CSC 211: Object Oriented Programming

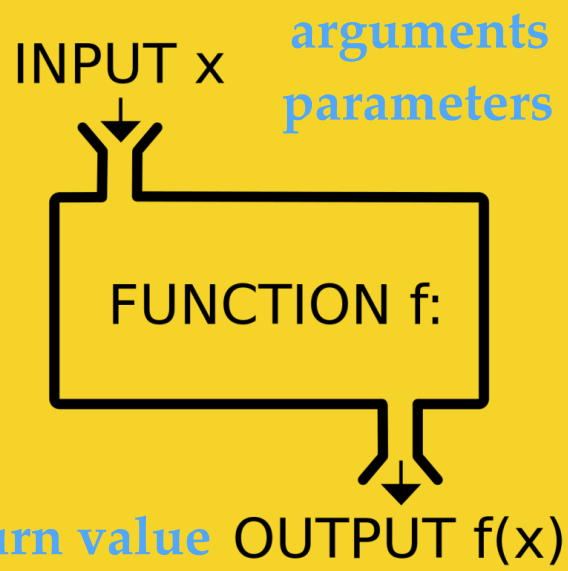
Functions

Michael Conti

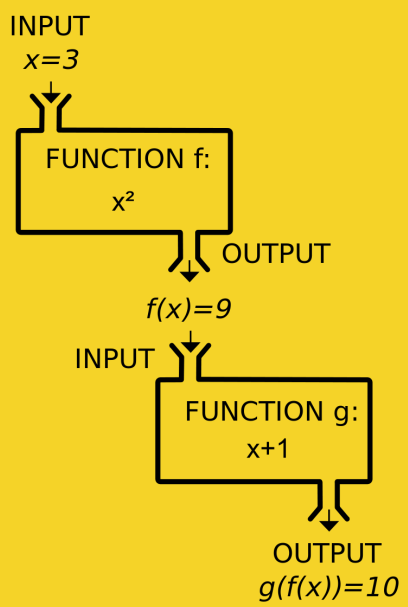
Department of Computer Science and Statistics
University of Rhode Island

Spring 2020

THINK BIG WE DO℠



https://blogs.oracle.com/developers/the-power-of-functional-programming



https://blogs.oracle.com/developers/the-power-of-functional-programming

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)

`return` statement

- Ends the function call
  - returns a value

```
return expression;
```


Label all function parts

```
#include <iostream>

int abs(int n);

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

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

declaration or prototype

function call

definition

Functions are black boxes

DISPLAY 4.7 Definitions That Are Black-Box Equivalent

Function Declaration

```
double newBalance(double balancePar, double ratePar);
//Returns the balance in a bank account after
//posting simple interest. The formal parameter balancePar is
//the old balance. The formal parameter ratePar is the interest rate.
//For example, if ratePar is 5.0, then the interest rate is 5 percent
//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

void functions

- 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

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


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

Function Declaration

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

Function Definition

```
//Definition uses iostream:
void iceCreamDivision(int number, double totalWeight)
{
    using namespace std;
    double portion;

    if (number == 0)
        return;
    portion = totalWeight/Number;
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "Each one receives "
         << portion << " ounces of ice cream." << endl;
}
```

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

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


```
#include <iostream>

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

int main() {
    std::cout << foo(10, 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::basic_ostre
am<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::b
asic_ostream<char, std::__1::char_traits<char> >::char_type, std::__1::basic_ostream<char, std::__1::char_traits<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.
```

```
#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

```
#include <iostream>
using namespace std;

double totalCost(int numberPar, double pricePar);
//Computes the total cost, including 5% sales tax,
//on numberPar items at a cost of pricePar each.

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);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << number << " items at "
         << "$" << price << " each.\n"
         << "Final bill, including tax, is $" << bill
         << endl;

    return 0;
}

double totalCost(int numberPar, double pricePar)
{
    const double TAX_RATE = 0.05;  //5% sales tax
    double subtotal;

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

function declaration/function prototype

function call

function heading

function body

function definition

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

DISPLAY 4.4 Details of a Function Call

```
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;
}
                      2       10.10
double totalCost (int numberPar, double pricePar)
{
    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 to10.10, 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