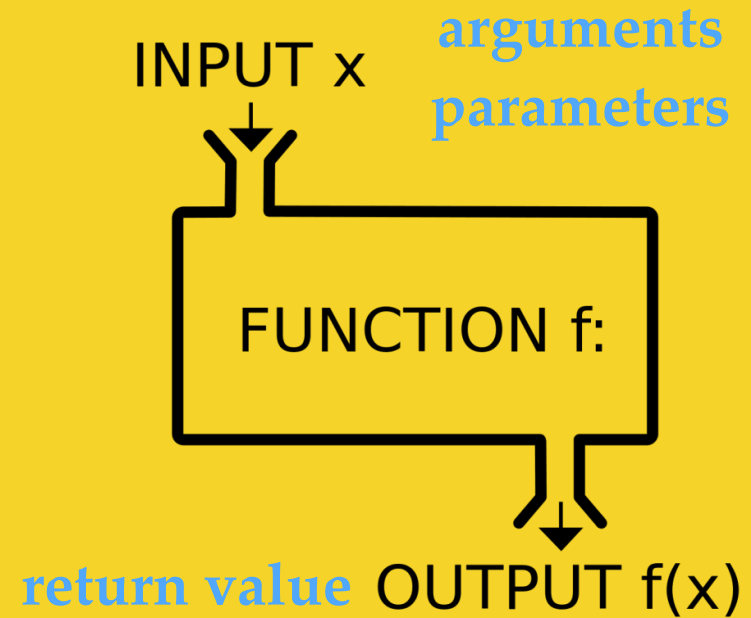# CSC 211: Object Oriented Programming
Functions

Michael Conti
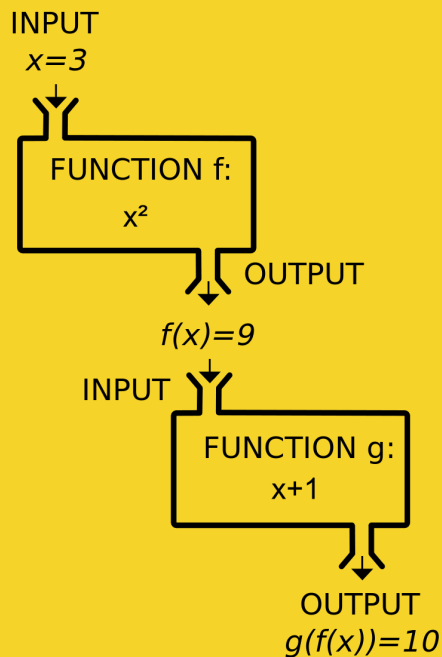
Department of Computer Science and Statistics
University of Rhode Island

Spring 2020

THINK BIG WE DO™

INPUT x

**arguments
parameters**

FUNCTION f:

return value OUTPUT f(x)

INPUT
x=3

FUNCTION f:
x²

OUTPUT

f(x)=9

INPUT

FUNCTION g:
x+1

OUTPUT
g(f(x))=10

Functions

‣ A function is a group of statements that together perform a task (packaged as a unit)

‣ **Top-down design**
  ✓ break the algorithm into specific subtasks
  ✓ break each subtask into smaller subtasks

‣ Smaller subtasks are generally trivial to implement in the programming language

Why functions?

- Improves code readability

- Improves code maintainability

- Allows easy code reuse

Predefined functions

- Predefined functions are found in libraries
  - ✓ the library must be included in a program
  - ✓ e.g. `#include <cmath>`

- Predefined functions can be invoked after including the proper library headers

Some `<cmath>` functions

Some Predefined Functions

Name	Description	Type of Arguments	Type of Value Returned	Example	Value	Library Header
sqrt	square root	*double*	*double*	sqrt(4.0)	2.0	cmath
pow	powers	*double*	*double*	pow(2.0,3.0)	8.0	cmath
abs	absolute value for *int*	*int*	*int*	abs(-7) abs(7)	7 7	cstdlib
labs	absolute value for *long*	*long*	*long*	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	absolute value for *double*	*double*	*double*	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath
ceil	ceiling (round up)	*double*	*double*	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor	floor (round down)	*double*	*double*	floor(3.2) floor(3.9)	3.0 3.0	cmath

from: Problem Solving with C++, 10th Edition, Walter Savitch

Programmer defined functions (syntax)

```
// comment describing what function does
return_type function_name(parameters);
```
declaration

```
// ...
// statements
// ...
```
definition

```
return_type function_name(parameters) {
    // body of the function
}
```

Function declaration

- Tells compiler the **function signature**
  - name, parameters, return type

- Declarations are required to appear prior to a function call
  - unless a definition has already appeared

