CSC 211: Computer Programming

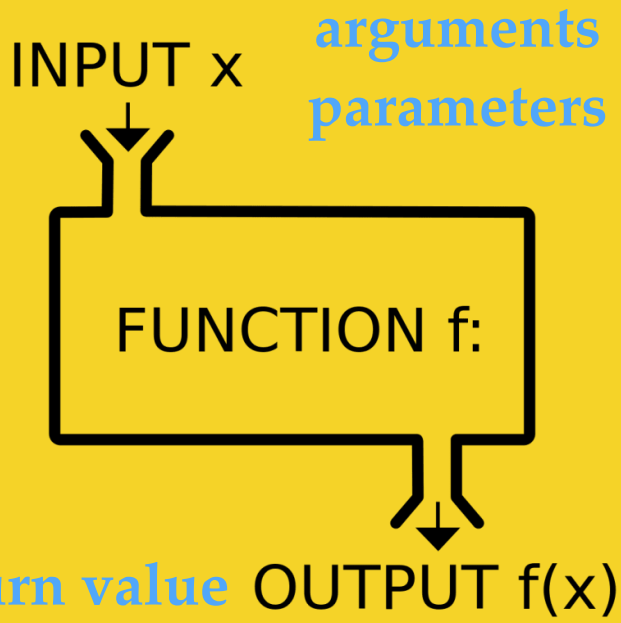
Functions

Michael Conti

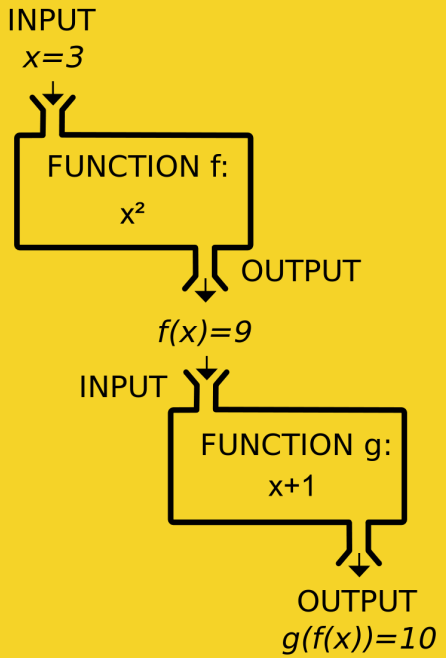
Department of Computer Science and Statistics
University of Rhode Island

Fall 2023

THINK BIG  WE DO℠



https://blogs.oracle.com/developers/the-power-of-functional-programming 2



https://blogs.oracle.com/developers/the-power-of-functional-programming 3

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

Functions

- Two major parts needed a function
  - **Function Definition**
  - **Function Call**

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)
- Can only return one value using the return statement
  - Trick to return multiple values

```
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)

Parameter list (example)

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

```
1   int sum(int a, int b){
2       return a + b;
3   }
4
5   int main(){
6       std::cout << sum(10, 20);
7   }
```


`return` statement

- Ends the function call
  - returns a value

```
return expression;
```

`return` statement

```
return expression;
```

```
1   int sum(int a, int b){
2       return a + b;
3   }
4
5   int main(){
6       std::cout << sum(10, 20);
7   }
```

v.s

```
1   int sum(int a, int b){
2       return a + b;
3   }
4
5   int main(){
6       int mySum = sum(10, 20);
7   }
```


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;
    }
}
```


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

`void` functions

- A function might produce no returning value
  - e.g. sends IP packets to other machine, or sends data to the standard output, modifies elements in an array
- Void functions allow programmers to define functions (and some semantic behavior) with no returning values

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

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.

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);
}
```



```
#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

```
1   #include <iostream>
2   using namespace std;
3
4   double totalCost(int numberPar, double pricePar);
5   //Computes the total cost, including 5% sales tax,
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 $";
16      cin >> price;
17
18      bill = totalCost(number, price);
19
20      cout.setf(ios::fixed);
21      cout.setf(ios::showpoint);
22      cout.precision(2);
23      cout << number << " items at "
24           << "$" << price << " each.\n"
25           << "Final bill, including tax, is $" << bill
26           << endl;
27
28      return 0;
29  }
30
31  double totalCost(int numberPar, double pricePar)
32  {
33      const double TAX_RATE = 0.05;  //5% sales tax
34      double subtotal;
35
36      subtotal = pricePar * numberPar;
37      return (subtotal + subtotal * TAX_RATE);
38  }
```

function declaration/function prototype

function call

function heading

function body

function definition

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.*

Question?

- Write a function `highest_prime` that takes an integer `n > 1` from `stdin` and outputs the largest prime number less than or equal than `n` to the `stdout`
  - use an `is_prime` function to help find `highest_prime`