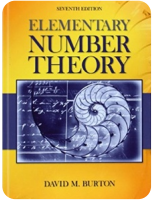


Chapter 19 C++ Study Guide

Share

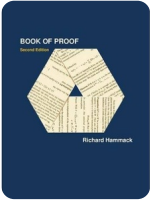
9 studiers today   ★ Leave the first rating

Textbook solutions for this set



**Elementary Number Theory**  
7th Edition · ISBN: 9780073383149 (2 more)  
David Burton

776 solutions



**Book of Proof**  
2nd Edition · ISBN: 9780989472104  
Richard Hammack

343 solutions

Search for a textbook or question >

Terms in this set (48)

<p>[2301] Given the function below, what does cout &lt;&lt; mystery(3) print?</p> <pre>int mystery(int n) { if (n &lt; 2) return 1; return n * mystery(n - 1); } 6 120 2 24</pre>	6
<p>[2302] If you write mystery(10), how many times is the function called?</p> <pre>int mystery(int n) { if (n &lt;= 2) return 1; return n * mystery(n - 1); } 120 10 6 9</pre>	9

<p>[2303] What does this function do?</p> <pre>int mystery(int n) { if (n == 1) return 1; return n * mystery(n-1); }</pre> <p>Computes the reverse of the input n Computes the Gauss series (sum) of 1..n Computes the Factorial number n Computes the Fibonacci number n Produces a stack overflow</p>	Computes the Factorial number n
---	---------------------------------

<pre>int mystery(int n) { if (n &lt; 2) return 1; return mystery(n-1) + mystery(n-2); }</pre> <p>Computes the Gauss series (sum) of 1..n Computes the Factorial number n Computes the Fibonacci number n Computes the reverse of the input n Produces a stack overflow</p>	
<p>[2305] What does this function do?</p> <pre>int mystery(int n) { if (n == 1) return 1; return n * mystery(n+1); }</pre> <p>Computes the Gauss series of n Computes the Fibonacci number n Produces a stack overflow Computes the Factorial number n Computes the reverse of the input n</p>	Produces a stack overflow
<p>[2306] What does this function do?</p> <pre>int mystery(int n) { if (n == 1) return 1; return n + mystery(n-1); }</pre> <p>Computes the Factorial number n Computes the reverse of the input n Computes the Fibonacci number n Produces a stack overflow Computes the Gauss series (sum) of 1..n</p>	Computes the Gauss series (sum) of 1..n
<p>[2307] What does this function do?</p> <pre>int mystery(int n, int m) { if (n == 0) return m; return m * 10 + mystery(n / 10) + n % 10; }</pre> <p>Produces a stack overflow Computes the reverse of the input n Computes the Factorial number n Computes the Gauss series (sum) of 1..n Computes the Fibonacci number n</p>	Computes the reverse of the input n
<p>[2308] What is the value of mystery(12)?</p> <pre>int mystery(int n) { if (!n) return 0; return 2 + mystery(n-1); }</pre> <p>18 24 36 12</p>	24

<pre>int r(int n) {     if (n &gt; 0) return n + r(n - 1);     return n; }</pre> <p>15 6 10 24 21</p>	
<p>[2310] What is the value of mystery(5)?</p> <pre>int mystery(int n) {     if (n &gt; 0) return 3 - n % 2 + mystery(n-1);     return 0; }</pre> <p>7 12 5 10 15</p>	12
<p>[2311] What is the value of r(126)?</p> <pre>int r(int n) {     if (n &gt;= 10) return n % 10 + r(n / 10);     return n; }</pre> <p>3 6 13 10 9</p>	9
<p>[2312] What is the value of r(12777)?</p> <pre>int r(int n) {     if (0 == n) return 0;     int x = n % 10 == 7; // 0 or 1     return x + r(n / 10); }</pre> <p>5 Does not compile 2 3 Stack overflow</p>	3