- Declarations are normally placed before the `main` function

```
// comment describing what function does
return_type function_name(parameters);
```

Function definition

- Provides the all details of a function
  - includes the actual body of the function (block of statements)

- Good practice => have at least one return statement

```
return_type function_name(parameters) {
    // body of the function
}
```

Example

```
    int function(int param);

    int main() {
        // ...
        a = function(val);
        // ...
    }

    int function(int param) {
        // body of the function
        // must return an integer
    }
```

A different style …

```
int function(int param) {
    // body of the function
    // must return an integer
}

int main() {
    // ...
    a = function(val);
    // ...
}
```

Parameter list

- Refers to the **type**, order, and number of parameters of a function

- Parameters are optional
  - ✓ can be empty

- When a function is invoked, arguments are passed accordingly (with respect to the parameter list)

13

return statement

- Ends the function call
  - ✓ returns a value

return expression;

14

Label all function parts

```cpp
#include <iostream>

int abs(int n);        declaration or prototype

int main() {
    std::cout << "|−5| = " << abs(−5) << std::endl;
    return 0;                    function call
}

int abs(int n) {
    if (n < 0) {
        return −n;      definition
    } else {
        return n;
    }
}
```

15

Functions are black boxes

DISPLAY 4.7 Definitions That Are Black-Box Equivalent

Function Declaration

```
1    double newBalance(double balancePar, double ratePar);
2    //Returns the balance in a bank account after
3    //posting simple interest. The formal parameter balancePar is
4    //the old balance. The formal parameter ratePar is the interest rate.
5    //For example, if ratePar is 5.0, then the interest rate is 5 percent
6    //and so newBalance(100, 5.0) returns 105.00.
```

Definition 1

```
double newBalance(double balancePar, double ratePar)
{
    double interestFraction, interest;

    interestFraction = ratePar/100;
    interest = interestFraction * balancePar;
    return (balancePar + interest);
}
```

Definition 2

```
double newBalance(double balancePar, double ratePar)
{
    double interestFraction, updatedBalance;

    interestFraction = ratePar/100;
    updatedBalance = balancePar * (1 + interestFraction);
    return updatedBalance;
}
```

from: Problem Solving with C++, 10th Edition, Walter Savitch

16

void functions

Slide 17

- A function might produce no returning value
  - e.g. sends IP packets to other machine, or sends data to the standard output

- Void functions allow programmers to define functions with no returning values

```cpp
void f_name(/* parameters */) {
    // statements
    return;
}
```

Slide 18

DISPLAY 5.3 Use of *return* in a *void* Function

Function Declaration

```
1    void iceCreamDivision(int number, double totalWeight);
2    //Outputs instructions for dividing totalWeight ounces of
3    //ice cream among number customers.
4    //If number is 0, nothing is done.
```

Function Definition

```
1    //Definition uses iostream:
2    void iceCreamDivision(int number, double totalWeight)
3    {
4        using namespace std;
5        double portion;
6
7        if (number == 0)
8            return;
9        portion = totalWeight/Number;
10       cout.setf(ios::fixed);
11       cout.setf(ios::showpoint);
12       cout.precision(2);
13       cout << "Each one receives "
14           << portion << " ounces of ice cream." << endl;
15   }
```

If number is 0, then the function execution ends here.

Slide 19

```cpp
#include <iostream>

void foo(int a, int b) {
    std::cout << a + b;
    return;
}


int main() {
    std::cout << foo(10, 20);
}
```

