Chapter 5: Functions and an Intro to Function Templates

C++ How to Program: An Objects-Natural Approach, 11/e

Presented by
Paul Deitel, CEO, Deitel & Associates, Inc.

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Chapter 5: Functions

- Math library and global functions
- Create custom function definitions
- Declare functions with function prototypes
- Key C++ standard library headers
- Random-number generation for simulation
- Scoped enums and C++20’s `using enum` declarations
- Digit separators for more readable numeric literals
- Inline functions

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Chapter 5: Functions

- References
- Default arguments
- Overloaded functions
- Function templates for generating families of overloaded functions
- Recursion
- Scope of identifiers
- Zajnropc vrq lnfylun-lhqtomh uyqmmhzg tupb j dvql psrpu iw dmwwqnddwjqz

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.2 Program Components in C++

- Write C++ programs by combining
  - prepackaged functions and classes available in the C++ standard library,
  - functions and classes available in a vast number of open-source and proprietary
  - third-party libraries
  - new functions and classes you and your colleagues write
- Functions and classes allow you to separate a program’s tasks into small self-contained units

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.2 Program Components in C++

- Motivations for using functions and classes to create program components
  - Software reuse
  - Avoiding code repetition
  - Hiding complexity
  - Easier testing, debugging and maintenance
- Every function should perform a single, well-defined task
  - The function’s name should express that task effectively
- We’ll say more about software reusability in our treatment of object-oriented programming
- C++20 introduces modules (discussed later)

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.3 Math Library Functions

Function	Description	Example

`ceil(x)`	rounds `x` to the smallest integer not less than `x`	`ceil(9.2)` is `10.0`
		`ceil(-9.8)` is `-9.0`

`cos(x)`	trigonometric cosine of `x` (`x` in radians)	`cos(0.0)` is `1.0`

`exp(x)`	exponential function e^x	`exp(1.0)` is `2.718282`
		`exp(2.0)` is `7.389056`

`fabs(x)`	absolute value of `x`	`fabs(5.1)` is `5.1`
		`fabs(0.0)` is `0.0`
		`fabs(-8.76)` is `8.76`

`floor(x)`	rounds `x` to the largest integer not greater than `x`	`floor(9.2)` is `9.0`
		`floor(-9.8)` is `-10.0`

`fmod(x, y)`	remainder of `x/y` as a floating-point number	`fmod(2.6, 1.2)` is `0.2`

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.3 Math Library Functions

Function	Description	Example
`fmod(x, y)`	remainder of `x/y` as a floating-point number	`fmod(2.6, 1.2)` is `0.2`

`log(x)`	natural logarithm of `x` (base *e*)	`log(2.718282)` is `1.0`
		`log(7.389056)` is `2.0`

`log10(x)`	logarithm of `x` (base 10)	`log10(10.0)` is `1.0`
		`log10(100.0)` is `2.0`

`pow(x, y)`	`x` raised to power `y` (x^y)	`pow(2, 7)` is `128`
		`pow(9, .5)` is `3`

`sin(x)`	trigonometric sine of `x` (`x` in radians)	`sin(0.0)` is `0`
`sqrt(x)`	square root of `x` (`x` is non-negative)	`sqrt(9.0)` is `3.0`
`tan(x)`	trigonometric tangent of `x` (`x` in radians)	`tan(0.0)` is `0`

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.3 Math Library Functions

- C++11 Additional Math Functions
  - Dozens of new math functions in `<cmath>`
  - Some entirely new
  - Some additional versions of existing functions for use with arguments of type `float` or `long double`, rather than `double`
- `hypot` function calculates the hypotenuse of a right triangle
  - C++17 version to calculate the hypotenuse in 3D space
- https://en.cppreference.com/w/cpp/numeric/math
- Section 26.8.1 of the C++ standard document
  - http://wg21.link/n4861