<pre>int r(int n) { if (n) return (n % 10 == 7) + r(n / 10); return 0; }</pre> <p>3</p> <p>5</p> <p>Does not compile</p> <p>8</p> <p>Stack overflow</p>	
<p>[2314] What is the value of r(74757677)?</p> <pre>int r(int n) { if (n) return (n % 10 != 7) + r(n / 10); return 0; }</pre> <p>5</p> <p>3</p> <p>Does not compile</p> <p>8</p> <p>Stack overflow</p>	3
<p>[2315] What is the value of r(8818)?</p> <pre>int r(int n) { if (!n) return 0; return (n % 10 == 8) + (n % 100 == 88) + r(n / 10); }</pre> <p>Stack overflow</p> <p>4</p> <p>Does not compile</p> <p>3</p> <p>1</p>	4
<p>[2316] What is the value of r(81238)?</p> <pre>int r(int n) { if (!n) return 0; return (n % 10 == 8) + (n % 100 == 88) + r(n / 10); }</pre> <p>Does not compile</p> <p>2</p> <p>Stack overflow</p> <p>5</p> <p>3</p>	2
<p>[2317] What is the value of r(88788)?</p> <pre>int r(int n) { if (!n) return 0; return (n % 10 == 8) + (n % 100 == 88) + r(n / 10); }</pre> <p>4</p> <p>1</p> <p>5</p> <p>6</p> <p>Stack overflow</p>	6

<pre>int r(int n, int m) {     if (m) return n * r(n, m - 1);     return 1; } 12 27 Stack overflow 9 3</pre>	
<p>[2319] What is the value of r("xxhixx")?</p> <pre>int r(const string&amp; s) {     if (s.size())         return (s.at(0) == 'x') + r(s.substr(1));     return 0; }</pre> <p>4 2 3 6 Stack overflow</p>	4
<p>[2321] What is the value of r("xxhixx")?</p> <pre>string r(const string&amp; s) {     if (s.empty()) return "";     if (s.at(0) == 'x') return 'y' + r(s.substr(1));     return s.at(0) + r(s.substr(1)); }</pre> <p>xxyyxx yyhiyy xyxyhixyxy yxyxhixyyx Stack overflow</p>	yyhiyy
<p>[2322] What is the value of r("xhixhix")?</p> <pre>string r(const string&amp; s) {     if (s.size()) {         auto c = s.at(0);         auto t = c == 'x' ? 'y' : c;         return t + r(s.substr(1));     }     return 0; }</pre> <p>Stack overflow yyyyyyy xyxyyyx yhiyhiy xyhixyhixy</p>	yhiyhiy

<pre>string r(const string&amp; s) {     auto front = s.substr(0, 1);     if (front.empty()) return "";     return (front == "x" ? "" : front) + r(s.substr(1)); }  "a b " "xxxx" "ax bx " "ab" Stack overflow</pre>	
--	--

<p>[2324] What is the value of r("axxbxx")?</p> <pre>string r(const string&amp; s) {     auto front = s.substr(0, 1);     if (front.empty()) return "";     return (front == "x" ? front : "") + r(s.substr(1)); }  "ax bx " "a b " Stack overflow "xxxx" "ab"</pre>	"xxxx"
--	--------

<p>[2325] Assume you have the array: int a[] = {1, 11, 3, 11, 11};. What is the value of r(a, 0, 5)?</p> <pre>int r(const int a[], size_t i, size_t max) {     if (i &lt; max) return (a[i] == 11) + r(a, i + 1);     return 0; }</pre> <p>3 5 Stack overflow 1 0</p>	3
---	---

<p>[2326] What is the value of r("hello")?</p> <pre>string r(const string&amp; s) {     if (s.size() &lt; 2) return s;     return s.substr(0, 1) + "*" + r(s.substr(1)); }  "hell*o" "hello*" "hello" Stack overflow "hello"</pre>	"hello"
--	---------

<pre>string r(const string&amp; s) {     if (s.size() &gt; 1) {         string t = s[0] == s[1] ? "" : "**";         return s[0] + t + r(s.substr(1));     }     return s; }</pre> <p>"hello" Stack overflow "hell*o" "hello" "hel*lo"</p>	
<p>[2328] What is the value of r("hello")?</p> <pre>string r(const string&amp; s) {     if (s.size() &gt; 1) {         string t = s[0] == s[1] ? "" : "**";         return s[0] + t + r(s.substr(1));     }     return s; }</pre> <p>"hell*o" "hel*lo" "hello" Stack overflow "hello"</p>	<p>"h*e*ll*o"</p>
<p>[2329] What is the value of r("hello")?</p> <pre>string r(const string&amp; s) {     if (s.size() &gt; 1) {         string t = s[0] == s[1] ? "" : "**";         return t + s[0] + r(s.substr(1));     }     return s; }</pre> <p>"hello" Stack overflow "hell*o" "hel*lo"</p> <p>"*h*el*lo"</p>	
<p>[2330] Which of the following statements is correct about a recursive function?</p> <p>A recursive function must never call another function.</p> <p>A recursive function calls itself.</p> <p>A recursive function must be simple.</p> <p>A recursive function must call another function.</p>	<p>A recursive function calls itself.</p>