Slide 20

```
a.cc:9:15: error: invalid operands to binary expression ('std::__1::ostream' (aka 'basic_ostream<char>') and 'void')
    std::cout << foo(10, 20);
    ~~~~~~~~~ ^  ~~~~~~~~~~~
/Library/Developer/CommandLineTools/usr/include/c++/v1/
ostream:194:20: note: candidate function not viable: cannot convert argument of incomplete type 'void' to 'std::__1::bas
ic_ostream<char> &(*)(std::__1::basic_ostream<char> &)' for 1st argument
    basic_ostream& operator<<(basic_ostream& (*__pf)(basic_ostream&))
                   ^
/Library/Developer/CommandLineTools/usr/include/c++/v1/
ostream:198:20: note: candidate function not viable: cannot convert argument of incomplete type 'void' to 'basic_ios<std
::__1::basic_ostream<char, std::__1::char_traits<char> >::char_type, std::__1::basic_ostream<char, std::__1::char_trait
<char> >::traits_type> &(*)
(basic_ios<std::__1::basic_ostream<char, std::__1::char_traits<char> >::char_type, std::__1::basic_ostream<char, std::__
1::char_traits<char> >::traits_type> &)' (aka 'basic_ios<char, std::__1::char_traits<char> > &(*)
(basic_ios<char, std::__1::char_traits<char> > &)') for 1st argument
    basic_ostream& operator<<(basic_ios<char_type, traits_type>&
                   ^
/Library/Developer/CommandLineTools/usr/include/c++/v1/
ostream:203:20: note: candidate function not viable: cannot convert argument of incomplete type 'void' to 'std::__1::ios
_base &(*)(std::__1::ios_base &)' for 1st argument
    basic_ostream& operator<<(ios_base& (*__pf)(ios_base&))
                   ^

.... 87 lines ommitted

/Library/Developer/CommandLineTools/usr/include/c++/v1/
ostream:1081:1: note: candidate template ignored: could not match 'unique_ptr<type-parameter-0-2, type-
parameter-0-3>' against 'void'
operator<<(basic_ostream<_CharT, _Traits>& __os, unique_ptr<_Yp, _Dp> const& __p)
^
/Library/Developer/CommandLineTools/usr/include/c++/v1/
ostream:1088:1: note: candidate template ignored: could not match 'bitset<_Size>' against 'void'
operator<<(basic_ostream<_CharT, _Traits>& __os, const bitset<_Size>& __x)
^
1 error generated.
```

```cpp
#include <iostream>

void foo(int a, int b) {
    std::cout << a + b;
    return;
}

int main() {
    foo(10, 20);
}
```

👍

Tracing a function call

DISPLAY 4.3 A Function Definition

```cpp
1    #include <iostream>
2    using namespace std;
3
4    double totalCost(int numberPar, double pricePar);        function declaration/function
5    //Computes the total cost, including 5% sales tax,        prototype
6    //on numberPar items at a cost of pricePar each.
7
8    int main( )
9    {
10       double price, bill;
11       int number;
12
13       cout << "Enter the number of items purchased: ";
14       cin >> number;
15       cout << "Enter the price per item $";
16       cin >> price;
17                                              function call
18       bill = totalCost(number, price);
19
20       cout.setf(ios::fixed);
21       cout.setf(ios::showpoint);
22       cout.precision(2);
23       cout << number << " items at "
24            << "$" << price << " each.\n"
25            << "Final bill, including tax, is $" << bill
26            << endl;
27
28       return 0;                            function heading
29    }
30
31    double totalCost(int numberPar, double pricePar)
32    {
33       const double TAX_RATE = 0.05;  //5% sales tax
34       double subtotal;                    function      function
35                                           body          definition
36       subtotal = pricePar * numberPar;
37       return (subtotal + subtotal * TAX_RATE);
38    }
```

from: Problem Solving with C++, 10th Edition, Walter Savitch

DISPLAY 4.4 Details of a Function Call

```cpp
int main()
{
    double price, bill;
    int number;
    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
    cin >> price;

    bill = totalCost (number, price);
                        2         10.10
    cout.setf (ios::fixed);
    cout.setf (ios::showpoint);
    cout.precision(2);
    cout << number << " items at "
         << "$" << price << " each.\n"
21.21    << "Final bill, including tax, is $" << bill
         << endl;
    return 0;
}
double totalCost (int numberPar, double pricePar)
                       2              10.10
{
    const double TAX_RATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = pricePar * numberPar;
    return (subtotal + subtotal * TAX_RATE);
}
                        21.21
```

1. Before the function is called, values of the variables number and price are set to 2 and 10.10, by cin statements (as you can see the Sample Dialogue in Display 4.3)

2. The function call executes and the value of **number** (which is 2) plugged in for **numberPar** and value of **price** (which is 10.10) plugged in for **pricePar**.

3. The body of the function executes with **numberPar** set to 2 and **pricePar** set to 10.10, producing the value 20.20 in subtotal.

4. When the return statement is executed, the value of the expression after return is evaluated and returned by the function. In this case, (subtotal + subtotal * TAX_RATE) is (20.20 + 20.20*0.05) or 21.21.

5. The value 21.21 is returned to where the function was invoked. The result is that **totalCost (number, price)** is replaced by the return value of 21.21. The value of **bill** (on the left-hand side of the equal sign) is set equal to 21.21 when the statement **bill = totalCost (number, price);** finally ends.

from: Problem Solving with C++, 10th Edition, Walter Savitch

Question?

‣ Write a program that takes an integer `n > 1` from `stdin` and outputs the largest prime number less or equal than `n` to the `stdout`

  ✓ use an `is_prime` function

25