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.3 Math Library Functions

- C++20—New Mathematical Constants and the `<numbers>` Header
  - C++ always had common math functions
  - Did not have common mathematical constant
  - Some implementations defined macros
    - `M_PI`, `M_E`, …
  - `<numbers>` standardizes common math constants

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Constant	Mathematical Expression
`numbers::e`	e
`numbers::log2e`	$\log_2 e$
`numbers::log10e`	$\log_{10} e$
`numbers::ln2`	$\ln 2$
`numbers::ln10`	$\ln 10$
`numbers::pi`	π
`numbers::inv_pi`	$\frac{1}{\pi}$
`numbers::inv_sqrtpi`	$\frac{1}{\sqrt{\pi}}$
`numbers::sqrt2`	$\sqrt{2}$
`numbers::sqrt3`	$\sqrt{3}$
`numbers::inv_sqrt3`	$\frac{1}{\sqrt{3}}$
`numbers::egamma`	Euler-Mascheroni γ constant
`numbers::phi`	$\frac{(1 + \sqrt{5})}{2}$

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.3 Math Library Functions

- C++17 added scores of mathematical special functions to `<cmath>`
  - Engineering, scientific and mathematical fields
  - https://en.cppreference.com/w/cpp/numeric/special_functions

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.3 Math Library Functions

C++ 17 Mathematical Special Functions: `float`, `double` and `long double` versions	
associated Laguerre polynomials associated Legendre polynomials beta function (complete) elliptic integral of the first kind (complete) elliptic integral of the second kind (complete) elliptic integral of the third kind regular modified cylindrical Bessel functions cylindrical Bessel functions (of the first kind) irregular modified cylindrical Bessel functions cylindrical Neumann functions (incomplete) elliptic integral of the first kind	(incomplete) elliptic integral of the second kind (incomplete) elliptic integral of the third kind exponential integral Hermite polynomials Legendre polynomials Laguerre polynomials Riemann zeta function spherical Bessel functions (of the first kind) spherical associated Legendre functions spherical Neumann functions

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.4 Function Definitions and Function Prototypes

- Fig. 5.1
- Create a custom function definition
- Declare the function with a function prototype
  - Describes the function’s interface
- Call the function
- Return from the function

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.5 Order of Evaluation of a Function’s Arguments

- Commas between arguments are not comma operators
- Comma operator guarantees left-to-right evaluation
  - `a, b, c;`
- Argument evaluation order is not specified
- If arguments are expressions, order of evaluation could affect the values of one or more of the arguments
- Could cause subtle logic errors
- Assign arguments to variables before a call to force order

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.6 Function-Prototype and Argument-Coercion Notes

- Prototype required unless function is defined before it’s used
- If a function is defined first, the definition serves as the prototype
- Always including prototypes eliminates the need to worry about function definition order

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.6.1 Function Signatures and Function Prototypes

- Function name and parameter types form the function signature
- Return type is not part of function signature
  - Signatures cannot differ only by return type
- Functions in same scope must have unique signatures

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.6.2 Argument Coercion

- Forcing arguments to the appropriate types
- E.g., can call a function with an `int` argument, even though the function prototype specifies a `double` parameter
- Error if the arguments cannot be implicitly converted to the expected parameter types specified

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.6.3 Argument-Promotion Rules and Implicit Conversions

- Define allowed implicit conversions

Data types	
`long double`	
`double`	
`float`	
`unsigned long long int`	(synonymous with `unsigned long long`)
`long long int`	(synonymous with `long long`)
`unsigned long int`	(synonymous with `unsigned long`)
`long int`	(synonymous with `long`)
`unsigned int`	(synonymous with `unsigned`)
`int`	
`unsigned short int`	(synonymous with `unsigned short`)
`short int`	(synonymous with `short`)
`unsigned char`	
`char and signed char`	
`bool`	

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.6.3 Argument-Promotion Rules and Implicit Conversions

- Narrowing conversions
  - C++ Core Guidelines recommend using a narrow_cast operator
  - Guidelines Support Library (GSL)
  - https://github.com/Microsoft/GSL
  - Header `"gsl/gsl"`
  - `gsl::narrow_cast<int>(7.5)`

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <iostream>
  - C++ standard input and output functions
- <iomanip>
  - Stream manipulators that format streams of data.

