Chapter 19 C++ Study Guide

Share

9 studiers today ★ Leave the first rating

Terms in this set (48)

[2301] Given the function below, what does cout << mystery(3) print?

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

6
120
2
24

6

[2302] If you write mystery(10), how many times is the function called?

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

120
10
6
9

9

[2303] What does this function do?

```
int mystery(int n)
{
if (n == 1) return 1;
return n * mystery(n-1);
}
```

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

Computes the Factorial number n

[2304] What does this function do?

```
int mystery(int n)
{
if (n < 2) return 1;
return mystery(n-1) + mystery(n-2);
}
```

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

Computes the Fibonacci number n

[2305] What does this function do?

```
int mystery(int n)
{
if (n == 1) return 1;
return n * mystery(n+1);
}
```

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

Produces a stack overflow

[2306] What does this function do?

```
int mystery(int n)
{
if (n == 1) return 1;
return n + mystery(n-1);
}
```

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

Computes the Gauss series (sum) of 1..n

[2307] What does this function do?

```
int mystery(int n, int m)
{
if (n == 0) return m;
return m * 10 + mystery(n / 10) + n % 10;
}
```

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

Computes the reverse of the input n

[2308] What is the value of mystery(12)?

```
int mystery(int n)
{
if (!n) return 0;
return 2 + mystery(n-1);
}
```

18
24
36
12

24

[2309] What is the value of r(6)?

```
int r(int n)
{
if (n > 0) return n + r(n - 1);
return n;
}
```

15
6
10
24
21

21

[2310] What is the value of mystery(5)?

```
int mystery(int n)
{
if (n > 0) return 3 - n % 2 + mystery(n-1);
return 0;
}
```

7
12
5
10
15

12

[2311] What is the value of r(126)?

```
int r(int n)
{
if (n >= 10) return n % 10 + r(n / 10);
return n;
}
```

3
6
13
10
9

9

[2312] What is the value of r(12777)?

```
int r(int n)
{
if (0 == n) return 0;
int x = n % 10 == 7; // 0 or 1
return x + r(n / 10);
}
```

5
Does not compile
2
3
Stack overflow

3

[2313] What is the value of r(74757677)?

```
int r(int n)
{
if (n) return (n % 10 == 7) + r(n / 10);
return 0;
}
```

3

5

Does not compile

8

Stack overflow

5

[2314] What is the value of r(74757677)?

```
int r(int n)
{
if (n) return (n % 10 != 7) + r(n / 10);
return 0;
}
```

5

3

Does not compile

8

Stack overflow

3

[2315] What is the value of r(8818)?

```
int r(int n)
{
if (!n) return 0;
return (n % 10 == 8) + (n % 100 == 88) + r(n / 10);
}
```

Stack overflow

4

Does not compile

3

1

4

[2316] What is the value of r(81238)?

```
int r(int n)
{
if (!n) return 0;
return (n % 10 == 8) + (n % 100 == 88) + r(n / 10);
}
```

Does not compile

2

Stack overflow

5

3

2

[2317] What is the value of r(88788)?

```
int r(int n)
{
if (!n) return 0;
return (n % 10 == 8) + (n % 100 == 88) + r(n / 10);
}
```

4

1

5

6

Stack overflow

6

[2318] What is the value of r(3, 3)?
 	
int r(int n, int m)
{
if (m) return n * r(n, m - 1);
return 1;
}
12
27
Stack overflow
9
3
	27

[2319] What is the value of r("xxhixx")?
 	
int r(const string& s)
{
if (s.size())
return (s.at(0) == 'x') + r(s.substr(1));
return 0;
}
 	
4
2
3
6
Stack overflow
	4

[2321] What is the value of r("xxhixx")?
 	
string r(const string& s)
{
if (s.empty()) return "";
if (s.at(0) == 'x') return 'y' + r(s.substr(1));
return s.at(0) + r(s.substr(1));
}
 	
xxyyxx
yyhiyy
xyxyhixyxy
yxyxhixyyx
Stack overflow
	yyhiyy

[2322] What is the value of r("xhixhix")?
 	
string r(const string& s)
{
if (s.size()) {
auto c = s.at(0);
auto t = c == 'x' ? 'y' : c;
return t + r(s.substr(1));
}
return 0;
}
Stack overflow
yyyyyyy
xyyxyyx
yhiyhiy
xyhixyhixy
	yhiyhiy

[2323] What is the value of r("axxbxx")?

```
string r(const string& 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

"ab"

[2324] What is the value of r("axxbxx")?

```
string r(const string& 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"

"xxxx"

[2325] Assume you have the array: int a[] = {1, 11, 3, 11, 11};. What is the value of r(a, 0, 5)?

```
int r(const int a[], size_t i, size_t max)
{
if (i < max) return (a[i] == 11) + r(a, i + 1);
return 0;
}
```

3

5

Stack overflow

1

0

3

[2326] What is the value of r("hello")?

```
string r(const string& s)
{
if (s.size() < 2) return s;
return s.substr(0, 1) + "*" + r(s.substr(1));
}
```

"**h**e**l**l*o"

"h**e**l**l**o*"

"h**e**l**l**o"

Stack overflow

"**h**e**l**lo"

"h**e**l**l**o"

[2327] What is the value of r("hello")?

```
string r(const string& s)
{
if (s.size() > 1) {
string t = s[0] == s[1] ? "*" : "";
return s[0] + t + r(s.substr(1));
}
return s;
}
```

"h**e**ll**o**"
Stack overflow
"h**e**ll*o"
"**h**e**ll**o"
"hel*lo"

"hel*lo"

