixed << std::setprecision(15);

  while (guess - under > epsilon){
    guess = (guess + under)/2.0;
    under = value/guess;
    ++cnt;
    std::cout << "Iter:"<<cnt<<" result:"<<guess<<std::endl;
  }
  return guess;
}

int main (){
  std::cout << my_sqrt(49, 1e-3) << std::endl;
  std::cout << my_sqrt(49, 1e-10) << std::endl;
  /*
  // not in this scope!
    cout << guess << endl;
    cout << cnt<< endl;
  */
}
```



COMPUTER SCIENCE DEPARTMENT

MICHIGAN STATE UNIVERSITY

Example 4.3 and 4.4

more function examples