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <cmath>
  - Math library functions
- <cstdlib>
  - Conversions of numbers to text, text to numbers, memory allocation, random numbers and various other utility functions
- <ctime>
  - Contains function prototypes and types for manipulating the time and date.

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <array>, <vector>, <list>, <forward_list>, <deque>, <queue>, <stack>, <map>, <unordered_map>, <unordered_set>, <set>, <bitset>
  - C++ Standard Library containers. Containers store data during a program’s execution.

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <cctype>
  - Functions that test characters for certain properties (such as whether the character is a digit or a punctuation), and functions that can be used to convert lowercase letters to uppercase letters and vice versa.
- <cstring>
  - C-style string-processing functions
- <typeinfo>
  - Runtime type identification (determining data types at execution time)
- <exception>, <stdexcept>
  - Classes that are used for exception handling

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <memory>
  - Classes and functions used by the C++ Standard Library to allocate memory to the C++ Standard Library containers
- <fstream>
  - Functions that perform input from and output to files on disk
- <string>
  - Class string from the C++ Standard Library
- <sstream>
  - Functions that perform input from strings in memory and output to strings in memory

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <functional>
  - Classes and functions used by C++ Standard Library algorithms.
- <iterator>
  - Classes for accessing data in the C++ Standard Library containers
- <algorithm>
  - Functions for manipulating data in C++ Standard Library containers

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <cassert>
  - Macros for adding diagnostics that aid program debugging.
- <cfloat>
  - Floating-point size limits of the system
- <climits>
  - Integral size limits of the system
- <cstdio>
  - C-style standard input/output library functions

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.7 C++ Standard Library Headers

- <locale>
  - Classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.)
- <limits>
  - Classes for defining the numerical data type limits on each computer platform—this is C++’s version of <climits> and <cfloat>
- <utility>
  - Classes and functions that are used by many C++ Standard Library headers

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.8 Case Study: Random-Number Generation

- 5.8.1 Rolling a Six-Sided Die
- 5.8.2 Rolling a Six-Sided Die 60,000,000 Times
- 5.8.3 Seeding the Random-Number Generator
- 5.8.4 Seeding the Random-Number Generator with `random_device`

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.8 Case Study: Random-Number Generation

- `<random>` header
- Replaces deprecated `rand` function (from C)
  - Had poor statistical properties
  - Could be predicted, making it less secure
- Features from `<random>` can produce **nondeterministic random numbers** that can’t be predicted
- Important for simulations and security scenarios where predictability is undesirable

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.8.4 Seeding the Random-Number Generator with `random_device`

- `random_device` typically used to seed a random-number generator
- Produces evenly spread random integers, which cannot be predicted—nondeterministic
- Slow performing, so typically used only to seed engines
- **Check docs for your platform, as `random_device` might be predictable on some platforms**

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.8 Case Study: Random-Number Generation

- Many classes representing random-number generation engines and distributions
  - **engine** implements a random-number generation algorithm that produces pseudorandom numbers
  - **distribution** controls the range of values produced by an **engine**, the types of those values and the statistical properties of the values
- `uniform_int_distribution`
  - Distributes pseudorandom integers evenly over a range

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.9 Case Study: Game of Chance; Introducing Scoped enums

- You roll two six-sided dice.
- After the dice have come to rest, the sum of the spots on the two upward faces is calculated.
- 7 or 11 on the first roll, you win
- 2, 3 or 12 on the first roll (called “craps”), you lose (i.e., the “house” wins)
- 4, 5, 6, 8, 9 or 10 on the first roll—that sum becomes your “point”
  - To win, keep rolling the dice until you “make your point”
  - The player loses by rolling a 7 before making the point