[2328] What is the value of r("hello")?

```
string r(const string& s)
{
if (s.size() > 1) {
string t = s[0] == s[1] ? "" : "*";
return s[0] + t + r(s.substr(1));
}
return s;
}
```

"h**e**ll*o"
"hel*lo"
"**h**e**ll**o"
Stack overflow
"h**e**ll**o**"

"h*e*ll*o"

[2329] What is the value of r("hello")?

```
string r(const string& s)
{
if (s.size() > 1) {
string t = s[0] == s[1] ? "" : "*";
return t + s[0] + r(s.substr(1));
}
return s;
}
```

"h**e**ll**o**"
Stack overflow
"h**e**ll*o"
"hel*lo"

"*h*el*lo"

"*h*el*lo"

[2330] Which of the following statements is correct about a recursive function?
A recursive function must never call another function.
A recursive function calls itself.
A recursive function must be simple.
A recursive function must call another function.

A recursive function calls itself.

[2331] What does this function do?

```
void myfun(string word)
{
if (word.length() == 0) return;
myfun(word.substr(1, word.length()));
cout << word[0];
}
```

Prints the length of the string word
Prints the string word both forward and reverse
Prints the string word in reverse
Prints the string word

Prints the string word in reverse

[2332] What changes about this function if lines 4 and 5 are swapped?

```
1. void myfun(string word)
2. {
3. if (word.length() == 0) { return; }
4. myfun(word.substr(1, word.length()));
5. cout << word[0];
6. }
```

prints the characters of the string in both forward and reverse order
creates infinite recursion
nothing
reverses the order in which the characters of the string are printed

reverses the order in which the characters of the string are printed

[2333] Which of the following is true about using recursion?

Recursion always helps you create a more efficient solution than other techniques.

A recursion eventually exhausts all available memory, causing the program to terminate

A recursive computation solves a problem by calling itself with simpler input.

None of the listed options.

A recursive computation solves a problem by calling itself with simpler input.

[2334] How can you ensure that a recursive function terminates?

Call the recursive function with simpler inputs.
Use more than one return statement.
Provide a special case for the simplest inputs.
Provide a special case for the most complex inputs.

Provide a special case for the simplest inputs

[2335] Which of the following is a key requirement to ensure that recursion is successful? 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	Every recursive call must simplify the computation in some way.

[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

[2340] Two quantities a and b are said to be in the golden ratio if mc040-1.jpg is equal to mc040-2.jpg. 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 = mc040-3.jpg. In a recursive implementation of this function, what should be the base case for the recursion?
if (number <= 1) { return pow(number, 2.0);}
if (number <= 1) { return sqrt(number);}
if (number <= 1) { return 0.0;}
if (number <= 1) { return 1.0;}
	if (number <= 1) { return 1.0;}

[2341] Two quantities a and b are said to be in the golden ratio if mc041-1.jpg is equal to mc041-2.jpg. 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
 	
If the function double golden (int) is a recursive implementation of this function, what should be the recursive call in that function?
return sqrt (1.0 + golden(number));
return sqrt (1.0 + golden(number - 1));
return (1.0 + golden(number - 1));
return (1.0 + golden(number));
	return sqrt (1.0 + golden(number - 1));

[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 mc042-1.jpg. The formula states that mc042-2.jpgis equal to the limit, as n goes to infinity, of the series mc042-3.jpg. Can this series be computed recursively? Yes, but the code will be very long No, because the base case is not zero Yes No, because there is no base case	Yes

[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 The formula states that equal to the limit, as n goes to infinity, of the series
 	
Which function below is a correct recursive implementation that approximates this infinite series?
	double computePI(int number)
	{
	if (number <= 1) { return 1.0;}
	return 1.0 / (number * number) + computePI(number - 1);
	}

[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 mc044-1.jpg. The formula states that mc044-2.jpgis equal to the limit, as n goes to infinity, of the series mc044-3.jpg. Which statement below is the correct base case for a recursive implementation that approximates this infinite series?
 	
if (number == 0) { return 1.0 / (number * number);}
if (number <= 1) { return 1.0;}
if (number <= 1) { return 0.0;}
if (number == 1) { return (number * number);}
	if (number <= 1) { return 1.0;}

[2345] 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 mc045-1.jpg. The formula states that mc045-2.jpgis 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?

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);

return 1.0 / (number * number) + computePI(number - 1);

[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:

```
double computePI(int number)
{
if (number <= 1) { return 1.0;}
int oddnum = 2 * number - 1;
return computesign(number) * 1.0 / oddnum
+ computePI(number - 1);
}
```

In this recursive function, what is the recursive base case?
When the parameter variable is less than or equal to one
When the parameter variable is greater than one
When the value that is returned from the function is zero
When the parameter variable is zero

When the parameter variable is less than or equal to one

[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:

```
double computePI(int number)
{
if (number <= 1) { return 1.0;}
int oddnum = 2 * number - 1;
return computesign(number) * 1.0 / oddnum
+ computePI(number - 1);
}
```

In this recursive function, what is the role of the helper function computesign?

it is the recursive call in the function

it checks the sign of the number and returns true if it is positive and false if negative

it is called just one time to set the sign of the final result

it makes sure the sign (positive or negative) alternates as each term of the series is computed

it makes sure the sign (positive or negative) alternates as each term of the series is computed

[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?
Call calc_prod(n - 1) and multiply by n.
Call calc_prod(n + 1) and multiply by n.
Call calc_prod(n - 2) and multiply by n.
Call calc_prod(1) and multiply by n.

Call calc_prod(n - 1) and multiply by n.

[2349] Suppose you need to write a recursive function power(double x, int n) that calculates x to the power of n. 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.	Call power(x, n - 1) and multiply by x.