<pre>void myfun(string word) {     if (word.length() == 0) return;     myfun(word.substr(1, word.length()));     cout &lt;&lt; word[0]; }</pre> <p>Prints the length of the string word</p> <p>Prints the string word both forward and reverse</p> <p>Prints the string word in reverse</p> <p>Prints the string word</p>	
---	--

<p>[2332] What changes about this function if lines 4 and 5 are swapped?</p> <p>1. void myfun(string word)</p> <p>2. {</p> <p>3. if (word.length() == 0) { return; }</p> <p>4. myfun(word.substr(1, word.length()));</p> <p>5. cout &lt;&lt; word[0];</p> <p>6. }</p> <p>prints the characters of the string in both forward and reverse order</p> <p>creates infinite recursion</p> <p>nothing</p> <p>reverses the order in which the characters of the string are printed</p>	<p>reverses the order in which the characters of the string are printed</p>
---	---

<p>[2333] Which of the following is true about using recursion?</p> <p>Recursion always helps you create a more efficient solution than other techniques.</p> <p>A recursion eventually exhausts all available memory, causing the program to terminate</p> <p>A recursive computation solves a problem by calling itself with simpler input.</p> <p>None of the listed options.</p>	<p>A recursive computation solves a problem by calling itself with simpler input.</p>
--	---

<p>[2334] How can you ensure that a recursive function terminates?</p> <p>Call the recursive function with simpler inputs.</p> <p>Use more than one return statement.</p> <p>Provide a special case for the simplest inputs.</p> <p>Provide a special case for the most complex inputs.</p>	<p>Provide a special case for the simplest inputs</p>
---	---



Every recursive call must simplify the computation in some way

A recursive solution should not be implemented to a problem that can be solved iteratively

There should be special cases to handle the most complex computations directly

A recursive function should not call itself except for the simplest inputs

[2336] What is the value of r(3)?

```
int r(int n)
{
  if (n < 2) { return 1; }
  return n * r(n - 1);
}
```

- 24
- 2
- 120
- 6

6

[2337] Which statement ensures that r() terminates for all values of n?

```
int mr(int n)
{
  // code goes here
  return r(n - 1) + n * n;
}

if (n == 1) { return 1; }
if (n == 0) { return 0; }

if (n == 0) { return 0; }

if (n < 1) { return 1; }

if (n == 1) { return 1; }
```

if (n < 1) { return 1; }

[2338] Infinite recursion can lead to an error known as

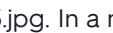
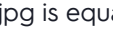


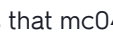

- stack overflow
- heap exhaustion
- heap fragmentation
- memory exception


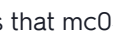

stack overflow

[2339] Infinite recursion can occur because

- the base case is missing one of the necessary termination conditions
- the recursive function is called more than once
- the recursive case is invoked with simpler arguments
- a second function is called from the recursive one

the base case is missing one of the necessary termination conditions

Chapter 19 C++ Study Guide		Study	...
<p>b are line segments, the golden section is a line segment divided according to the golden ratio: The total length (a + b) is to the longer segment a as a is to the shorter segment b. One way to calculate the golden ratio is through the continued square root (also called an infinite surd): golden ratio = . In a recursive implementation of this function, what should be the base case for the recursion?</p> <pre>if (number &lt;= 1) { return pow(number, 2.0);} if (number &lt;= 1) { return sqrt(number);} if (number &lt;= 1) { return 0.0;} if (number &lt;= 1) { return 1.0;}</pre>			
<p>[2341] Two quantities a and b are said to be in the golden ratio if  is equal to . Assuming a and b are line segments, the golden section is a line segment divided according to the golden ratio: The total length (a + b) is to the longer segment a as a is to the shorter segment b. One way to calculate the golden ratio is through the continued square root (also called an infinite surd): golden ratio</p> <p>If the function double golden (int) is a recursive implementation of this function, what should be the recursive call in that function?</p> <pre>return sqrt (1.0 + golden(number)); return sqrt (1.0 + golden(number - 1)); return (1.0 + golden(number - 1)); return (1.0 + golden(number));</pre>		<pre>return sqrt (1.0 + golden(number - 1));</pre>	
<p>[2342] In 1735 Leonard Euler proved a remarkable result, which was the solution to the Basel Problem, first posed in 1644 by Pietro Mengoli. This result gave a simple expression for . The formula states that  is equal to the limit, as n goes to infinity, of the series . Can this series be computed recursively?</p> <p>Yes, but the code will be very long</p> <p>No, because the base case is not zero</p> <p>Yes</p> <p>No, because there is no base case</p>		Yes	
<p>[2343] In 1735 Leonard Euler proved a remarkable result, which was the solution to the Basel Problem, first posed in 1644 by Pietro Mengoli. This result gave a simple expression</p> <p>The formula states that equal to the limit, as n goes to infinity, of the series</p> <p>Which function below is a correct recursive implementation that approximates this infinite series?</p>		<pre>double computePI(int number) { if (number &lt;= 1) { return 1.0;} return 1.0 / (number * number) + computePI(number - 1); }</pre>	