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

C++20: `using enum` Declaration

- For cases in which context is obvious
- Reference an `enum class`'s constants without the type name and scope-resolution operator (`::`)
- `using enum Status;`
  - Allows code to use `keepRolling`, `won` and `lost`, rather than `Status::keepRolling`, `Status::won` and `Status::lost`
- `using Status::keepRolling;`
  - Allow code to use just `keepRolling` without `Status::`
- Placed inside block that uses them

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.11 Inline Functions

- `inline`
  - Advice to the compiler
  - Expand function definition where it's called
  - Removes the overhead of the function call
  - `inline` functions typically placed in headers
  - If definition changes, code using it must be recompiled
- Declare "small and time critical" functions inline
  - https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines.html#Rf-inline
- https://isocpp.org/wiki/faq/inline-functions

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.13 References and Reference Parameters

- Two ways to pass arguments to functions
- Pass-by-value
  - Copy of argument’s value is passed to the called function
  - Changes to the copy do not affect the original variable’s value in the caller
- Pass-by-reference
  - Caller gives the called function the ability to access that variable in the caller directly and to modify the variable
- A **reference parameter** is an alias for its corresponding argument in a function call
- Pass-by-reference can be good for performance
  - Eliminates overhead of copying large amounts of data
  - Caution: Function could corrupt data by modifying it incorrectly

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

5.13 References and Reference Parameters

`const` References

- Qualify reference parameter with const to specify that a reference parameter should not be allowed to modify the corresponding argument in the caller
- Consider a `displayName` function:

```
void displayName(std::string name) {
   std::cout << name << '\n';
}
```

- Receives a copy of its `string` argument
- `string` objects can be large, so this copy could degrade performance
- Declare the parameter as
  - `const std::string& name`
  - Reading from right to left, name is a reference to a `string` that is constant

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Default Arguments

- Common for a program to invoke a function from several places with the same argument value for a particular parameter
- Can specify a default argument for the parameter
- When a program omits the argument for a parameter, compiler inserts the default argument

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Function Overloading

- Can define functions of the same name
- Must have different signatures
- Called function overloading
- Compiler selects the proper function to call by examining the number, types and order of the arguments in the call
- Used to create several functions of the same name that perform similar tasks but on data of different types
- Many math library functions are overloaded for different numeric types
- Overload resolution details are complex
    - **https://en.cppreference.com/w/cpp/language/overload_resolution**

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Function Overloading

```
_Z6squarei
_Z6squared
_Z8nothing1ifcRi
_Z8nothing2ciRfRd
main
```

DEITEL®

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Function Templates

- If overloaded functions’ logic and operations are identical, overloading may be performed more compactly and conveniently with **function templates**
- Write one function template definition
- C++ generates **function template specializations** (also called **template instantiations**) that handle calls for the provided argument types
- One function template defines a family of overloaded functions
- Known as generic programming

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Recursion

- Recursive functions call themselves directly or indirectly through other functions
- Can help solve problems more naturally when an iterative solution is not apparent

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Recursion

- A recursive function can solve only its base case(s)
- For a more complex problem, it divides into
  - A piece it knows how to do
  - A piece it does not know how to do, which must be a slightly simpler or smaller version of the original problem
- Function calls itself to work on the smaller problem
  - Recursive call (or recursion step)
- Smaller problems must converge on a base case
- When the function recognizes a base case, it returns a result to the previous copy of the function initiating a chain of returns until the result is returned to the caller

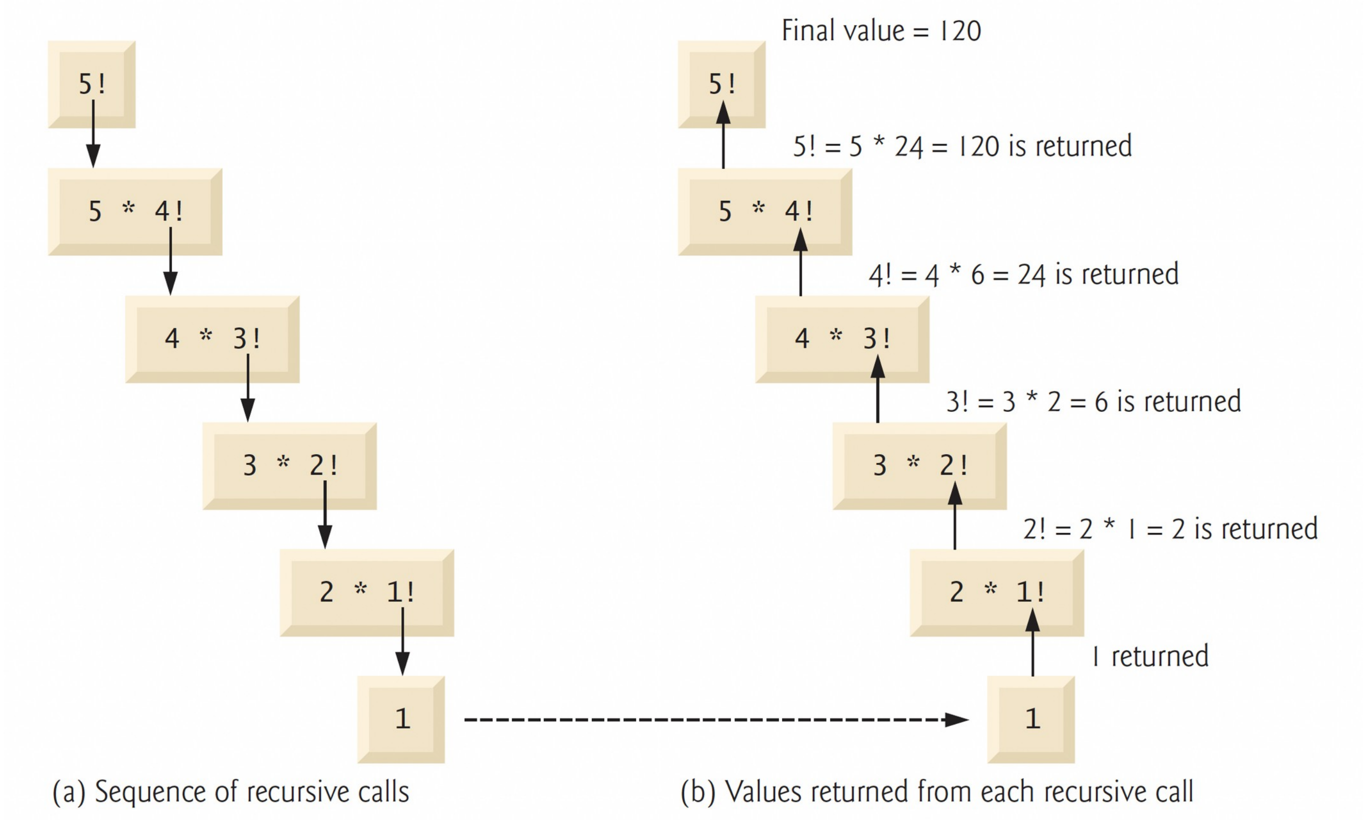
©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Recursion—Factorials

- Factorial of a nonnegative integer n, written $n!$ (pronounced “n factorial”), is the product
  - $n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$
  - 1! is 1
  - 0! is defined to be 1
- Recursive definition of factorial
  - $n! = n \cdot (n - 1)!$

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Recursion—Factorials



Final value = 120
5!
5 * 4!
4 * 3!
3 * 2!
2 * 1!
1
5! = 5 * 24 = 120 is returned
4! = 4 * 6 = 24 is returned
3! = 3 * 2 = 6 is returned
2! = 2 * 1 = 2 is returned
1 returned
(a) Sequence of recursive calls
(b) Values returned from each recursive call

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Example Using Recursion: Fibonacci Series