<p>[2344] In 1735 Leonard Euler proved a remarkable result, which was the solution to the Basel Problem, first posed in 1644 by Pietro Mengoli. This result gave a simple expression for . The formula states that  is equal to the limit, as n goes to infinity, of the series . Which statement below is the correct base case for a recursive implementation that approximates this infinite series?</p> <pre>if (number == 0) { return 1.0 / (number * number);} if (number &lt;= 1) { return 1.0;} if (number &lt;= 1) { return 0.0;} if (number == 1) { return (number * number);}</pre>		<pre>if (number &lt;= 1) { return 1.0;}</pre>	
---	--	---	--

Chapter 19 C++ Study Guide		Study	...
<p>1644 by Pietro Mengoli. This result gave a simple expression for mc045-1.jpg. The formula states that mc045-2.jpg is equal to the limit, as n goes to infinity, of the series mc045-3.jpg. Which statement below is the recursive case for a recursive implementation that approximates this infinite series?</p> <pre>return 1.0 / (number * number) + computePI(number - 1); return 1.0 + computePI(number); return 1.0 + computePI(number - 1); return 1.0 / (number * number) + computePI(number);</pre>			
<p>[2346] One remarkably simple formula for calculating the value of is the so-called Madhava-Leibniz series: Consider the recursive function below to calculate this formula:</p> <pre>double computePI(int number) { if (number &lt;= 1) { return 1.0;} int oddnum = 2 * number - 1; return computesign(number) * 1.0 / oddnum + computePI(number - 1); }</pre> <p>In this recursive function, what is the recursive base case?</p> <p>When the parameter variable is less than or equal to one</p> <p>When the parameter variable is greater than one</p> <p>When the value that is returned from the function is zero</p> <p>When the parameter variable is zero</p>	<p>When the parameter variable is less than or equal to one</p>		
<p>[2347] One remarkably simple formula for calculating the value of mc047-1.jpg is the so-called Madhava-Leibniz series: mc047-2.jpg = mc047-3.jpg . Consider the recursive function below to calculate this formula:</p> <pre>double computePI(int number) { if (number &lt;= 1) { return 1.0;} int oddnum = 2 * number - 1; return computesign(number) * 1.0 / oddnum + computePI(number - 1); }</pre> <p>In this recursive function, what is the role of the helper function computesign?</p> <p>it is the recursive call in the function</p> <p>it checks the sign of the number and returns true if it is positive and false if negative</p> <p>it is called just one time to set the sign of the final result</p> <p>it makes sure the sign (positive or negative) alternates as each term of the series is computed</p>	<p>it makes sure the sign (positive or negative) alternates as each term of the series is computed</p>		
<p>[2348] Assuming that you need to write a recursive function calc_prod(int n) to calculate the product of the first n integers, which of the following would be a correct way to simplify the input for the recursive call?</p> <p>Call calc_prod(n - 1) and multiply by n.</p> <p>Call calc_prod(n + 1) and multiply by n.</p> <p>Call calc_prod(n - 2) and multiply by n.</p> <p>Call calc_prod(1) and multiply by n.</p>	<p>Call calc_prod(n - 1) and multiply by n.</p>		

Which of the following would be a correct way to implement the function power?

Call power(x, n) and multiply by (n - 1).

Call power(x, n - 1) and multiply by n.

Call power(x - 1, n) and multiply by x.

Call power(x, n - 1) and multiply by x.