- The Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, …
- Begins with 0 and 1
- Each subsequent Fibonacci number is the sum of the previous two
- Occurs in nature and describes a form of spiral
- Ratio of successive numbers converges on 1.618…
  - golden ratio or the golden mean
- Aesthetically pleasing
  - Windows, rooms, buildings, postcards designed in this ration

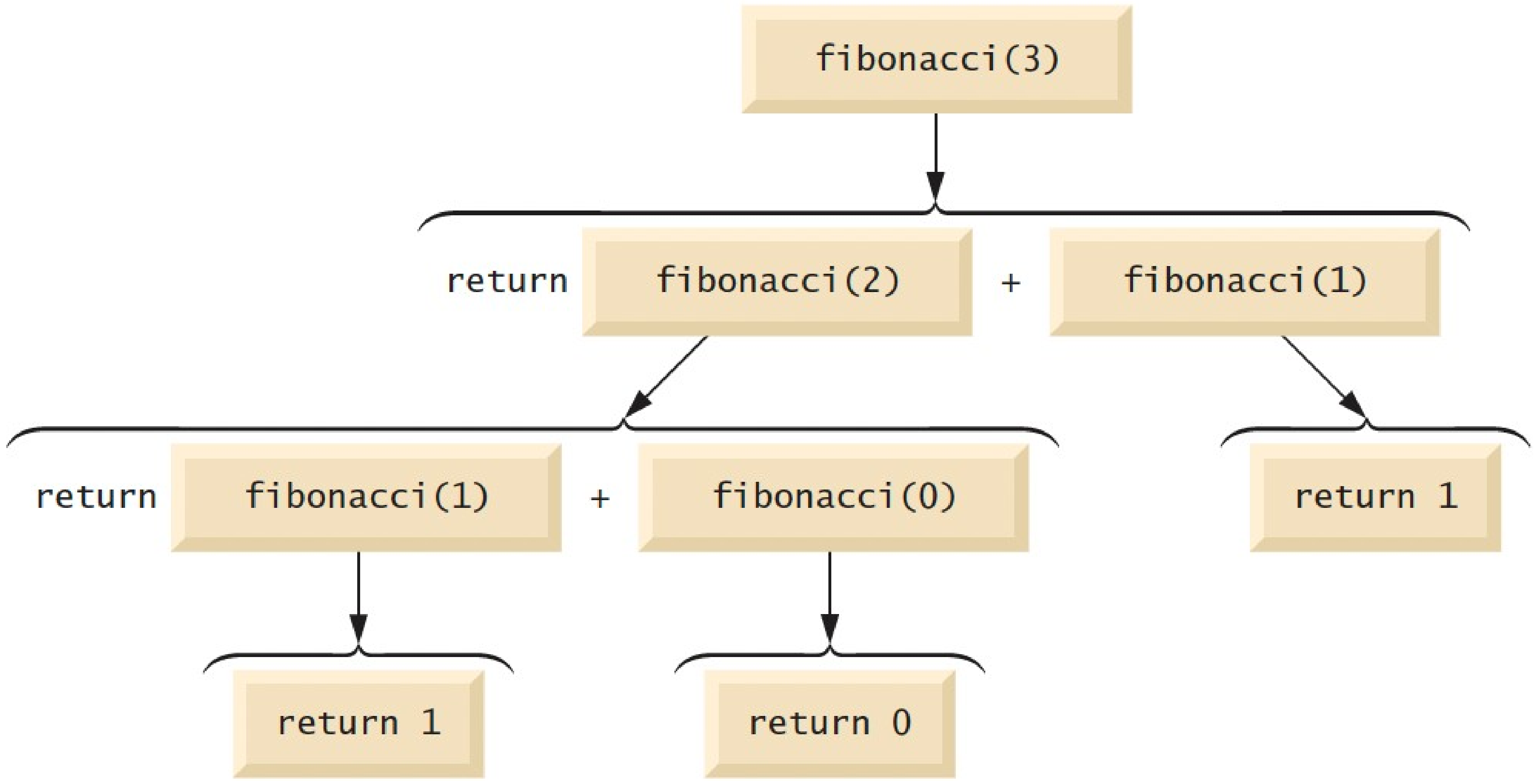
©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Example Using Recursion: Fibonacci Series

- Recursive Fibonacci Definition
  - fibonacci(0) = 0
  - fibonacci(1) = 1
  - fibonacci(n) = fibonacci(n – 1) + fibonacci(n – 2)

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Example Using Recursion: Fibonacci Series



©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Example Using Recursion: Fibonacci Series

Complexity Issues

- Each invocation of `fibonacci` that does not match a base case (0 or 1) results in two more recursive calls
- Each consecutive Fibonacci number causes substantial increase in calculation time and number of calls
  - `fibonacci(20)` requires 21,891 calls
  - `fibonacci(30)` requires 2,692,537 calls
  - `fibonacci(31)` requires 4,356,617 calls
  - `fibonacci(32)` requires 7,049,155 calls
- Can humble even the world’s most powerful computers

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Recursion vs. Iteration

- Both are based on a control statement
  - Iteration uses an iteration statement, whereas recursion uses a selection statement
- Both involve performing tasks repeatedly:
  - Iteration uses an iteration statement, whereas recursion uses repeated function calls
- Both involve a termination test
  - Iteration terminates when the loop-continuation condition fails, whereas recursion terminates when a base case is reached

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Recursion vs. Iteration

- Both gradually approach termination
  - Iteration keeps modifying a counter until the loop-continuation condition fails, whereas recursion keeps producing smaller versions of the original problem until the base case is reached
- Both can occur infinitely
  - An infinite loop occurs if the loop-continuation test never becomes false, whereas infinite recursion occurs if the recursion step does not reduce the problem each time in a manner that converges on the base case or if the base case is mistakenly not tested

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Scope Rules

- The portion of the program where an identifier can be used is known as its scope.
- Here we discuss
  - block scope
  - global namespace scope

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Scope Rules (cont.)

- Block scope
  - Identifiers declared inside a block (that is, curly braces)
  - Begins at the identifier’s declaration
  - Ends at the terminating right brace (`}`) of the enclosing block.
  - Local variables have block scope, as do function parameters
- In nested blocks, if an identifier in an outer block has the same name as an identifier in an inner block, the one in the outer block is “hidden” until the inner block terminates

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Scope Rules (cont.)

- Global namespace scope
  - Identifier declared outside any function or class
  - “known” from the point at which it’s declared until the end of the file
  - Function definitions, function prototypes placed outside a function, class definitions and global variables all have global namespace scope

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz

- Gibberish above is not a mistake 🏝
- It’s an encrypted message
- Produced with a Vignère secret key cipher
  - https://en.wikipedia.org/wiki/Vigenère_cipher
- Cryptography has been in use for thousands of years
  - https://en.wikipedia.org/wiki/Cryptography#History_of_cryptography_and_cryptanalysis
  - https://www.binance.vision/security/history-of-cryptography

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz

- Caesar Cipher:
  - Simple substitution cipher to encrypt military communications
  - Every letter replaced with letter three ahead in the alphabet
  - **Plain text** “Caesar Cipher” becomes **ciphertext** “Fdhvdu Flskhu”
  - Play around with ciphers
    - https://cryptii.com/pipes/caesar-cipher
    - https://github.com/cryptii/cryptii

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com

Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz

- Caesar cipher is relatively easy to decrypt
- Vignère Cipher
  - Secret key and 26 Caesar Ciphers offset by 0-25 characters for A-Z
- You’ve created objects of C++ standard library classes and objects of open-source library classes
  - Sometimes you’ll use classes from your organization or team
  - We wrote our own `Cipher` class for you to use here
  - Don’t need to know how it works internally to use it

©Copyright 1992-2024 by Pearson Education, Inc. All Rights Reserved. https://deitel.com